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

ВАСИЛЬКІВ Дмитро Сергійович

Алгоритми виявлення шкідливого програмного забезпечення на основі гешу чутливого до локалізації/ Locale-Sensitive Hash-Based Malware Detection Algorithms

спеціальність: 125 – Кібербезпека та захист інформації

освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи
КБм -21
Д.С. Васильків

Науковий керівник
к.т.н., доцент Н.Г. Яцків

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
ВАСИЛЬКІВ ДМИТРО СЕРГІЙОВИЧ
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Алгоритми виявлення шкідливого програмного забезпечення на основі гешу чутливого до локалізації/ Locale-Sensitive Hash-Based Malware Detection Algorithms
керівник роботи к.т.н., доцент Н.Г. Яцків

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:
– висвітлити популярні типи шкідливого програмного забезпечення;
– провести аналіз сучасних методів виявлення шкідливого програмного забезпечення;
– дослідити принципи роботи LSH та гешів чутливих до місцевості;
– розробити алгоритми виявлення шкідливого програмного забезпечення на основі гешу чутливого до місцевості LSH;
– дослідити ефективність та практичну значущість запропонованих алгоритмів;
– провести аналіз експериментів обраних методів;

5. Перелік графічного матеріалу у роботі:
– генератор гешів SHA1 та SHA2;
– python файл nilsimsa_hash.py;

– результати гешування методами Nilsimsa, TLSH, SSDeep;
– Платформа наявності файлів в базі Virus Total;

6. Консультанти розділів кваліфікаційної роботи

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 12 грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз та дослідження наявних вразливостей	12.2023 р. – 03.2024 р.	
2	Розробка алгоритмів виявлення ШПЗ на основі LSH	03.2024 р. – 06.2024 р.	
3	Експериментальне дослідження та аналіз результатів	06.2024 р. – 11.2024 р.	

Студент _____ Васильків Д.С.
(підпис)

Керівник роботи _____к.т.н., доцент Н.Г. Яцків

АНОТАЦІЯ

Кваліфікаційна робота на тему «Алгоритми виявлення шкідливого програмного забезпечення на основі гешу чутливого до локалізації» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека та захист інформації» освітньо-професійної програми «Кібербезпека» написана обсягом 72 сторінок і містить 14 ілюстрацій, 1 таблиця, 1 додаток та 35 джерел за переліком посилань.

Метою цієї магістерської роботи є дослідження ефективності алгоритмів виявлення шкідливого програмного забезпечення на основі гешування, чутливого до локальності, з використанням локалізації гешування.

Результати дослідження: досліджено принципи роботи LSH та гешів чутливих до місцевості.

Проведено аналіз сучасних методів виявлення шкідливого програмного забезпечення.

Розроблено алгоритми виявлення шкідливого програмного забезпечення на основі гешу чутливого до місцевості LSH.

Досліджено ефективність та практичну значущість методів TLSH, Nilsimsa, SSDEEP

Ключові слова: ГЕШ-ЗНАЧЕННЯ, ШКІДЛИВЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ГЕШІ ЧУТЛИВІ ДО ЛОКАЛЬНОСТІ, НЕЧІТКІ ГЕШІ.

ABSTRACT

The qualification work titled "Algorithms for Detecting Malware Based on Locality-Sensitive Hashing" for obtaining the Master’s degree in the specialty 125 Cybersecurity and Information Protection of the educational and professional program Cybersecurity is written on 72 pages and includes 14 illustrations, 1 table, 1 appendix, and 35 references.

The objective of this master’s thesis is to study the effectiveness of malware detection algorithms based on locality-sensitive hashing (LSH) using localized hashing techniques.

Research Results: examined the principles of operation of LSH and locality-sensitive hashes.

Analyzed modern methods for detecting malware.

Developed malware detection algorithms based on locality-sensitive hashing (LSH).

Investigated the efficiency and practical significance of methods such as TLSH, Nilsimsa, and SSDEEP.

Keywords: HASH VALUES, MALWARE, LOCALITY-SENSITIVE HASHES, FUZZY HASHES.

ЗМІСТ

ВСТУП

Актуальність роботи. З появою технологій людство все більше і більше використовує їх можливості. Європейські держави активно сприяють їх впровадженню в свою діяльність та подальшому розвитку, що забезпечує комфорт громадянам. З 2014 року Україна також стала на шлях діджиталізації, яка полегшує життя людям. Збільшення комп'ютеризації в державному сегменті має позитивні наслідки, проте з ними існують ризики атак з метою крадіжки даних. Під час них злочинці здебільшого використовують шкідливе програмне забезпечення. У свою чергу кіберзахисники, використовують різні способи його дослідження для подальшої боротьби з ним та полегшення його виявлення в майбутньому. Одним з таких методів є аналіз, на основі гешу чутливого до локальності.

Мета і завдання дослідження. Метою цієї магістерської роботи є підвищення ефективності алгоритмів виявлення шкідливого програмного забезпечення на основі гешування, чутливого до локальності. В рамках дослідження для досягнення визначеної мети будуть вирішені такі завдання:

– провести аналіз сучасних методів виявлення шкідливого програмного забезпечення;

– дослідити принципи роботи LSH та гешів чутливих до місцевості;

– розробити алгоритми виявлення шкідливого програмного забезпечення на основі гешу чутливого до місцевості LSH;

– дослідити ефективність та практичну значущість запропонованих алгоритмів.

Об'єкт дослідження - процеси виявлення шкідливого програмного забезпечення.

Предмет досліджень - методи та алгоритми виявлення ШПЗ на основі гешу чутливого до місцевості LSH.

Методи досліджень. При вирішені завдань поставлених у даній магістерській роботі застосовано: аналітичний метод, проведення експериментів для оцінки ефективності розроблених алгоритмів та моделювання процесів виявлення шкідливого програмного забезпечення.

Наукова новизна одержаних результатів. Наукова новизна дослідження полягає в розробці алгоритмів виявлення ШПЗ на основі гешу чутливого до локальності LSH, які мають ряд переваг порівняно з існуючими методами, а саме: підвищену стійкість до еволюції, кращу продуктивність та практичну значущість. Використання цих алгоритмів призведе до підвищення рівня захисту інформаційних систем та зробить значний внесок в розвиток методів виявлення шкідливого програмного забезпечення.

Практичне значення отриманих результатів. Практична значущість дослідження полягає в зборі даних та виконання тестування на реальних прикладах ШПЗ та розробці нових методів виявлення ШПЗ, які можуть бути використані для захисту інформаційних систем.

Публікації та апробація до КР. За результатами наукових досліджень, проведених у кваліфікаційній роботі, підготовлено дві тези доповіді.

1. Васильків Д.С. Виявлення шкідливого програмного забезпечення на основі хешу подібності. Матеріали студентської науково-технічної конференції «Перспективні мережні та комп’ютерні технології» ПерСиК 2024), Харків, 2024.

2. Васильків Д.С. Аналіз файлів за допомогою SSDEEP. Матеріали науково-практична конференція молодих вчених, аспірантів та студентів «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ - 2024), Тернопіль, 2024 – С. 39-42.

1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ НАЯВНИХ ВРАЗЛИВОСТЕЙ

1.1 Огляд основних типів шкідливого програмного забезпечення та їх характеристик

У 2024 році існує безліч різновидів шкідливого програмного забезпечення, проте основних є не так багато. Здебільшого воно ділиться за властивостями, способами реалізації та метою.

Шкідливе програмне забезпечення - це шкідливі програми або набори програм, які мають на меті завдати шкоди людям (користувачам), комп'ютерам або цілим мережам. Ці програми націлені на різні дії без відома чи згоди користувача на них, або отримання її обманим шляхом.

Для спеціалістів з кібербезпеки корисно досліджувати ШПЗ, це допоможе їм вдало типізувати і характеризувати його, а це вже призведе до позитивних наслідків. Таких як: вчасне виявлення та правильне реагування на загрози, відповідно до їх можливих впливів на мережу або систему, розробка та вдосконалення методів захисту, задля здійснення актуального опору, також результати дослідження можна використати для навчання користувачів або навіть персоналу компаній, яким вони надають свої послуги, а це вже сприятиме полегшенню їх діяльності та можливості захисту від простих атак та швидшому виявленню підозрілої активності або повноцінних атак. Ці наслідки фактично допомагають попереджувати атаки та мінімізувати ризики під час них.

Для користувачів ШПЗ може загрожувати: діями з даними, а саме їх крадіжками, блокуванням, руйнуванням або їх видаленням; шпигунство за людьми та їх діями; фінансовими атаками; використанням їхнього ПК задля участі в ботнеті або для розповсюдження з них спаму, також можуть бути інші шкідливі дії. ШПЗ може існувати у різних формах, таких як: віруси (Virus), трояни (Trojan), бекдори (Backdoor), дропери (Dropper), завантажувачі (Downloader), тули (Tool), рекламні програми (adware), (Dialer), черв’яки (Worms), експлойти (Exploit), руткіти (Rootkit) та багато

інших форм шкідливого ПЗ [35]. Далі в тексті будуть подані їх визначення та характеристики.

Вірусами називають програми, які здатні копіювати себе і поширюватися на інші пристрої. Їх основними завданнями є блокування роботи користувача, видалення інформації чи використання пристрою для подальших атак. Основними методами поширення вірусів є сервіси електронного спілкування, заражені носії даних, піратське ПЗ або вразливості в програмах, також є можливість використання мережевих атак. Наслідки діяльності вірусів залежать від їх мети, а саме за ціль може ставиться видалити або пошкодити файли, використати пристрій для пропагації, це розповсюдження вірусу з зараженого пристрою для інфікування інших комп'ютерів і викликання масштабних атак. Найпопулярнішою метою є зараження для створення ботнету для здійснення кібератак, наприклад таких як DDoS. Також, здебільшого заражають пристрої для отримання матеріальних винагород, одним з таких прикладів є шпигунство та крадіжка інформації, задля подальшого продажу, зараження пристрою з метою шифрування даних, та вимагання викупу за розшифрування. Ефективний захист передбачає використання антивірусного ПЗ, обережність під час роботи з файлами з невідомих джерел та регулярне оновлення програм.

Наступним типом ШПЗ є Троянські програми (Trojans). це шкідливе ПЗ, приховане в легітимних додатках, назва якого походить від легенди про Троянського коня. Вперше термін "троян" згадувався у 1974 році, а популярність здобув у 1980-х роках. Сьогодні трояни є найпоширенішою категорією шкідливих програм. Основні призначення цього типу – це проникнення на пристрій користувача та відкриття бекдорів. збір та передачі злочинцеві конфіденційної інформації, блокування пристрою з метою подальшого викупу, шпигування за жертвою (запис натискань клавіш, скріншоти) та віддаленого керування системою. Основною особливістю даного типу є їх самостійність, яка полягає в їх можливості виконувати

зловмисні дії без відомі чи згоди користувача. Троянські програми несуть серйозні наслідки для користувача починаючи втратою контролю над системою та закінчуючи фінансовими втратами. Для уникнення цього типу потрібне навчання користувачів поводження в інтернеті та з програмами та звісно постійне проведення заходів безпеки [6].

Ще одною досить поширеною шкідливою програмою, що дозволяє зловмиснику захопити контроль над системою шляхом отримання доступу до неї є бекдор. Дослівно з англійської бекдор перекладається як задні двері. В цьому перекладі можна передати суть роботи бекдору, а саме подібно до будівника, що залишив непомітний вхід до будинку, хакер використовує бекдор, щоб отримувати доступ до пристрою жертви в потрібний момент. Бекдор встановлюється разом із початковим вірусом або завантажується з мережі після зараження. Він функціонує у фоновому режимі, залишаючись невидимим для користувача, і може: стежити за діями, змінювати або видаляти дані, встановлювати інші програми та викрадати конфіденційну інформацію. Бекдори часто вбудовуються в програми віддаленого доступу чи потрапляють через фішингові атаки, заражені файли або вразливості програмного забезпечення. Захист передбачає уникнення підозрілих вкладень, регулярне оновлення ПЗ, встановлення антивірусів та контроль доступу до системи. [8].

Важливим типом ШПЗ є Дроппер (англ. Dropper), фактично це програма, основною метою якої є таємне встановлення шкідливих програм, наприклад троянців, на пристрій жертви. Даний вид програм досить популярний серед різних зловмисників, через здатністю обходити сигнатури антивірусів. Варто зазначити що дропери як і їх аналоги діляться на два типи, а саме, постійні та непостійні. Інфікування можливе через заражені вкладення, шкідливі посилання, інфіковані сайти або пристрої збереження даних. Для реалізації своїх завдань дропери використовують дірки в системі безпеки, часто маскуються під відомі файли або програми що перебувають на комп’ютері, деколи вони входять у безкоштовні програми, встановлюючи

ШПЗ разом із ними. Для захисту від дроперів CISA рекомендують реалізовувати принципи привілеїв, дотримуватись стратегії zero-trust strategy, блокувати невідомі вкладення, які не вдається перевірити антивірусами та використовувати сегментацію та відокремлення мереж і функцій. та використовувати сегментацію та відокремлення мереж і функцій. [10].

Інший тип ШПЗ це Downloader, поверхнево він не здається шкідливим, і це правда, проте він може виконувати одну шкідливу функцію, а саме оновлення вже наявних вірусів, з метою запобігання їх виявлення антивірусним ПЗ. Також, він може завантажити інше ШПЗ, при умові що це прописали у його коді. Цей підвид вірусів має дуже маленький розмір, тому що основною їх метою є відкриття посилання (звідки завантажується вірус) і саме завантаження. Зазвичай це ШПЗ встановлюється зловмисниками при отримані першого доступу до системи [34].

Небезпечним типом ШПЗ є тули. Вони створені для злому програмного забезпечення, наприклад, HackTool: Win32/Keygen, що використовується для злому Microsoft Office. Вони завдають шкоди не лише компаніям, але й користувачам, які ризикують заразити пристрої вірусами під час використання таких програм. Здебільшого вони націлені на отримання прав адміністратора і після досягнення цього тули можуть: завантажувати інше ШПЗ, змінювати систему, залишатися прихованими до моменту значного пошкодження системи. Антивіруси часто безсилі, якщо користувач ігнорує попередження та дозволяє запуск тула. Тому для захисту від даного типу потрібно дотримуватись трьох правил. Перше, мати встановлений ліцензійний антивірус та регулярно оновлювати його. Друге, повністю відмовитись від програм даного типу та незаконно зламаних програмних продуктів. Третє це, аналізувати попередження антивірусів перед встановленням програм. Фактично, ці правила не захистять вас на 100%, але максимально мінімізують ризик проникнення ШПЗ даного типу [33].

Розповсюдженим типом ШПЗ є Adware (рекламне). Adware – це шкідливе ПЗ, мета якого – показувати небажану рекламу. У 2017 році 45%

атак на ПК в Україні були пов’язані з Adware (дані Zillya!). Воно часто встановлюється разом із легальними програмами та збирає дані про користувача для персоналізації реклами. Також даний тип може спрямовувати на шкідливі сайти, що поширюють віруси, знижувати продуктивність пристрою, вносити зміни в браузери (спливаючі вікна, нові панелі інструментів), встановлювати інше ШПЗ. Основною ознакою пристрою є поява несанкціонованої реклами або додатків. Дієвими методами захисту від даного ШПЗ є встановлення антивірусного ПЗ та уважне встановлення програм. Також варто уникати взаємодії зі спливаючими банерами. Якщо все ж Adware потрапив на ПК, можна використати інструменти для видалення рекламного ПЗ, проте рекомендується попередньо створити резервні копії даних, щоб запобігти їх втраті [32].

Цікавим типом шкідливого програмного забезпечення є Dialer (номеронабирачі). Dialer — це особливий вид троянів, який виконує одну з двох функцій. Він отримавши доступ до вашого пристрою автоматично набирає номер на ньому до якого підключений преміум-тариф, в іншому випадку щоб отримати номер з преміум-тарифом, замінює номер у налаштуваннях комутованого з’єднання з Інтернетом. Варто зазначити що на даний час він рідко використовується в Україні. Проте обов’язково потрібно запобігати зараженню комп'ютера будь-якими вірусами, і для цього використовуються антивірусне ПЗ [13].

Одним з найвідоміших типів ШПЗ є Worms (хробак). Черв’яки – це шкідливе ПЗ, що швидко розмножується та поширюється через мережі, споживаючи ресурси і знижуючи продуктивність систем. Вони здатні видаляти файли, змінювати їх і поширювати інші шкідливі програми. Черв’яки потрапляють на пристрої завдяки вразливостям ПЗ, спам розсилкам та через підключення до попередньо заражених накопичувачів. Атака шляхом зараження черв’яком, фактично, має три стадії. На першій стадії відбувається Інфікування пристрою через вразливості або шкідливі файли. На другій, черв’як поширюється на інші пристрої в мережі, перевантажуючи

систему, знижуючи пропускну здатність і продуктивність. На останній стадії зловмисник отримує права адміністратора, краде дані або завдає шкоди системам. Після поширення цього ШПЗ по всій системі, він продовжує автоматично поширюватися, по даному алгоритму. Таким він розповсюджується і на інші системи. Для захисту від атаки хробаків також використовується спеціальний алгоритм дій. На першому етапі локалізує зараження пристрою для обмеження поширення. Під час другого кроку, проводиться перевірка та усунення вразливостей на інших пристроях. Третя дія яку виконує захисник якщо видалити не можливо, це подає пристрій на карантин. І на останньому етапі може знадобитися переустановлення системи для повного очищення. Фактично усуває атаку та виправляє наслідки для систем та пристроїв [31].

Якщо розглядати експлойти детально, можна зрозуміти що це програми, що використовують вразливості систем або програм для отримання доступу до них. Вони можуть застосовуватися законно, але часто стають інструментом зловмисників. Найчастіше експлойти атакують через браузери, де знаходять слабкі місця безпеки.. Існують дві основні категорії експлойтів, невідомі та відомі. Невідомі експлойти використовують вразливості, які ще не виправлені розробниками, а відомі експлойти знайдені та виправлені у рамках оновлень ПЗ. Також експлойти можуть автономно поширювати інше ШПЗ, наприклад трояни віддаленого доступу (RAT), що надають хакерам контроль над даними користувача. Для захисту потрібно регулярно встановлюйте оновлення безпеки для системи, браузерів і плагінів, також варто правильно налаштувати та ввімкнути брандмауер та використовувати антивірус Ці дії значно знижують ризики, пов’язані з експлойтами. [22].

І останніми з наведених в моєму переліку типів ШПЗ, будуть руткіти. Це ШПЗ, що приховує інші віруси й створює бекдори для віддаленого доступу. Вони містять набори інструментів для зараження пристроїв, вимкнення антивірусів та викрадення даних. Залежно від цілі зараження

можна виділити п’ять типів руткітів це: завантажувача, пам’яті, додатків і режиму ядра та апаратні. Останні з перелічених це ті що заражають жорсткі диски, BIOS або мікропрограми на материнській платі, впливають на маршрутизатори. Руткіти режиму ядра дозволяють хакерам отримати глибокий контроль над пристроєм, тому що вони модифікують ядро ОС. Руткіти додатків націлені на стандартні програми, такі як Word або Notepad. Наступні ті руткіти що працюють з оперативною пам’яттю задля зниження продуктивності. Ще є ті що модифікують головний завантажувальний запис (MBR) це руткіти завантажувача. Варто зазначити що кожен тип руткітів є складним для виявлення, проте до ефективних методів цього можна віднести сканування сигнатур, аналіз дампу пам’яті та поведінковий аналіз. Варто зазначити що існують різні програми не лише для виявлення але й видалення даного типу ШПЗ, проте перед тим як їх використовувати, потрібно здійснювати регулярне резервне копіювання даних, щоб уникнути втрат. Для ефективного видалення потрібен перехід у безпечний режим та використання кількох сканерів. У складних випадках, як зараження BIOS, може знадобитися переустановлення ОС або очищення пристрою. Щоб уникнути зараження, слід дотримуватись правил безпеки: уникати фішингу, регулярно оновлювати ПЗ, використовувати антивіруси, моніторити мережевий трафік. [30].

