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

КОНДРАТЮК Андрій Володимирович

Алгоритми виявлення спаму на основі гешу подібності / Spam Detection Algorithms Based on Hash Similarity

спеціальність: 125 – Кібербезпека

освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи
КБм -22
А.В. Кондратюк

Науковий керівник
д.т.н., професор В.В.Яцків

Кваліфікаційну роботу
допущено до захисту:

«____» _____ ___2023 р.

Завідувач кафедри
_____ **В.В.Яцків**

ТЕРНОПІЛЬ - 2023

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

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ В.В.Яцків
«____» _____ 2022 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

КОНДРАТЮК АНДРІЙ ВОЛОДИМИРОВИЧ
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Алгоритми виявлення спаму на основі гешу подібності / Spam Detection Algorithms Based on Hash Similarity

керівник роботи д.т.н., професор В.В. Яцків

затверджені наказом по університету від 1 грудня 2022 року № 491

2. Строк подання студентом закінченої кваліфікаційної роботи 1 грудня 2023 р.

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

4. Основні питання, які потрібно розробити:
– провести аналіз підходів до виявлення спаму;
– розкрити принципи функціонування методів гешування з урахуванням локальності;
– провести порівняння методів гешування з урахуванням локальності;
– провести оцінка подібності методом Нільсімса.
– розробити алгоритм ідентифікація спаму з використанням SSDeep.

5. Перелік графічного матеріалу у роботі:
порівняння результатів нечіткого гешування;
приклади обчислення геш значення з використанням SSDeep;
приклади обчислення геш значення на основі TLSH;
оцінка подібності повідомлень;
порівняння гешів подібності.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 8 грудня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз підходів до виявлення спаму	12.2022 р. – 03.2023 р.	
2	Методи гешування з урахуванням локальності	03.2023 р. – 05.2023 р.	
3	Ідентифікація спаму з використанням SSDeep	05.2023 р. – 11.2023 р.	

Студент _____ Кондратюк А.В.
(підпис)

Керівник роботи _____ д.т.н., професор В.В. Яцків
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Алгоритми виявлення спаму на основі гешу подібності» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека та захист інформації» освітньо-професійної програми «Кібербезпека» написана обсягом 75 сторінок і містить 9 ілюстрації, 2 таблиці, 1 додаток та 27 джерел за переліком посилань.

Метою кваліфікаційної роботи є дослідження та оцінка ефективності алгоритмів гешування з урахуванням локальності для виявлення спаму.

Методи досліджень. Для розв’язання поставлених задач у даній кваліфікаційній роботі використано: криптографічні методи гешування, нечіткі методи гешування та методи гешування з урахуванням локальності.

Результати дослідження: проведено порівняння продуктивності методів гешування TLSH, SDHASH і SSDEEP, показано, що метод TLSH показує кращий рівень хибних позитивних результатів.

Розроблено алгоритм ідентифікації спаму з використанням SSDeep, який забезпечує виявлення подібних файлів а також спам листів.

Результати роботи можуть бути застосовані при виборі методів при розробці систем ідентифікації спаму на основі гешу подібності.

Ключові слова: ГЕШ-ЗНАЧЕННЯ, СПАМ, НЕЧІТКІ ГЕШІ, КРИПТОГРАФІЧНІ ГЕШІ.

ABSTRACT

Qualification work on "Spam Detection Algorithms Based on Hash Similarity" for the degree of "Master" in the specialty 125 "Cybersecurity" educational and professional program "Cybersecurity" is written in 75 pages and contains 9 illustrations, 2 tables, 1 appendice and 27 source according to the list of links.

The purpose of the qualification work is to research and evaluate the effectiveness of hashing algorithms taking into account locality.

Research methods. To solve the problems in this qualification work, cryptographic hashing methods, fuzzy hashing methods and locality-based hashing methods were used.

Research results: a comparison of the performance of hashing methods TLSH, SDHASH and SSDEEP was made, it was shown that the TLSH method shows a better rate of false positive results.

A spam identification algorithm was developed using SSDeep, which provides detection of similar files as well as spam emails.

The results of the work can be applied in the selection of methods in the development of spam identification systems based on the similarity hash.

Keywords: HASH VALUES, SPAM, FUZZY HASHES, CRYPTOGRAPHIC HASHES.

ЗМІСТ

ВСТУП

Актуальність роботи. Електронні листи та SMS є найпопулярнішими засобами сучасної комунікації, і зі збільшенням кількості користувачів електронних листів та SMS-повідомлень кількість спаму також зростає. Спам, який часто вважають небажаною поштою кіберсвіту, перетворився на постійну проблему для користувачів Інтернету, компаній і онлайн-платформ. Значення спаму в контексті кібербезпеки стосується будь-яких небажаних і часто нерелевантних або недоречних повідомлень, які надсилаються через Інтернет, як правило, великій кількості користувачів, головним чином для реклами, фішингу, розповсюдження шкідливого програмного забезпечення чи інших подібних цілей. Цей термін найчастіше асоціюється з небажаними повідомленнями електронної пошти, але він також стосується повідомлень, надісланих за допомогою інших електронних засобів, таких як миттєві повідомлення, платформи соціальних мереж або мобільні програми [1].

Спамові електронні листи та SMS спричиняють значну втрату ресурсів через непотрібне переповнення мережевих посилань. Хоча більшість спаму надходять від рекламодавців, які прагнуть просувати свої продукти, деякі з них набагато зловмисніші, як-от фішингові електронні листи, які мають на меті змусити жертв надати конфіденційну інформацію, як-от дані для входу на веб-сайт або дані кредитної картки. Цей тип кіберзлочинності відомий як фішинг. Щоб протидіяти спаму зроблено багато досліджень для створення детекторів спаму, здатних фільтрувати повідомлення та електронні листи такі як спам або не спам [2].

Отже, дослідження методів автоматичного виявлення спаму, мета яких класифікувати електронні листи на категорії спаму та не спаму є актуальною задачею.

Мета і завдання дослідження. Метою роботи є дослідження та оцінка ефективності алгоритмів гешування з урахуванням локальності для виявлення спаму.

Досягнення визначеної мети передбачає вирішення таких завдань:

– провести аналіз підходів до виявлення спаму;

– розкрити принципи функціонування методів гешування з урахуванням локальності;

– провести порівняння методів гешування з урахуванням локальності;

– провести оцінка подібності методом Нільсімса.

– розробити алгоритм ідентифікація спаму з використанням SSDeep.

Об’єкт дослідження – процеси ідентифікації спаму в електронній пошті;

Предмет дослідження – методи та алгоритми гешування з урахуванням локальності.

Методи досліджень. Для розв’язання поставлених задач у даній кваліфікаційній роботі використано: криптографічні методи гешування, нечіткі методи гешування та методи гешування з урахуванням локальності.

Наукова новизна одержаних результатів. Проведено порівняння методів гешування з урахуванням локальності з використанням методу Нільсімса.

Практичне значення отриманих результатів. Розроблено алгоритм ідентифікації спаму з використанням SSDeep, який забезпечує виявлення подібних файлів а також спам листів.

Публікації та апробація КР.

1. Кондратюк А.В., Кметик В.В. Виявлення програм-вимагачів. Матеріали науково-практичного симпозіуму «Захист інформації», Тернопіль, 2023. – С. 96-97.

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

1 АНАЛІЗ ПІДХОДІВ ДО ВИЯВЛЕННЯ СПАМУ

1.1 Види спам-атак

Неприємність спаму неминуче потрапить у майже всі типи цифрових комунікаційних засобів, які використовуються в сучасну епоху. Серед них спам через електронні листи завжди був однією з найбільш використовуваних арен для шахраїв.

Частини даних електронної пошти складаються з кількох різних блоків, як показано на рисунку 1.1, тоді як таблиця 1.1 узагальнює ці блоки [3].

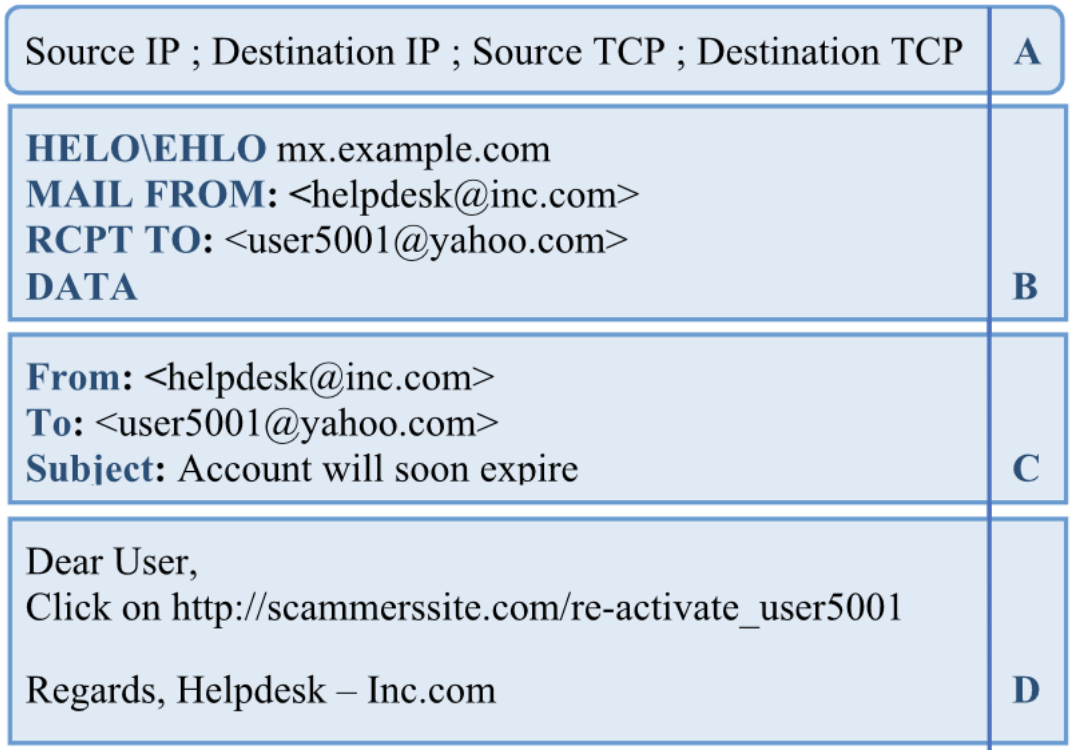

Рисунок 1.1 – Частини даних електронного листа

Існує низка атак, якими постійно користуються шахраї у всьому світі, такі як спуфінг електронної пошти, фішинг, варіанти фішингових атак, такі як spear phishing, clone phishing, whaling, приховане перенаправлення тощо. Окрім спуфінгу та фішингу, у спам-листах також з’явилися такі різновиди, як клікджекінг. Хакери навіть пішли дальше, щоб приховати текст за зображеннями, щоб боротися з антиспамовими програмами [4 - 7].

Таблиця 1.1 – Пояснення частин даних електронної пошти

Частини даних	Ідентифіковані як	Пояснення
A	TCP/IP Header	Корисне навантаження повідомлень загорнуто в пакети TCP/IP. IP-адреси джерела та призначення є тут найважливішою інформацією.
B	SMTP Envelop	Починається з ідентифікації поштових обмінників (може містити MAIL FROM є вихідним джерелом, а адреса одержувача переходить до сегмента RCPT TO. Ці дві адреси є фактичними електронними адресами джерела та призначення. Однак вони необов'язкові та можуть бути підроблені. EHLO та DATA – це дві команди SMTP.
C	SMTP Header	Поля «Від» і «Кому» в цій частині призначені лише для перегляду (відображаються в клієнтах електронної пошти), шахраї можуть заповнити їх даними, щоб приховати фактичний маршрут. За замовчуванням вони беруть значення з конверта.
D	Body	Підчастина заголовка SMTP. Вміст електронної пошти міститься в цій частині, включаючи вкладення

Фішингові сповіщення можуть призвести до значного негативного прибутку на акції або ринкової вартості глобальних компаній [8]. Також є ряд, які вказують на руйнівний вплив на компанії, чиї електронні листи були

позначені системами захисту від спаму як спам, але насправді не були спамом.

1.1.1 Фішинг електронної пошти

Фішинг електронної пошти є одним із найпоширеніших способів здійснення спам-атак на відправників, який досягається шляхом маніпулювання частиною даних C (поле «Від») або B. Мета полягає в тому, щоб створити повідомлення таким чином, щоб воно виглядало надісланим від когось або звідкись, часто відомого користувачеві, крім фактичного джерела. Спамери також втручаються в домен, який передається в операторі HELO, так що здається, що лист надійшов з якогось відомого домену [9, 10].

Це вказує на те, що спуфінг також може статися в частині B даних (SMTP Envelop). У 2017 році малайзійська нафторозподільча компанія зазнала значних фінансових збитків понад 1 мільйон доларів США через підробку електронної пошти [11].

1.1.2 Spear phishing

На практиці «Spear Phishing» є формою загального сімейства фішингу електронної пошти, оскільки він вводить в оману легітимні повідомлення. Фішингове шахрайство може додатково надати посилання на фіктивний веб-сайт, де від кінцевого користувача вимагається ввести конфіденційну фінансову та особисту інформацію [12 - 14].

Він діє на тілі електронної пошти, тобто на частині даних D. Такий тип фішингу також може містити вкладення, які можуть містити зловмисне програмне забезпечення. Також можна, в основному використовуючи трюки соціальної інженерії, створити повідомлення таким чином, без шкідливих посилань або вкладень, що користувач буде змушений виконати певні дії на основі вмісту електронної пошти, що в кінцевому підсумку принесе користь шахраям [15].

Спам у випадку Spear Phishing створюється з використанням особистої інформації про користувача, яка часто збирається за допомогою методів соціальної інженерії та надсилається здається, з надійного джерела. Ці типи атак часто важче виявити за допомогою традиційних фільтрів через складну персоналізацію вигляду та вмісту. Фішинг можна використовувати для створення серйозної атаки, відомої як «Advanced Persistent Threats» (APT). Наприклад, в 2018 році конфіденційна фінансова інформація співробітників ABC Bus Company була скомпрометована через фішингові електронні листи [16].