1.2 Принцип роботи алгоритмів LSH та їх використання в області виявлення ШПЗ

Цікавим методом виявлення шкідливого програмного забезпечення, є спосіб на основі використання алгоритмів гешу чутливого до локальності. Для повного висвітлення даного питання варто почати з пояснення що таке гешування. Отже, це процес, який перетворює вхідні дані будь якого розміру у вихідний бітовий рядок фіксованої довжини. Вихідні результати бувають різного розміру, проте вони залежні від алгоритмів які мають постійні

розміри. Серед них алгоритм SHA-256 видає 256-бітні виходи, а SHA-1 завжди 160-бітні. Важливим є те що геш при будь якій зміні у вхідних даних є різним. А саме на першому зображені було перетворено фразу Приклад гешу, з використанням алгоритму SHA-256 та отримано геш *66DD908C210090B80146983DA60541C3DE8C40218B8013F5BB73FE30E99C9412*. На другому зображені її було змінено, а саме замість великої букви у першому слові поставлено малу, а у другому навпаки. В результаті отримано геш *72621B4615861E5A01052E8E8E2C4AEBFDAEE6F0493954222016C29E4D82BC01*, що у порівняні з попереднім є іншим. Отже це є перша властивість гешу, а саме невелика зміна в повідомленні змінює геш настільки сильно, що нове та старе значення здаються некорелюючими. Варто зазначити що при використанні інших алгоритмів результат буде інший тому що вони мають виходи іншої довжини, хоча при проведені аналогічного експерименту їх результати також будуть змінюватись. Якщо вже говорити про основні алгоритми то їх абревіатура SHA перекладається як безпечні алгоритми шифрування, а в оригіналі це Secure Hash Algorithms. Фактично до них входять геш-функції, які включають алгоритми SHA-0 та SHA-1 і групи SHA-2 та SHA-3. До групи SHA-2 входять поширені алгоритми SHA-256 та SHA-512. Важливо що лише групи SHA-2 та SHA-3 зараз вважаються безпечними. Стосовно застосування самого процесу гешування то він є досить поширеним та використовується у різних сферах, найчастіше у програмуванні, криптографії та кібербезпеці.

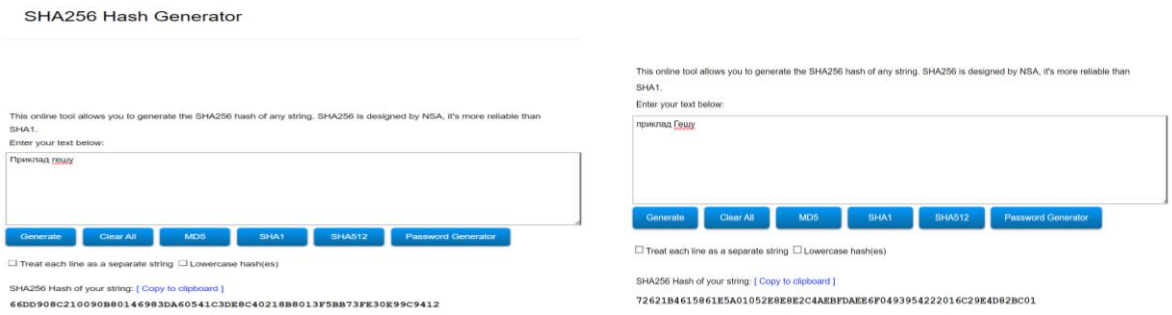

Рисунок 1.1 – Приклад гешу, з використанням алгоритму SHA-1 та SHA-256

Гешування з урахуванням локальності (LSH) − це метод, який дає можливість ефективного наближення пошуку схожих елементів шляхом зменшення розміру даних, при цьому дає можливість зберегти відстані між близькими точками. Основна ціль LSH полягає в застосування гешуванні точок даних таким чином, щоб схожі фрагменти з високою ймовірністю потрапляли в одні й ті ж геш-кошики. Ці дії дозволяють суттєво зменшити простір пошуку та прискорити процес знаходження подібних об'єктів.

Алгоритм LSH складається з трьох основних етапів:

1. Гешування: на цьому етапі створюються кілька геш-функцій, кожна з яких генерує геш-код для кожної з точок даних. У більшості випадків ці геш-коди містять у собі двійкові рядки однакової довжини.

2. Групування: тут відбувається певне сортування, головну роль у якому відіграють геш-коди, які переміщують у геш-кошики. Однакові геш-коди потрапляють в один геш-пакет, а це означає що і точки відповідають одна одній та опиняються в одному сегменті.

3. Запит: під час цього етапу до нас потрапляє елемент запиту, який ми пропускаємо через той самий процес гешування для отримання геш-коду. Потім ми шукаєм геш-пакет в який потрапив геш-код елемента запиту. Варто зазначити, що всі елементи які знаходяться у цьому геш-пакеті, вважаються потенційно схожими із запитом, для остаточного вердикту ми виконуємо уточнену перевірку. Саме тоді відбувається точний аналіз подібності в якому використовуються вихідні багатовимірні дані, які допоможуть визначити реальну схожість.

Ефективність LSH обумовлена трьома основними факторами:

1. Кількість геш-функцій (K), чим більше їх тим вища ймовірність того, що схожі елементи потраплять в один геш-кошик. Проте це впливає на два важливі аспекти, а саме збільшення тривалості обчислення та витрат на нього.

2. Довжина геш-кодів (L) чим вона більша тим точнішими будуть результати, проте даний фактор співвідноситься з розміром пам’яті

потрібним для його зберігання. Не завжди можливо знайти стільки вільного простору у пам’яті пристрою та часто це буває не доречно. Тому потрібно враховувати компроміс між довжиною та точністю залежно від умов та бажаного результату.

3. Кількість геш-таблиць (M) цей фактор сприяє виконати кращий та достовірний пошук, тому що ми можемо шукати схожі елементи серед більшої їх кількості.

Головною особливістю методу LSH є використання геш-функції, яка зберігає локальність. Дана функція зберігає відносну локальність точок даних у геш-просторі так само, як і в оригінальному просторі. Іншими словами, дану геш-функцію, можна висвітлити у формулі. Нехай геш-функція називатиметься буквою *h*, якщо $dist(h(q),h(r))>dist(h(p),h(q))$ і $dist(q,r)>dist(p,q)$, то дана h буде геш-функцією із збереженням локальності.

Цікаво те що коло використання методу LSH є досить широке. Для розуміння цього варто розглянути 3 практичних приклади. Для кожного з них будуть змінюватись та використовуватись інші технології це потрібно для демонстрації різноманітності підходів для вирішення індивідуальних проблем. Першим прикладом буде розпізнавання сутностей, а наступними будуть пошуки відповідності відбитків пальців та статей (публікацій).

Розпізнавання сутностей використовується для виявлення об’єкту за допомогою аналізу та звірки наявних записів. Наприклад на практиці розглянемо колекцію записів, де кожен запис містить інформацію про певний об’єкт (найчастіше це люди, але також можуть бути компанії, фізичні локації, події тощо). Проблема розпізнавання сутностей полягає у виявленні наборів записів, які відносяться до одного й того самого об’єкта, та злитті їх в один узагальнений запис, що містить усю наявну інформацію про сутність. Це завдання є складнішим, ніж може здаватися на перший погляд. Наприклад, записи про людей часто містять імена. Спочатку здається, що згрупувати записи за іменами не повинно викликати труднощів. Однак у великій колекції можуть бути люди з однаковими іменами, і це може призвести до

об'єднання записів про різних осіб. Крім того, ім'я однієї й тієї ж людини може бути записане по-різному в різних записах, а також можуть бути орфографічні помилки. Щоб компенсувати такі розбіжності, ми можемо використовувати інші поля з записів, як-от номери телефонів чи адреси. Якщо два записи мають схожі імена, але ідентичні номери телефонів, то ми можемо припустити, що це записи про одну й ту саму особу. Уявімо ситуацію, де у нас є дві бази даних: одна з клубу любителів книг, а інша з групи фанів комп'ютерних ігор. Наше завдання — знайти людей, які є учасниками обох груп, але зареєструвалися у групі фанів комп'ютерних ігор після їх запрошення туди з клубу любителів книг. Обидві бази містять по мільйону записів, кожен з яких містить ім’я, прізвище, номер мобільного телефону, дату народження, дату реєстрації, електронну пошту та хобі. На перший погляд, це виглядає просто: можна порівняти лише імена та прізвища. Проте тут виникає низка проблем: люди можуть записувати свої імена по-різному. Наприклад, "Дмитро" може бути записано як "Dmytro", або ж використовувати скорочення, як-от "Діма". Такі варіанти написання ускладнюють пошук відповідників між двома базами даних. Щоб уникнути таких неточностей, можна залучити додаткові критерії для порівняння: номери телефонів, електронні адреси, дати народження та хобі. Для визначення збігів ми можемо надати кожному полю певну кількість балів. Наприклад, загальна кількість балів становить 100, а кожен збіг за певним критерієм приносить 20 балів. Якщо три поля збігаються, то запис отримає 60 балів, і ми можемо вважати, що це одна й та сама особа. Наступний крок — створення п'яти геш-функцій. Перша функція обробляє імена та прізвища, друга — номери телефонів, третя — електронні адреси, четверта — дати народжень, а п'ята використовує хобі як критерій. Спочатку сортуємо записи за іменами, після чого всі записи з однаковими іменами з'являються послідовно у списку. Це дозволяє підрахувати кількість пар з однаковими іменами. Аналогічну процедуру застосовуємо до номерів телефонів. Ідея полягає в тому, що ми гешуємо записи за певними полями, розміщуючи їх у

кошики. Пари кандидатів, які перебувають в одному кошику за хоча б одним з критеріїв, стають потенційними відповідниками. Звичайно, іноді деякі пари можуть бути пропущені через мінімальні розбіжності, але їх кількість буде незначною. Наприклад, пара записів може отримати 60 балів зі 100 можливих. Це свідчить про значну схожість між записами, хоча вона й не є повною. Чи належать ці записи одній і тій самій людині? Дуже ймовірно, що так. Щоб бути впевненими, ми можемо використовувати додаткові методи оцінки, як-от дата створення записів. Якщо середня різниця між датами реєстрації становить 10 днів, то це свідчить про те, що записи, найімовірніше, стосуються однієї особи. Якщо ж різниця в 45 днів, то це, швидше за все, різні люди. Такий підхід дозволяє оцінити ймовірність правильного збігу, враховуючи додаткові фактори.

Пошуки відповідності відбитків пальців. У цьому прикладі локально-чутливого гешування є велика база відбитків пальців, з якої потрібно визначити, які пари належать одній і тій самій людині. Завдання полягає не в тому, щоб одразу знайти всі збіги, а в тому, щоб порівнювати новий відбиток з тими, що вже є в базі. Локально-чутливе гешування ефективно організовує базу даних, дозволяючи швидко перевіряти, чи є збіг у невеликій кількості кошиків. Спершу слід зрозуміти, як представлені відбитки. Вони аналізуються на основі "дрібниць" (minutiae) – унікальних особливостей, таких як закінчення або злиття ліній. Зображення відбитка можна подати у вигляді набору координат у двовимірному просторі, які відповідають цим особливостям. Кожен відбиток пальця покривається сіткою, яка має бути масштабована та орієнтована таким чином, щоб, якщо два зображення одного пальця мали різні розміри, їх сітки все одно збігалися. Потім кожен відбиток представляється множиною квадратів цієї сітки, які містять деталі. Іноді деякі деталі розташовані близько до межі квадратів, тому їх варто вважати такими, що належать одразу кільком квадратам. Задача пошуку збігів тепер зводиться до пошуку схожих наборів квадратів сітки, що містять ці деталі. Основна проблема полягає в тому, що сітка не може бути занадто

дрібною, інакше складно буде визначити, до якого саме квадрата належать деталі. У цьому випадку мінгешинг буде менш ефективним. Припустимо, що деталі займають 20% площі сітки. Якщо два відбитки пальців належать одній людині, тоді принаймні 80% квадратів з одного відбитка мають збігатися з іншим. Якщо ж відбитки належать різним людям, то ймовірність того, що вони виявляться в одному кошику, дуже мала. Наприклад, ймовірність, що відбитки з різних пальців матимуть деталі в усіх квадратах, дорівнює (0.2) 6 = 0.000064. У випадку двох відбитків одного пальця ймовірність значно вища. Якщо для одного квадрата ймовірність наявності деталей становить 0.2, а для другого — 0.8, то ймовірність для обох одночасно дорівнює (0.2*0.8)3 = 0.004096. Ця ймовірність у 64 рази більша, ніж у випадку різних пальців. Ми маємо 1024 набори з трьох квадратів, і щоб пара відбитків стала кандидатом на збіг, достатньо, щоб вони потрапили разом хоча б в один кошик. Ймовірність цього становить 98,5%.

І останній приклад це пошуки відповідності статей (публікацій). Ця проблема стосується пошуку новинних статей, які описують одну й ту ж подію. Оскільки різні джерела можуть представляти одну історію по-різному, це нагадує задачу пошуку схожих документів. Проте, у цьому випадку стандартна технологія шинглінгу не підходить, оскільки вона розглядає всі частини документа однаково. Натомість, ми хочемо ігнорувати деякі частини, наприклад, рекламу або посилання на інші статті, що не є частиною новини. Виявляється, що текст основної статті та текст, який з'являється у заголовках або рекламі, значно відрізняються. Основна частина новини містить набагато більше стоп-слів (наприклад, "і", "але"). Стоп-слова — це часто вживані слова, які можна ігнорувати для покращення аналізу тексту. Для обробки тексту ми можемо використовувати шингли (послідовності слів). Наприклад, припустимо, що ми визначаємо шингл як стоп-слово з двома наступними за ним словами.

Таким чином, рекламна фраза "Вболівай за Збірну" не міститиме шинглів, оскільки вона дуже коротка і не відповідає цьому формату. Проте

речення "Нам потрібно вболівати за збірну України у відборі на ЧС2026" буде представлено чотирма шинглами: " Нам потрібно ", " вболівати за збірну України", "у відборі на ЧС2026", і " у відборі на х" (де х — це будь-яке наступне слово). Якщо у нас є дві веб-сторінки, кожна з яких на половину складається з новин, а на іншу половину — з реклами або іншого матеріалу, що містить мало стоп-слів, ми можемо порівняти їх. Якщо новинний контент ідентичний, але інший матеріал різний, то значна частина шинглів буде збігатися, і жакардова схожість між веб-сторінками може становити 75%. Якщо ж новинний контент різний, а навколишній матеріал схожий, схожість між шинглами може знизитися до 25%. Якщо б використовувався стандартний метод шинглінгу (наприклад, послідовності з 10 символів), ми б очікували приблизно однакову жакардову схожість незалежно від того, чи співпадає новинний текст, чи навколишній контент.

Узагальнюючи, LSH є ефективним методом для роботи з великими обсягами даних. Його основна перевага – швидкий пошук схожих елементів без перевірки всіх можливих пар, що зменшує обчислювальну складність. Цей метод широко застосовується для виявлення схожих документів, зображень чи аудіо файлів у великих базах даних, пошуку найближчих сусідів у системах рекомендацій, кластеризації даних для виявлення шаблонів і структур. Він базується на геш-функціях, які групують схожі елементи в однакові сегменти, зменшуючи простір пошуку. Завдяки своїй гнучкості та можливості адаптації до різних метрик подібності, LSH залишається ключовим інструментом у сфері аналізу даних і машинного навчання.

1.3 Порівняння алгоритмів виявлення ШПЗ на основі LSH з іншими підходами

Виявляти шкідливе ПЗ з року в рік стає все складнішим завданням. Це пов’язано із тим, що його автори аналізують традиційні методи його

виявлення, та сприяють його розвитку та мутації. В сучасному світі існують різні способи для дослідження та виявлення ШПЗ. Серед популярних підходів є використання методів на основі схожості, фактично до них належить і досліджуваний нами Locality-Sensitive Hashing (LSH). Також відомими підходами є машинне навчання, сигнатурний та поведінковий аналізи. Всі ці вони мають свої переваги й недоліки, що впливають на їхню ефективність у конкретних сценаріях виявлення загроз.

Locality-Sensitive Hashing (LSH) та інших підходів варто розпочати з основних принципів роботи. LSH це метод гешування, що призначений для виявлення схожих об'єктів у великих обсягах даних. Алгоритм LSH перетворює вхідні дані на геші таким чином, що схожі об'єкти мають великі шанси потрапити до одного й того ж "відра" (bucket). Це дозволяє швидко шукати схожі файли та виявляти ШПЗ, яке маскується як легітимне програмне забезпечення, але має схожу структуру або поведінку з відомими загрозами.

У свою чергу машинне навчання використовуючи свої алгоритми, найчастіше це методи класифікації та регресії, використовують різні характеристики файлів, щоб навчити модель відрізняти шкідливі та легітимні програми. Ці моделі створюються на основі аналізу великих наборів даних, що включають як шкідливі, так і легальні зразки. Методи аналізу включають нейронні мережі, дерева рішень, SVM, градієнтний бустинг та інші [28].

Виявлення ШПЗ на основі сигнатурного аналізу відбувається за рахунок використання заздалегідь зібраних "сигнатур" відомих зразків ШПЗ. Сигнатури це унікальні послідовності байтів або інструкцій, що відображають структуру шкідливого коду. Сигнатурні бази даних регулярно підлягають оновленню для включення нових загроз, і при скануванні файлів відбувається порівняння їх вмісту з цими сигнатурами [5].

Поведінковий аналіз зосереджується на вивченні дій програмного забезпечення під час його виконання. Замість аналізу статичних характеристик, поведінковий аналіз відстежує, як програма взаємодіє з

системою, які системні виклики вона робить, які файли або мережеві ресурси використовує, і на основі цього визначає, чи є програма шкідливою. Це дозволяє виявляти загрози, які можуть обійти статичний аналіз через обфускацію або динамічні зміни коду [24].

Наступним пунктом для висвітлення різниці між алгоритмами стануть швидкість і продуктивність. LSH дозволяє швидко знаходити схожі файли навіть у великих наборах даних, оскільки процес гешування та розподілу в "кошики" зменшує кількість порівнянь. Це робить алгоритм ефективним для аналізу великих баз даних, де потрібно знаходити схожі об'єкти. Висока продуктивність досягається завдяки скороченню кількості файлів, які потребують детального аналізу. Однак ефективність може знизитися при налаштуванні надто великої кількості геш-функцій, при роботі з даними, які погано піддаються гешуванню або мають багато шумових елементів та при роботі з файлами які не містять достатньо схожих ознак для точного порівняння.

Машинне навчання може бути ресурс затратним і займати багато часу. Проте після завершення тренувань моделів їх використання для класифікації є швидким. Моделі машинного навчання можуть працювати на великих наборах даних і водночас забезпечувати високу продуктивність при правильному налаштуванні гіперпараметрів. Проте, як вже писалось раніше їх навчання потребує значної кількості даних і часу.

Сигнатурний аналіз працює швидко для виявлення відомих загроз, оскільки процес зводиться до простого порівняння з базою даних. Однак для нових або модифікованих загроз цей підхід є менш ефективним. Він відносно продуктивний при роботі з великим обсягом файлів, але це залежить від оновлення бази сигнатур. Без цього його ефективність різко падає, коли виникають нові загрози.

Поведінковий аналіз є повільнішим у порівнянні з LSH та іншими методами, оскільки він вимагає виконання коду в середовищі, яке відстежує його поведінку. Це включає моделювання або віртуалізацію роботи файлу та

спостереження за його діями через необхідність запуску програм у контрольованому середовищі (пісочниця або емулятор), він може бути значно повільнішим та вимагати більше ресурсів для аналізу великої кількості зразків одночасно.

Іншим критерієм оцінювання є ефективність даних методів. LSH може бути ефективним для виявлення модифікованих або обфускованих зразків ШПЗ, оскільки він орієнтований на пошук схожих даних, а не точних збігів. Це дозволяє виявляти нові версії відомого шкідливого ПЗ. Проте він менш ефективний для виявлення абсолютно нових загроз, які не мають схожих ознак з уже відомими зразками. Його основний фокус на схожості обмежує здатність виявляти суттєво відмінні типи загроз.

Моделі машинного навчання також можуть виявляти нові типи загроз, якщо навчальні дані включають різноманітні характеристики шкідливого ПЗ. Ці методи можуть враховувати поведінкові ознаки, що робить їх гнучкими до нових, незнаних загроз. Але нажаль його ефективність залежить від якості та обсягу навчальних даних. Якщо модель навчалася на обмеженому наборі зразків, її здатність виявляти нові загрози буде обмеженою.

Сигнатурний аналіз є дуже точним та надійним для виявлення відомих загроз, він має низький рівень помилкових спрацьовувань, оскільки алгоритм працює з точними сигнатурами. Але на відміну від попередніх методів він вкрай вразливий до нових загроз, які не присутні в базі даних. Віруси, трояни або руткіти, які були модифіковані, можуть уникнути виявлення, оскільки їхня сигнатура відрізняється від тієї, що зберігається у базі.

Поведінковий аналіз один з найефективніших методів для виявлення нових загроз, оскільки він орієнтується не на структуру файлів, а на їхню активність у системі. Це дозволяє виявляти раніше невідомі загрози, які можуть маскуватися за допомогою статичних методів. Проте іноді навіть цей метод можуть сповільнити або навіть обійти складні методи обфускації або специфічні техніки ухилення.

Ще одним індикатором порівняння алгоритмів виявлення ШПЗ є можливість їх адаптації до нових атак. LSH добре підходить для виявлення модифікованих або обфускованих версій відомих загроз, оскільки він здатен виявляти схожість між файлами. Проте цей підхід обмежений в адаптації до нових типів атак, які суттєво відрізняються від попередніх. Для його адаптації до нових атак потрібна зміна геш-функцій або методів гешування, що може ускладнити його використання для повністю нових видів ШПЗ.

У свою чергу методи машинного навчання більш гнучкі, оскільки можуть вивчати нові зразки та оновлювати модель на основі нових даних. Вони можуть бути легко адаптовані до нових загроз шляхом додавання нових зразків у тренувальні дані та проведення повторного навчання.

Поведінковий аналіз також може адаптуватися до нових загроз, оскільки він не залежить від структури файлу. Будь-яка підозріла активність у системі може бути виявлена, навіть якщо вона не відповідає жодним відомим сигнатурам. Проте для складних атак, які використовують специфічні техніки ухилення, може знадобитися його додаткове налаштування.

Наступним пунктом для висвітлення різниці стане інформація про помилкові спрацьовування. LSH може генерувати більшу кількість помилкових позитивних спрацьовувань (false positives) ніж сигнатурний аналіз, оскільки орієнтується на схожість між файлами, яка не завжди означає шкідливу активність. Легітимні файли можуть мати подібні структури з шкідливими. Машинне навчання може також давати помилкові позитивні або негативні результати, особливо якщо модель була неправильно навчена або якщо вибірка для навчання була нерепрезентативною. Поведінковий аналіз зазвичай має нижчий ризик помилкових спрацьовувань, оскільки він зосереджується на аналізі дій програми. Програми, які поводяться нормально, не будуть визначені як шкідливі, навіть якщо їх структура схожа на шкідливі програми. Проте, часто поведінка, а саме запити навіть легітимних програм, можуть бути схожими на запити шкідливих програм. Сигнатурний аналіз зазвичай має найнижчий рівень помилкових

спрацьовувань, оскільки він порівнює точні сигнатури з відомими шкідливими програмами.

Якщо ми розглянули інформацію про помилкові спрацьовування, то варто приділити увагу і близькій за сенсом точності виявлення. LSH добре працює для виявлення відомих або модифікованих загроз, що дозволяє виявляти навіть трохи змінені зразки ШПЗ, проте точність виявлення може знижуватися для абсолютно нових загроз або загроз, які суттєво відрізняються від раніше відомих.

Висока точність машинного навчання можлива тільки при наявності достатньої кількості навчальних даних, що включають різні варіанти ШПЗ, але при відсутності якості даних і здатності моделі виявляти патерни у нових типах загроз, точність нівелюється.

Поведінковий аналіз може забезпечити високу точність виявлення, оскільки зосереджується на аналізі фактичної поведінки програми. Це дозволяє виявляти навіть нові загрози, що не мають прямих аналогів серед відомих зразків, але іноді складні техніки ухилення можуть знизити точність виявлення, особливо якщо шкідлива програма здатна ховати свою шкідливу активність або запускати її лише в певних умовах.

І на завершення варто відзначити застосування в реальних умовах. LSH підходить для швидкого аналізу великих обсягів даних і виявлення загроз, що мають схожі структури. Він може бути корисним у хмарних середовищах або великих корпоративних мережах, де потрібно обробляти багато файлів одночасно. З складного у цьому методі є налаштування його алгоритму для забезпечення балансу між швидкістю і точністю в реальних умовах.

Машинне навчання ефективне для складних середовищ, де загрози можуть мати різноманітні поведінкові або структурні ознаки. Машинне навчання застосовується у складних системах захисту від загроз та автоматизованих платформах. Проте його використання потребує даних у великих обсягах для тренування моделей та постійного оновлення для забезпечення актуальності захисту.

Сигнатурний аналіз найкраще підходить для середовищ, де основна загроза полягає у відомих видах шкідливих програм, а швидке виявлення є критично важливим (наприклад, антивірусні програми). Проте йому також потрібне регулярне оновлення бази сигнатур, щоб забезпечити належний рівень захисту, що може вимагати великих зусиль при появі нових загроз.

Поведінковий аналіз підходить для виявлення нових і складних загроз у динамічних середовищах, де шкідливі програми можуть використовувати техніки обфускації або змінювати свою структуру під час виконання. З його основних мінусів це те що він вимагає більше ресурсів і часу для виконання, що може бути проблематичним при великій кількості зразків або при аналізі в реальному часі.

Підводячи підсумок, то усі методи мають свої плюси та мінуси. Алгоритми виявлення ШПЗ на основі LSH є ефективними для швидкого пошуку схожих загроз, особливо в великих наборах даних, але можуть не виявити радикально нові загрози. Методи машинного навчання, в свою чергу, забезпечують більшу гнучкість та здатність адаптуватися до нових загроз, але потребують більших ресурсів для навчання та підтримки. Водночас сигнатурний аналіз забезпечує точне та швидке виявлення відомих загроз, але не є ефективним у випадках, коли загроза змінюється або є новою. Поведінковий аналіз, з іншого боку, забезпечує вищу гнучкість і точність при виявленні нових загроз, оскільки він аналізує дії програми в реальному часі. В ідеальному випадку всі ці підходи можуть доповнювати один одного, забезпечуючи як точність виявлення нових загроз, так і ефективність у роботі з модифікованими зразками. Це все потрібне для досягнення максимального захисту.

2 РОЗРОБКА АЛГОРИТМІВ ВИЯВЛЕННЯ ШПЗ НА ОСНОВІ LSH

2.1 Проектування архітектури системи виявлення ШПЗ на основі LSH

Розробка алгоритмів виявлення ШПЗ на основі LSH досить важкий та затратний по часу процес, його варто розпочинати з проектування архітектури системи виявлення. Для початку слід назвати основні компоненти системи, серед них є різні модулі, а саме попередньої обробки даних, гешування, індексації та пошуку, виявлення та аналізу, інтерфейс користувача.

Модуль попередньої обробки даних є ключовим елементом системи виявлення шкідливого програмного забезпечення (ШПЗ) на основі локально чутливого гешування (LSH). Його основна мета - підготувати вхідні дані для подальшого аналізу, забезпечуючи їхню однорідність і готовність до ефективного гешування та порівняння. Серед його основних функцій є збір даних, фільтрація даних, їх нормалізація та фрагментація. Перший крок полягає в зборі даних для аналізу. Це можуть бути бінарні файли, фрагменти коду або інші форми даних, що містять потенційні ознаки ШПЗ. Вони можуть бути взяті з різних репозиторіїв зразків ШПЗ, скомпрометованих систем або спеціалізованих баз даних, що містять відомі шкідливі файли. Для того щоб відсіяти некорисні або нерелевантні дані для аналізу ми також використовуємо цей модуль. Саме він допомагає видалити шумові або неінформативні фрагменти, які можуть вплинути на точність подальшого гешування. Нормалізація забезпечує перетворення цих даних у стандартний формат, зручний для аналізу, тому що вхідні дані можуть мати різні формати та структури. Наприклад, це може бути приведення різних бінарних форматів до єдиної структури або вилучення ключових ознак з тексту. Нормалізація також може включати масштабування даних, обробку пропущених значень, а також видалення зайвих символів або зайвої інформації, яка не впливає на процес виявлення ШПЗ. Якщо вихідні дані представлені у вигляді великих файлів або довгих фрагментів коду, цей модуль може виконати їхню

фрагментацію, а саме розділити дані на менші блоки. Це дозволяє більш детально аналізувати і порівнювати частини файлів, що може бути критично важливим для виявлення невеликих або прихованих сегментів ШПЗ. Ця функція допомагає виявляти схожість між частинами різних файлів, навіть якщо вони обфусцифіковані або трохи змінені. Також варто відзначити що модулем попередньої обробки даних ставиться завдання з виділення особливих ознак або характеристик, які можуть вказувати на наявність шкідливого ПЗ. Це можуть бути характерні послідовності байтів, сигнатури, строки, команди або інші унікальні патерни. Ці виділені ознаки передаються далі на етап гешування для їхнього подальшого аналізу за допомогою LSH.

Модуль гешування є центральним компонентом системи виявлення ШПЗ на основі LSH. Його основне завдання полягає у перетворенні підготовлених даних у геші, що зберігають інформацію про схожість між різними фрагментами даних. Це забезпечує швидкий і ефективний пошук та порівняння даних. Цьому модулю притаманні функції вибору, створення гешів, їх індексація та оптимізація ну і звісно зберігання. Для виконання функції вибору потрібно знайти та використати потрібні геш-функції, які зможуть вловити схожість між даними. У випадку LSH використовуються спеціальні геш-функції, які забезпечують високий рівень колізій для схожих даних. В залежності від типу даних можуть використовуватися різні варіанти геш-функцій, такі як MinHash, SimHash, або інші, спеціально адаптовані під специфіку вхідних даних. Після вибору геш-функцій, дані з попереднього модуля обробки передаються для створення гешів. Гешування може виконуватися як для всього об'єкта (наприклад, файлів), так і для його окремих фрагментів. Коли геші створюються для кожного фрагмента даних, то це сприяє ефективному порівняні різні фрагменти між собою. Важливо, щоб схожі дані давали однакові або схожі геші, що є основою для подальшого пошуку та виявлення ШПЗ. Створені геші не зберігаються у сирому вигляді, а індексуються для швидкого доступу. Індексація дозволяє організувати геші таким чином, щоб забезпечити миттєвий пошук подібних

даних. Можуть використовуватися різні методи індексації, зокрема побудова дерев, геш-таблиць, або спеціальних структур даних для організації гешів. Детальніше ці методи будуть розглянуті в самому модулі індексації, тому що їх робота досить схожа. І останньою функцією модуля гешування є зберігання цих гешів. Вони зберігаються у спеціальній базі даних або іншій структурі зберігання. І це забезпечує швидкий доступ до гешів для подальшого пошуку та порівняння нових зразків з існуючими.

Модуль індексації та пошуку є критичним компонентом системи виявлення шкідливого програмного забезпечення (ШПЗ) на основі локально чутливого гешування (LSH). Його основна мета - забезпечити ефективне зберігання, організацію та швидкий пошук гешів, створених модулем гешування, для виявлення схожих зразків ШПЗ. Його основними функціями є індексація гешів, пошук схожих зразків та його оптимізація, управління даними та оновлення індексу і на завершення обробка результатів пошуку. Після створення гешів модулем гешування, модуль індексації відповідає за їх організацію в структурі, яка дозволяє швидкий доступ і пошук. Основна задача індексації - організувати геші так, щоб схожі дані були розташовані близько один до одного. Одним із найпростіших і поширених методів індексації є геш-таблиці. Вони дозволяють швидко знаходити геші за ключем, що зменшує час пошуку [2]. Геш-таблиця працює за принципом гешування, де кожен геш обробляється за допомогою геш-функції, що перетворює його у індекс (геш-значення), який вказує на позицію в масиві або іншій структурі, де зберігається відповідне значення. Відповідно до обчисленого індексу, значення (або набір значень) зберігається в певній позиції геш-таблиці. Це дозволяє швидко знаходити дані за ключем, використовуючи ту саму геш-функцію. Оскільки можливих ключів може бути більше, ніж доступних індексів у геш-таблиці, два різних ключі можуть генерувати однакове геш-значення. Це називається колізією. Для їхнього вирішення використовуються різні методи. Одним із найпоширеніших методів є ланцюгове гешування (Chaining), при його використані всі

значення, що відповідають одному індексу, зберігаються у вигляді списку або зв'язаного списку (linked list). Якщо виникає колізія, новий елемент просто додається до списку. Також відомий метод відкритого адресного простору (Open Addressing). У цьому методі при виникненні колізії виконується пошук іншої вільної позиції у геш-таблиці для зберігання даних. Найбільш поширені підходи включають лінійне пробування, квадратичне пробування та подвійне гешування. В загальному геш-таблиці забезпечують дуже швидкий доступ до даних – у середньому час доступу, вставки та видалення елемента складає (O(1)). Це означає, що ці операції виконуються за постійний час, незалежно від кількості даних у таблиці. Проте, у випадку численних колізій час доступу може зрости до (O(n)) в найгіршому випадку, де (n) — кількість елементів у геш-таблиці. Однак при правильному виборі геш-функції та достатньому розмірі таблиці ймовірність цього сценарію мінімальна. У системах виявлення ШПЗ геш-таблиці використовуються для швидкого знаходження гешів, створених на основі файлів або сегментів файлів. Це дозволяє миттєво перевірити, чи був даний зразок раніше виявлений як шкідливий. Також геш-таблиці дозволяють ефективно зберігати велику кількість гешів, що необхідно для забезпечення масштабованості системи та можливості роботи з великими наборами даних. У контексті LSH геш-таблиці можуть бути використані для організації гешів у бакети, де кожен бакет містить геші, що відповідають певним критеріям схожості. Ефективність геш-таблиці значною мірою залежить від якості геш-функції. Вона повинна розподіляти ключі рівномірно по таблиці, щоб мінімізувати кількість колізій. Важливо правильно підібрати розмір геш-таблиці. Якщо таблиця занадто маленька, колізії будуть виникати частіше, що призведе до зниження продуктивності. Занадто велика таблиця може витрачати багато пам’яті. Деякі геш-таблиці можуть активно змінювати свій розмір (ресайзинг) при перевищенні певного коефіцієнта заповнення (load factor). Це дозволяє зберігати високу продуктивність навіть при зростанні кількості даних. До переваг даного методу варто віднести простоту реалізації та

використання, гнучкість у вирішенні колізій та високу швидкість доступу та роботи з даними.

Наступним методом індексації є LSH-бакети. Вони являть собою спеціальні структури, які групують геші на основі їхньої схожості. Геші зберігаються в бакетах (корзинах), що дозволяє ефективно знаходити схожі зразки за допомогою простого пошуку по відповідним бакетам. Для кожного зразка (наприклад, файлу або фрагменту файлу) система генерує кілька гешів за допомогою набору локально чутливих геш-функцій. Ці геші відображають особливості зразка таким чином, що схожі зразки мають схожі геші. Зазвичай використовується кілька різних геш-функцій, і кожна функція створює свій геш для одного і того ж зразка. Це дозволяє зменшити ймовірність того, що схожі зразки потраплять у різні бакети, підвищуючи точність кластеризації. Після обчислення гешів кожен зразок розподіляється в один або кілька бакетів (bucket) залежно від значень його гешів. Бакети можна уявити як кошики, в які поміщаються зразки з однаковими або схожими гешами. У деяких системах використовуються багатовимірні бакети, що дозволяють краще враховувати складні зв’язки між різними характеристиками зразків. Коли потрібно знайти зразки, схожі на новий зразок, система обчислює геші для цього зразка і перевіряє відповідні бакети. Зразки, які знаходяться в тих самих бакетах, що й новий зразок, вважаються потенційно схожими. Зразок може бути перевірений відразу в кількох бакетах, що дозволяє знаходити всі можливі варіанти схожості з мінімальними витратами на обчислення. Оптимальне налаштування параметрів LSH, таких як кількість геш-функцій і кількість бакетів, є ключовим для ефективної роботи системи. Це впливає на точність кластеризації та швидкість пошуку. Важливо забезпечити, щоб бакети не були переповнені або порожніми. Це досягається шляхом налаштування геш-функцій і періодичного аналізу розподілу зразків у бакетах. У деяких системах можлива адаптація бакетів у процесі роботи, коли система активно змінює кількість або структуру бакетів у залежності від характеру даних. З плюсів цього методу це швидкість, тому що система може

ефективно працювати з великими наборами даних, зберігаючи високу продуктивність. Також варто відзначити гнучкість, що надає можливість налаштування параметрів LSH для досягнення оптимального балансу між швидкістю та точністю. Варто зазначити що цей метод як і інші методи LSH не гарантують, що всі схожі зразки будуть знайдені, оскільки це імовірнісні методи. Однак, він забезпечує досить високу точність у більшості випадків. Важливо враховувати що для реалізації потрібне ретельне налаштування геш-функцій і структури бакетів для досягнення оптимальних результатів [19].

В деяких випадках для індексації можуть використовуватися дерева, які розбивають простір гешів на підпростори, що дозволяє зменшити кількість необхідних порівнянь. Дерева для пошуку найближчих сусідів, такі як KD-Tree та Ball-Tree, є ефективними структурами даних, які використовуються для прискорення задач пошуку схожих об'єктів у багатовимірних просторах. У контексті виявлення шкідливого програмного забезпечення (ШПЗ), ці дерева допомагають швидко знаходити зразки, схожі на новий підозрілий файл, що значно покращує ефективність системи виявлення. KD-Tree - це структура даних для ефективного зберігання точок у багатовимірних просторах (k-вимірний простір). Вона розбиває простір на області, організовані у вигляді дерева, що дозволяє швидко здійснювати пошук найближчих сусідів. KD-Tree побудовано на основі рекурсивного розбиття простору на підпростори. На кожному рівні дерева вибирається одна координата (вимір), за якою дані розбиваються на дві частини: ліву і праву. Вузли дерева містять точки, що визначають ці поділи, якщо дерево побудоване рівномірно, кожен вузол дерева розділяє приблизно половину точок, що залишаються на відповідному підпросторі, що забезпечує ефективний пошук. Пошук найближчих сусідів у KD-Tree починається з перевірки вузла, що відповідає за поточний підпростір. Потім перевіряються сусідні підпростори, якщо вони можуть містити точки, що ближчі до цільової точки. KD-Tree простий в реалізації та простий в розумінні, ефективний

пошук у просторах середньої розмірності (до 20-30 вимірів) та має підтримку динамічних операцій, таких як вставка та видалення точок, проте складність пошуку зростає експоненційно із збільшенням розмірності простору (висока вимірність) та балансування дерева може бути складним у разі динамічних змін у даних [17]. Ball-Tree - це ще одна структура даних для пошуку найближчих сусідів, яка також працює у багатовимірних просторах, але використовує дещо інший підхід для розбиття простору порівняно з KD-Tree. Ball-Tree розділяє простір на "сфери" (balls), де кожен вузол дерева відповідає за певну підмножину точок, що знаходяться в межах цієї сфери. Сфера визначається центром і радіусом, що охоплює всі точки, що потрапляють до неї. Простір рекурсивно ділиться на підсфери, поки не залишиться точок, які можна вважати однією підмножиною. Кожен вузол має двох дочірніх вузлів, які охоплюють підмножини, що не перетинаються. Під час пошуку найближчого сусіда у Ball-Tree перевіряються сфери, що потенційно можуть містити сусідів, і відкидаються ті, що точно не містять ближчих сусідів. Підходить для обробки високо вимірних даних та краще працює, ніж KD-Tree, в просторах з великою кількістю вимірів. Ефективніше розподіляє точки в разі нерівномірного розподілу даних у просторі. Проте порівняно з KD-Tree цей метод складніший в реалізації та більш час затратний, оскільки потрібно обчислювати центри та радіуси сфер. Ball-Tree підходить для задач, де важливо враховувати високо вимірні характеристики ШПЗ, такі як складні векторні ознаки файлів, також він дозволяє ефективно кластеризувати схожі зразки ШПЗ, зменшуючи час, необхідний для пошуку потенційно шкідливих файлів у великому масиві даних [7].

геші можуть бути представлені у вигляді векторів в багатовимірному просторі, що дозволяє використовувати методи пошуку найближчих сусідів (k-nearest neighbors) для виявлення схожих зразків. Вектори схожості є математичним інструментом, який використовується для представлення об'єктів у багатовимірному просторі таким чином, щоб можна було легко вимірювати їхню схожість. У контексті виявлення шкідливого програмного

забезпечення (ШПЗ), вектори схожості дозволяють представляти характеристики файлів або їхніх фрагментів у вигляді числових векторів, що значно полегшує порівняння, кластеризацію та пошук схожих об'єктів. Кожен об'єкт, наприклад файл або фрагмент коду, можна представити у вигляді векторів. Кожен елемент векторів відповідає певній характеристиці або ознаці об'єкта. Наприклад, для файлів це можуть бути різні статистичні дані, геші частин файлу, або інші показники. Кількість вимірів залежить від кількості ознак, які використовуються для опису об'єктів. Спочатку визначаються ознаки (features), які будуть використовуватись для побудови векторів. Це можуть бути статистичні ознаки, ознаки на основі частотного аналізу, біти кодової секції, або інші характеристики, які можуть вказувати на схожість між зразками. Часто дані потрібно нормалізувати, щоб усі ознаки мали однаковий вплив на обчислення схожості. Це може бути досягнуто шляхом масштабування значень у певний діапазон або застосуванням інших методів нормалізації. Кожен об'єкт перетворюється на вектор шляхом обчислення значень вибраних ознак для цього об'єкта. Векторний простір, у якому знаходяться ці вектори, використовується для подальших обчислень. Один з найпоширеніших методів обчислення схожості між двома векторами - це косинусна схожість. Вона обчислюється як косинус кута між двома векторами. Якщо косинус дорівнює 1, вектори ідеально схожі, якщо 0 - вони ортогональні (не мають спільних характеристик), якщо -1 - протилежні. Ще один метод - це обчислення Евклідової відстані між векторами. Чим менша ця відстань, тим більше схожі об'єкти. Проте цей метод може бути менш ефективним у високо вимірних просторах. Також використовується Манхеттенська відстань (також відома як відстань L1), яка обчислюється як сума абсолютних різниць між відповідними компонентами векторів. Вектори схожості можна використовувати для класифікації файлів, порівнюючи новий файл з уже відомими зразками ШПЗ. Якщо вектор нового файлу схожий на вектори відомих зразків ШПЗ, файл може бути ідентифікований як потенційно шкідливий. Системи можуть групувати файли у кластери на

основі їх схожості, що дозволяє виявляти нові родини ШПЗ або варіанти вже відомих загроз. Коли новий зразок аналізується, вектори схожості дозволяють швидко знайти найближчі до нього відомі зразки, що може допомогти в швидкій ідентифікації типу загрози або оцінці ризику. Вектори схожості дозволяють швидко порівнювати великі набори даних, використовуючи математичні методи для обчислення подібності. Також, можна використовувати різні набори ознак для побудови векторів залежно від конкретної задачі та типу даних. І ще одним плюсом є те що векторні методи добре масштабуються і можуть бути застосовані до великих обсягів даних з високою розмірністю. З мінусів є те що зі збільшенням кількості ознак (вимірів) зростає складність пошуку і ризик "прокляття розмірності", коли ефективність алгоритмів може знижуватись. Важливо враховувати що потрібно правильно вибрати ознаки, що впливають на побудову векторів, оскільки це впливає на точність і ефективність аналізу. У деяких випадках інтерпретація результатів обчислення схожості може бути складною, особливо якщо ознаки складні або мають неочевидні зв’язки [9].