1.1.3 Whaling

Різновид фішингових атак, у цій формі атака часто спрямована на високопосадовців компанії. У випадку китобійного промислу або «шахрайської атаки на генерального директора» веб-сторінка/електронна пошта, що імітує себе, матиме більш серйозну форму на рівні керівника. Оскільки це також працює в частині даних D, буде створено контент, який здебільшого націлений на керівника вищої ланки, який має певний рівень допуску в організації та майже завжди є або терміновою проблемою керівника, що впливає на всю компанію, або скарга клієнта. Електронні листи будуть отримані з підробленого джерела, маскуючись під законну ділову установу (та сама чи інша компанія) або навіть генеральний директор самої хост-компанії. Ризики та небезпека подібні до інших форм фішингу та спуфінгу [17].

Основний європейський виробник електричних кабелів і проводів, Leone AG, втратив величезні 40 мільйонів євро через складну корпоративну електронну аферу з використанням комбінації фішингових і китобійних атак [18]. Під час нещодавньої китобійної атаки у 2018 році французька мережа кінотеатрів Pathé втратила понад 21 мільйон доларів [19].

1.2 Сучасні системи боротьби зі спамом

Розглянемо поширені системи захисту від спаму, більшість із яких доступні на різних платформах, наприклад, автономні програми або онлайн-рішення. Вони не застосовують жодних підходів на основі штучного інтелекту (ШІ).

1.2.1 Схеми авторизації/автентифікації сервера

Нижче наведено деякі з відомих схем авторизації/автентифікації сервера

1. Пошта, ідентифікована доменними ключами (DKIM). Один із найскладніших фреймворків, які використовуються сьогодні і в яких весь процес реалізується за допомогою шифрування з відкритим ключем. Однак, через досить низький рівень впровадження цієї досить потужної структури ESP, певний електронний лист із нульовим полем DKIM не може бути позначений як підтверджений спам. DKIM також схильний до підробки DKIM працює як у частині C, так і в частині D (див. рисунок 1.1). Метод «шифрування з відкритим ключем (PKE)» вважається одним із актуальних методів шифрування, які використовуються досі. Зазвичай у процесі задіяно два ключа. «Відкритий ключ», один із двох ключів, призначених кожній стороні, публікується у відкритому каталозі в місці, де будь-хто може легко його знайти, наприклад, за адресами електронної пошти. Другий є «Закритий ключ», секретний ключ, який зберігається кожною стороною. До завершення успішного циклу шифрування та дешифрування за допомогою PKE необхідно виконати декілька кроків [20].

1. Знайдіть P і Q, два великих (наприклад, 1024-розрядних) простих числа.

2. Виберіть E так, щоб $E > 1, E < PQ$, а E і $(P-1)\,(Q-1)$ були взаємно простими, тобто вони не мали спільних простих множників. E не

обов’язково має бути простим числом, але воно має бути непарним. $(P-1)(Q-1)$ не може бути простим, оскільки це парне число.

3. Обчисліть D так, щоб $DE = 1\ (mod\ (P-1)(Q-1))$

4. Функція шифрування

$$C = (T\hat{}E)\ mod\ PQ,$$

де C – зашифрований текст (ціле додатне число), T – відкритий текст (ціле додатне число). Повідомлення, що шифрується, T, має бути меншим за модуль PQ.

5. Функція дешифрування – це $T = (C\hat{}D)\ mod\ PQ$, де C – зашифрований текст (ціле додатне число), T – відкритий текст (ціле додатне число).

Відкритий ключ – це пара (PQ, E), а D – закритий ключ. Основною перевагою цієї криптографії є те, що можна вільно публікувати свій відкритий ключ, оскільки не існує відомих простих методів обчислення D, P або Q, наданих лише (PQ, E) – відкритий ключ. Крім того, такі популярні постачальники послуг електронної пошти (ESP), як-от Gmail, тепер забезпечують наскрізне шифрування через S/MIME (безпечні/багатоцільові розширення Інтернет-пошти), яке само по собі базується на шифруванні з відкритим ключем. Однак такий тип криптографії часто повільніший за інші доступні методи.

1.2.2 Політика відправника

Фреймворк політики відправника (Sender Policy Framework, SPF) сьогодні став одним із найважливіших механізмів автентифікації електронної пошти, який часто використовується разом із DKIM.

Однак ця технологія сама по собі є автономною структурою та протоколом перевірки електронної пошти, створеним для виявлення та блокування підробки електронної пошти, надаючи систему, яка дозволяє обмінникам електронної пошти автентифікувати, що вхідна пошта з домену справді надійшла з IP-адреси, авторизованої адміністратором цього домену.

SPF фактично запобігає шахраям розповсюджувати електронні листи від чиєїсь особи.

Сервер-одержувач вхідного повідомлення шукатиме запис SPF сервера-відправника разом із повідомленням. Запис SPF матиме список дозволених IP-адрес, з яких дозволено надсилати електронні листи цього конкретного відправника (або користувача). Отже, якщо список не містить IP-адреси сервера, який надіслав повідомлення на сервер-одержувач, сервер-одержувач не дозволить прийняти повідомлення. SPF працює як у частині A, так і в частині B (див. рисунок 1.1) [21].

Незважаючи на те, що наразі не всі поштові сервери реалізували SPF, ця технологія впроваджується швидкими темпами.

1.2.3 Пропозиції на основі архітектурних модифікацій

Простий і невибагливий дизайн SMTP протягом тривалого часу вважався причиною низки спам-атак. Деякі стверджують, що хакери навіть підробляють поле «Дата» в заголовку SMTP, щоб зберігати свої спам-повідомлення у верхній частині папки «Вхідні» одержувача, щоб можна було негайно привернути увагу [22].

Для боротьби з цим недоліком, запропоновано використовувати спеціальний «сервер відміток часу» для автентифікації дати надсилання кожного електронного листа. Також запропоновані зміни (наприклад, модифікацію деяких SMTP етапів транзакцій) у плані SMTP, щоб зробити його більш безпечним і надійним як кращий вибір для протоколу передачі пошти. Однак такі кроки не завжди практичні з багатьох причин.

Попереднє приймальне тестування електронних листів було обговорено Esquivel та ін. [23], який працює шляхом аналізу особливостей окремих транзакцій SMTP, таких як послідовності повідомлень EHLO/HELO. Їх можна далі розділити на різні категорії на основі робочого механізму, наприклад «дефекти протоколу». Дефект протоколу може виявити будь-які

додаткові підозрілі блоки даних у вхідному буфері до того, як відбудеться транзакція повідомлення EHLO/HELO.

Фільтри в мережевих серверах блокування спаму повинні використовувати засоби виявлення підозрілих моделей поведінки абонентів спаму VoIP, які можуть бути вбудовані в протоколи сигналізації, що використовуються в VoIP, такі як SIP (Session Initiation Protocol). SIP – це протокол прикладного рівня, який активно використовується для створення, зміни та завершення мультимедійного сеансу (потокове відео, онлайн-ігри, обмін миттєвими повідомленнями тощо) через Інтернет-протокол [24].

SIP також може використовувати автентифікацію Message Digest (MD5) з метою безпеки. Щоб виявити підозрілу поведінку, SIP за своєю природою може застосовувати автоматизовані структури, які можуть аналізувати повідомлення, щоб визначити, чи повідомлення є синтаксично неправильним, чи воно не має явного значення, чи його важко інтерпретувати чи може призвести до тупикової ситуації [25].

Наведені вище приклади є важливими кроками у напрямку зміцнення структури SMTP на кореневому рівні. З іншого боку, сама популярність і прийняття SMTP на першому етапі як де-факто обраного протоколу для спілкування електронною поштою встановило деякі суворі стримуючі фактори. Наприклад, незначна модифікація протоколу може внести хвилю змін в інші взаємопов’язані сервіси, необхідні для успішної доставки пошти, як щодо ефективності, так і щодо корисності. Таким чином, такі структурні модифікації, необхідні для базової інфраструктури зв’язку електронною поштою, безсумнівно, ускладнять роботу. Це обмеження SMTP було проблемою вже давно, і спамери використовували недоліки з дуже ранньої стадії.

1.2.4 Моделі співпраці

Згідно зі стратегіями спільного моделювання фільтрації спаму, кожне повідомлення доставляється певній кількості одержувачів. Конкретне

повідомлення напевно буде отримано та оцінено іншим користувачем. Спільні моделі демонструють процес захоплення, запису та опитування цих ранніх суджень.

Згодом ці колекції стають достатньо значними, щоб винести вердикт на певний електронний лист. Низка методів доступна для досягнення різних етапів успішної спільної роботи [26].

1. Криптографічне гешування. Надзвичайно успішний метод у попередні часи спілкування електронною поштою, де великі постачальники електронної пошти (Hotmail, Yahoo!, AOL тощо) математично обчислювали алфавітно-цифрові рядки від 32 до 128 символів, відомі як підпис електронної пошти (значення гешу), для зберігання це в базі даних. Постачальники працюють над ідеєю, що для досягнення мети спамери розсилатимуть серію спам-листів, і деякі з цих спам-листів потраплять до їхніх облікових записів honeypot, тобто облікових записів, які були створені спеціально для перехоплення спаму. Ці постачальники також покладаються на той факт, що згенерований підпис буде значною мірою відрізнятися для спаму та неспамових електронних листів [20].

Тому, щойно шаблон підпису збігається зі шаблоном спаму, він додається до бази даних для підпису спаму, і коли інші електронні листи надходять до будь-якого іншого облікового запису клієнта, вони миттєво відкидаються, якщо буде виявлено спам – через відповідність підпису (обчислюється за допомогою того самого методу, із заголовка та тіла, таким чином, метод працює як у частині C, так і в частині D (див. рисунок 1.1) до того, що зберігається в базі даних, за умови, що запис знайдено. Постачальники надають базу даних іншим постачальникам послуг електронної пошти (ESP), і, таким чином, коли електронний лист визначається як спам, він оновлюється в кількох із цих баз даних, розташованих по всьому світу.

Message Digest 5 (MD5) був одним із популярних варіантів для криптографічного гешування. Це криптографічний алгоритм, який приймає

вхідні дані будь-якої довжини та генерує дайджест повідомлення довжиною 128 біт, часто відомий як «відбиток» або «геш» вхідних даних. MD5 дуже корисний, коли потенційно довге повідомлення потрібно швидко обробити та/або порівняти.

Проблема з такою технікою полягає в тому, що спамери вже досягли успіху у розробці інструментів, які можуть фактично зламати алгоритми гешування. Крім того, питання оновлення бази даних також іноді є вузьким місцем, оскільки воно оновлюється автоматично, але із затримкою, і цього вікна достатньо для багатьох шахраїв [21]. Крім того, якщо цю базу даних зламано, то це завіса для ESP. Таким чином, з роками точність техніки знизилася. SHA-3 – це нова розробка, яка повністю замінює MD5 та його близькі варіації.

2. Нечітке гешування. Наступний підхід це використання принципу гешування (наприклад, нечітке гешування) для виявлення спам-кампаній шляхом кластеризації електронних листів на основі подібних цілей.

Нечітке гешування можна ефективно використовувати для вимірювання схожості двох послідовностей символів шляхом обчислення балів на основі подібності спам-повідомлень. Нечітке гешування базується як на функції «традиційного гешування» (наприклад, MD5), так і на функції «постійного гешування». Значення Hash рядка M = m0... mn−1 можна отримати з (1), де p – модуль, а 0 ≤ a < p – основа a p, і b зазвичай є простими числами і не мають спільних простих множників.

$$h(M) = \left(\sum_{i=1}^{n-1} a^{n-1-i} \cdot m_i\right) mod\ p \qquad (1.1)$$

При нечіткому гешуванні вхідні дані ділить на частини довільного розміру. Потім ці фрагменти гешуються за допомогою традиційної геш-функції.

Конкатенація геш-значень, отриманих після гешування всіх фрагментів, утворює «нечітке геш-значення» заданого вмісту. Геш-значення

часто є значно компактнішим, ніж вихідний рядок символів, оскільки вони дають вихідні дані фіксованої довжини, незалежно від довжини вхідних даних. Таким чином, вміст, який не зовсім ідентичний, але дещо відрізняється певним чином, все одно може бути згрупований під одним значенням.

Даний висновок базується на точці зору, що електронні листи від однієї кампанії матимуть вищий показник схожості між собою, тоді як такі показники будуть сильно відрізнятися від електронних листів від різних спам-кампаній. Було також показано, що електронні листи з однієї кампанії мають подібний тип URL-адреси або електронної адреси. Недоліком роботи є те, що вона не розглядає питання щодо «Асиметричного обчислення відстані», де оцінка кластерної відстані може стати недетермінованою, якщо змінюється порядок введення [22].

Через ряд недоліків MD5, зокрема, невелика зміна вхідних даних різко змінює відповідне геш-значення, що не завжди бажано, оскільки часто вміст кількох спам-листів дещо відрізняється, але вони все одно вважаються спамом і часто з однієї кампанії. Застосування локально чутливого алгоритму гешування, відомого як «Nilsimsa», значно зросло для цілей гешування, він генерує оцінку від 0 (несхожі об’єкти) до 128 (дуже схожі об’єкти або ідентичні). Nilsimsa використовує 5-байтове ковзне вікно фіксованого розміру, яке аналізує вхідні дані побайтно перед тим, як генерувати триграми (групу з трьох послідовних символів) можливих комбінацій вхідних символів.

Триграми відображаються в 256-бітний масив для отримання бажаного гешу. Ще одна контрольована близька варіація Nilsimsa на основі k-NN, відома як TLSH (Trend Locality Sensitive Hashing), привернула велику увагу в ці дні. Він надає оцінку подібності від 0 до 1000+, де будь-яка оцінка <=100 ідентифікує два файли, як подібні один до одного, і походять переважно з одного джерела [23].

Ще одним важливим прикладом є проект SSDEEP, програма часткового гешування, що запускається контекстом.

3. Центр розподіленої контрольної суми. DCC (Distributed Checksum Clearinghouse), інша структура обміну гешами проти спаму, працює, підраховуючи, скільки разів певне повідомлення було позначено як спам. Він бере контрольну суму тіла повідомлення та зберігає її в центрі обміну даними або на сервері. Таким чином, із кожним додатковим повідомленням до інформаційного центру про те, що повідомлення є спамом, кількість контрольної суми збільшується на 1. Таким чином масову пошту можна впевнено ідентифікувати, оскільки кількість відповіді та кількість контрольної суми зазвичай набагато вищі. Контрольні суми є нечіткими за своєю природою, і часто в процесі обміну контрольними сумами беруть участь декілька серверів DCC [13].

4. Внесення в сірий список. Сірий список передбачає, що законний відправник повторно надішле електронний лист, якщо початкова спроба виявиться невдалою, тоді як спамери просто перейдуть до наступного відправника й не потурбуються перевірити, чи було доставлено електронний лист. Однак цей підхід можна просто обійти, повторно надіславши спам [14].

5. Чорний та білий списки DNS. Чорний список DNS (сервер доменних імен) виконується двома різними варіантами. Перший передбачає підтримку списку IP-адрес поштового сервера, ідентифікованих як джерело або розповсюджувач спаму, у централізованій базі даних [15]. Інший спосіб – позначати спам на основі уніфікованих ідентифікаторів ресурсів (URI), як правило, доменних імен або веб-сайтів; тоді чорні списки складаються з таких шкідливих URI. Потім можна надати доступ до цих чорних списків або баз даних відомих IP-адрес або доменів, що надсилають спам, адміністраторам безкоштовно або платно. Сервер електронної пошти, який використовує цю службу, виконає додатковий DNS-запит на хості, який надсилає повідомлення, щоб визначити статус джерела; у цьому випадку запитуваний DNS-сервер буде сервером, наданим службою чорного списку

DNS. Однак усі такі чорні списки страждають від неможливості раннього виявлення зловмисних фішингових URL-адрес після атаки, оскільки процес оновлення бази даних є недостатньо швидким.

Проблеми з чорними списками полягають у тому, що спамери можуть часто змінювати адресу джерела. Крім того, сама адреса джерела може бути підроблена, як згадувалося раніше. У випадку чорних списків, що складаються з URI, спамери постійно створюють нові дешеві домени, перш ніж розпочати новий цикл масового спаму, залишаючи чорним спискам дуже мало часу для миттєвої реакції. Крім того, список часто оновлюється досить повільно, а отже, досить неефективний проти фішингових загроз електронної пошти, які орієнтуються на відвідування користувачами короткочасних фішингових веб-сайтів. Білий список – це практика ведення списку поштових серверів, якими керують лише перевірені законні адміністратори, або для прийому вмісту від сумлінних користувачів.

Різні організації мають власні білі списки, щоб полегшити роботу клієнтів. Коли справа доходить до білого та чорного списків джерел спаму, проект Spamhaus, започаткований у 1998 році, став робочою моделлю. Деякі провайдери та сервери електронної пошти використовують списки, щоб зменшити кількість спаму, що надходить до їхніх користувачів. Spamhaus також надає інформацію про певні домени та головний сервер для навмисного надання послуг підтримки спаму з метою отримання прибутку. Зараз у проекту понад 600 мільйонів підписників [10].

1.3 Архітектура мовної моделі BERT

Удосконалення в моделях машинного навчання, зокрема в обробці природної мови (NLP), дозволило значно вдосконалити різні цифрові продукти. Одним із помітних проривів є інтеграція BERT (Bidirectional

Encoder Representations from Transformers) у пошук Google, що Google вважає одним із найважливіших досягнень в історії пошуку.

BERT – це архітектура мовної моделі, яка змінила завдання обробки природної мови. На відміну від попередніх моделей, які покладалися на безконтекстне вбудовування слів, BERT представив двонаправлений підхід для захоплення контекстної інформації як з лівого, так і з правого контекстів слова в реченні.

Архітектура BERT базується на моделі Transformer, яка складається з кількох рівнів нейронних мереж самоконтролю та прямого зв'язку. Він призначений для попереднього навчання глибоких двонаправлених представлень із текстових даних без міток. Цей процес попереднього навчання дозволяє BERT вивчати загальні мовні моделі та зв'язки між словами.

Однією з ключових переваг BERT є його здатність вловлювати контекстне значення слів. Традиційні моделі, такі як word2Vec або GloVe, генерують представлення вбудованого слова для кожного слова, незалежно від його контексту. Навпаки, BERT враховує навколишні слова під час представлення слова, що призводить до більш точних і нюансованих представлень.

Щоб адаптувати BERT для конкретних завдань NLP, таких як аналіз настроїв або відповіді на запитання, поверх попередньо навченої моделі BERT додається вихідний рівень для конкретного завдання. Потім цей додатковий рівень налаштовується з використанням позначених даних, пов'язаних із конкретним завданням, що дозволяє BERT робити розширені прогнози або отримувати високоякісні мовні функції.

Принцип роботи BERT. BERT використовує попередню підготовку без нагляду з подальшим контрольованим налаштуванням, щоб використовувати силу розуміння мови. Етап попередньої обробки тексту включає токенізацію вхідних даних і поєднання маркерів, сегментації та позиційних вставок для створення вхідного представлення для кодера BERT.

Попередні навчальні завдання для BERT включають моделювання замаскованої мови, де передбачені замасковані лексеми, і передбачення наступного речення, яке оцінює зв’язок між двома реченнями.

Обмеження BERT. Незважаючи на те, що BERT, безсумнівно, значно вдосконалив обробку природної мови (NLP), важливо визнати, що жодна модель не позбавлена своїх обмежень. Ось деякі недоліки BERT.

1. Обчислювальні ресурси. BERT – це велика та складна модель із численними параметрами, яка потребує значних обчислювальних ресурсів як для навчання, так і для розгортання. Це може створити проблеми для дослідників і розробників з обмеженим доступом до потужного апаратного забезпечення.

2. Швидкість логічного висновку. Через розмір і архітектуру швидкість логічного висновку BERT може бути нижчою порівняно з меншими моделями. Це може викликати занепокоєння в програмах, які вимагають обробки текстових даних у режимі реального часу або майже в реальному часі.

3. Вимоги до пам’яті. Вимоги до пам’яті BERT можуть бути значними, особливо під час обробки довгих документів або кількох документів одночасно. Це може призвести до обмеження пам’яті на пристроях з обмеженими ресурсами.

4. Складність тонкого налаштування. хоча BERT можна точно налаштувати для конкретних завдань, цей процес може бути складним і трудомістким. Правильне тонке налаштування вимагає ретельного вибору гіпер параметрів і навчальних даних, що може бути неможливим для всіх проектів.

5. Адаптація до конкретного домену. Попередньо підготовлені ваги BERT можуть не бути оптимальними для кожного домену чи ніші. Тонке налаштування для спеціалізованих доменів потребує специфічних для домену даних і досвіду, що ускладнює процес впровадження.

6. Відсутність пояснень. Як і багато інших моделей глибокого навчання, процес прийняття рішень BERT може бути важко інтерпретувати. Ця відсутність пояснення може викликати занепокоєння в програмах, де розуміння модельних рішень має вирішальне значення.

7. Залежність від даних. Продуктивність BERT значною мірою залежить від якості та різноманітності навчальних даних. Якщо навчальні дані є упередженими, неповними або нерепрезентативними, продуктивність моделі може погіршитися.

8. Слова поза словниковим запасом. Словниковий запас BERT обмежений словами, з якими він навчався. Слова поза словниковим запасом можуть бути проблемою, оскільки BERT може важко з ними ефективно працювати.

9. Великий розмір моделі. Розмір BERT може ускладнити його розгортання на пристроях з обмеженими ресурсами, обмежуючи його використання в сценаріях периферійних обчислень.

10. Багатомовні обмеження. Хоча BERT підтримує кілька мов, його продуктивність може відрізнятися для різних мов, і деякі мови можуть не мати добре налаштованих попередньо навчених моделей.

Незважаючи на ці обмеження, BERT залишається потужним інструментом для вирішення різноманітних завдань НЛП. Однак важливо уважно розглянути ці недоліки та оцінити, чи є BERT правильним вибором для конкретного проекту, особливо в сценаріях, де критичними факторами є обчислювальні ресурси, швидкість або вимоги до домену.

Застосування BERT у різних галузях промисловості. Універсальні можливості BERT у обробці природної мови (NLP) призвели до його широкого впровадження в різних галузях. Основні галузі, де BERT широко використовується:

1. Електронна комерція та роздрібна торгівля: BERT використовується для покращення функціональності пошуку на платформах електронної

комерції. Він покращує рекомендації щодо продукту та точніше розуміє наміри користувача, що сприяє покращенню взаємодії з клієнтами.

2. Охорона здоров'я та науки про життя: BERT допомагає в аналізі медичної документації, розумінні записів пацієнтів і вилученні відповідної інформації з медичної літератури. Це допомагає в розробці систем підтримки прийняття клінічних рішень і автоматизації медичного кодування.

3. Фінанси та банківська справа: BERT покращує аналіз настроїв для фінансових новин, виявляє шахрайські дії та аналізує відгуки клієнтів, щоб покращити фінансові продукти та послуги.

4. Підтримка та обслуговування клієнтів: BERT використовується в чат-ботах і віртуальних помічниках, щоб надавати більш контекстно точні та змістовні відповіді на запити клієнтів, що сприяє кращому залученню клієнтів.

5. Юридичні питання та відповідність: BERT допомагає в аналізі документів для юридичних досліджень, перевірки контрактів і процесів належної перевірки. Це прискорює вилучення відповідної правової інформації з величезних обсягів текстових даних.

6. Медіа та розваги: BERT використовується для рекомендацій вмісту, аналізу настроїв у відгуках і створення підсумків вмісту, що дозволяє медіа-компаніям краще зрозуміти вподобання аудиторії.

7. Маркетинг і реклама: BERT допомагає в аналізі настроїв у соціальних мережах, оптимізації рекламних кампаній і створенні більш релевантного та привабливого вмісту для цільової аудиторії.

8. Автомобільна промисловість і виробництво: BERT допомагає аналізувати відгуки клієнтів, претензії щодо гарантії та технічну документацію, покращуючи якість продукції та задоволеність клієнтів.

9. Освіта та електронне навчання: BERT використовується в автоматизованих системах оцінювання, інтелектуальних системах навчання та платформах рекомендацій щодо вмісту, пропонуючи персоналізований досвід навчання.

10. Подорожі та готельний бізнес: BERT вдосконалює системи рекомендацій щодо подорожей, аналіз настроїв відгуків і аналіз відгуків клієнтів, сприяючи кращому плануванню подорожей і обслуговуванню клієнтів.

11. Урядові та державні послуги: BERT використовується для аналізу настроїв громадської думки, автоматизованого аналізу правових документів і підвищення ефективності державних послуг.

12. Енергетика та комунальні послуги: BERT допомагає в аналізі технічної документації, журналів технічного обслуговування та відгуків клієнтів для оптимізації роботи та покращення обслуговування клієнтів в енергетичному секторі.

13. Виявлення спаму. BERT це також революційне розуміння мови для виявлення спаму. Використання потужного мовного моделювання BERT для точної класифікації SMS-спаму.

Це лише кілька прикладів того, як BERT справляє значний вплив на різні галузі. Його здатність розуміти контекст і нюанси мови зробила його цінним інструментом для отримання розуміння, покращення процесів прийняття рішень і покращення досвіду клієнтів у різних секторах.

Таким чином, BERT – це новаторський прогрес у обробці природної мови (NLP). Його двонаправлений підхід і архітектура трансформатора зробили революцію в розумінні мови. BERT відмінно справляється з різними завданнями NLP, такими як класифікація тексту та аналіз настроїв.

Хоча BERT має такі обмеження, як вимоги до ресурсів і складність тонкого налаштування, він залишається незамінним у таких галузях, як електронна комерція, охорона здоров’я, фінанси та підтримка клієнтів. Його вплив охоплює від покращення функцій пошуку до покращення взаємодії з клієнтами та отримання інформації.

По суті, вплив BERT відчувається в усіх секторах, надаючи підприємствам можливість краще приймати рішення та персоналізовані

послуги. Незважаючи на свої обмеження, BERT є потужним інструментом, який просунув NLP вперед, змінивши мовну взаємодію та інтерпретацію.

Алгоритм впровадження BERT для виявлення SMS-спаму.

Щоб застосувати BERT для завдання виявлення SMS-спаму, необхідно виконати такі дії.

1. Імпортуйте необхідні бібліотеки та набір даних SMS-спаму.

2. Розділіть набір даних на тренувальний і тестовий набори.

3. Імпортуйте попередньо навчену модель BERT, наприклад BERT-base-uncased.

4. Токенізуйте та кодуйте послідовності SMS за допомогою токенізатора BERT.

5. Перетворіть закодовані послідовності в тензори.

6. Створіть завантажувач даних для ефективного завантаження даних під час навчання.

7. Визначте архітектуру моделі, використовуючи BERT як основу.

8. Тонко налаштуйте модель на наборі даних SMS-спаму.

9. Робіть прогнози щодо нових SMS-повідомлень за допомогою навченої моделі.

Переваги BERT для виявлення SMS-спаму. Використовуючи потужні можливості моделювання мови BERT, ми можемо досягти точнішої класифікації SMS-спаму. Двонаправлений підхід BERT дає змогу моделі фіксувати детальну контекстну інформацію, дозволяючи їй з більшою точністю розрізняти законні повідомлення та спам.

Це покращує взаємодію з користувачем, захищає від потенційних загроз безпеці та забезпечує оперативні дії проти небажаних повідомлень.

BERT, завдяки своєму двонаправленому підходу до моделювання мови, приніс значні успіхи в розумінні та обробці природної мови.

Впровадивши BERT для виявлення спаму через SMS, можна створити систему, яка точно класифікує повідомлення як спам чи ні, покращуючи взаємодію з користувачем і підвищуючи заходи безпеки. Інтеграція BERT у

різноманітні завдання NLP демонструє його трансформаційний вплив на обробку мови та закладає основу для подальшого прогресу в машинному навчанні.

2 МЕТОДИ ГЕШУВАННЯ З УРАХУВАННЯМ ЛОКАЛЬНОСТІ

2.1 Схема відбитків Рабіна

Ідея генерації більш гнучкого та надійного відбитка для двійкових даних була запропонована Рабіним у 1981 році [13]. Відтоді значні дослідження були зосереджені на розробці все більш складних методів зняття відбитків, але основна ідея Рабіна була перенесена з відносно невеликими варіаціями. Схема Рабіна заснована на випадкових поліномах, і її початковою метою було «створити простий алгоритм зіставлення рядків у реальному часі та процедуру для захисту файлів від несанкціонованих змін» [13].