Після індексації гешів, модуль індексації та пошуку, розпочинає пошук схожих зразків. Коли новий зразок ШПЗ надходить на аналіз, модуль пошуку відповідає за порівняння його гешу з уже існуючими гешами, індексованими в системі. Основна мета - знайти всі схожі геші, що можуть вказувати на наявність схожого або того ж самого шкідливого ПЗ. Після індексації новий геш використовується як запит для пошуку в базі даних. Пошук відбувається спочатку в найбільш релевантних бакетах або сегментах, що містять схожі геші. Знайдені геші порівнюються з гешем нових даних для визначення рівня схожості. Порівняння може проводитися за допомогою метрик схожості, таких як косинусна схожість, Hamming distance або Jaccard index. Якщо рівень схожості перевищує визначений поріг, даний зразок класифікується як підозрілий або потенційно шкідливий. Для забезпечення швидкості та точності пошуку модуль використовує різні оптимізації, які можуть включати: скорочення кількості порівнянь, пошук за кількома бакетами та

ранжування результатів. Скорочення кількості порівнянь включає відсіювання явно несхожих гешів на ранніх етапах пошуку, що значно скорочує кількість необхідних порівнянь. Пошук за кількома бакетами застосовується у випадку, якщо геш може належати кільком різним бакетам, пошук може проводитися одночасно по кількох із них, щоб збільшити ймовірність знайти схожий зразок. Ранжування результатів означає можливість того що результати пошуку можуть бути відсортовані за рівнем схожості, що дозволяє швидко ідентифікувати найбільш релевантні результати. Модуль також відповідає за постійне оновлення індексів з урахуванням нових даних. При додаванні нових зразків в систему, їхні геші додаються до відповідних індексів. Оновлення індексу може включати і видалення старих або мало релевантних гешів, щоб забезпечити актуальність та ефективність пошуку. Після того як знайдені схожі зразки, результати пошуку передаються в інший модуль для аналізу. На основі цього аналізу приймається рішення про те, чи є новий зразок шкідливим, або ж він не має загрози. Модуль індексації та пошуку є центральним елементом, що забезпечує ефективність і продуктивність системи виявлення ШПЗ. Він дозволяє швидко знайти схожі зразки серед великої кількості даних, що критично важливо для своєчасного виявлення та запобігання розповсюдженню шкідливого програмного забезпечення.

Модуль виявлення та аналізу є фінальним етапом у системі виявлення шкідливого програмного забезпечення (ШПЗ) на основі локально чутливого гешування (LSH). Його основне завдання полягає у прийнятті рішень на основі результатів, отриманих від модуля пошуку, і подальшому аналізі потенційно шкідливих зразків. Після того, як модуль пошуку знаходить схожі зразки в базі даних, модуль виявлення та аналізу приймає ці результати для подальшої обробки. Він оцінює рівень схожості між новим зразком і знайденими екземплярами ШПЗ. Важливим аспектом є встановлення порогу схожості, вище якого зразок вважається підозрілим. Цей поріг може бути налаштований залежно від вимог системи або специфіки даних. Важливим

аспектом є встановлення порогу схожості, вище якого зразок вважається підозрілим. Цей поріг може бути налаштований залежно від вимог системи або специфіки даних. Модуль також повинен мати механізми для обробки помилкових спрацювань, коли невинні файли можуть бути помилково класифіковані як шкідливі. Для цього може використовуватися додаткова фільтрація або перехресна перевірка з іншими методами. На основі аналізу схожості модуль виявлення приймає рішення щодо класифікації нового зразка як шкідливого або безпечного. Це може бути проста бінарна класифікація (шкідливий або ні) або більш складна багаторівнева класифікація з різними рівнями загрози. Модуль може використовувати різні алгоритми класифікації, включаючи машинне навчання, для підвищення точності виявлення. Наприклад, нейронні мережі або методи ансамблю можуть бути застосовані для покращення результатів. В деяких випадках модуль може враховувати контекстні дані, такі як історія взаємодії користувача з системою або поведінкові патерни, для прийняття більш точних рішень. Після виявлення потенційного ШПЗ модуль оцінює рівень ризику, який він представляє. Ця оцінка може включати аналіз можливих наслідків від дії шкідливого коду та рекомендації щодо подальших дій. Модуль створює детальні звіти про виявлені загрози, включаючи інформацію про знайдені схожі зразки, рівень схожості, потенційні ризики та запропоновані заходи щодо нейтралізації загрози. Звіти можуть бути автоматично відправлені адміністраторам або використовуватися для автоматичного реагування на загрози. У випадку складних систем безпеки модуль виявлення може бути інтегрований з іншими системами для обміну інформацією або активації автоматичних заходів захисту, таких як ізоляція заражених систем або блокування шкідливого ПЗ. Модуль виявлення та аналізу повинен постійно вдосконалюватися для адаптації до нових видів загроз. Для цього він може використовувати механізми машинного навчання, які дозволяють системі самонавчатися на основі нових даних. Регулярне оновлення баз даних шкідливих зразків і моделей класифікації дозволяє

системі залишатися актуальною та ефективною у боротьбі з новими видами ШПЗ. Модуль може використовувати зворотний зв'язок від користувачів або адміністраторів системи для поліпшення алгоритмів виявлення та зниження кількості помилкових спрацювань. Модуль може також включати інструменти для візуалізації даних і результатів аналізу, що дозволяє адміністраторам системи легше інтерпретувати дані і приймати обґрунтовані рішення. Інтерактивні панелі управління можуть надавати інформацію в режимі реального часу про виявлені загрози, тенденції поширення ШПЗ, а також дозволяти проводити детальний аналіз окремих інцидентів. Модуль виявлення та аналізу відіграє ключову роль в кінцевому прийнятті рішень щодо безпеки системи. Він забезпечує комплексний аналіз виявлених загроз, надає інструменти для оцінки ризиків і пропонує відповідні заходи для запобігання потенційних атак. Завдяки цьому модулю система виявлення ШПЗ може ефективно реагувати на нові виклики та забезпечувати високий рівень захисту.

Цікаво, що інтерфейс користувача (UI) є також важливим компонентом системи виявлення шкідливого програмного забезпечення (ШПЗ) на основі локально чутливого гешування (LSH). Він забезпечує зручний та інтуїтивно зрозумілий доступ до функціоналу системи, дозволяючи користувачам ефективно взаємодіяти з нею, отримувати результати аналізу та приймати обґрунтовані рішення щодо виявлених загроз. На мою думку, UI має містити панель управління де є головна панель, яка повинна надавати користувачеві загальний огляд системи, включаючи ключові показники безпеки, останні виявлені загрози, статус системи, та інші важливі дані, також можлива наявність інтерактивних віджетів, що відображають графіки, статистику, та іншу інформацію в реальному часі. Це можуть бути діаграми частоти виявлення ШПЗ, теплові карти активності загроз, або віджети, що відображають поточний стан мережі. Важливо щоб панель управління надавала можливість персоналізації, дозволяючи користувачам налаштовувати відображення даних і обирати, які показники для них

найбільш важливі. Інтерфейс повинен підтримувати автоматичне генерування звітів на основі виявлених загроз. Користувачі можуть налаштувати параметри звітів, такі як періодичність, формат і тип даних, що включаються в звіти. Звіти повинні бути інтерактивними та візуально зрозумілими. Графіки, діаграми, та таблиці повинні надавати глибокий аналіз даних, таких як тренди розповсюдження ШПЗ, динаміка виявлення загроз, і порівняння з історичними даними. Інтерфейс має дозволяти експорт звітів у різних форматах, таких як PDF, Excel, або CSV, для подальшого використання чи обміну з іншими користувачами або системами. Система повинна надавати можливість налаштування сповіщень про виявлені загрози або інші важливі події. Повідомлення можуть надходити у вигляді спливаючих вікон, електронної пошти, або повідомлень у месенджерах. Інтерфейс має містити журнал подій, що дозволяє користувачам переглядати історію всіх важливих інцидентів, включаючи деталі про час виявлення, тип загрози, та дії, які були вжиті. Користувачі повинні мати можливість налаштування порогових значень, при досягненні яких система буде надсилати повідомлення. Це дозволяє уникнути перевантаження повідомленнями та зосередитися на найважливіших загрозах. Інтерфейс повинен забезпечувати можливість детального аналізу кожної виявленої загрози, включаючи інформацію про геші, рівень схожості з іншими зразками, можливі шляхи поширення, та рекомендації щодо реагування. Плюсом для інтерфейсу буде можливість порівняння кількох зразків ШПЗ у візуальному вигляді, що дозволяє виявити подібності або відмінності між ними. Чудовою фішкою є візуалізація розповсюдження ШПЗ на карті, що показує географічне поширення загроз і можливі джерела атак. Повертаючись до взаємодії інтерфейсу з користувачами, варто зазначити що він має підтримувати систему ролей, яка дозволяє адміністратору призначати різні рівні доступу для різних користувачів. Наприклад, звичайні користувачі можуть мати обмежений доступ до функціоналу, тоді як адміністратори мають повний контроль над системою, також він повинен вести журнал дій

користувачів, що дозволяє відслідковувати, хто і коли виконував ті чи інші дії у системі. Це важливо для забезпечення прозорості та безпеки. Механізми безпеки, такі як двох факторна автентифікація, шифрування даних, і періодичне оновлення паролів, повинні бути інтегровані в систему для захисту від несанкціонованого доступу. Інтерфейс має містити вбудовані інструкції, підказки, та відео-уроки, що допоможуть новим користувачам швидко освоїти систему, варто зауважити що функція інтерактивної підтримки, наприклад, через чат або систему тікетів, дозволяє користувачам швидко отримувати допомогу у вирішенні проблем. Інтерфейс повинен надавати можливість інтеграції з іншими системами безпеки або управління. Це може включати API для автоматичного обміну даними або активації відповідних дій у разі виявлення загроз. Інтерфейс може мати можливість інтеграції з іншими інструментами аналізу та моніторингу, що дозволяє користувачам отримувати ще більш повну картину безпеки системи. Інтерфейс користувача в системі виявлення ШПЗ на основі LSH має бути зручним, функціональним і гнучким, забезпечуючи користувачів всіма необхідними інструментами для ефективного управління безпекою. Він допомагає не лише виявляти та аналізувати загрози, а й приймати оперативні та обґрунтовані рішення для їхньої нейтралізації.

2.2 Розробка алгоритмів виявлення ШПЗ з використанням LSH

Цей підрозділ має описує конкретні алгоритми, які використовуються для виявлення ШПЗ за допомогою LSH. Основними аспектами тут є: розробка схеми гешування, алгоритм побудови LSH-індексів, процедури пошуку і порівняння та аналіз продуктивності.

При розробці схеми гешування для локально чутливого гешування (LSH) важливо вибрати та налаштувати геш-функції, які зберігають схожість між вхідними даними. LSH базується на ідеї того, що схожі об'єкти з високою ймовірністю будуть мати однакові геші або геші, що належать до

тієї ж групи, тоді як несхожі об'єкти мають різні геші. Це робить LSH ефективним інструментом для обробки великих обсягів даних у таких задачах, як виявлення шкідливого програмного забезпечення. Існує декілька типів геш-функцій, які використовуються для різних видів вхідних даних, залежно від типу схожості, яку потрібно зберегти. Наприклад, Hamming Distance (для бінарних даних) у цьому випадку використовуються геш-функції на основі випадкових бітів вхідних даних. При цьому, кілька випадкових бітових позицій вибираються для створення гешу. Два схожі об'єкти матимуть однакові біти в багатьох з цих позицій. У випадках, коли дані можуть бути представлені як вектори в багатовимірному просторі, наприклад, текстові дані або зображення, застосовують геш-функції для збереження косинусної схожості (Cosine Similarity). Випадкові гіперплощини вибираються для розбиття простору. Точки (вектори), що знаходяться по один бік гіперплощини, отримують однаковий геш. Випадкові проекції вхідних векторів на напрямки у просторі використовуються якщо потрібно зберігати схожість на основі Евклідової відстані (для дійсних чисел). Гешування полягає в розбитті простору за випадковими гіперплощинами. Якщо два вектори розташовані близько один до одного в просторі, їхні геші будуть співпадати. Для даних, які представляються у вигляді множин (наборів елементів), часто використовують MinHash для збереження Jaccard схожості. Випадкові перестановки елементів множини застосовуються до кожного об'єкта, і гешування базується на першому елементі після перестановки.

При розробці схеми гешування потрібно використовувати налаштування геш-функцій, яке враховує кількість гешів (k) і "відер" (l) та виборів випадкових параметрів. Параметр k визначає кількість окремих геш-функцій, які застосовуються до одного об'єкта для створення одного "супергешу". Це допомагає збільшити ймовірність того, що схожі об'єкти потраплять у ту саму "геш-групу". Вибір k залежить від компромісу між точністю та ефективністю. Чим більше k, тим меншою стає ймовірність, що

випадкові об'єкти матимуть однаковий геш, але зростає складність обчислень. Параметр l визначає кількість різних "груп геш-функцій" або "відер", які будуть використовуватися для зберігання схожих об'єктів. Збільшення l підвищує ймовірність того, що схожі об'єкти будуть гешовані в одне з однакових "відер". У багатьох випадках геш-функції використовують випадкові параметри (наприклад, випадкові гіперплощини або випадкові біти). Правильний вибір або генерація цих випадкових параметрів є ключем до ефективного LSH, оскільки від цього залежить, наскільки добре геш-функція буде зберігати схожість.

Алгоритм побудови LSH-індексів дозволяє ефективно організувати пошук подібних зразків шкідливого програмного забезпечення (ШПЗ) у великому наборі даних. Процес індексації базується на принципі того, що об'єкти з подібними гешами будуть з високою ймовірністю знаходитися в однакових відрах (buckets), що дозволяє значно скоротити кількість порівнянь під час пошуку схожих зразків. Алгоритм побудови LSH-індексів дозволяє ефективно організувати пошук подібних зразків шкідливого програмного забезпечення (ШПЗ) у великому наборі даних. Процес індексації базується на принципі того, що об'єкти з подібними гешами будуть з високою ймовірністю знаходитися в однакових відрах (buckets), що дозволяє значно скоротити кількість порівнянь під час пошуку схожих зразків. Перед тим як будувати індекси, необхідно отримати геші для кожного зразка, який потрібно індексувати. Використовуються кілька геш-функцій, які зберігають схожість (наприклад, для бінарних даних це може бути SimHash, для векторних даних - Cosine LSH або MinHash для множин). Для кожного зразка обчислюються k-геш-функцій, які будуть використовуватись для кожного з множин геш-функцій (групування відбувається по декількох групах для зменшення ймовірності хибних результатів). Щоб зменшити кількість відер (buckets) та підвищити ефективність пошуку, результати декількох геш-функцій можна об'єднати у так званий "супергеш". Кілька (наприклад, 5–10) геш-функцій можуть бути

об'єднані в одну групу, результатами якої буде набір гешів для об'єкта. Об'єднані геші для одного об'єкта створюють унікальний супергеш. Таким чином, один об'єкт може мати кілька супергешів (залежно від кількості груп), які будуть використовуватися для пошуку. На основі супергешів відбувається розподіл об'єктів у спеціальні відра (buckets). Кожен супергеш визначає одне або кілька відер, у які поміщаються об'єкти. Залежно від параметра l (кількість відер), кожен об'єкт може потрапити до кількох відер. Це підвищує ймовірність того, що схожі об'єкти потраплять до одного відра. Після того як всі об'єкти розподілені по відрах, формується індекс, який дозволяє швидко знаходити потенційно схожі об'єкти в одному або кількох відрах. Індекс зберігає відповідність між гешами та відрами, в яких знаходяться об'єкти. Варто зауважити, що кожен об'єкт можна індексувати кілька разів (залежно від кількості геш-функцій і груп). Після побудови LSH-індексів, процес пошуку подібних зразків шкідливого ПЗ стає значно швидшим, оскільки пошук здійснюється тільки в тих відрах, де ймовірно можуть бути схожі зразки. Алгоритм пошуку виглядає так: спочатку відбувається гешування запиту, а саме шкідливий зразок, який розшукається, спочатку проходить через ті ж самі геш-функції, які використовувалися для індексування даних та обчислюється кілька супергешів для цього запиту. Далі на їх основі визначаються відра, в яких можуть знаходитись схожі об'єкти, важливо що вибираються всі об'єкти з цих відер для подальшого аналізу. Після цього усі об'єкти з відер порівнюються з запитом за допомогою метрики схожості (наприклад, Hamming відстань або інші метрики для конкретного типу даних), але краще якщо порівнюються тільки ті об'єкти, які знаходяться в тих самих відрах, що й запит, це значно скорочує кількість порівнянь у порівнянні з повним перебором всіх можливих об'єктів. І на останок найбільш схожі об'єкти повертаються як результати пошуку. Якщо потрібно, можуть бути використані додаткові евристики або уточнення, щоб зменшити кількість хибно позитивних результатів.

При розробці алгоритму та процедурах пошуку та порівняння важливо пам’ятати про оптимізацію ефективності LSH, вона полягає в прискорені пошуку та підвищення його точності. Спочатку оптимізуються кількості геш-функцій і відер. Збільшення кількості геш-функцій k підвищує точність (зменшує ймовірність того, що несхожі об'єкти потраплять в одне відро). Однак при занадто великій кількості геш-функцій збільшується ймовірність того, що схожі об'єкти будуть розкидані по різних відрах. Тому k вибирається так, щоб забезпечити компроміс між швидкістю пошуку та точністю. Збільшення кількості відер l підвищує ймовірність того, що схожі об'єкти потраплять до одного з них. З іншого боку, занадто велика кількість відер збільшує кількість об'єктів, які потрібно порівнювати в кожному пошуку, що впливає на ефективність. Після цього варто використати можливість мультипробного пошуку, це полягає в тому що замість того щоб обмежувати пошук лише кількома відрами, в яких знаходиться новий зразок, multi-probe LSH пропонує використовувати додаткові відра для пошуку. Це дозволяє знайти потенційні кандидати з відер, які близькі до гешу нового зразка. Multi-Probe LSH дозволяє досягти більш високої точності без значного збільшення кількості обчислень. Третім етапом оптимізації можна вважати зменшення кількості хибно позитивних результатів. Для його виконання можна застосовувати кілька рівнів гешування для більш точної кластеризації об'єктів. Наприклад, спочатку використовується один набір геш-функцій для грубого фільтрування, а потім інший набір для більш детального аналізу. А якщо основний алгоритм продовжує давати занадто багато хибно позитивних результатів, можна додатково використовувати більш точні метрики або евристики для остаточного порівняння. Наприклад, для порівняння двох зразків ШПЗ можуть бути використані сигнатури або поведінкові характеристики. Після цього етапу виконується паралельна обробка, вона застосовується у випадку великого набору даних, тоді обчислення гешів і пошук у відрах можуть бути розподілені між кількома потоками або серверами. Це дозволяє паралельно обробляти кілька

кандидатів, зменшуючи час пошуку. Важливо що гешування та пошук можуть виконуватися незалежно один від одного для кожного відра, що легко піддається паралелізації. І на завершальному етапі відбувається адаптивна оптимізація порогів. Замість фіксованих порогів для метрики схожості можна використовувати адаптивні пороги, які налаштовуються залежно від результатів пошуку та особливостей зразків ШПЗ. Наприклад, якщо в певному випадку зразки демонструють велику схожість, пороги можуть бути знижені для більш точного пошуку. Підсумовуючи варто зазначити що ці п’ять етапів є незалежні один від одного, та можуть використовуватись в довільному порядку, або взагалі вибірково, хоча для ефективного застосування методу вони усі потрібні.