Відбиток Рабіна можна розглядати як контрольну суму з низькою кількісно визначеною ймовірністю зіткнення, яку можна використовувати для ефективного виявлення ідентичних об’єктів. У 1990-х роках відновився інтерес до роботи Рабіна в контексті пошуку подібних об’єктів, з акцентом на текст, зокрема, для кількісного визначення подібності між текстовими файлами в інструменті sif для Unix [8]; в схемі виявлення копій [2]; для пошуку синтаксичних подібностей на веб-сторінках [3].

Основна ідея, яку називають фрагментацією або сегментацією, полягає у використанні ковзного відбитка Рабіна над вікном фіксованого розміру, яке розбиває дані на частини. Геш-значення h обчислюється для кожного вікна розміром w. Значення ділиться на константу c, а залишок порівнюється з іншою константою m. Якщо два значення рівні (тобто $m \equiv h \bmod c$), то дані у вікні оголошуються як початок блоку (прив’язки), а ковзне вікно переміщується на одну позицію. Цей процес триває, доки не буде досягнуто кінця даних. Для зручності значення c зазвичай дорівнює степеню двох ($c = 2^k$), а m є фіксованим числом від нуля до $c - 1$. Після визначення прив’язки базової лінії його можна використовувати кількома способами для вибору характерної риси. Наприклад, фрагменти між якорями можна вибрати як функції. Альтернативно, l байтів, що починаються з позицій прив’язки,

можуть бути обрані як функції, або можуть бути використані кілька вкладених функцій.

Хоча схеми вибирають випадкову вибірку ознак, вони є детермінованими, тобто, враховуючи однакові вхідні дані, створюють однакові характеристики. Крім того, вони локально чутливі, оскільки визначення точки прив'язки залежить лише від попередніх w байтів вхідних даних, де w може бути лише кількома байтами. Цю властивість можна використовувати для вирішення проблеми крихкості традиційного гешування на основі файлів і блоків.

Розглянемо дві версії одного документа. Один документ можна розглядати як похідний від іншого, вставляючи та видаляючи символи. Наприклад, сторінку HTML можна перетворити на звичайний текст, видаливши всі теги HTML. Зрозуміло, що це змінило б низку функцій, але фрагменти неформатованого тексту залишалися б недоторканими та створювали б деякі оригінальні функції, дозволяючи автоматично співвідносити дві версії документа. Для фактичного порівняння функцій геш-значення вибраних функцій зберігаються та використовуються як економічне представлення «відбитка».

2.1.1 Оцінка підходів до отримання відбитків

Рандомізована модель відбитків Рабіна в середньому працює добре, але страждає від проблем, пов'язаних із охопленням і хибнопозитивними показниками. Обидві ці проблеми можна пояснити тим фактом, що базові дані можуть мати значні варіації інформаційного вмісту. Як наслідок, розмір/розподіл функцій може сильно відрізнятися, що робить покриття відбитків дуже спотвореним. Подібним чином функції з низькою ентропією створюють аномально високі показники помилкових позитивних результатів, що робить відбиток ненадійною основою для порівняння.

Дослідження в області корисного навантаження створили більш складні версії відбитків Рабіна. Ці методи керують процесом вибору функцій,

щоб уникнути великих прогалин або кластерів. Однак вони не розглядають хибні спрацьовування через слабкі (неідентифікуючі) ознаки. Важливо визнати, що охоплення та помилкові спрацьовування фундаментально пов'язані; вибір слабких функцій для покращення покриття безпосередньо збільшує ризик хибнопозитивних результатів.

Зняття відбитків не за Рабіном. Загальна ідея будь-якої схеми подібності полягає у виборі кількох характерних (інваріантних) ознак з об'єкта даних і порівняння їх із ознаками, вибраними з інших об'єктів. Набір функцій можна розглядати як цифровий відбиток або підпис. Функцію можна визначити на кількох рівнях абстракції, де вищі рівні потребують більш спеціалізованої обробки. Для даної роботи функцію визначаємо як послідовність бітів. Іншими словами, розглядаємо двійкові дані як синтаксичну сутність і не намагаємося їх проаналізувати чи інтерпретувати.

Вибір статистично малоймовірних ознак. Процес вибору статистично малоймовірних ознак дещо схожий на використання Amazon статистично малоймовірних фраз для характеристики публікацій. Мета полягає в тому, щоб вибрати властивості об'єкта, які найменш імовірно випадково з'являться в інших об'єктах даних. Проблема полягає в тому, що цей підхід має працювати з двійковими даними (а не лише з текстом), і, отже, неможливо проаналізувати чи інтерпретувати дані.

Виберемо розмір блоку $B = 64$ байти, які використовуються для ідентифікації об'єктів у дискових блоках і мережевих пакетах. Однак немає концептуальних або реалізаційних відмінностей у використанні іншого розміру функції. Проте, існує фундаментальний компроміс: чим менші функції, тим вища деталізація, тим більші дайджести та більше обробки.

У всіх випадках процес вибору функції включає такі кроки:

1. Ініціалізація: показник ентропії H_{norm}, ранг пріоритету R_{prec} і рейтинг популярності R_{pop} – ініціалізуються до нуля.

2. Розрахунок H_{norm}: Ентропія Шеннона спочатку обчислюється для кожної функції (послідовність B-байтів):

$$H = -\sum_{i=0}^{255} P(X_i) \log P(X_i)$$

де $P(X_i)$ – емпірична ймовірність зустрічі з кодом i ASCII. Потім оцінка ентропії обчислюється як $H_{norm} = \lfloor 1000 \times H / log\ 2\ B \rfloor$.

R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				1														
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				2														
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				3														
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4														
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1													
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1						
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1		1				
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1		2				
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1		3				
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1		4				
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1		5				

Рисунок 2.1– Приклад розрахунку R_{pop}

3. Розрахунок R_{prec}: значення попереднього рангу R_{prec}отримується шляхом відображення норми показника ентропії H на основі емпіричних спостережень.

4. Розрахунок R_{pop}: для кожного ковзного вікна з W послідовних об’єктів ідентифікується крайня ліва об’єкта з найнижчим рангом пріоритету R_{prec}.

Показник популярності R_{pop} ідентифікованої функції збільшується на одиницю.

5. Вибір функції: обираються функції з рейтингом популярності $R_{pop} >= t$, де t є пороговим параметром.

Рисунок 2.1 ілюструє обчислення R_{pop} і етапи вибору функції. Використовується фрагмент 18 попередніх чисел R з фактичного обчислення; вікно $W = 8$ використовується для розрахунку R_{pop}. Якщо припустити, що поріг $t = 4$ і розмір функції $B = 64$, дві функції вибираються для представлення 82-байтової частини даних.

Основне спостереження полягає в тому, що переважна більшість ознак дорівнює нулю або одиниці; це дуже типовий результат. Для інтуїтивного пояснення розглянемо гістограму оцінки ентропії для набору стиснутих даних на рисунку 2.2. Загальна ентропія близька до максимуму, але ентропія окремих ознак фактично розподілена нормально.

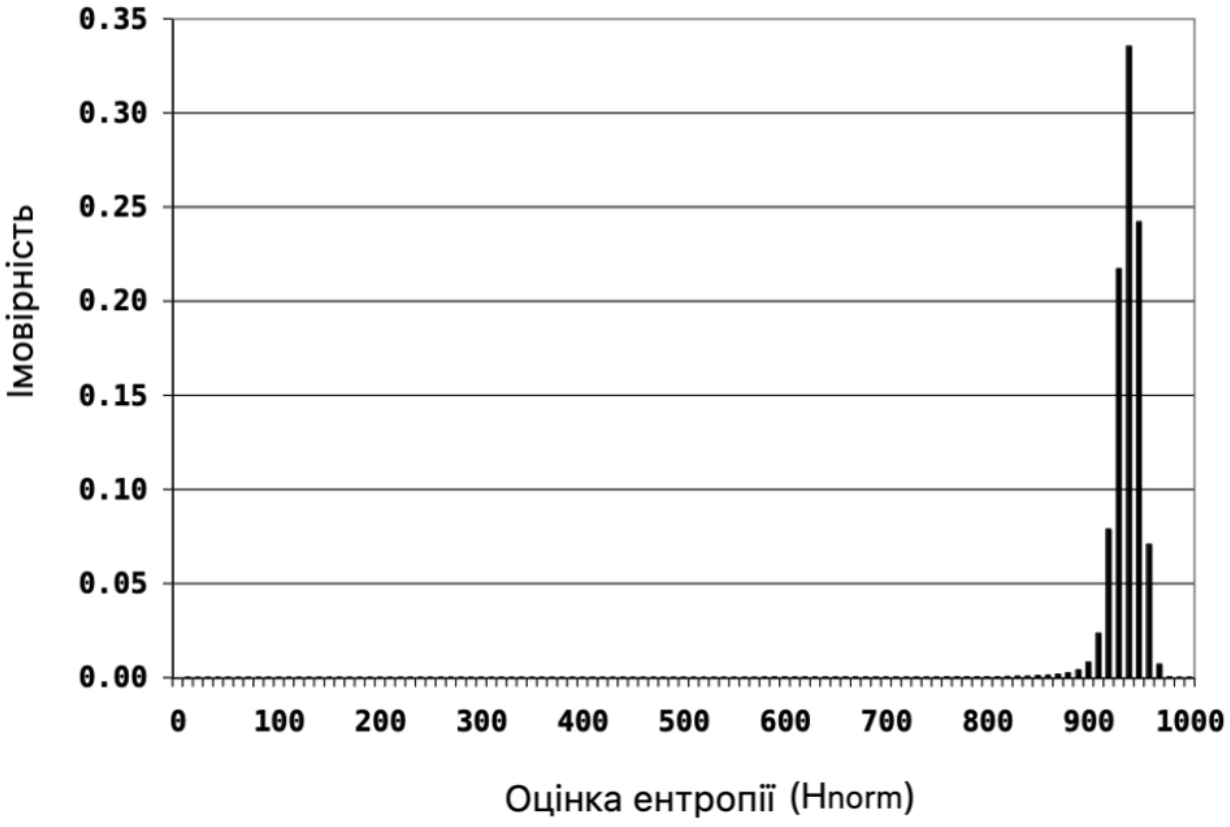

Рисунок 2.2 – Ентропія для набору стиснутих даних

Це означає, що, незважаючи на те, що показники ентропії сусідніх функцій сильно корельовані, час від часу (випадково) зустрічається показник

ентропії, який зустрічається рідше. З іншого боку, сама функція, ймовірно, буде зустрічатися рідше. Згідно зі статистикою, це проявляється як локальний мінімум у пріоритеті, що зрештою призводить до вищого показника популярності.

Загалом описана вище процедура вибору ознак працює з будь-якими типами даних, для яких доступне прийнятне (не обов'язково ідеальне) наближення розподілу ентропії ознак.

2.1.2 Порівняння гешів

Основним будівельним блоком для порівняння хещів є порівняння двох фільтрів Блума. Загалом, перекривання між двома сумісними фільтрами є мірою перекриття між наборами, які вони представляють – кількість спільних бітів зростає лінійно зі кількістю перекривання набору.

Фільтри Блума. Одним із багатообіцяючих підходів до прискорення операцій пошуку та зменшення потреб у просторі є фільтри Блума.

Вперше представлені Бертоном Блумом, вони широко використовуються в таких сферах, як мережева маршрутизація та фільтрація трафіку. Фільтр Блума – це бітовий вектор розміру m, усі біти якого спочатку встановлені на нуль. Основна ідея полягає в тому, щоб представити кожен елемент набору як унікальну комбінацію з k бітових розташувань. Для цього нам потрібен набір із k незалежних геш-функцій, $h_1, \dots, h_k$ які створюють значення в діапазоні від 0 до m – 1. Щоб вставити елемент (двійковий рядок) S1, ми застосовуємо кожну геш-функцію до це, що дає нам k значень. Для кожного значення: $h_1(S1), \dots, h_k(S1)$ –ми встановлюємо біт із відповідним числом в одиницю (встановлення біта двічі має той самий ефект, що й одноразове встановлення). На рисунку 2.3 показано приклад вставки двох послідовних елементів — $S1$ і $S2$ – за допомогою чотирьох геш-функцій: h_1, h_2, $h_3$3 і h_4.

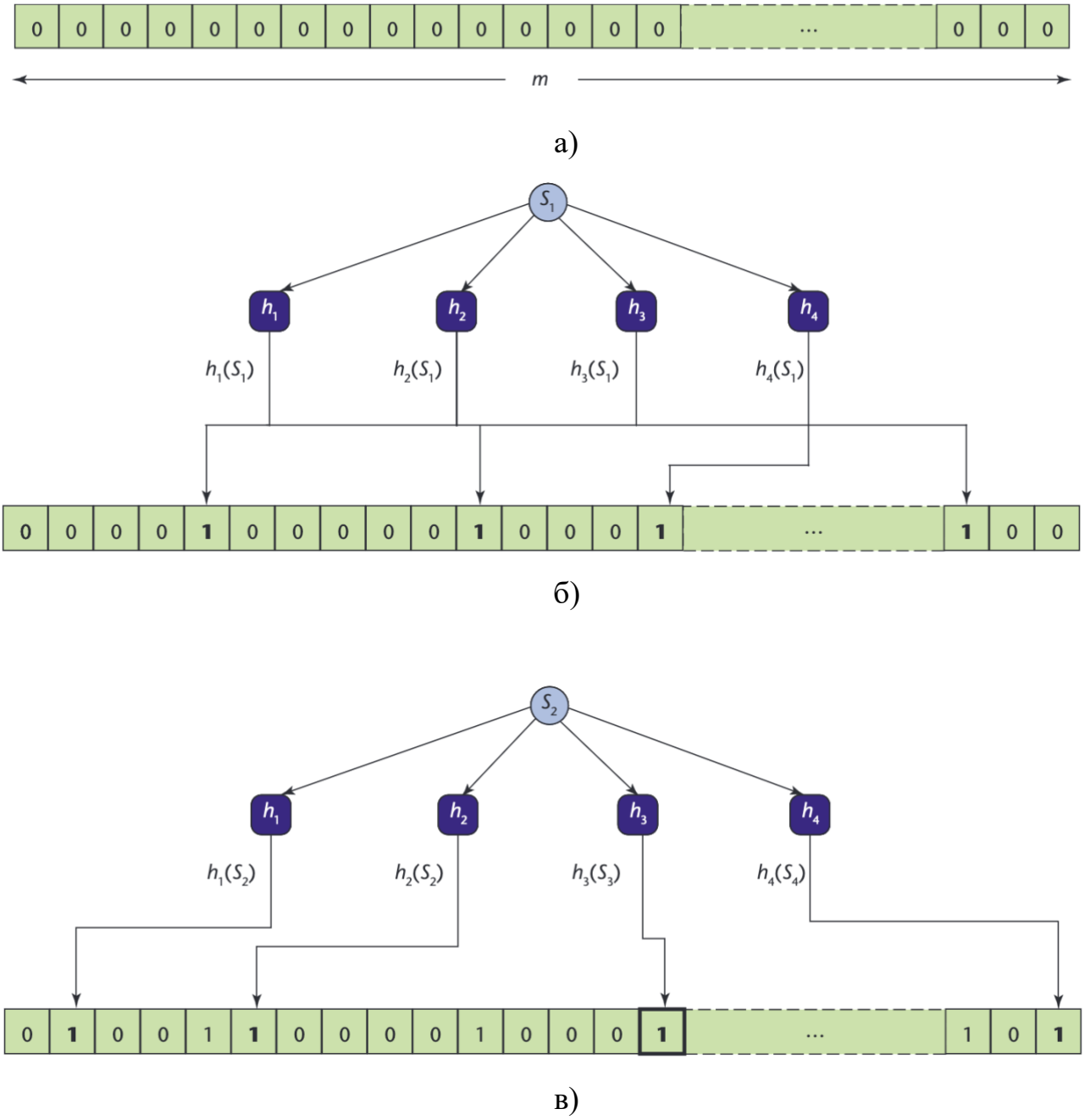

Рисунок 2.3 – Вставка двох елементів у фільтр Блума за допомогою чотирьох геш-функцій: (а) порожній фільтр Блума; (б) фільтр Блума після вставки одного елемента, S1; і (в) фільтр Блума після вставки другого елемента, S2. Кожна вставка встановлює чотири біти у фільтрі; деякі біти можуть бути обрані різними елементами – $h4\,(S1) = h3\,(S3)$ – що може призвести до помилкових спрацьовувань.

Щоб знайти елемент, гешуємо його за допомогою всіх геш-функцій і перевіряємо відповідні біти – якщо всі вони встановлені в один, повертаємо «так»; інакше «ні».

Фільтр ніколи не поверне помилково негативний результат; тобто якщо елемент було вставлено, відповідь завжди буде «так». Однак можемо мати помилковий позитивний результат – відповідь «так» для елемента, який ніколи не вставлявся, але чиї біти були випадково встановлені іншими вставками елементів.

Помилкові спрацьовування – це ціна, яку ми платимо за підвищення стиснення. Срібна підкладка полягає в тому, що оскільки можна кількісно оцінити частоту хибнопозитивних результатів аналітично, відповідно, можна їх контролювати. Загалом, після вставки n елементів ймовірність того, що фільтр поверне хибнопозитивний результат, є нелінійною функцією бітів на співвідношення елементів m/n і кількість геш-функцій k.

Хибнопозитивні показники. Використання криптографічних гешів у цифровій криміналістиці дозволяє легко вводити в процес фільтри Блума. Замість того, щоб обчислювати k окремих гешів, можна взяти криптографічний геш об’єкта, розділити його на кілька субгешів, які не перекриваються, і використовувати їх так, ніби їх створили різні геш-функції. Наприклад, ми могли б розділити 128-бітний геш MD5 на чотири 32-бітні геші, що дозволило б нам працювати з 1-Гбайтним фільтром і чотирма геш-функціями.

Якщо вставити 50 мільйонів гешів, очікувана кількість хибнопозитивних результатів буде менше 0,3 на мільйон, що майже у всіх випадках буде цілком прийнятним. Натомість ми отримаємо чотири звернення до пам’яті замість 26. У багатьох ситуаціях, наприклад під час початкового відбору спам листів, допустимі значно вищі хибнопозитивні показники, щоб зменшити обсяг даних, що розглядаються. Наприклад, ми могли б збільшити кількість гешів з 50 до 500 мільйонів і очікувати на хибнопозитивний рівень 0,2 відсотка.

Розглянемо два фільтри Блума $f1$ і $f2$ розміром m бітів, що містять $n1$ і $n2$ елементів ($n1 \leq n2$), відповідно, і n 12 спільних елементів. Нехай k – кількість використовуваних геш-функцій, а e_1, e_2 і e_{12} – кількість бітів, встановлених на одиницю в f_1, f_2 і $f_1 \cap f_2$ відповідно. Використовуючи класичний аналіз фільтра Блума [4], оцінка кількості очікуваних загальних бітів, встановлених на одиницю, є:

$$E_{12} = m(1 - p^{ks1} - p^{ks2} + p^{k(s1+s2-s12)}), p = 1 - 1/m.$$

Крім того, оцінки максимальної та мінімальної кількості можливих бітів, що перекриваються через випадковість, такі [15]:

$$E_{max} = min(n_1, n_2); E_{min} = m\,(1 - p^{ks1} - p^{ks2} + p^{k(s1+s2)})$$

Далі визначаємо порогову точку C, нижче якої всі збіги бітів вважаються випадковими:

$$C = \alpha(E_{max} - E_{min}) + E_{min}.$$

Таким чином, оцінка SF фільтра схожості двох фільтрів визначається як:

$$SF_{score}(f_1, f_2) = \begin{cases} -1 & \text{якщо } n_1 < N_{min} \\ 0 & \text{якщо } e_{12} < C \\ \left[100\ \dfrac{e_{12} - C}{E_{max} - C}\right] & \text{інакше} \end{cases}$$

де N_{min} – мінімальна кількість елементів, необхідних для обчислення значущої оцінки. У нашій реалізації використовується експериментально отримане значення $N_{min} = 6$.

Дано два дайджести SD1 і SD2, що складаються з фільтрів Блума $f_1^1, \dots, f_8^1$, $f_1^2, \dots, f_t^2$ відповідно ($s \leq t$), оцінка дайджесту подібності формально визначається як:

$$SD_{score}\,(SD_1,\, SD_2) = \frac{1}{s}\sum_{i=1}^{s} \max_{1 \leq j \leq t} SF_{score}(f_i^1, f_j^2)$$

Неформально перший фільтр із коротшого дайджесту (SD1) порівнюється з кожним фільтром у другому дайджесті (SD2) і вибирається максимальний бал подібності. Цю процедуру повторюють для решти фільтрів у SD1, а результати усереднюють для отримання єдиної складеної оцінки.

Обґрунтування цього розрахунку полягає в тому, що фільтр Блума представляє всі функції в безперервній частині вихідних даних.

Таким чином, порівнюючи два фільтри, порції вихідних даних порівнюються неявно.

Отже, розмір фільтрів стає критичним проектним рішенням – більші фільтри пришвидшують порівняння, тоді як менші фільтри забезпечують більшу точність.

2.2 Нечітке гешування з урахуванням місцевості

Щоб зрозуміти локальне (нечітке) гешування (LSH), використаємо криптографічні геш-функції, оскільки вони є гарним прикладом поясненням нечіткого гешування.

Для криптографічного гешу два дуже схожих входу дають два дуже різні геші. Ми можемо спростити це до твердження:

Криптографічна геш-функція оптимізована, щоб уникнути колізій гешів.

Геш-колізія описує випадок, коли два різні вхідні дані призводять до того самого гешу. Цілком очевидно, що це буде поганою функцією для криптографічного гешу, який використовується, наприклад, для перевірки цілісності вхідних даних.

Приклад результату криптографічного гешування.

The fox jumps over the **blue** dog	83AF788830E2248AB0448D2D0B31EF79
The fox jumps over the **red** dog	B60C4BBDD1AB7AF4FBD1AC19B4DE431C

У той час як криптографічний геш намагається уникнути зіткнень будь-якою ціною, геш, чутливий до місцевості, намагається майже навпаки. Хоча геш має бути різним для двох різних вхідних даних, подібності чи відмінності та їх розташування мають бути представлені в геші. Ми можемо спростити це до твердження: нечітке гешування максимізує ймовірність часткової колізії.

Поштучне перемішування. Покрокове або нечітке гешування описує техніку, за якої вхідні дані розділяються на частини, ми обчислюємо геш для кожної частини, а потім використовуємо окремі геші, щоб об’єднати їх у кінцевий результат по частинах.

10101111101010100101101010101001010001001010010100101010101010100100

A K 4 Y 7

10101111101010100101101010101011010001001010010100101010101010100100

A K T Y 7

Псевдо часткове гешування. У наведеному вище прикладі ми обчислюємо частковий геш AK4Y7 на основі вхідних даних. Потім ми вносимо зміни до вхідних даних, що призводить до варіації в одній із частин

вхідних даних, її геші та, зрештою, у результуючому вхідному геші: AKTY7. Тепер ми можемо припустити, просто дивлячись на два геші, що два вхідних файли відрізняються лише на одну частину. Це основна концепція поштучного гешування.

Гешування з урахуванням місцевості. У попередньому прикладі показано зменшення розмірності. Складність вхідних даних скоротили до гешів. Іншим прикладом того, як алгоритм гешування зменшує розмірність, є фільтр Блума, де ми зменшуємо складність за рахунок точності (рисунок 2.4).

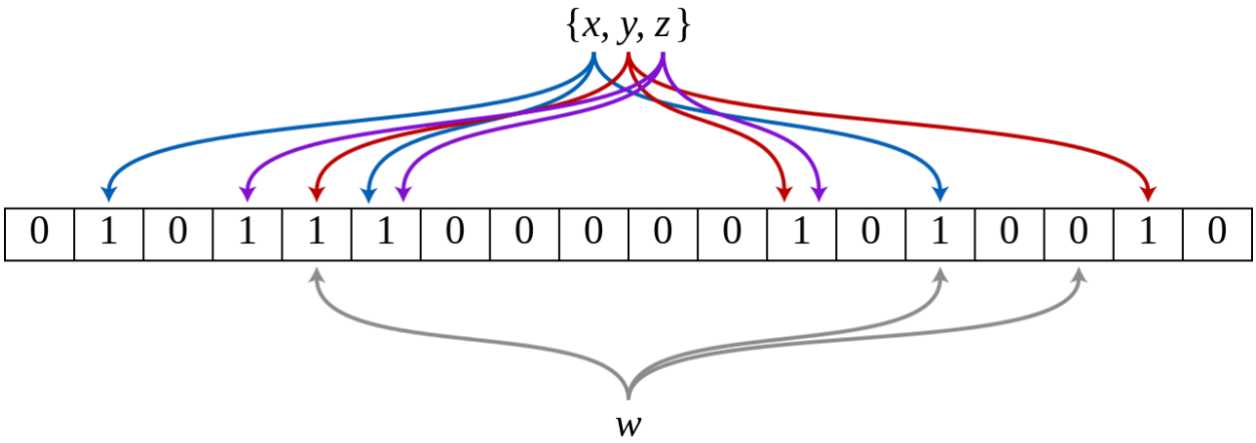

Рисунок 2.4 – Вхідні дані зведені до сегментів

В фільтрі Блума вхідні дані поміщаються в сегменти, що зменшує складність. Ще одна особливість, яка є результатом конкатенації гешів у порядку фрагментів, - це збереження локальності. Результат називається гешем, чутливим до місцевості (рисунок 2.5).

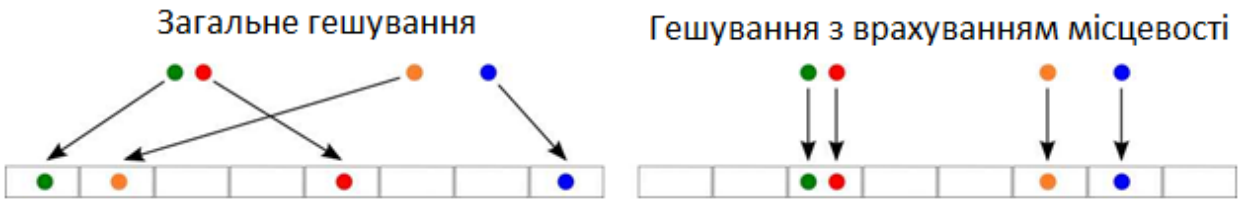

Рисунок 2.5 – Конкатенація гешів

Як можна зрозуміти з цього прикладу, розташування фрагмента у вхідному файлі відображається у розташуванні гешу фрагмента у вхідних файлах, чутливого до розташування нечіткого гешу.

Порівняння результатів нечіткого гешування. Маючи нечіткий геш, необхідно застосувати оптимізацію до колізій, щоб можна використовувати геші, для визначення подібноті вхідних файлів. Для цього, використовуючи локальність, поміщаємо два геші поруч один з одним і порівняти їх символ за символом. Чим більше збігів ми отримуємо, тим схожішими є два вхідних файли (рисунок 2.6).

Рисунок 2.6 – Порівняння результатів нечіткого гешування

Різні реалізації нечітких гешів мають дещо різні підходи до фактичного порівняння двох гешів, але в своїй основі вони покладаються на концепцію відстані редагування. Як відомо, можна легко перевести операції видалення, заміни та вставки під час порівняння двох гешів у зміни введення. Отже, чим більша відстань редагування між двома гешами, тим менш схожими є вхідні файли.