В цьому пункті також важливо висвітлити аналіз продуктивності алгоритмів виявлення шкідливого програмного забезпечення (ШПЗ) із використанням локально чутливого гешування (LSH) який дозволяє оцінити, наскільки цей підхід ефективний для великих наборів даних, порівняно з іншими методами. У цьому аналізі розглядаються ключові аспекти продуктивності, такі як швидкість пошуку, точність виявлення, проблеми хибно позитивних/хибно негативних результатів. Усіх їх ми вже розглядали, проте з для підрахунку ефективності потрібно знати що Точність (precision) визначає, яка частка об’єктів, що були класифіковані як схожі, насправді є подібними. Вона вираховується за допомогою формули: $Precision = \frac{\text{TP}}{\text{TP+FP}}$ у ній TP - кількість справжніх позитивів (правильно знайдені схожі зразки) та FP - кількість хибних позитивів (несхожі об'єкти, які помилково вважалися схожими). Повнота (recall) визначає, яка частка справжніх схожих об'єктів була правильно знайдена. Вона вираховується за допомогою формули: $Recall = \frac{\text{TP}}{\text{TP+FN}}$, тут TP це те саме що й в минулій формулі, а FN - кількість хибних негативів (схожі об'єкти, які не були знайдені). Баланс між точністю та повнотою можна оцінити за допомогою метрики $F1$-score $F1 = 2 * \frac{\text{Precision}*\text{Recall}}{\text{Precision+Recall}}$. Основна причина хибнопозитивів - недостатньо "гострі" геш-

функції, які не можуть чітко відрізняти схожі об'єкти від несхожих, або занадто велика кількість відер, що збільшує кількість об'єктів, які потрапляють до одного відра, але насправді не є подібними. У свою чергу хибно негативи виникають, коли схожі об'єкти потрапляють до різних відер. Це може відбуватися через недостатню кількість геш-функцій або надто малі параметри l, погане налаштування геш-функцій також може призвести до того, що важливі ознаки схожості не будуть враховані під час гешування. Варто зазначити, що оптимізація цих метрик в LSH залежить від правильного налаштування параметрів k та l. LSH добре масштабується для великих наборів даних, проте деякі проблеми можуть виникати з часом, а саме збільшення кількості об'єктів і динамічне оновлення індексів. Зі збільшенням кількості зразків ШПЗ може збільшуватися кількість хибно позитивних результатів, оскільки більше об'єктів буде потрапляти до тих самих відер. Тому що індексація нових зразків ШПЗ здійснюється шляхом додавання їх до існуючих відер, потрібно забезпечити ефективне динамічне оновлення без перерахунку всього індексу. Оптимізації для покращення продуктивності, за допомогою Multi-Probe LSH ми вже розглядали, проте ще є каскадне гешування та паралельне обчислення. Каскадне гешування дозволяє використовувати кілька рівнів гешування. Спочатку проводиться грубе гешування для відсіювання явно несхожих зразків, а потім - більш точне гешування для порівняння кандидатів, що залишились. А паралельне обчислення дає змогу легко проводити паралельно індексацію та пошук. Це сприяє одночасному обчисленню гешу та виконання пошуку серед різних відер на різних серверах або потоках. Також для оцінювання потрібно враховувати аналіз складності. Вона складається з просторової та часової складностей. Просторова складність залежить від кількості відер і геш-функцій. Збільшення кількості відер призводить до більшого використання пам'яті для збереження індексів. А часова складність, полягає в тому що для одного зразка обчислення гешів займає лінійний час щодо кількості геш-

функцій, а пошук у відрах - сублінійний щодо загальної кількості об'єктів [20].

Підсумовуючи можна сказати, що розробка схеми гешування для алгоритму Locality Sensitive Hashing (LSH) є важливим кроком у побудові системи для пошуку схожих об'єктів у великих наборах даних. Основна ідея схеми гешування полягає в тому, щоб створити геш-функції, які з високою ймовірністю відображатимуть схожі об'єкти в один і той самий геш-бакет. А правильна схема гешування може значно скоротити час пошуку, зменшуючи кількість порівнянь між об'єктами, що особливо важливо при роботі з великими базами даних.

Для прикладу реалізації розглянемо алгоритм виконання тестування файлів за допомогою TLSH, розпочинається з встановлення інструменту TLSH, після цього йде генерація гешів для файлів, далі збереження гешів для бази порівняння, після нього вже відбувається саме порівняння і аналіз результатів. Варто ще зазначити що доцільно доповнити алгоритм автоматизацією процесу та інтерпретація результатів.

Алгоритм складається з шести етапів. Розглянемо кожен з етапів детальніше.

1. Встановлення інструменту TLSH. Для початку роботи потрібно встановити цей інструмент. Процес встановлення залежить від операційної системи. Це передбачає завантаження, встановлення та перевірка:

– для ОС Windows потрібно перейти на GitHub-репозиторій TLSH: *TLSH на GitHub*, завантажити вихідний код або зібрані бінарні файли, далі можливо буде необхідність зібрати програму за допомогою компілятора C++ (наприклад, GCC або Visual Studio), після цього потрібно додати шлях до змінної середовища PATH. Для перевірки введіть tlsh у командному рядку. Якщо з’являється довідка про використання, встановлення виконано успішно;

– для більшості дистрибутивів Linux TLSH доступний у стандартних репозиторіях. Для Debian/Ubuntu виконується команда sudo apt install tlsh-tools;

Варто зазначити що у користувачів можуть виникати помилки, незрозумілості або інші перешкоди, але головне не зупинятись на цьому технічному етапі. Якщо виникли проблеми, потрібно перевірити документацію в репозиторії TLSH. Також важливо завжди використовувати останню версію репозиторію для отримання актуальних функцій і виправлень помилок.

2. Генерація гешів для файлів. Після встановлення TLSH, можна створити розмиті геші для файлів. Це ключовий етап, який дозволяє отримати унікальні рядки для подальшого порівняння.

Для генерація гешів потрібно відкрити термінал (або командний рядок). Написати команду *tlsh -f MediaCreationTool_22H2.exe*, важливо що *MediaCreationTool_22H2.exe* – файл, для якого потрібно згенерувати геш. Результат може виглядати приблизно так: T1DA17DF0176E99214F1B36B34A9B59291096BBC119935C43EEE7C760DC B3F9B08A7C732. Це і є розмитий геш MediaCreationTool_22H2 (завантажувача windows 10). також є можливість генерації гешів для декількох файлів. Для цього використовується рекурсивне створення гешів у каталозі завдяки команді *find directory/ -type f -exec tlsh -f {} *. Результат: у терміналі буде виведено геші для кожного файлу. *Directory/* – шлях до каталогу з файлами [3].

На цьому етапі ми фактично отримуємо потрібну інформацію для створення основи.

3. Створення бази гешів Щоб здійснювати порівняння нових файлів з існуючими, необхідно створити базу гешів. Ця база містить геші всіх файлів, які будуть використовуватись як еталон.

Для початку нам потрібно згенерувати геші для всіх файлів у каталозі, найпростіше це зробити за допомогою команди *find /path/to/directory -type f -*

exec tlsh -f {} \; > hashes.txt Фактично вона складається з двох частин, перша це directory/ – шлях до каталогу з файлами, яка нам знайома з попереднього етапу. А друга hash_database.txt – файл, у який зберігаються результати. Структура цього файлу складається з 3 частин. Перша – розмитий геш, друга (після символу :) – зменшений під хеш для оптимізації та третя – ім'я файлу. На практиці це може виглядати так:

File: /path/to/directory/file1.exe

TLSH:

T1A85F134487FDC9E2F2B4809A7CD837F1D2A345612EC34512A8B34D1C56A78D

File: /path/to/directory/file3.txt

TLSH:

T1B93F134B8FFDC9F2F3B4813B7CD839F1D2B345612EC34512A9C44D1E45A78E. Важливо, що у користувачів є можливість оновлювати базу гешів. Якщо потрібно додати нові файли до бази то це можна робити двома способами, а саме згенерувати геш для окремого файлу або для кількох файлів у новому каталозі. Це робиться завдяки команді *tlsh -f new_file.txt >> hashes.txt* або *find /path/to/directory -type f -exec tlsh -f {} \>> hashes.txt* відповідно до потреби. Важливо що обов’язково потрібно ставити символ >>, тому що він додає геш до існуючої бази без її перезапису.

Перевірка бази гешів відбувається досить просто, варто тільки відкрити файл *hashes.txt*, щоб переконатись, що всі записи є. Також варто перевіряти правильність шляхів до файлів, особливо якщо база використовуватиметься на іншій машині.

Завдяки даному етапу ми закладаєм певну основу для подальшого порівняння. Для комфортнішого та безпечнішого комікування з базою нам краще використовувати добре організовані каталоги, щоб уникати плутанини, зберігати копію бази на випадок її пошкодження та якщо буде потрібно автоматизувати обробку, як варіант можна форматувати базу в JSON або CSV.

4. Порівняння файлів. Після того як база гешів створена, наступний етап – порівняння нових файлів із базою, щоб визначити схожість.

Якщо потрібно перевірити окремий файл на схожість із файлами в баз, то ми просто порівнюємо файл, який потрібно перевірити з файлом із гешами бази, а саме *target_file.txt* та *hashes.txt.* Потрібна команда виглядає так: *tlsh -f target_file.txt > hashes.txt*

База гешів (hashes.txt) має вигляд:

/path/to/file1

T1A85F134487FDC9E2F2B4809A7CD837F1D2A345612EC34512A8B34D1C56A78D

/path/to/file2

T12945134BCFFDC9E123A4813A5CD837E1D2B345A12EC34912B8C44D1F45B78E

Порівняти новий файл із базою можна за допомогою простого скрипта:

```
while IFS= read -r line; do
    base_file=$(echo "$line" | awk '{print $1}')
    base_hash=$(echo "$line" | awk '{print $2}')
    target_hash=$(cat target_hash.txt)
    distance=$(tlsh -c "$target_hash" "$base_hash")
    echo "$distance $base_file"
done < hashes.txt
```

Результати порівняння зберігаються у файл для подальшого аналізу:

./compare_script.sh > comparison_results.txt

Для порівняння кількох файлів з базою використовують рекурсивну перевірку, вона проводиться командою *find new_directory/ -type f -exec tlsh -f {} \; > new_hashes.txt.* Потім потрібно порівняти результати із базою за допомогою іншого скрипта:

```
while IFS= read -r target_line; do
    target_file=$(echo "$target_line" | awk '{print $1}')
    target_hash=$(echo "$target_line" | awk '{print $2}')
```

```
    while IFS= read -r base_line; do
        base_file=$(echo "$base_line" | awk '{print $1}')
        base_hash=$(echo "$base_line" | awk '{print $2}')
        distance=$(tlsh -c "$target_hash" "$base_hash")
        echo "$distance $target_file $base_file"
    done < hashes.txt
done < new_hashes.txt > comparison_results.txt
```

На цьому етапі ми завершили отримання даних для проведення аналіз, після цього залишається тільки підвести підсумки та як варіант покращення автоматизувати дану роботу. І ще для систем моніторингу можна налаштувати скрипти, які аналізуватимуть результати порівняння та генеруватимуть сповіщення, якщо буде знайдено високий рівень схожості.

5. Аналіз результатів. Після порівняння файлів за допомогою TLSH необхідно інтерпретувати результати, щоб зробити висновки про схожість і потенційні загрози. Розтлумачити відсотки схожості можна за такою шкалою:

distance = 0: файли ідентичні.

1–50: файли схожі, можливо, з невеликими змінами (метадані, правки коду).

51–100: часткові збіги (модифікації коду чи зміна структури).

>100: низька схожість, малоймовірна зв’язок між файлами.

Аналіз результатів TLSH дозволяє ідентифікувати схожі файли та оцінити рівень змін. Використовувати порогові значення можна для автоматизації класифікації та зберігання результатів для подальшого моніторингу. Наступний етап – інтеграція в робочий процес.

6. Автоматизація процесу. Щоб ефективно використовувати TLSH для аналізу файлів, можна автоматизувати весь процес – від генерації гешів до аналізу результатів. Це зменшить ручну роботу, скоротить час обробки та підвищить точність. Автоматизація процесу з використанням Bash або Python дозволяє ефективно створювати базу, аналізувати нові файли та отримувати

сповіщення про підозрілі збіги. Розклад запуску скриптів забезпечить регулярність перевірок.

2.3 Реалізація та тестування розроблених алгоритмів на реальних даних та відомих прикладах ШПЗ

Реалізація та тестування алгоритмів виявлення шкідливого програмного забезпечення (ШПЗ) на основі Locality-Sensitive Hashing (LSH) є критично важливими етапами, які забезпечують перевірку їхньої ефективності та надійності. Цей процес складається з кількох ключових етапів, а саме реалізація алгоритмів, підготовка тестових даних, саме тестування алгоритмів, аналіз його результатів та документування і зворотній зв'язок.

На етапі реалізації алгоритмів, буде розроблюватись спеціальне програмне забезпечення, мова його написання залежить від вимог замовника, навичок розробника та потреб для вирішення. Популярними мовами для написання є Python, Java, C++, проте це практикується і на інших. Варто врахувати що написання має відбуватись з використанням бібліотек для обробки даних і гешування. Також на даному етапі забезпечується інтеграція всіх компонентів системи, таких як збір даних, гешування, порівняння, аналіз і класифікація та розробляється та впроваджується зручний інтерфейс користувача для налаштування параметрів, запуску аналізу та перегляду результатів.

Після завершення етапу реалізації алгоритмів, розпочинається підготовка тестових даних. Спочатку збираються різноманітні набори даних, які містять відомі зразки шкідливого ПЗ (з баз даних, таких як VirusTotal або MalwareBazaar), легітимні програми для уникнення помилкових спрацьовувань та модифіковані версії відомих загроз для оцінки здатності алгоритму виявляти нові варіанти. І потім готуються дані для тестування, включаючи їхню нормалізацію та форматування для зручності обробки.

Одним з основних етапів є тестування алгоритмів, тут реалізуються автоматизовані тестові сценарії для виявлення ШПЗ на основі розроблених алгоритмів. Основні тестові сценарії можуть включати статичний та динамічний тести. Статичний тест це коли виконується аналіз статичних даних без виконання програм для перевірки наявності шкідливого коду. А коли відбувається запуск підозрілого ПЗ в контрольованому середовищі для моніторингу його поведінки, то це буде динамічний тест. Також під час тестування алгоритмів вимірюється час виконання, використання пам'яті та інші ключові показники продуктивності алгоритмів під час тестування.

Ключовим етапом є аналіз результатів тестування. На ньому збираються дані про всі виявлені загрози, включаючи правильні спрацьовування (true positives), помилкові спрацьовування (false positives) та пропущені загрози (false negatives). Після цього проводиться аналіз ефективності алгоритмів за такими метриками, як: точність (precision), повнота (recall) та F1-міра. Проаналізувавши результати тестування ми зможемо виявити сильні та слабкі сторони алгоритмів, а також можливості для покращення.

Важливим та завершальним етапом є документування та зворотний зв'язок. Фактично він вимагає створення докладної документації, яка описує реалізацію, тестування та результати, що допоможе у подальшому вдосконаленні системи. Також дозволяє провести обговорення з користувачами та фахівцями з безпеки для отримання відгуків про ефективність алгоритмів і їхню доцільність у реальних умовах.

Підсумовуючи варто наголосити що підготовка реальних даних та відомих прикладів ШПЗ є ключовими для перевірки алгоритмів у умовах, близьких до реальних. Успішне тестування забезпечує впевненість у надійності системи виявлення загроз і її готовності до впровадження в практику.

3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Організація експериментів для оцінки ефективності та продуктивності виявлення вірусів методом Нільсімса

Сучасні шкідливі програми активно використовують техніки мутації та обфускації для уникнення виявлення традиційними антивірусними засобами. Це створює значні виклики для систем кібербезпеки, які покладаються на сигнатурний аналіз. У зв’язку з цим, зростає попит на підходи, здатні ідентифікувати схожість між зразками шкідливого програмного забезпечення незалежно від їхніх модифікацій.

Метод Нільсімса це алгоритм гешування, основною ціллю якого є використання чутливості до локалізації, для боротьби зі спамом. Він був розроблений як специфічний інструмент для боротьби зі спамом у текстових даних. Його винахідником є Ернесто Даміані, який описав метод у статті 2004 року. Основна ідея полягала в створенні механізму для виявлення схожості між текстовими повідомленнями, навіть якщо вони були модифіковані для уникнення виявлення. під назвою «Техніка виявлення спаму на основі відкритого дайджесту» [12]. Основна ідея полягала в створенні механізму для виявлення схожості між текстовими повідомленнями, навіть якщо вони були модифіковані для уникнення виявлення. Nilsimsa побудований на концепції локально чутливого гешування (LSH), що дозволяє порівнювати об'єкти за схожістю на основі їхніх гешів. На відміну від криптографічних геш-функцій (таких як MD5 або SHA-256), які прагнуть уникнути зіставлення схожих даних, Nilsimsa спеціально оптимізований для пошуку таких збігів. Метод Nilsimsa реалізує алгоритм, що базується на ковзному вікні фіксованої довжини (зазвичай 5 байтів). Метод Nilsimsa, хоча й розроблений для текстових даних, може бути

адаптований для роботи з бінарними файлами, що є важливим для аналізу шкідливих програм. Основні етапи алгоритму:

Аналіз вхідних даних: дані аналізуються побайтно, формуючи триграми можливих комбінацій символів.

Проекція триграм: для шкідливих програм триграми формуються на основі послідовностей байтів у файлі. Наприклад, файл із вмістом 0x45, 0x67, 0x89 генерує триграму 0x456789. Nilsimsa підраховує частоту появи таких триграм і проектує їх у 256-бітний акумулятор

Порогове значення: після обробки триграм кожна позиція масиву перевіряється на відповідність пороговому значенню. Якщо значення перевищує поріг, позиція отримує значення 1, інакше – 0.

Генерація дайджесту: після обробки всіх триграм формується 32-байтовий дайджест, який слугує «відбитком» файлу. Цей відбиток чутливий до змін у структурі файлу, але не до його назви чи розширення.

Особливістю Nilsimsa є те, що його геш можна порівнювати для визначення рівня схожості. Величина схожості вимірюється в діапазоні від 0 до 128, де 128 означає ідентичність.

Основними перевагами можна назвати швидкість обчислень та високу чутливість до змін у вхідних даних.

А до недоліків те що він працює лише з символьними даними і не завжди адаптований до бінарних файлів.

Основною метою проведення експерименту є перевірка дієздатності даного методу для виявлення схожості між оригінальним зразком вірусу та його дублікатом з іншою назвою та розширення і відсутністю помилкового спрацювання на легітимну програму, не пов’язану з ним.