Покрокове гешування, викликане контекстом (CTPH). Важливою задачею є вибір частин в алгоритмі гешування. Скажімо, можна розділити

вхідні дані на фрагменти однакової довжини від початку введення. Якщо тепер додамо дані до вхідних даних, наше покрокове гешування розпадеться:

10101111101010100101101010101001010001001010010100101010101010100100

A K 4 Y 7

11010111110101010010110101010100101000100101001010010101010101010010

H 6 O G 3

Додавання даних до вхідних даних зсуває всі частини. Замість використання фрагментів фіксованої довжини та зсуву, можна використовувати тригер на основі контексту для меж фрагментів. У наступному прикладі вибираємо рядок Cat як тригер для наших частин і гешуємо їх, як робили раніше. Якщо тепер вставимо новий вміст у наш вхід, це вплине лише на геш для зміненого фрагмента:

What do **Cat**s like to read? **Cat**alogues!
H Y 7

What do smart **Cat**s like to read? **Cat**alogues!
B Y 7

Покрокове гешування ініційоване вмістом. Однією з найвідоміших реалізацій покрокового гешування, що запускається контекстом, є SSDeep.

Для генерації тригерів SSDeep використовує постійний геш-алгоритм, який створює псевдовипадкове значення лише на основі поточного контексту вхідних даних. Ковзне вікно передається як вхідні дані, і щоразу, коли згенерований ковзаючий геш збігається з нашим тригером, ми обчислюємо

традиційний геш для поточної частини. Виходячи з алгоритму spamsum, SSDeep використовує термін розмір блоку для значення тригера:

$$b_{init} = b_{min} 2^{\left\lfloor log_2((\frac{n}{Sb_{min}}) \right\rfloor},$$

де b_{min} – постійний мінімальний розмір блоку, n – довжина вхідних даних у байтах, а S – мінімальна довжина Spamsum. b_init називається початковим розміром блоку, оскільки він може отримати коригування під час обчислення гешу. наприклад якщо кінцева довжина гешу не відповідає мінімальним вимогам, розмір блоку може бути зменшено, а геш перераховано. Як можна собі уявити, це серйозна проблема продуктивності, особливо для невеликих файлів або файлів з низькою ентропією.

Кожного разу, коли досягається тригерне значення за допомогою змінного гешу, отримуємо кінець нової частини даних, для яких потім обчислюємо геш Фаулера–Нолла–Во (FNV). FNV не є криптографічним гешем, але його оптимізація для геш-таблиць і контрольних сум робить його гарним кандидатом при реалізації програми. Потім беремо значення шести молодших значущих бітів (LS6B) гешу FNV у кодуванні base64 і додаємо його до першої частини остаточного підпису.

Порівнюючи два геші SSDEEP, ми починаємо з розміру блоку. Ми можемо порівнювати лише геші з однаковим розміром блоку. Наступним кроком є усунення повторюваних послідовностей. Послідовності вказують на шаблони у вхідному файлі, які містять мало інформації і тому мають незначний вплив на вміст. Нарешті, обчислюємо зважену відстань редагування між двома отриманими гешами. Відстань редагування представляє мінімальну кількість мутацій, необхідних для переходу від одного входу до іншого. Вага встановлюється:

$$distance = inserts + deletes + 3changes + 5swaps.$$

Порівняння гешів SSDEEP. Нормалізований бал відповідності представляє консервативний зважений відсоток того, наскільки два вхідні дані є впорядкованими гомологічними послідовностями. Це означає, скільки бітів ідентичних і в тому самому порядку. Два несхожі файли можуть мати однаковий геш SSDEEP, якщо їх відмінності не впливають на точки запуску, а геш FNV створює колізію в LS6B. Можливість можна зменшити, замінивши FNV на криптографічний алгоритм гешування.

2.3 Порівняння методів гешування з урахуванням локальності

Гешування з урахуванням локальності (Locality Sensitive Hash, LSH) – це відносно нове сімейство алгоритмів зменшення розмірності, яке зосереджено на створенні стислого представлення заданих вхідних даних, які пізніше можна використовувати для порівняння. На практиці це означає, що використання методів LSH для майже ідентичних файлів виведе майже ідентичні геші. Порівнюючи це з методами криптографічного гешування, такими як SHA256, де гешування двох майже ідентичних файлів дасть два кардинально різні геші, які не можна порівняти один з одним.

2.3.1 Метод Nilsimsa

Nilsimsa. Одним із методів гешування, чутливих до місцевості, є Nilsimsa. Цей метод був продемонстрований як корисний у сфері виявлення спаму, який був представлений у статті [4].

Алгоритм працює шляхом обходу рядків байтів вхідних даних і об’єднання сусідніх символів у трійки. Ці трійки потім гешуються за допомогою алгоритму tran53, який виробляє ключ, що визначає місцевість, до якої належала трійка.

Алгоритм tran53 повертає значення в діапазоні від 0 до 255. Потім підраховуються всі знайдені місцевості, і створюється вектор, що містить кількість кожної конкретної знайденої місцевості. Шляхом відображення вектора в масив із додатними бітами, де значення більше медіани, можна отримати 32-байтовий код.

Остаточний створений геш – це простий 32-байтовий код, який у формі base64 складається з 64 символів.

2.3.2 TLSH

TLSH (Trend Micro Locality Sensitive Hash) – це алгоритм LSH від Oliver et al. від Trend Micro, вперше описаний у статті TLSH - A Locality Sensitive Hash [18].

Цей метод LSH був розроблений для виявлення шкідливих програм і кластеризації. Геш створюється шляхом аналізу байтового рядка за допомогою ковзного вікна розміром п’ять, а потім зіставлення відповідних значень у сегменти за допомогою гешу Пірсона, швидкої некриптографічної функції гешування. Далі тіло дайджесту, яке представляється як шістнадцятковий рядок, створюється з масиву сегментів шляхом поділу його на різні квартилі q1, q2 і q3 залежно від кількості сегментів. Перші три байти гешу - це заголовок дайджесту. Таким чином, остаточний геш представляється як шістнадцятковий рядок із 70 символів.

По суті, цей метод LSH подібний до методу гешування Nilsimsa, де значні відмінності полягають у тому, як кінцевий геш будується з вектора сегмента/функції. У TLSH об’єкти відображаються після поділу на квартилі залежно від випадків появи сегментів, а в Nilsimsa вони відображаються залежно від того, чи є вони більшими за медіану вектора ознак.

2.3.3 SSDeep

Ssdeep – це алгоритм LSH (або, як його називає автор, алгоритм часткового гешування, що запускається контекстом) Джессі Корнблюма,

вперше описаний у статті «Ідентифікація майже ідентичних файлів за допомогою контекстно активованого гешування по частинах» [17].

SSDeep є продовженням алгоритму spamsum Ендрю Триджелла, розробленого для захисту від спаму. Ssdeep став де-факто алгоритмом LSH, який використовується для виявлення зловмисного програмного забезпечення, і є єдиним гешем, чутливим до локалізації, який підтримується провідним у галузі сховищем аналізу зловмисного програмного забезпечення VirusTotal [12].

Геш SSDeep створюється за допомогою двох методів гешування, один із яких є частковим гешуванням, який є лише довільним алгоритмом гешування, але використовується для невеликих ділянок рядка байтів замість усього файлу. Так, наприклад, геш складається з перших 256 байтів, а потім ще один для наступних 256 і так далі.

Алгоритм гешування, який SSDeep використовує для гешування, – це геш-функція Фаулера-Нолла-Во від Фаулера та ін., її було обрано в першу чергу через те, що це некриптографічна геш-функція з акцентом на швидкості [5].

Іншим методом гешування є постійне гешування, яке є алгоритмом гешування, який створює псевдовипадкове значення на основі поточного контексту вхідних даних. Поворотний геш використовується для визначення того, коли відбувся відповідний діапазон байтів часткового гешування. Коли цей тригер спрацьовує, накопичений частковий геш додається до остаточного гешу. Кінцевий геш має такий вигляд:

block_size : hash1 : hash2

Розмір блоку – це обчислене значення, яке визначає розмір кожного блоку коду перед додаванням накопиченого поштучного гешу до остаточного гешу. Як видно, є два різних геші. Це пов’язано з наявністю двох тригерів для часткового гешування. Перший тригер виникає, коли змінний геш створює значення, яке дорівнює за модулем розміру блоку розміру блоку мінус одиниця. Другий тригер трапляється, коли змінний геш створює

значення, яке дорівнює подвоєному розміру блоку або подвійному розміру блоку мінус одиниця. Однак, оскільки тригери базуються на розмірі блоку файлу, геші не мають фіксованої довжини, замість цього реалізується максимальна довжина всього гешу, яка становить 148 символів у кодуванні base64.

2.3.4 SDHASH

Алгоритм SDHASH, спочатку запропонований Roussev у статті Data fingerprinting with identity digests, є більш новим методом LSH [20].

SDHASH використовує дещо іншу систему для представлення введених даних порівняно з іншими методами, що знаходяться в фокусі. Він працює за допомогою ковзного вікна, яке перетинає вхідні дані, а програма обчислює нормалізовану ентропію Шеннона кожного вікна, щоб визначити, чи має вміст вікна статистичні дані.

Значення. Якщо на цьому етапі воно вважається статистично значущим, воно присвоює вікну номер функції. Після пошуку всіх функцій у файлі програма відфільтровує всі функції, які вважаються слабкими. Після того, як усі об’єкти відфільтровано, об’єкти додаються до фільтра розквіту, щоб уникнути дублювання об’єктів у кінцевих результатах. Потім фільтр Блума використовується для побудови шістнадцяткового геш-рядка.

На практиці найбільш суттєвою відмінністю між SDHASH та іншими алгоритмами LSH є те, що SDHASH не має обмежень на розмір своїх гешів і що вони суттєво відрізняються за розміром залежно від вхідних даних, оскільки великі файли можуть виводити геш, який складається з понад 10 тисяч символів, що означає, що їх важче використовувати в моделях нейронних мереж, які вимагають фіксованих не надто великих розмірів вхідних даних.

3 ДОСЛІДЖЕННЯ МЕТОДІВ ГЕШУВАННЯ ПОДІБНОСТІ

3.1 Обчислення геш значення методом TLSH

Під час аналізу зловмисного програмного забезпечення та схожості тексту часто доводиться ідентифікувати подібні області даних у файлах. Одним з методів є TLSH (A Locality Sensitive Hash), який був визначений Олівером та ін. [25]. Він використовується компанією VirusTotal разом із SSDeep- для ідентифікації геш-значення зловмисного програмного забезпечення. TLSH – це метод нечіткого гешування, який вимагає принаймні 50 байт даних. Сам геш має довжину 35 байтів із «T1» (номером версії) на початку та за ним 70 шістнадцяткових символів, наприклад:

tlsh.hash: T106C08057F5705755C14DD383310775C5770DD1DB3101CD144449C5417714A6516FB259

Розглянемо приклади обчислення геш значення на основі TLSH:

Приклад 1. Рядок 1 рядок 2 ідентичні.

String 1: Тема: Алгоритми виявлення спаму на основі гешу подібності.

String 2: Тема: Алгоритми виявлення спаму на основі гешу подібності.

tlsh.hash:
T1B1B0920A264B289A02105509D2243A0D392418BEBBA00A0606BC7D7B25D6390A8A242A

tlsh.hash:
T1B1B0920A264B289A02105509D2243A0D392418BEBBA00A0606BC7D7B25D6390A8A242A

tlsh.diff(hex1, hex2) 0;

Отже, різниця між гешами дорівнює 0.

Приклад 2. Тепер, додавши в кінець першого рядка крапку, побачимо, що геш змінився.

String 1: Тема: Алгоритми виявлення спаму на основі гешу подібності.

String 2: Тема: Алгоритми виявлення спаму на основі гешу подібності

tlsh.hash:
T1B1B0920A264B289A02105509D2243A0D392418BEBBA00A0606BC7D7B2
5D6390A8A242A
tlsh.hash:
T101B0920A264B289A02105509D2243A0D392418BEBBA00A0605BC7D7B2
5D6390A8A242A
tlsh.diff(hex1, hex2) 2

В результаті, бачимо, що ідеальна оцінка (0) тепер змінена на 2. Зміна в геші відбулася в 3 і 52 символі.

Приклад 3. Тепер, додавши знак оклику в кінці, ми побачимо, що геш змінився:

String 1: Тема: Алгоритми виявлення спаму на основі гешу подібності.
String 2: Тема: Алгоритми виявлення спаму на основі гешу подібності!
tlsh.hash:
T1B1B0920A264B289A02105509D2243A0D392418BEBBA00A0606BC7D7B2
5D6390A8A242A
tlsh.hash:
T1ABB0920A264B289A02105509D2243A0E392418BEBBA00A0605BC7D7B2
5D6390A8A242A
tlsh.diff(hex1, hex2) 3

В результаті, бачимо, що ідеальна оцінка (0) тепер змінена на 3. Зміна в геші в основному локалізована на початку гешу.

Приклад 4. В цьому прикладі змінити одне зі слів.

String 1: Тема: Алгоритми виявлення спаму на основі гешу подібності.
String 2: Тема: Алгоритми виявлення спаму на базі гешу подібності.
tlsh.hash:
T1B1B0920A264B289A02105509D2243A0D392418BEBBA00A0606BC7D7B2
5D6390A8A242A

tlsh.hash:
T1F0B0920A174B249E42204505D1243A097924087EBBA00A010A783D6B25D73A098A1468
tlsh.diff(hex1, hex2) 28

В цьому прикладі змінили одне зі слів.

Приклад 5. В цьому прикладі змінили два сліва.

String 1: Тема: Алгоритми виявлення спаму на основі гешу подібності.
String 2: Тема роботи: Алгоритми виявлення спаму на базі гешу подібності.
tlsh.hash:
T1B1B0920A264B289A02105509D2243A0D392418BEBBA00A0606BC7D7B25D6390A8A242A
tlsh.hash:
T1DEB0922E575B248E4220A509E1283B0D7525087FBFA40B010BB83D6B25E32A098A14AC
tlsh.diff(hex1, hex2) 47

Відповідно, різниця суттєво збільшилася і становить 47.

Приклад 6: Тепер використаємо два абсолютно різні рядки:

String 1: Тема: Алгоритми виявлення спаму на основі гешу подібності.
String 2: Topic: Spam Detection Algorithms Based on Similarity Hash.
tlsh.hash:
T1B1B0920A264B289A02105509D2243A0D392418BEBBA00A0606BC7D7B25D6390A8A242A
tlsh.hash:
T11FA002217955753444CEB65B1541A9407056521509B0510269294D84E53FAA6AD07B3E
tlsh.diff(hex1, hex2) 258

В результаті проведених експериментів, бачимо, що оцінка залежить від змін між одним набором даних і де ми можемо змінювати порогове значення залежно від кількості внесених змін. Для фішингу ми часто бачимо електронні листи, які відрізняються лише іменем особи, а отже:

String 1: Шановний Тарас,

Ви виграли 1 000 доларів у лотереї.