Для експерименту був обраний зразок шкідливої програми з сайту Malware Bazar. Також попередньо на віртуальну машину з системою Kali linux було налаштовано та створено Python файл *nilsimsa_hash.py* в якому був прописаний код

import nilsimsa

file_path = "Malvare.exe" # Замінюється на свій шлях до потрібного файлу

with open(file_path, 'rb') as f:

file_data = f.read()

hash_value = nilsimsa.hash(file_data)

print(f"Nilsimsa hash: {hash_value}")

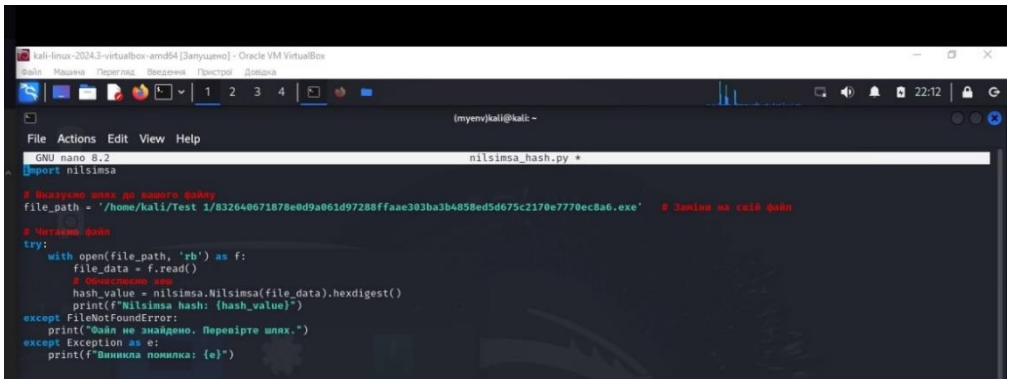

Рисунок 3.1 – Код використання nilsimsa у редакторі nano (зразок вірусу)

В першому випадку *file_path* буде дорівнювати: *'/home/kali/Test1/83264067187.exe'*

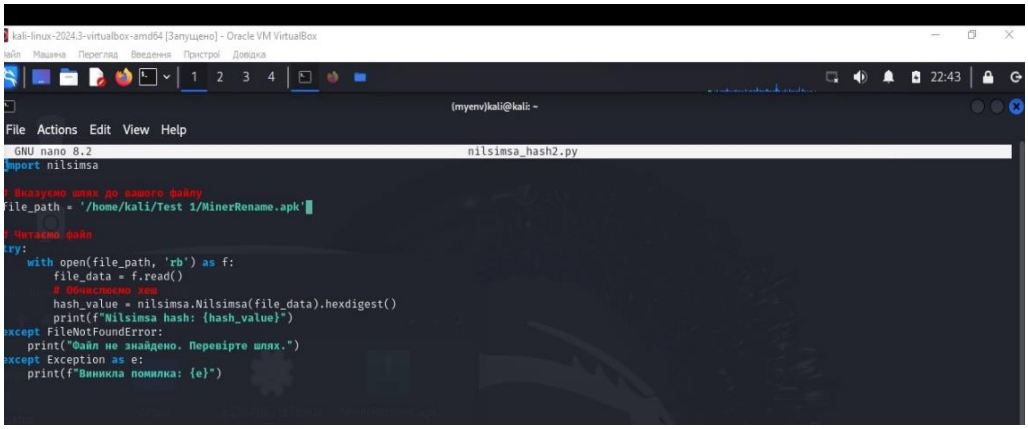

Рисунок 3.2 – Код використання nilsimsa у редакторі nano (дублікат вірусу)

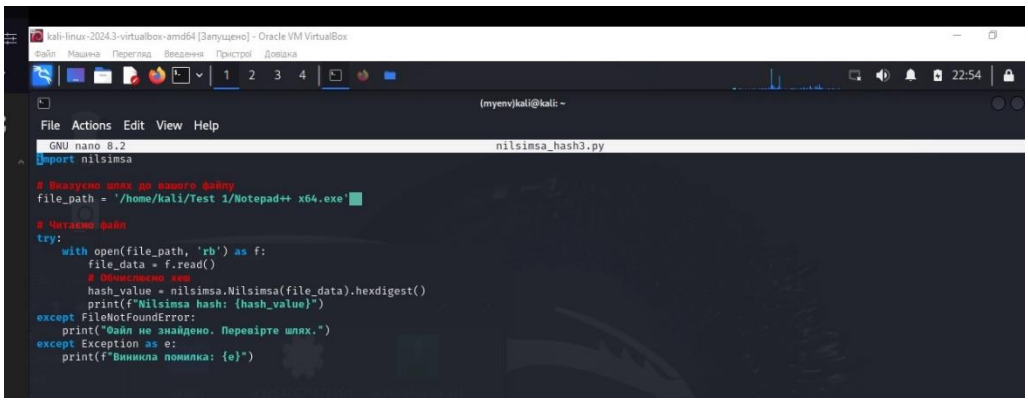

Рисунок 2.3 – Код використання nilsimsa у редакторі nano (легітимна програма)

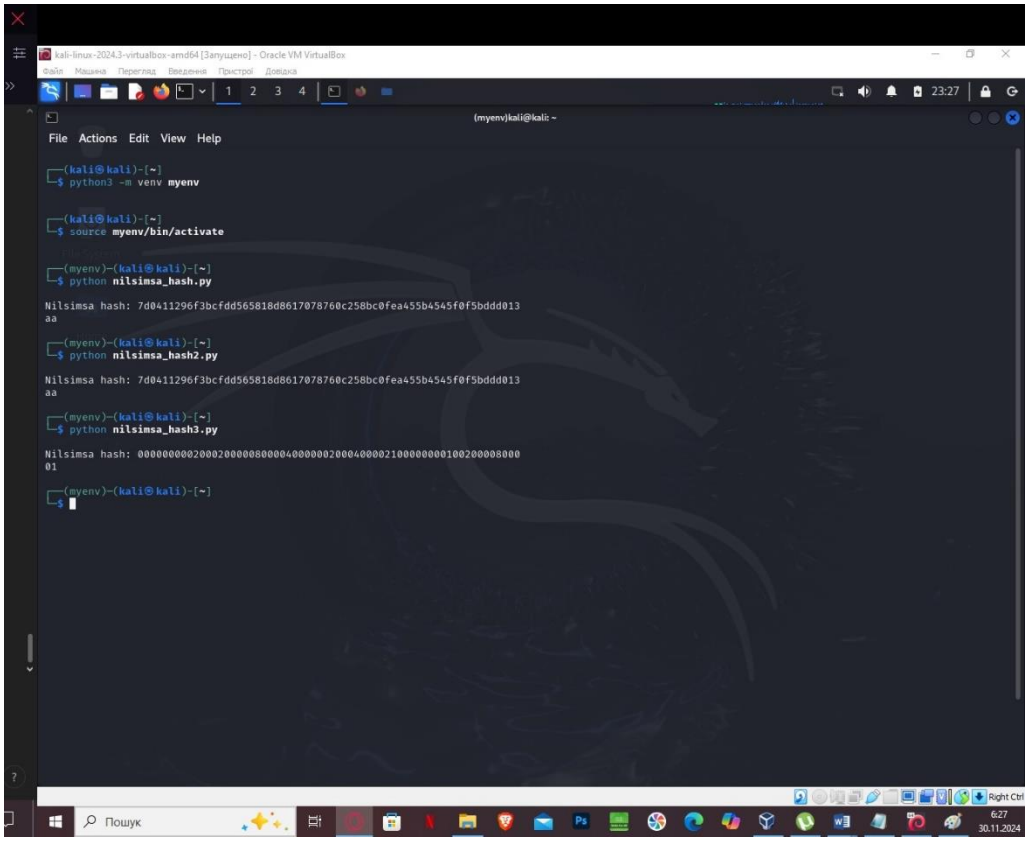

Рисунок 3.4 – Результати гешування методом nilsimsa

Також, замість усіх цих дій і перевірок кожного файлу окремо можна створити дещо модернізований код, вклавши його також у файл з розширенням *PY* .

import nilsimsa

def get_nilsimsa_hash(file_path):

with open(file_path, 'rb') as f:

```
        file_data = f.read()
    return nilsimsa.hash(file_data)
file1 = 'Miner.exe '
file2 = 'MinerRename.apk'
file3 = 'Notepad++.exe'
hash1 = get_nilsimsa_hash(file1)
hash2 = get_nilsimsa_hash(file2)
hash3 = get_nilsimsa_hash(file3)

print(f"Nilsimsa hash for {file1}: {hash1}")
print(f"Nilsimsa hash for {file2}: {hash2}")
print(f"Nilsimsa hash for {file3}: {hash3}")
```

Після його збереження даний скрипт можна запустити, фактично він повинен виконати такі самі функції що й попередні, але з економіє часу.

Тепер можна розпочинати порівняння гешів:

Між першим і другим файлом: ці геші однакові, тому схожість буде максимальною (128).

Між першим і третім файлом: геші сильно відрізняються, тому схожість буде низькою.

Між другим і третім файлом: схожість буде така сама, як і між першим і третім, оскільки геші теж сильно відрізняються.

Якщо порівнювати їх в діапазоні від 0 до 128, то ми можемо отримати щось подібне до:

Схожість між першим і другим: 128 (однакові геші).

Схожість між першим і третім: 24 (ці геші сильно відрізняються).

Схожість між другим і третім: 24 (це те саме, що між першим і третім).

Ці дані можна приведені у таблиці 3.1.

Таблиця 3.1 – Порівняння рівня схожості

Пара файлів	Рівень схожості (%)	Висновок
Оригінальний майнер — Notepad++	24	Жодної подібності, помилкове спрацювання відсутнє
Оригінальний майнер — дублікат	124	Файли ідентичні, зміна назви не впливає
Дублікат — Notepad++	24	Жодної подібності, помилкове спрацювання відсутнє

Отже, даний метод зміг виявити схожості між оригінальним зразком вірусу та його дублікатом з іншою назвою та розширення і зумів не спрацювати на легітимну програму, не пов’язану з ним, тому можна вважати що експеримент пройшов успішно. Підсумовуючи варто сказати що метод Нільсімса є перспективним підходом до вирішення цієї проблеми. Він дозволяє порівнювати зразки за їхньою структурою, що робить його стійким до обфускацій та мутацій. Для оцінки практичної цінності цього методу необхідно провести серію експериментів, спрямованих на аналіз його ефективності та продуктивності в умовах, наближених до реальних.

3.2 Зібрання даних та виконання тестування методу TLSH на реальних прикладах черв’яків

У сучасному різновиді шкідливого програмного забезпечення особливе місце посідають черв’яки, вони здатні до швидкого поширення через мережеві або фізичні канали. У процесі боротьби з такими загрозами критично важливим є використання методів, які дозволяють ефективно ідентифікувати схожість між зразками ШПЗ для їх класифікації, створення сигнатур та прогнозування потенційних загроз.

TLSH був створений компанією Trend Micro для вирішення завдання класифікації шкідливого програмного забезпечення. Основна мотивація – забезпечити механізм для швидкого та ефективного пошуку схожості між файлами, враховуючи їхні структурні особливості. Метод був представлений у наукових публікаціях компанії на початку 2010-х років.

TLSH застосовує комбінацію технік локально чутливого гешування та аналізу структури даних, а саме: збір статистики, розбиття файлу на сегменти, формування геша та їх порівняння. Для збору статистики алгоритм розраховує контрольні суми на основі байт-файлів і формує геш, який відображає загальну структуру файлу. Після цього дані розбиваються на блоки фіксованого розміру, що дозволяє враховувати локальні зміни. Стосовно формування геша, то TLSH має змінну довжину, але завжди містить специфічну сигнатуру файлу, включаючи контрольну суму та структурні атрибути. Для порівняння гешу обчислюється відстань між гешами, це дозволяє визначати схожість між файлами. Також застосовується метрика Hamming Distance, яка дозволяє отримати високу точність порівняння.

` Метод TLSH (Trend Micro Locality Sensitive Hashing) забезпечує можливість вимірювання схожості між файлами, враховуючи їх структурні особливості. Це робить його перспективним інструментом у задачах виявлення модифікацій шкідливого коду [1].

TLSH є ефективним інструментом для аналізу схожості між файлами завдяки своїм структурним особливостям, а саме незалежність від розміру файлу, гнучкість у роботі з великими файлами та стійкість до обфускації. Стосовно першого то воно проявляється у тому що на відміну від криптографічних геш-функцій, які повністю змінюють результат при найменшій зміні вхідних даних, TLSH адаптований до ідентифікації схожості навіть при локальних модифікаціях. Фактично ця властивість робить його надзвичайно корисним для аналізу мутацій шкідливого програмного забезпечення, яке зазвичай зазнає мінімальних змін для уникнення

виявлення. Стосовно гнучкості можна сказати що TLSH оптимізований для аналізу файлів різних розмірів, дозволяючи ідентифікувати схожість навіть між великими файлами, що містять значну кількість структурної інформації. Ну а стійкість іншими словами це те що Шкідливе програмне забезпечення часто застосовує техніки обфускації для ускладнення аналіз, а TLSH демонструє стійкість до цих технік, оскільки фокусуючи увагу на загальній структурі файлу, а не на окремих його частинах.

Попри вище перелічені переваги TLSH також має певні обмеження, які важливо враховувати, а саме чутливість до великих змін відносну чутливість до типу даних та необхідність доповнення іншими методами. Чутливість до великих змін полягає в тому що значні структурні модифікації у файлі можуть призвести до створення суттєво відмінного геша, навіть якщо файл належить до того ж класу шкідливого ПЗ. Це вимагає додаткових інструментів аналізу для покращення загальної ефективності. Також, що до відносної чутливості то це пояснюється тим що TLSH найкраще працює з бінарними файлами, але може мати труднощі з ефективною обробкою певних форматів даних, таких як сильно стиснені або зашифровані файли.

Для оцінювання ефективності методу TLSH (Trend Micro Locality Sensitive Hashing) було проведено роботу, яка полягала у виявленні схожості між зразками. А саме черв'яка та його дублікату, і перевірці наявності відповідного гешу в базі даних ШПЗ на сайті Virus Total. Звісно для перевірки помилкового збігу ми дізнаємось геш легітимної програми, не пов'язаної з ШПЗ.

Для початку нам потрібно встановити пакет tlsh-tools. Перше ми оновлюємо списки пакетів в kali linux вводячи команду *sudo apt update.* А потім встановлюємо пакет за допомогою *sudo apt install tlsh-tools.* Після успішного встановлення ми можем користуватися TLSH. [16].

Розглянемо перевірку TLSH на конкретному прикладі. Після завантаження черв'яка з Malware bazar, ми можемо його геш. В конкретному випадку ми вводим дану команду:

┌──(kali㉿kali)-[~]

└─$ tlsh –f '/home/kali/Test2/abb727416ce1cbf43185b9f0e01ea2d5dedb7dbc1d8d262fc382219ca4120e2a.exe'

Після цього отримуємо такий результат:

8D658D61D601083EC0A110FEC6BB4EBE6A605F285B4151DF47A13AD7A6376FC3A31DA7 /home/kali/Test2/abb727416ce1cbf43185b9f0e01ea2d5dedb7dbc1d8d262fc382219ca4120e2a.exe

Аналогічно виконуєм з іншими потрібними файлами. А саме:

┌──(kali㉿kali)-[~]

└─$ tlsh -f '/home/kali/Test 2/Rename_worm.apk'

8D658D61D601083EC0A110FEC6BB4EBE6A605F285B4151DF47A13AD7A6376FC3A31DA7 /home/kali/Test 2/Rename_worm.apk

┌──(kali㉿kali)-[~]

└─$ tlsh -f '/home/kali/Test 2/Notepad++ x64.exe'

8D6633002642CD7DE87F543082798B7C1B37AEA17B80D2437B9DAA5F192362CFA55736 /home/kali/Test 2/Notepad++ x64.exe

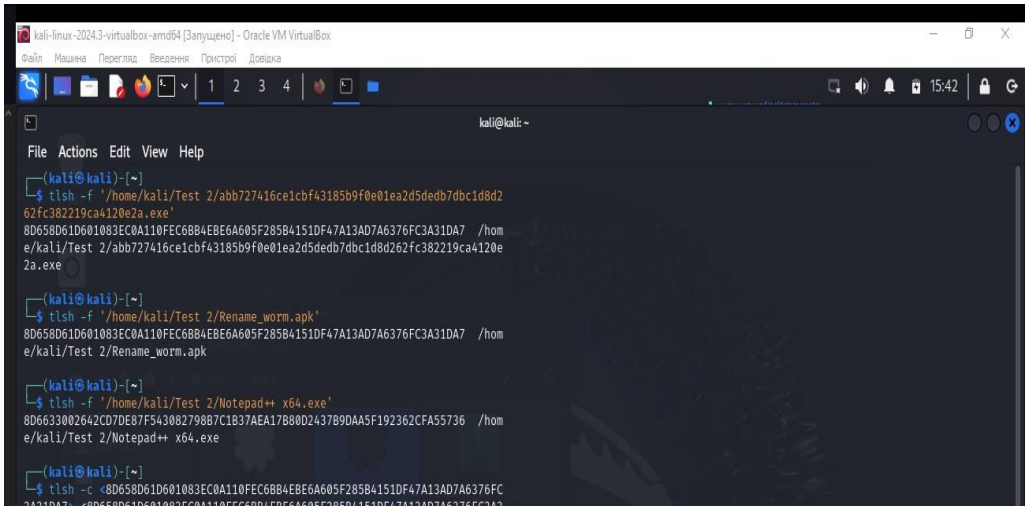

Рисунок 3.5– результати гешування методом TLSH

Як ми бачимо геші файлів *abb727416ce1cbf43185b9f0e01ea2d5dedb7dbc1d8d262fc382219ca4120e2a.exe* та *Rename_worm.apk* ідентичні. Це означає, що ці два файли є однаковими або мають мінімальні відмінності (настільки незначні, що не вплинули на гешування TLSH). А геш файлу *Notepad++ x64.exe* відрізняється від перших двох, тому цей файл не схожий на *abb727416ce1cbf43185b9f0e01ea2d5dedb7dbc1d8d262fc382219ca4120e2a.exe* чи *Rename_worm.apk.*

Для підтвердження достовірності перевірки файлу типу ШПЗ, а саме черв’яка нами було перевірено його наявність в базі Virus Total. Аналогічно було зроблено і з безпечним файлом.

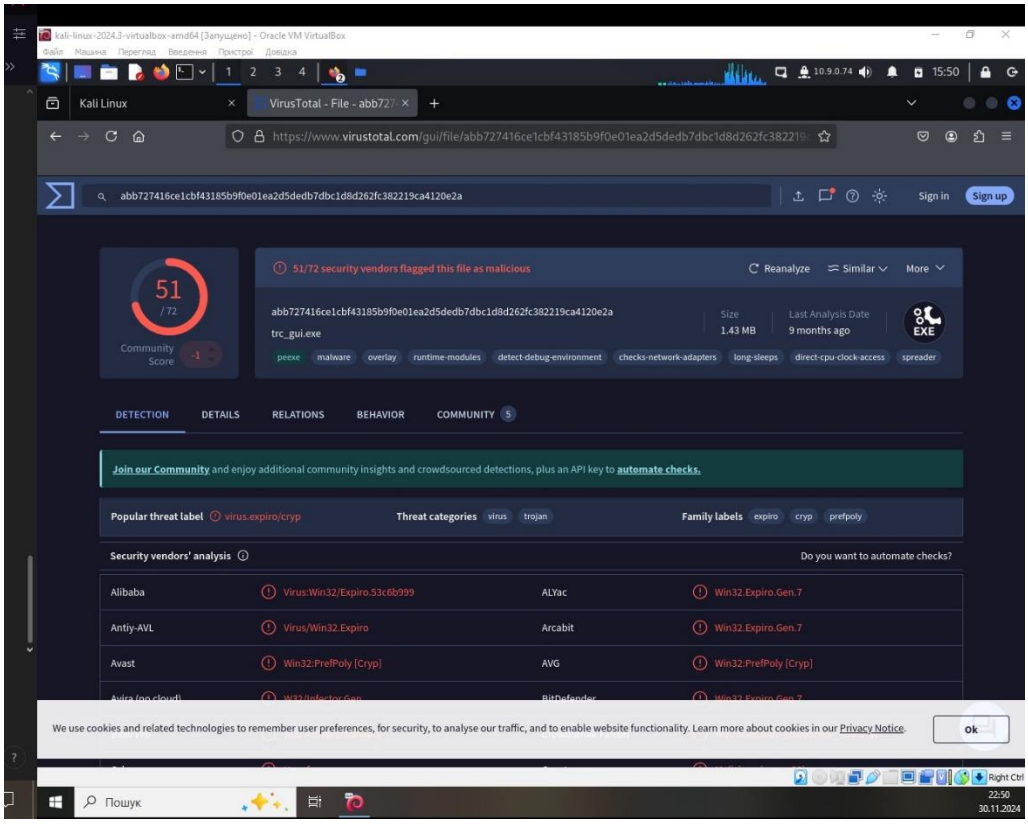

Рисунок 3.6– наявність черв’яка в базі Virus Total

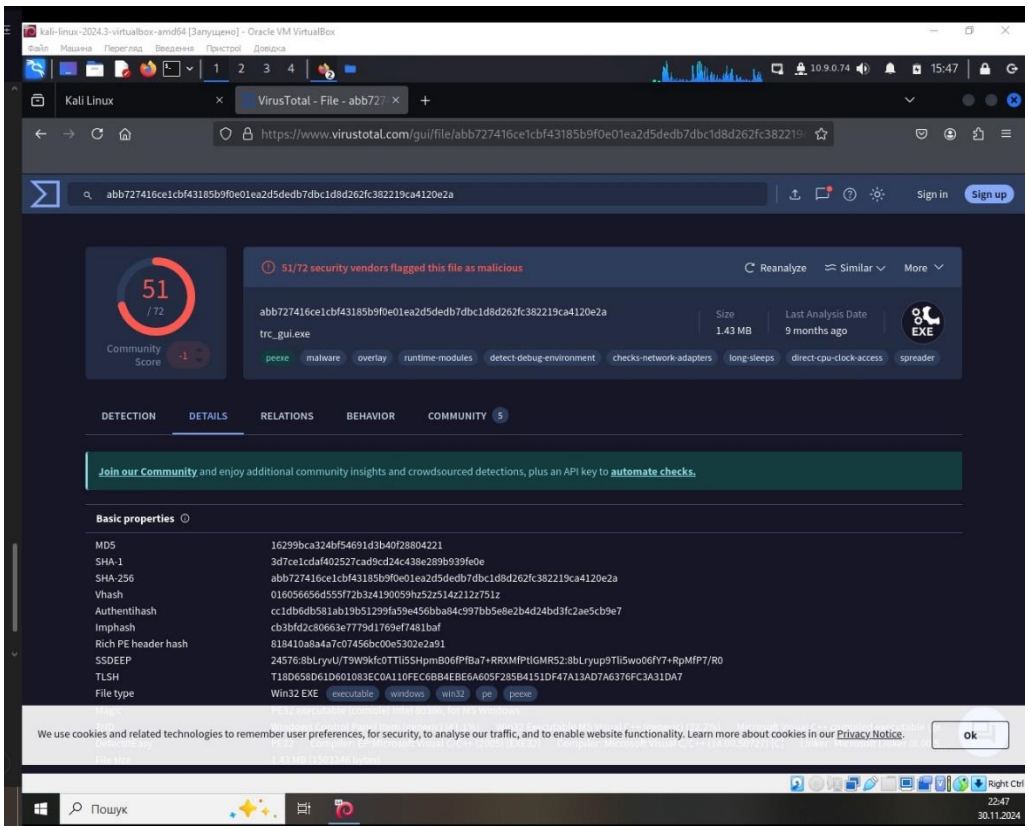

Рисунок 3.7 – деталі черв’яка в базі Virus Total

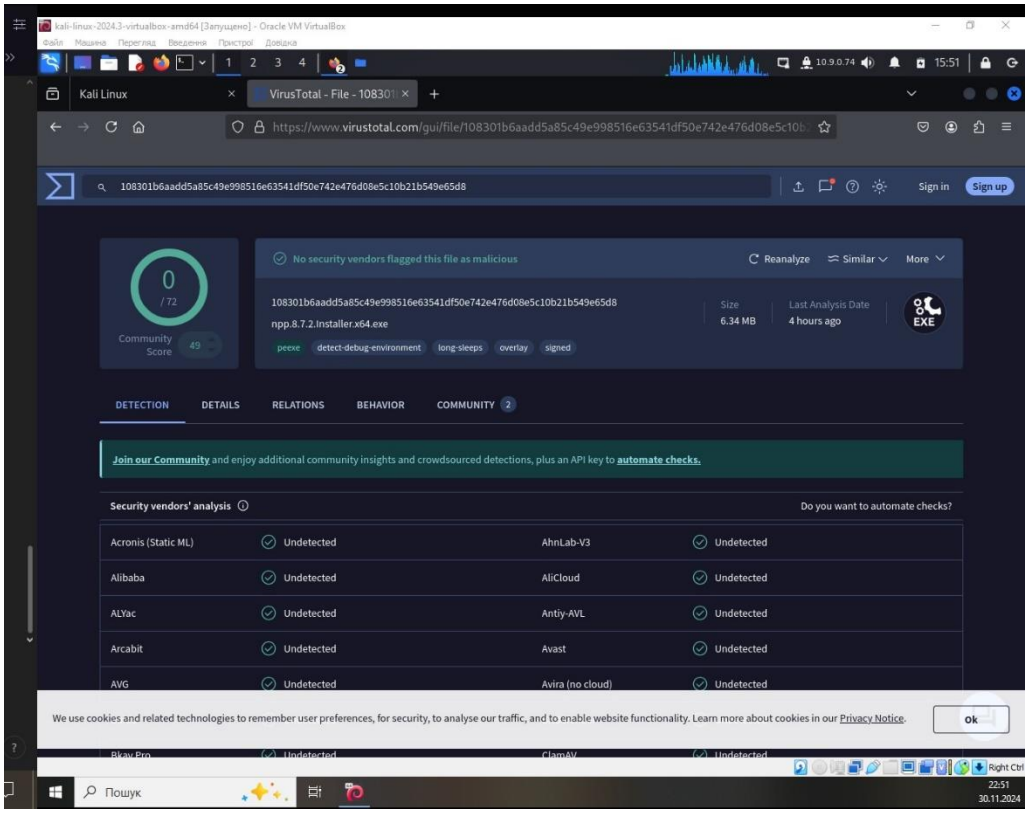

Рисунок 3.8 – відсутність легітимної програми в базі Virus Total

На основі проведених експериментів було встановлено, що метод TLSH є ефективним інструментом для аналізу шкідливого програмного забезпечення. Він продемонстрував високу точність у визначенні схожості одного черв'яка. Це підтверджує придатність TLSH для задач класифікації та побудови дерев родинної належності ШПЗ. Метод TLSH дозволяє не лише класифікувати зразки шкідливого ПЗ, але й будувати дерева спорідненості (фамільні дерева) для аналізу еволюції шкідливих програм. Це відкриває можливості для прогнозування потенційних атак на основі споріднених загроз, створення баз даних для швидкого виявлення нових модифікацій відомих вірусів та вдосконалення систем раннього попередження про можливі загрози. Проте варто завжди пам'ятати про необхідність доповнення TLSH іншими методами аналізу для підвищення загальної ефективності системи виявлення ШПЗ. Якщо конкретизувати переваги то основними є підтримка роботи з великими файлами та адаптивність до структурних змін. А з недоліків, чутливість до значних змін у файлі може зменшувати точність.

3.3 Аналіз результатів експериментів виявлення Троянів за допомогою SSDeep

SSDeep реалізує підхід нечіткого гешування (fuzzy hashing), що дозволяє оцінювати схожість між файлами навіть за умови їх часткової модифікації. Основу інструменту складає алгоритм context triggered piecewise hashing (CTPH), який розбиває файл на блоки відповідно до контексту. Для кожного блоку обчислюється геш, а потім формується контрольна сума, що зберігає структурну інформацію про файл коду [26]. Результатом є можливість визначати рівень схожості між файлами, виражений у відсотках (від 0 до 100%).

Цей підхід є особливо корисним для аналізу шкідливих програм, оскільки дозволяє виявляти подібність між оригінальним зразком і його модифікаціями, навіть якщо структура файлу зазнала змін.

Для аналізу методу було виконано низку експериментальних завдань, що включали виявлення схожості між зразками шкідливих програм та аналіз їх відповідності гешам у базі даних VirusTotal. Зокрема, проводився аналіз оригінального зразка троян та його дубліката з іншою назвою та розширенням, а також перевірка легітимної програми, не пов'язаної зі шкідливим програмним забезпеченням, для виключення помилкових спрацювань.

Варто зазначити, що попередньо для проведення експериментів було створено віртуальне середовище на основі операційної системи Kali Linux, яка надає широкий спектр інструментів для аналізу кіберзагроз. До системи було завантажено та налаштовано SSDeep, що забезпечило можливість обчислення нечітких гешів і виконання порівнянь між файлами.

Для завантаження та встановлення SSDEEP, який доступний у репозиторії Kali Linux потрібно ввести команду *sudo apt install ssdeep*. Після встановлення, щоб переконатися, що програма успішно встановлена і працює можна ввести команду для перевірки версії SSDEEP, а саме *ssdeep –V*.

Після успішної перевірки можна розпочинати генерацію нечітких гешів для кожного зі зразків (троян, його дублікат, легітимна програма) за допомогою SSDeep.

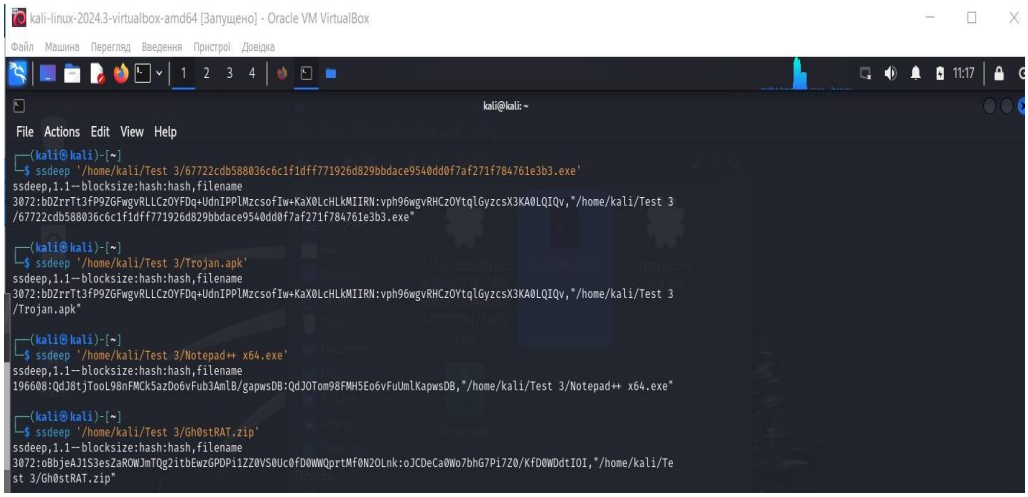

Рисунок 3.9 – Результати гешування методом TLSH

Коли ми отримали дані результати можна порівняти генеровані геші між собою для визначення рівня схожості. Результати цього порівняння для зручності варто помістити в таблицю 3.2.

Таблиця 3.2 –

Пара файлів	Рівень схожості (%)	Висновок
Оригінальний троян — Notepad++	0	Жодної подібності, помилкове спрацювання відсутнє
Оригінальний троян — дублікат	100	Файли ідентичні, зміна назви не впливає
Дублікат — Notepad++	0	Жодної подібності, помилкове спрацювання відсутнє

Також для оцінки відповідності з відомими шкідливими файлами можна провести перевірку отриманих гешів у базі даних VirusTotal.

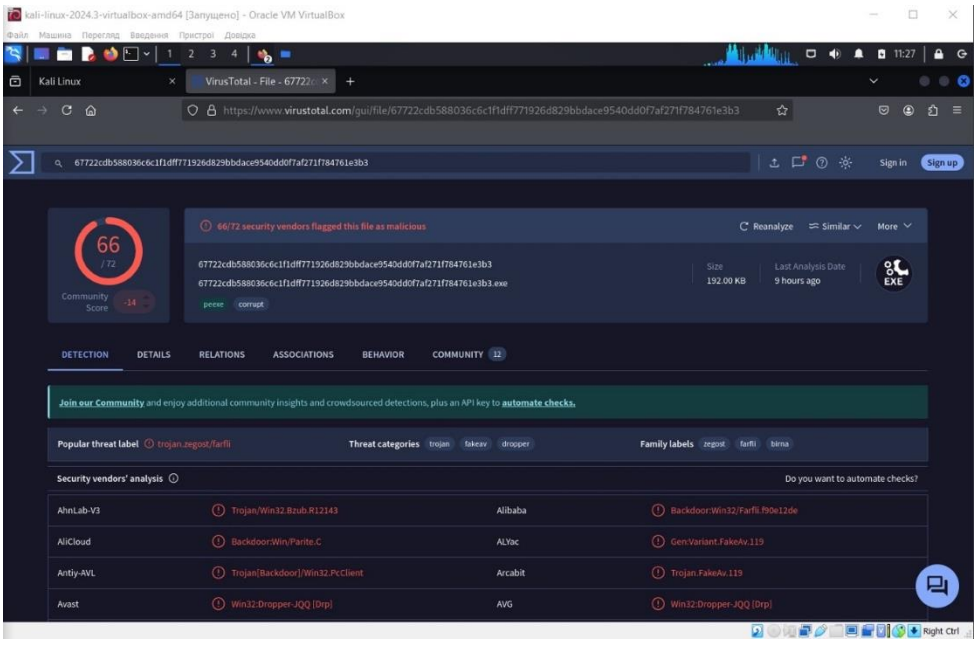

Рисунок 3.10 – наявність трояну в базі Virus Total

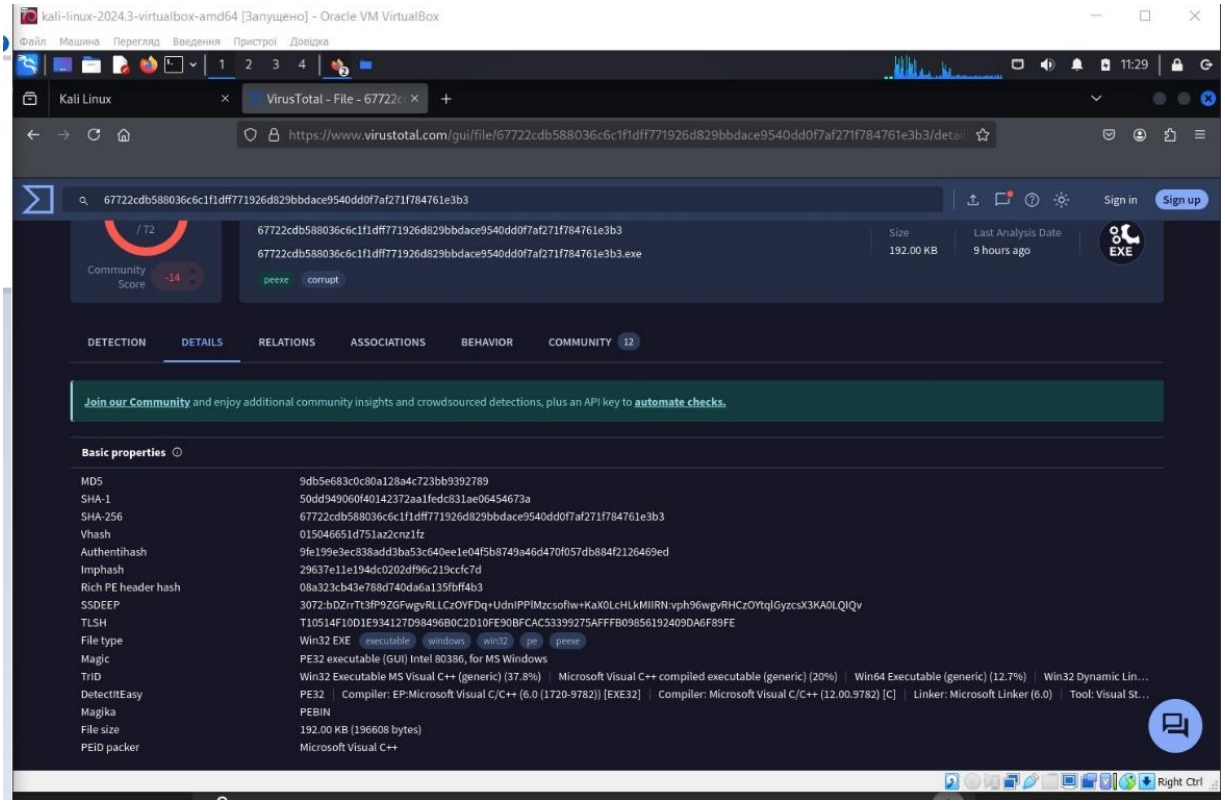

Рисунок 3.11 – Деталі трояну в базі Virus Total

Як ми бачимо що дана перевірка підтвердила відповідність файлу до відомих шкідливих файлів. Далі ми проведемо окрему перевірку гешу легітимного файлу, щоб визначити, чи він є присутнім у базі даних як шкідливий.

Дана перевірка підтвердила що легітимний файл відсутній у базі даних як шкідливий.

Цей підхід дозволив оцінити як точність SSDeep у визначенні схожості, так і його здатність працювати в інтеграції з глобальними базами даних для виявлення шкідливого ПЗ.

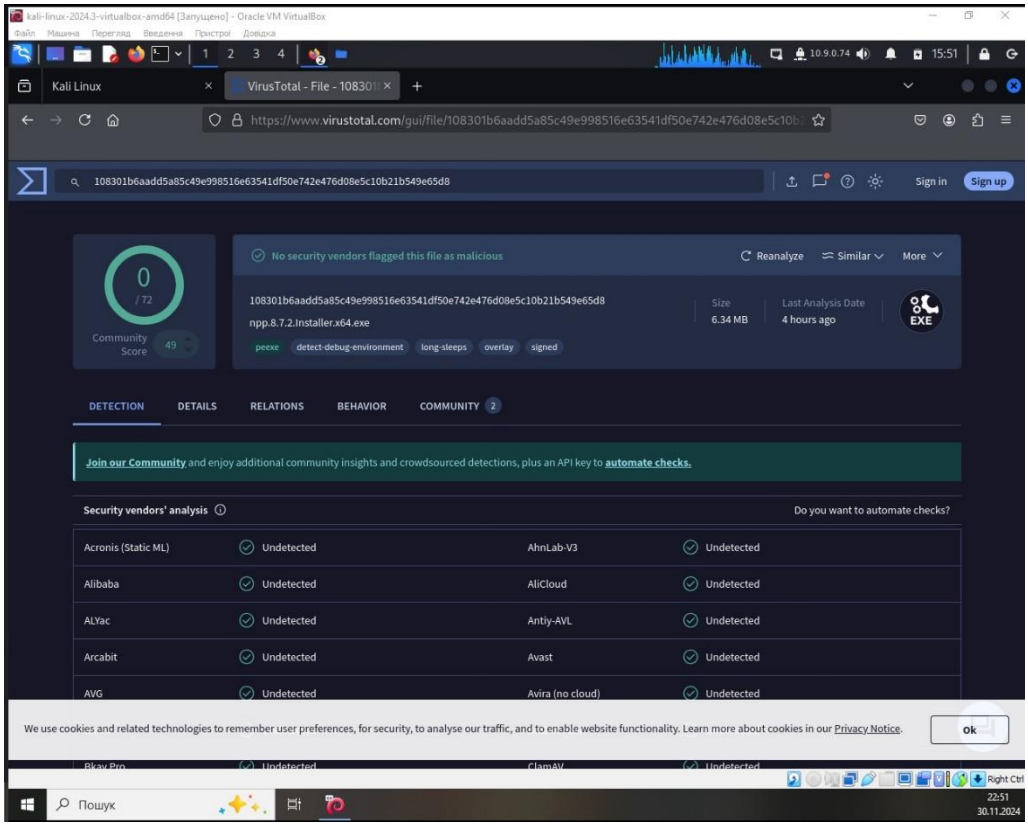

Рисунок 3.12 – Результати перевірки на відсутність легітимної програми в базі Virus Total

Весь цей проведений аналіз показав, що SSDeep ефективно визначає ідентичність між файлами, навіть за умов зміни їхньої назви та розширення. При порівнянні троянів із легітимною програмою схожість не була виявлена, що свідчить про відсутність помилкових спрацювань у даній вибірці.

ВИСНОВКИ

В кваліфікаційній роботі розв’язано актуальну задачу дослідження ефективності алгоритмів виявлення шкідливого програмного забезпечення на основі гешування, чутливого до локальності, з використанням локалізації гешування. При цьому отримано наступні результати.

1. Проведено аналіз сучасних методів виявлення шкідливого програмного забезпечення, серед який виділено наступні: Сигнатурний аналіз, Евристичний аналіз, Аналіз поведінки (Behavioral Analysis), Методи машинного навчання (Machine Learning) та Методи на основі локально чутливого гешування (LSH)

2. Досліджено принципи роботи LSH та гешів чутливих до місцевості. Зокрема такі: близькості, спеціальних геш-функцій та ефективності. Близькості – це те що близькі об'єкти у вихідному просторі мають потрапляти до одного "відра" з високою ймовірністю, а далекі - до різних. Спеціальних геш-функцій це те що функції гешування ґрунтуються на тому що враховують локальну схожість даних.

3. Розроблено алгоритми виявлення шкідливого програмного забезпечення на основі гешу чутливого до місцевості LSH. Ці алгоритми дозволяють швидко виявляти ШПЗ завдяки зменшенню обсягу порівнянь.

4. Досліджено ефективність та практичну значущість запропонованих алгоритмів на реальних прикладах шкідливого програмного забезпечення. LSH ефективно виявляє варіації відомих троянів, черв'яків чи майнерів та зменшує потребу в повному скануванні файлів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кондратюк А. В. Алгоритми виявлення спаму на основі гешу подібності / Spam Detection Algorithms Based on Hash Similarity : Кваліфікаційна робота : 125. Тернопіль, 2023. 75 с. URL: http://dspace.wunu.edu.ua/bitstream/316497/50189/1/КР_Кондратюк.pdf.

2. Лекції. Кафедра обчислювальної математики. URL: http://om.univ.kiev.ua/users_upload/15/upload/file/pr_lecture_25.pdf (дата звернення: 27.09.2024).

3. Нечітке хешування SSDeep. SPY-SOFT.NET. URL: https://spy-soft.net/ssdeep/ (дата звернення: 03.12.2024).

4. Смірнов Д.С., Іваніцкий Н.Ю., Кондратюк А.В. Виявлення невідомих шкідливих програм за допомогою SSDeep. Матеріали науковопрактична конференція молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ - 2023), Тернопіль, 2023. – С. 115-116.

5. Тітова В. Ю., САВЧУК С. О., ЧЕРНИШ В. Ю. КОНЦЕПТУАЛЬНА МОДЕЛЬ СИСТЕМИ ЗАХИСТУ ІНФОРМАЦІЇ В СУЧАСНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ. Вісник Хмельницького національного університету. 2019. № 3. С. 164–167.