Будь ласка, зв'яжіться зі мною, щоб отримати виграш.

String 2: Шановний Ігоре,

Ви виграли 1 000 доларів у лотереї.

Будь ласка, зв'яжіться зі мною, щоб отримати виграш.

tlsh.hash: T14BA0120C50025D610062D348DB4764A85305C100C10485 42042088DC305C1FF084108D

tlsh.hash: T13AA0120C50021D610062D704DF0754ADE305C110C10485 4204204099245C1AF18410CD

tlsh.diff(hex1, hex2) 26

У даному випадку існує досить хороший збіг між двома вхідними даними (листами спаму).

3.2 Оцінка подібності методом Нільсімса

Nilsimsa – це геш подібності, який використовується як метод гешування, орієнтований на локалізацію, для боротьби зі спамом. Він дає оцінку від 0 (несхожі об’єкти) до 128 (ідентичні або дуже схожі об’єкти).

Nilsima був створений Damiani et al. для виявлення спаму [25, 26]. Він використовує 5-байтове ковзне вікно фіксованого розміру, яке аналізує вхідні дані побайтно та створює триграми можливих комбінацій введених символів. Триграми відображаються в 256-бітний масив (відомий як акумулятор), щоб створити геш, і кожного разу, коли здійснюється доступ до певної позиції, її значення збільшується. Наприкінці обробки, якщо значення перевищують певний поріг, значення встановлюється на 1, інакше воно буде нульовим. Це створює 32-байтовий дайджест.

Щоб порівняти два геші, метод перевіряє кількість ідентичних бітів червоного кольору на однакову позицію. Це дає оцінку від 0 (несхожі об’єкти) до 128 (ідентичні або дуже схожі об’єкти).

Метод Нільсімса.

String 1: Dear Taras,

You won $1,000 in the lottery.

Please contact me to claim your prize.

String 2:Dear Igor,

You won $1,000 in the lottery.

Please contact me to claim your prize.

Рядок 1. Геш - значення:

50ac9d4a0375ad80af02288b95560b6165095d356c20845faf61aac1e8780829

Рядок 2. Геш - значення:

40ad9d4a0377ad88af83288a951603c1610d5d356f208557af63aac1f8280a29

Score: 106

Акумулятор (Рядок 1)

[3, 2, 1, 4, 1, 3, 2, 2, 2, 2, 1, 4, 2, 2, 1, 1, 2, 2, 1, 4, 3, 5, 3, 2, 2, 2, 2, 3, 2, 3, 4, 3, 3, 1, 2, 0, 2, 1, 6, 5, 2, 3, 1, 4, 2, 4, 0, 3, 4, 2, 1, 1, 1, 3, 4, 2, 3, 3, 3, 4, 1, 7, 1, 3, 3, 3, 6, 3, 4, 2, 4, 2, 2, 1, 5, 2, 0, 1, 1, 4, 1, 2, 0, 1, 0, 5, 1, 1, 2, 2, 3, 3, 2, 3, 6, 2, 5, 1, 4, 2, 4, 5, 0, 0, 4, 2, 3, 3, 4, 1, 4, 2, 3, 2, 2, 3, 1, 1, 2, 0, 4, 1, 3, 2, 0, 3, 4, 2, 3, 0, 0, 2, 1, 3, 4, 2, 3, 4, 2, 3, 2, 2, 1, 1, 0, 5, 4, 2, 3, 1, 3, 2, 7, 2, 5, 2, 3, 1, 2, 6, 3, 4, 1, 6, 1, 1, 2, 5, 1, 0, 2, 4, 1, 3, 2, 1, 2, 4, 1, 1, 1, 2, 1, 2, 6, 6, 4, 3, 2, 5, 0, 3, 1, 2, 2, 2, 1, 1, 0, 5, 3, 2, 4, 3, 1, 3, 1, 3, 4, 2, 3, 1, 3, 4, 6, 2, 3, 3, 1, 1, 2, 1, 1, 1, 1, 3, 1, 6, 2, 2, 5, 2, 3, 2, 5, 3, 5, 1, 1, 6, 2, 2, 7, 4, 2, 3, 1, 6, 2, 1, 2, 2, 3, 2, 4, 0]

Акумулятор (Рядок 2)

[3, 2, 1, 4, 1, 3, 1, 2, 2, 3, 1, 4, 2, 1, 1, 1, 2, 2, 1, 3, 2, 5, 2, 2, 1, 2, 2, 4, 3, 3, 4, 3, 3, 1, 2, 0, 2, 1, 5, 6, 1, 3, 1, 3, 2, 4, 1, 3, 4, 3, 1, 2, 0, 3, 5, 2, 3, 3, 3, 4, 1, 7, 1, 4, 3, 3,

6, 2, 4, 1, 4, 1, 3, 1, 4, 2, 0, 0, 1, 5, 1, 1, 1, 0, 0, 4, 1, 1, 4, 4, 3, 3, 2, 3, 5, 2, 4, 2, 3, 2, 4, 5, 0, 0, 4, 2, 3, 3, 3, 1, 5, 1, 3, 2, 3, 3, 1, 1, 2, 0, 3, 1, 2, 2, 0, 3, 4, 2, 3, 1, 0, 1, 1, 1, 4, 3, 3, 3, 2, 2, 2, 2, 1, 2, 1, 5, 4, 2, 4, 1, 2, 1, 8, 2, 6, 2, 3, 1, 1, 5, 2, 5, 2, 6, 2, 1, 2, 5, 1, 0, 1, 3, 2, 3, 0, 1, 4, 3, 0, 1, 1, 2, 2, 3, 6, 7, 5, 3, 2, 4, 0, 3, 1, 2, 2, 3, 1, 1, 0, 6, 3, 2, 5, 4, 1, 3, 1, 3, 4, 3, 3, 1, 3, 4, 5, 1, 3, 3, 1, 2, 1, 1, 1, 1, 1, 4, 1, 6, 2, 2, 4, 1, 3, 2, 5, 3, 5, 2, 1, 6, 3, 1, 5, 3, 2, 3, 1, 5, 2, 1, 2, 2, 2, 2, 4, 0]

Приклад оцінювання подібності різних повідомлень методом Нільсімса приведений в таблиці 3.1

Таблиця 3.1 – Оцінка подібності повідомлень

Повідомлення 1	Повідомлення 2	Оцінка
You won $1,000 in the lottery.	You won $1,000 in the lottery.	128
You won $1,000 in the lottery.	You won $2,000 in the lottery.	109
Dear X, You won $1,000 in the lottery.	Dear Y, You won $2,000 in the lottery.	102
Dear Andrew	Dear andrew	90
Spam Detection Algorithms Based on Similarity Hash.	spam detection algorithms based on similarity hash.	67
Spam Detection Algorithms Based on Similarity Hash.	spam evolves as rapidly as anti-spam techniques improve.	4

Як видно з таблиці 3.1 найвище значення оцінки (128), коли повідомлення однакові, відповідно, при різних повідомленнях оцінка буде 4.

У цьому випадку існує досить хороший збіг між двома вхідними даними. На рисунку 3.1 показано продуктивність TLSH порівняно з sdhash і

ssdeep. Як видно з рисунку 3, рівень хибних позитивних результатів TLSH є кращий порівняно з іншими сетодами.

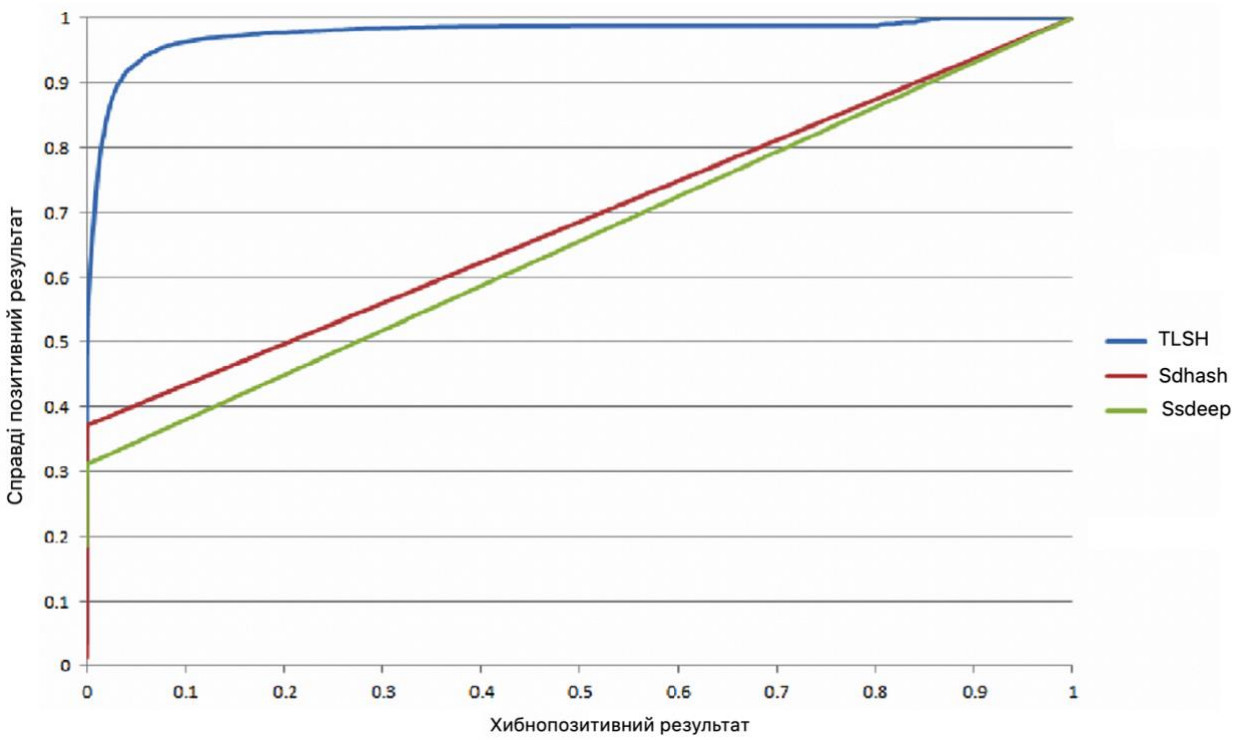

Рисунок 3.1 – Порівняння гешів подібності

На відміну від криптографічних гешів гешування подібності мають на меті показати, де все не змінилося. Під час фішингу можна часто бачити шаблонні повідомлення, у яких змінюються лише невеликі частини повідомлення. Те ж саме може трапитися зі зловмисним програмним забезпеченням, яке є лише невеликою частиною змін у зловмисному програмному забезпеченні – і воно уникає методів виявлення сигнатур, які використовують геш-значення для шкідливого коду.

3.3 Ідентифікація спаму з використанням SSDeep

SSDeep – це інструмент із відкритим кодом, який використовується для створення та порівняння нечітких гешів. Базуючись на алгоритмі частково

керованого гешування (CTPH), він генерує нечітке геш-порівняння значень, розбиваючи дані на сегменти змінних розмірів, що гарантує, що невеликі зміни суттєво не змінюють його геш-значення; що полегшує визначення схожості. Нечітке гешування з SSDeep активно використовується в цифровій криміналістиці та аналізі шкідливих програм.

SSDeep Hash – це нечіткий алгоритм гешування, розроблений для швидкої ідентифікації та порівняння файлів. На відміну від традиційних криптографічних геш-функцій, таких як MD5 або SHA-1, SSDeep враховує навіть незначні зміни у файлах, що робить його безцінним активом для аналізу зловмисного програмного забезпечення, цифрової експертизи та інших програм, орієнтованих на безпеку.

Нечітке гешування швидко стало одним із популярних методів цифрової криміналістики та аналізу шкідливих програм, особливо серед аналітиків. Одним із відомих інструментів, який використовується для аналізу нечіткого гешування, є SSDeep, який дозволяє швидко й ефективно порівнювати файли на схожість.

SSDeep працює за наступним алгоритмом: спочатку порівнює вміст двох файлів і створює унікальні геші для обох, а потім порівнює ці геші, щоб оцінити будь-яку схожість між ними та знайти будь-які випадки незначних змін, які відбулися між ними. Завдяки цьому процесу SSDeep визначає файли, які мають схожі характеристики, хоча вони незалежно обробляються кількома програмами чи службами.

Наприклад, розглянемо два файли, A і B. Традиційні алгоритми гешування вироблятимуть для них дуже різні геші, якщо буде змінено хоча б один біт. Геші на основі подібності SSDeep дозволяють швидко визначити, що ці два файли пов’язані.

Варіанти ефективного застосування SSDeep.

1. Аналіз шкідливих програм. SSDeep використовується фахівцями з безпеки для швидкого виявлення варіантів шкідливих програм і виявлення зв’язків між, здавалося б, різними зразками.

Аналітики шкідливого програмного забезпечення та дослідники безпеки покладаються на SSDeep як на безцінний інструмент для відстеження розвитку шкідливого програмного забезпечення. Швидко порівнюючи нові зразки зловмисного програмного забезпечення з відомими загрозами, SSDeep дозволяє дослідникам і аналітикам швидко виявляти подібності, а також розкривати варіанти існуючих сімейств сімейств зловмисного програмного забезпечення, які, можливо, ще не були виявлені.

Порівнюючи нечіткі геш-значення, аналітики можуть зрозуміти та визначити зв’язки між різними зразками шкідливого програмного забезпечення та простежити його розвиток з часом. Це розуміння не тільки допомагає приписувати атаки конкретним суб’єктам загрози, але й допомагає створювати більш комплексні та ефективні стратегії захисту від нових кіберзагроз.

Здатність SSDeep визначати схожість серед зразків зловмисного програмного забезпечення прискорює зворотне проектування та інші інструменти для розшифрування шкідливого коду. Швидко знаходячи спільні компоненти або функції, аналітики можуть ефективніше зосередити свої зусилля на розумінні конкретних аспектів кожного зразка; в кінцевому підсумку прискорення розробки контрзаходів і методів виявлення.