6. Троян – вірус, замаскований в програмному забезпеченні. ESET. URL: https://www.eset.com/ua/support/information/entsiklopediya-ugroz/troyan/?utm_source=admitad&utm_medium=afiliados&tagtag_uid=08bc862d2f9a4501367feec5bbfeb73a&utm_content=2075099 (дата звернення: 02.06.2024).

7. Тригуб А. П. Система прогнозування кредитного ризику на основі поєднання методів навчання з вчителем і без вчителя: Дипломна робота. Київ, 2023. 99 с. URL: https://ela.kpi.ua/server/api/core/bitstreams/73e915a8-9333-4f1a-b8c0-97c222efdaba/content (дата звернення: 01.10.2024).

8. Що таке бекдор і який його вплив на комп'ютер. UAEU Українська ТОП Газета. URL: https://uaeu.top/digital-online/shcho-take-bekdor-i-yakij-jogo-vpliv-na-komp-yuter.html (дата звернення: 03.06.2024).

9. Що таке пошук векторної подібності та чим він корисний?. Unite.AI. URL: https://www.unite.ai/uk/what-is-vector-similarity-search-how-is-it-useful/ (дата звернення: 01.10.2024).

10. Contributor T. What is dropper? | Definition from TechTarget. WhatIs. URL: https://www.techtarget.com/whatis/definition/dropper (date of access: 23.06.2024).

11. Contributor to Wikimedia projects. Nilsimsa Hash - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Nilsimsa_Hash (date of access: 03.12.2024).

12. Damiani, E., De Capitani Di Vimercati, S., Paraboschi, S., Samarati, P. An Open Digest-based Technique for Spam Detection. In D. A. Bader, A. A. Khokhar. 17th ISCA International Conference on Parallel and Distributed Computing Systems 2004, PDCS, 2004, pp. 559-564.

13. Dealing with the dialer virus. Spam – Antivirus - Identity Theft - Scams and Fraud: STOP IT. URL: https://www.spamlaws.com/dialer-virus.html (date of access: 27.06.2024).

14. Dunlea J. Machine Learning Techniques for Advanced Malware Detection. Akkio. URL: https://www.akkio.com/post/malware-detection-using-machine-learning (date of access: 25.09.2024).

15. GitHub - ssdeep-project/ssdeep: Fuzzy hashing API and fuzzy hashing tool. GitHub. URL: https://github.com/ssdeep-project/ssdeep (19.11.2024).

16. GitHub - trendmicro/tlsh. GitHub. URL: https://github.com/trendmicro/tlsh (date of access: 30.11.2024).

17. Hristov H. Introduction to K-D trees. URL: https://www.baeldung.com/cs/k-d-trees (date of access: 27.09.2024).

18. Jakobs C., Lambertz M., Hilgert J.-N. Ssdeeper: evaluating and improving ssdeep. Forensic science international: digital investigation. 2022. Vol. 42. P. 301402. URL: https://doi.org/10.1016/j.fsidi.2022.301402 (19.11.2024).

19. Locality sensitive hashing | data science with apache spark. index | Data Science with Apach Spark Test. URL: https://george-jen.gitbook.io/data-science-and-apache-spark/locality-sensitive-hashing (date of access: 27.09.2024).

20. Locality Sensitive Hashing (LSH): The Illustrated Guide | Pinecone. URL: https://www.pinecone.io/learn/series/faiss/locality-sensitive-hashing/ (date of access: 03.12.2024).

21. Locality-sensitive hashing - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Locality-sensitive_hashing (date of access: 03.12.2024).

22. Malware exploit: threat and critical security vulnerability - overt software. Overt Software. URL: https://www.overtsoftware.com/malware-exploit-threat-and-critical-security-vulnerability/#t-1674208785392 (date of access: 30.06.2024).

23. Master ssdeep hash for improved file security & identification. Blue Team Resources. URL: https://blueteamresources.in/ssdeep-hash/ (date of access: 19.11.2024).

24. Moisiuk O., Samila A. Investigation about electromagnetic field topology of the spiral coil for nqr detector of explosive and narcotic substances. Herald of Khmelnytskyi National University. 2021. Vol. 299, no. 4. P. 101–107. URL: https://doi.org/10.31891/2307-5732-2021-299-4-101-107 (date of access: 25.09.2024).

25. Oliver J., Cheng C., Chen Y. TLSH - A Locality Sensitive Hash. Threat Encyclopedia | Trend Micro (US). URL: https://documents.trendmicro.com/assets/wp/wp-locality-sensitive-hash.pdf (date of access: 03.12.2024).

26. Optimizing ssDeep for use at scale. Virus Bulletin: Home. URL: https://www.virusbulletin.com/virusbulletin/2015/11/optimizing-ssdeep-use-scale (date of access: 19.11.2024).

27. Operationalizing TLSH Fuzzy Hashing - Magonia Research. Magonia Research. URL: https://www.magonia.io/blog/operationalizing-tlsh-fuzzy-hashing-for-detection (date of access: 03.12.2024).

28. Priyanka J. Malware Detection Using Machine Learning Techniques. eInfochip. URL: https://www.einfochips.com/blog/malware-detection-using-machine-learning-techniques (date of access: 25.09.2024).

29. Playbook of the Week: Uncovering Unknown Malware Using SSDeep - Palo Alto Networks Blog. URL: https://www.paloaltonetworks.com/blog/security-operations/playbook-of-the-week-uncovering-unknown-malware-using-ssdeep/ (date of access: 03.12.2024).

30. What is a rootkit? How to defend and stop them? | fortinet. Fortinet. URL: https://www.fortinet.com/resources/cyberglossary/rootkit (date of access: 30.06.2024).

31. What is a worm?. Cisco. URL: https://www.cisco.com/c/en/us/products/security/what-is-a-worm.html#~steps-of-a-worm-attack (date of access: 27.06.2024).

32. What is adware? | types of cyber threats | ESET. URL: https://www.eset.com/uk/types-of-cyber-threats/adware/?utm_source=admitad&utm_medium=afiliados&tagtag_uid=d0dc6dc480e27def75a4b7a420cf977f&utm_content=2075099 (date of access: 26.06.2024).

33. Zillya! - HackTool: той, що «ламає» все. Zillya! Антивірус – український антивірус. URL: https://zillya.ua/hacktool-toi-shcho-lama-vse (дата звернення: 26.06.2024).

34. Zillya! - Типи шкідливого ПЗ: downloader. Zillya! Антивірус – український антивірус. URL: https://zillya.ua/tipi-shkidlivogo-pz-downloader (дата звернення: 26.06.2024).

35. Zillya! - Типи шкідливих програм: від Trojan до Rootkit. Zillya! Антивірус – український антивірус. URL: https://zillya.ua/tipi-shkidlivikh-program-vid-trojan-do-rootkit (дата звернення: 02.06.2024).

ДОДАТОК А.

Копії публікацій

УДК 004.056

ВИЯВЛЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ ХЕШУ ПОДІБНОСТІ

Васильків Д.С., студент гр. КБм-11
Науковий керівник: д.т.н., професор Яцків В. В.
Західноукраїнський національний університет

Швидке зростання кількості нових та модифікованих версій шкідливого програмного забезпечення (ШПЗ) вимагає розробки нових та удосконалення існуючих методів та засобів їх ефективного виявлення.

Метою даної роботи є дослідження ефективності виявлення шкідливого програмного забезпечення на основі хешу подібності.

Для виявлення ШПЗ на основі хешу подібності використовують різні типи хеш-функцій. Основна відмінність даних алгоритмів від звичайних методів хешування є те, що хеш-колізії максимізуються, а не мінімізуються. Одними з найпоширеніших хеш-функцій даного типу є TLSH, Ssdeep та Sdhash.

Даний метод діє за простим алгоритмом, а саме, спочатку створюється або отримується хеш значення файлу, що потребує перевірки, потім він завантажується в програму, яка під'єднана до баз даних відомих зразків ШПЗ та проходить перевірку шляхом порівняння. Кінцевим кроком є вжиття заходів, які залежать від результату перевірки.

Метод виявлення шкідливого програмного забезпечення на основі хешу подібності, має як переваги так і недоліки. До переваг варто віднести швидкість, вона пов'язана з відсутністю потреби сканування всього файлу. Ще одною перевагою є точність тому що хеш-значення є унікальним для кожного файлу. Також до переваг можна віднести можливість виявлення навіть модифікованого ШПЗ Найбільшим недоліком цього методу є постійна необхідність оновлення баз даних та неможливість виявлення нового ШПЗ.

Варто зазначити, що алгоритми на основі хешу подібності є ефективним інструментом для виявлення модифікованих версій ШПЗ.

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

ЗМІСТ

СИСТЕМИ ТА ТЕХНОЛОГІЇ КІБЕРБЕЗПЕКИ

УДК 004.056.5

Дмитро ВАСИЛЬКІВ

Західноукраїнський національний університет

АНАЛІЗ ФАЙЛІВ ЗА ДОПОМОГОЮ SSDEEP

Вступ. Враховуючи стрімко зростаючі обсяги цифрових даних та збільшення кількості атак на них, або їх використання у здійснені атак, компаніям та їхнім співробітникам сфери кібербезпеки потрібно все більше автоматизувати процеси фільтрації. Здебільшого ці процеси використовують для відсіювання безпечних файлів від загального потоку серед якого є підозрілі або нецікаві файли [1].

Можливість порівнювати та знаходити схожі файли є одною з ключових для вирішення даної проблеми, проте як вона працює в масштабі. Насправді часто ми стикаємось з неможливістю обробити та перевірити усі потрібні файли, тому що нам довелося б провести нездійсненну кількість порівнянь. Ще гірше, якщо ці порівняння мали б відбуватися поза базою даних. Проте для полегшення даних процесів спеціалісти розробили оптимізацію ssDeep порівнянь, фактично завдяки їй ми можемо зменшити час на дані операції [2].

Мета: аналіз файлів з метою виявлення шкідливого програмного забезпечення за допомогою SSDEEP.

1. Алгоритм аналізу файлів

SSDeep Hash - це нечіткий хеш-алгоритм, розроблений для швидкого визначення та співставлення файлів. MD5 або SHA-1 (і наступні версії) це традиційні криптографічні хеш-функцій, вони ефективні, якщо потрібно точно перевірити збігу файлах, проте коли нам потрібно перевірити навіть незначні зміни ми можемо використовувати саме SSDeep. Він враховує ці варіації у файлах і фактично це дозволяє його використовувати для аналізу шкідливого програмного забезпечення, цифрової експертизи та інших безпекових завдань. SSDeep - це інструмент з відкритим кодом, який використовується для створення та порівняння нечітких хешів. В його основу покладений алгоритм частково керованого хешування (CTPH), який надає можливість ділити дані на сегменти змінних розмірів. Це гарантує що невеликі зміни не призводять до суттєвої різниці у хеш-значенні, що полегшує визначення схожості між файлами. Алгоритм роботи цього є хешу є досить маштабним, та широко висвітленим у наукових працях, проте в даній роботі, я б хотів висвітлити алгоритм дій під час та перед застосуванням SSDEEP [3, 4].

Алгоритм виконання тестування файлів за допомогою SSDEEP, розпочинається з встановлення інструменту SSDEEP, після цього йде генерація хешів для файлів, далі збереження хешів для бази порівняння, після нього вже відбувається саме порівняння і аналіз результатів. Варто ще зазначити що доцільно доповнити алгоритм автоматизацією процесу та інтерпретація результатів. Алгоритм складається з шести етапів. Розглянемо кожен з етапів детальніше.

1. Встановлення інструменту SSDEEP. Для початку роботи потрібно

встановити цей інструмент. Процес встановлення залежить від операційної системи. Це передбачає завантаження, встановлення та перевірка:

- для ОС Windows потрібно перейти на офіційний сайт SSDEEP або GitHub-репозиторій: SSDEEP на GitHub, завантажити потрібні файли, далі можливо буде необхідність розпакувати архів додати каталог з утилітою до змінної середовища PATH, щоб мати доступ до команди з будь-якого місця треба у розділі Змінні середовища додати шлях до SSDEEP. Для перевірки нам треба буде відкрити командний рядок (Cmd) та ввести команду ssdeep, якщо з'являється довідка або інформація про використання, встановлення завершено успішно;
- для більшості дистрибутивів Linux SSDEEP доступний у стандартних репозиторіях. Для Debian/Ubuntu виконується команда sudo apt update sudo apt install ssdeep, для Fedora/RHEL sudo dnf install ssdeep, а для Arch Linux sudo pacman -S ssdeep;
- в macOS встановлення проходить через Homebrew. Якщо він ще не встановлений потрібно ввести команду:

/bin/bash -c $(curl -fsSL

https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh),

після цього команда *brew install ssdeep* .встановлює SSDEEP, для перевірки можна ввести *ssdeep* у терміналі. У разі успіху з'явиться список команд або інформація про програму.

Варто зазначити що у користувачів можуть виникати помилки, незрозумілості або інші перешкоди, але головне не зупинятись на цьому технічному етапі. Якщо виникли проблеми, потрібно перевірити документацію в репозиторії SSDEEP. Також важливо завжди використовувати останню версію репозиторію для отримання актуальних функцій і виправлень помилок.

2. Генерація хешів для файлів. Після встановлення SSDEEP, можна створити розмиті хеші для файлів. Це ключовий етап, який дозволяє отримати унікальні рядки для подальшого порівняння.

Для генерація хешів поотрібно відкрити термінал (або командний рядок). Написати команду *ssdeep MediaCreationTool_22H2.exe*, важливо що *MediaCreationTool_22H2.exe* - файл, для якого потрібно згенерувати хеш. Результат може виглядати приблизно так: 196608:MmtHa+5hH1km/Sf7byFXKEBmih9S5rQ5FNFl001p4Ki:Y+5RB/SDbyFBH9 cQD/l00/4. Це і є розмитий хеш MediaCreationTool_22H2 (завантажувача windows 10). також є можливість генерації хешів для декількох файлів. Для цього використовується рекурсивне створення хешів у каталозі завдяки команді *ssdeep -r directory/.* Результат: у терміналі буде виведено хеші для кожного файлу. *Directory/* - шлях до каталогу з файлами [3].

На цьому етапі ми фактично отримуємо потрібну інформацію для створення основи.

3. Створення бази хешів Щоб здійснювати порівняння нових файлів з існуючими, необхідно створити базу хешів. Ця база містить хеші всіх файлів, які будуть використовуватись як еталон.

Для початку нам потрібно згенерувати хеші для всіх файлів у каталозі, найпростіше це зробити за допомогою команди *ssdeep -r directory/ > hash_database.txt.* Фактично вона складається з двох частин, перша це directory/ -

шлях до каталогу з файлами, яка нам знайома з попереднього етапу. А друга hash_database.txt - файл, у який зберігаються результати. Структура цього файлу складається з 3 частин. Перша - розмитий хеш, друга (після символу :) - зменшений підхеш для оптимізації та третя - ім'я файлу. На практиці це може виглядати так:

384:4iVhXoM/3kuTazPOr0Zp5e...:oM/kuTazPOr0Z... file1.txt

192:Wr12kk8RGmN3rzEF4AG0jN...:2k8RGmN3rzEF4... file2.txt

Важливо, що у користувачів є можливість оновлювати базу хешів. Якщо потрібно додати нові файли до бази то це можна робити двома способами, а саме згенерувати хеш для окремого файлу або для кількох файлів у новому каталозі. Це робиться завдяки команді *ssdeep new_file.txt >> hash_database.txt* або *ssdeep -r new_directory/ >> hash_database.txt* відповідно до потреби. Важливо що обов'язково потрібно ставити символ >>, тому що він додає хеш до існуючої бази без її перезапису.

Перевірка бази хешів відбувається досить просто, варто тільки відкрити файл *hash_database.txt*, щоб переконатись, що всі записи є. Також варто перевіряти правильність шляхів до файлів, особливо якщо база використовуватиметься на іншій машині.

Завдяки даному етапу ми закладаєм певну основу для подальшого порівняння. Для комфортнішого та безпечнішого комікування з базою нам краще використовувати добре організовані каталоги, щоб уникати плутанини, зберігати копію бази на випадок її пошкодження та якщо буде потрібно автоматизувати обробку, як варіант можна форматувати базу в JSON або CSV.

4. Порівняння файлів. Після того як база хешів створена, наступний етап - порівняння нових файлів із базою, щоб визначити схожість.

Якщо потрібно перевірити окремий файл на схожість із файлами в баз, то ми просто порівнюємо файл, який потрібно перевірити з файлом із хешами бази, а саме *target_file.txt* та *hash_database.txt*. Потрібна команда виглядає так: *ssdeep -k hash_database.txt target_file.txt*.

У результаті буде виведено список схожих хешів та відсоток схожості (число перед ім'ям файлу) наприклад:

ssdeep,1.1--blocksize:hash:hash,filename

35:Wr12kk8RGmN3rzEF4AG0jN...:2k8RGmN3rzEF4... file1.txt

50:Hd72mx9YGq4DszAB8IJ0kM...:mx9YGq4DszAB8... file2.txt

Для порівняння кількох файлів з базою використовують рекурсивну перевірку, вона проводиться командою *ssdeep -r new_directory/ -k hash_database.txt* [3].

На цьому етапі ми завершили отримання даних для проведення аналіз, після цього залишається тільки підвести підсумки та як варіант покращення автоматизувати дану роботу. Також потрібно не забувати зберігати результати у файл для подальшого аналізу, це робиться командою ssdeep -k hash_database.txt target_file.txt > comparison_results.txt. І ще для систем моніторингу можна налаштувати скрипти, які аналізуватимуть результати порівняння та генеруватимуть сповіщення, якщо буде знайдено високий рівень схожості.

5. Аналіз результатів. Після порівняння файлів за допомогою SSDEEP необхідно інтерпретувати результати, щоб зробити висновки про схожість і

потенційні загрози. Розтлумачити відсотки схожості можна за такою шкалою:

- 100% - файли ідентичні;
- 80 - 99% - файли майже однакові, можливі невеликі зміни (наприклад, метадані чи зміни коду без зміни структури);
- 50 - 79% - часткова схожість, яка може свідчити про наявність змін у структурі або частині даних, також це можуть бути версії того самого файлу;
- 30 - 49% - низька схожість. Може вказувати на використання однієї бібліотеки або фрагментів коду, нечастий збіг через збіг частини даних;
- менше 30% - ймовірно, файли не пов'язані між собою.

Аналіз результатів SSDEEP дозволяє ідентифікувати схожі файли та оцінити рівень змін. Використовувати порогові значення можна для автоматизації класифікації та зберігання результатів для подальшого моніторингу. Наступний етап - інтеграція в робочий процес.

6. Автоматизація процесу. Щоб ефективно використовувати SSDEEP для аналізу файлів, можна автоматизувати весь процес - від генерації хешів до аналізу результатів. Це зменшить ручну роботу, скоротить час обробки та підвищить точність. Автоматизація процесу з використанням Bash або Python дозволяє ефективно створювати базу, аналізувати нові файли та отримувати сповіщення про підозрілі збіги. Розклад запуску скриптів забезпечить регулярність перевірок.

Висновок. SSDEEP дозволяє виявляти модифіковані або частково змінені файли, що особливо корисно для виявлення зловмисного програмного забезпечення або дублікованих даних. SSDEEP показує високий рівень ефективності в завданнях пошуку частково ідентичних файлів, але потребує правильної інтерпретації результатів для зменшення помилкових спрацьовувань. Автоматизація процесу дозволяє значно спростити інтеграцію цього алгоритму в робочий цикл, забезпечуючи оперативність та масштабованість. Таким чином, SSDEEP є надійним інструментом для забезпечення безпеки та аналізу даних, який можна легко адаптувати до різних сценаріїв використання. Також варто пам'ятати, що файли з високою схожістю можна перевіряти детальніше: наприклад, використати інші методи аналізу (статичний чи динамічний).

Перелік використаних джерел.

1. Jakobs C., Lambertz M., Hilgert J.-N. Ssdeeper: evaluating and improving ssdeep. Forensic science international: digital investigation. 2022. Vol. 42. P. 301402. [Електронний ресурс].- Режим доступу: https://doi.org/10.1016/j.fsidi.2022.301402 (date of access: 19.11.2024).

2. Virus Bulletin :: Optimizing ssDeep for use at scale. Virus Bulletin: Home. [Електронний ресурс].- Режим доступу: https://www.virusbulletin.com/virusbulletin/2015/11/optimizing-ssdeep-use-scale (date of access: 19.11.2024).

3. GitHub - ssdeep-project/ssdeep: Fuzzy hashing API and fuzzy hashing tool. GitHub. [Електронний ресурс].- Режим доступу: https://github.com/ssdeep-project/ssdeep (date of access: 19.11.2024).

4. Master ssdeep hash for improved file security identification. Blue Team Resources. [Електронний ресурс].- Режим доступу: https://blueteamresources.in/ssdeep-hash/ (date of access: 19.11.2024).