Використання SSDeep в аналізі зловмисного програмного забезпечення виявилося неоціненним для виявлення загроз і реагування на них, ставши важливим елементом сучасних заходів із кібербезпеки.

Використання SSDeep в аналізі зловмисного програмного забезпечення, безсумнівно, підвищило ефективність і точність виявлення загроз і реагування на них, що зробило його незамінним компонентом сучасних зусиль у сфері кібербезпеки.

2. Захист інтелектуальної власності на програмне забезпечення. Нечітке гешування може допомогти виявити плагіат, крадіжку коду або необґрунтоване використання захищеного авторським правом матеріалу.

Захист інтелектуальної власності – ще одна сфера, де SSDeep досягає успіху. Захист власного коду та алгоритмів розробки програмного забезпечення має надзвичайно важливе значення, у той час як плагіат, крадіжка коду або несанкціоноване копіювання іноді можуть статися. SSDeep може допомогти ідентифікувати екземпляри, порівнюючи нечіткі геші оригінальних виконуваних файлів із файлами, які ймовірно порушують авторські права.

Компанії та розробники можуть використовувати цю можливість для виявлення будь-якого несанкціонованого повторного використання свого коду, навіть якщо порушник вносить велику кількість невеликих змін у свій шкідливий файл, намагаючись замаскувати свою крадіжку.

Крім того, SSDeep може допомогти організаціям захистити комерційні таємниці та конфіденційні дані, приховані в документах, зображеннях або цифрових активах. Порівнюючи нечіткі геш-значення захищеного вмісту з геш-функцією загальнодоступних або витікаючих файлів, вони можуть ефективніше контролювати та захищати права інтелектуальної власності.

У все більш цифровому світі, де крадіжки та зловживання інтелектуальною власністю стали надто частими, SSDeep надає ефективні та економічні засоби для захисту активів.

3. Запобігання втраті даних. SSDeep може допомогти у виявленні та запобіганні витоку даних, порівнюючи файли з великою базою даних, що містить конфіденційну інформацію.

Запобігання втраті даних (DLP) є ще одним важливим застосуванням SSDEep в інформаційній безпеці. Організації постійно стикаються з ризиком втрати конфіденційних даних через випадковий або навмисний витік, внутрішні загрози або цілеспрямовані кібератаки.

SSDeep може виявляти спам і запобігати витокам даних, порівнюючи файли з базою даних відомої конфіденційної інформації. Наприклад, компанії можуть генерувати нечіткі геш-значення, щоб показувати спам-повідомлення або позначати конфіденційними файли, що містять

конфіденційну інформацію, як-от фінансові звіти, відомості про клієнтів або власні дослідження.

SSDeep можна використовувати для сканування всіх повідомлень електронної пошти, вкладень, передачі файлів і сховищ хмарних сховищ на предмет криптографічних гешів, які збігаються з конфіденційними файлами. У разі виявлення таких збігів SSDeep сповіщає персонал служби безпеки або блокує передачу цих даних, тим самим запобігаючи потенційному витоку даних до того, як він відбудеться.

Нечітке гешування може надати організаціям більш ефективні засоби захисту конфіденційних даних, залишаючись сумісними з правилами захисту даних, зрештою зменшуючи ризик дорогого витоку даних і шкоди репутації.

Отже, SSDeep Hash це ефективний і гнучкий інструмент для ідентифікації файлів, який забезпечує безпечне порівняння та аналіз, навіть якщо файли зазнали незначних змін.

До собливостей SSDeep необхідно віднести [27].

1. Сумісність із різними платформами: це програмне забезпечення можна запускати на різних платформах, включаючи Windows, macOS і Linux, щоб задовольнити потреби користувачів у низці операційних систем.

2. Інтерфейс командного рядка: SSDeep надає простий інтерфейс командного рядка для полегшення використання та інтеграції в існуючі робочі процеси.

3. Бібліотека Python для можливостей нечіткого гешування: за допомогою бібліотеки SSDeep Python користувачі можуть інтегрувати можливості нечіткого гешування безпосередньо у свої проекти Python.

4. Підтримка форматів файлів: ssdeep пропонує підтримку багатьох різних форматів файлів, включаючи текстові файли, двійкові файли та виконувані файли.

Алгоритм роботи з SSDeep, містить наступні кроки [27].

1. Завантаження та встановлення SSDeep. Завантажити останню версію програми можна з репозиторію SSDeep GitHub.

2. Обчислення SSDeep Fuzzy Hashes для файлів. Після встановлення SSDeep відкриваємо інтерфейс командного рядка та переходимо до каталогу, де знаходяться файли для порівняння. Для створення гешів SSDeep всіх файлів у каталозі виконаємо команду:

```
ssdeep -r *
```

Ця команда генеруватиме геші SSDeep для всіх файлів, розташованих у каталозі та його вкладених папках.

3. Порівняння гешів SSDeep. Щоб порівняти геші SSDeep двох файлів, виконаємо команду:

```
ssdeep -s test1.ssdeep test2.ssdeep
```

В результаті SSDeep згенерує результат із оцінкою подібності від 0 до 100, де 100 позначає ідентичні файли, тоді як нижчі оцінки означають зростаючий ступінь відмінності.

Порівняння кількох файлів.

Якщо потрібно швидко порівняти багато файлів, створюємо геш-список і використовуємо опцію -m:

```
ssdeep -r directory_path > hashes.txt
ssdeep -m hashes.txt target_file.txt
```

Розглянемо приклад. Ось два повідомлення з різницею лише в 1 байт:

Повідомлення 1. Spam Detection Algorithms Based on Hash Similarity

Повідомлення 2. Spam Detection Algorithms Based on Hash Similarit**Y**

Обчислимо геш функції SHA256 використовуючи бібліотеку Openssl:

```
% echo -n "Spam Detection Algorithms Based on Hash Similarity" |
openssl dgst -sha256
```

7d6ac100b53635babc40a19888ba3fdaecfd516f4b5b498f6810339c9e8f6e7d

```
% echo -n "Spam Detection Algorithms Based on Hash SimilaritY" |
openssl dgst -sha256
511a21ad202878764322b07e72050d60b3ba6159a954e26c9078fac4
ab053bfc
```

Як бачимо, для кожного повідомлення створюється зовсім інший геш SHA-256.

Тепер обчислимо SSDeep геші цих повідомлень:

```
% ssdeep -r *
ssdeep,1.1--blocksize:hash:hash,filename
3:XIbAmG3uWGW/X/y:XI0+E//y,"/file1.txt"
3:XIbAmG3uWGW/X/S:XI0+E//S,"/file2.txt"
```

Отже, геш SSDeep майже ідентичний, за винятком різниці в одному символі.

CTPH – це постійний геш, який включає кілька традиційних криптографічних гешів для одного або кількох сегментів фіксованого розміру у файлі. Одним із найпопулярніших методів CTPH є SSDeep. Він був створений у 2006 році Корнблюмом та ін. і використовує нечіткі геші [27].

Він зіставляє подібні послідовності байтів, незалежно від того, чи існують відмінності між цими послідовностями. Для цього розбиваємо файл на фрагменти за допомогою постійної геш-функції. Потім ми створюємо невеликий геш для кожного з цих фрагментів. Потім вони агрегуються для отримання повного гешу файлу. Під час порівняння підписів файлів ми розглядаємо результуюче геш-значення як рядок, а потім використовуємо метод редагування відстані для порівняння.

При цьому SSDeep зберігає стан на основі лише кількох останніх байтів як вхідних даних. Байти додаються до стану, а потім видаляються, коли додаються інші байти – це схоже на наявність вікна, яке переміщується над вхідними даними. Області, які мають однакові послідовності байтів у частинах, створюватимуть ту саму послідовність вихідних гешів для цього сегмента.

Під час порівняння підписів файлів ми розглядаємо результуюче геш-значення як рядок, а потім використовуємо метод редагування відстані для порівняння. Розглянемо приклади використання SSDeep.

Приклад 1. Повідомлення однакові.

Рядок 1. Spam Detection Algorithms Based on Hash Similarity: 3:XIbAmG3uWGW/X/ml:XI0+E//

Рядок 2. Spam Detection Algorithms Based on Hash Similarity: 3:XIbAmG3uWGW/X/ml:XI0+E//

Як видно з результатів обчислення, однакові повідомлення дають однакові геші.

Приклад 2. Різні повідомлення.

Spam Detection Algorithms Based on Hash Similarity:

Рядок 1. 3:XIbAmG3uWGW/X/ml:XI0+E//

Рядок 2. Also called fuzzy hashes, it allows identifying :

3:AXGBicFliMJWFAu1ll:AXGH+MYmuv

Приклад 3. Повідомлення відрізняються одним словом.

Spam Detection Algorithms Based on Hash Similarity:

3:XIbAmG3uWGW/X/ml:XI0+E//

Spam Detection Algorithms Based on Hash not Similarity:

3:XIbAmG3uWGW/X/+**il**:XI0+E//

Отже, результат гешування починається і закінчується однаковими символами. Дані геші відрізняються трьома символами в середині гешу.

Оскільки вставляються або видаляються байти, подібні послідовності завжди збігатимуться. Це ідеально підходить для виявлення зловмисного програмного забезпечення, оскільки все ще можна виявити збіг, навіть якщо байти не вирівняні.

Однією з систем, яка використовує SSDeep геш для ідентифікації зловмисного програмного забезпечення, є VirusTotal (рисунок 3.2).

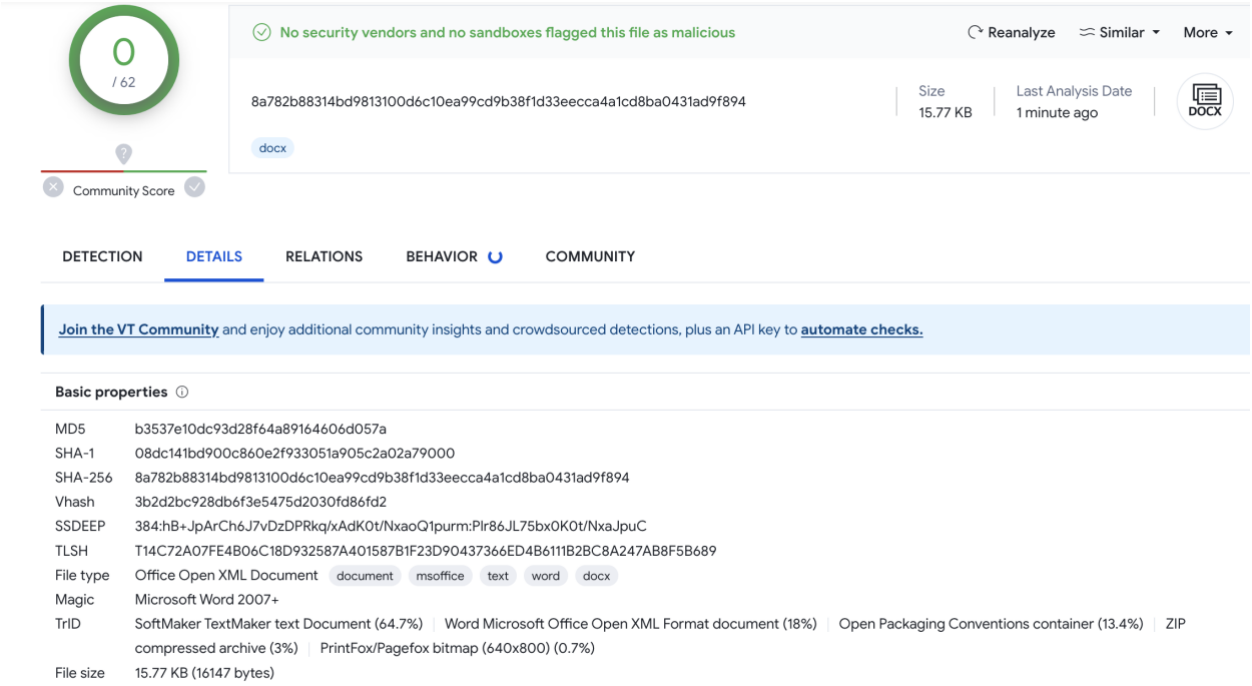

Рисунок 3.2 – Використання SSDeep системою VirusTotal

Як видно з рисунку 3.2 VirusTotal також використовує геші подібності SSDeep та TLSH, що свідчить про їх ефективність.

ВИСНОВКИ

В кваліфікаційній роботі розв’язано актуальну задачу дослідження та оцінки ефективності алгоритмів гешування з урахуванням локальності для виявлення спаму. При цьому отримано наступні результати.

1. Проведено аналіз підходів до виявлення спаму, серед яких виділено наступні: криптографічне гешування, Нечітке гешування, центр розподіленої контрольної суми, внесення в сірий список, чорний та білий списки DNS.

2. Розкрито принципи функціонування методів гешування з урахуванням локальності. Показано використання фільтру Блума для обчислення гешу подібності.

3. Проведено порівняння продуктивності методів гешування TLSH, SDHASH та SSDEEP, показано, що метод TLSH демонструє кращий рівень хибних позитивних результатів.

4. Проведено оцінку подібності методом Нільсімса в якому найвище значення оцінки (128) при однакових повідомленнях, відповідно, при різних повідомленнях оцінка буде близька до нуля.

5. Розроблено алгоритм ідентифікація спаму з використанням SSDeep. Наведено приклади використання SSDeep для ідентифікації спаму.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Email Spam Statistics to Know in 2023. [Електронний ресурс]. - Режим доступу: https://www.mailmodo.com/guides/email-spam-statistics/

2. What Is Spam? [Електронний ресурс]. - Режим доступу: https://www.proofpoint.com/au/threat-reference/spam

3. What is computer spam? [Електронний ресурс]. - Режим доступу: https://www.malwarebytes.com/spam

4. How Email Spam Filters Work and Ways to Safeguard Against Them. [Електронний ресурс]. - Режим доступу: https://www.mailmodo.com/guides/spam-filters/

5. Mehmet Ali Abdulhayoglu and Bart Thijs. 2018. Use of locality sensitive hashing (LSH) algorithm to match Web of Science and Scopus. Scientometrics 116, 2 (2018), 1229–1245.

6. Osamah M Al-Qershi and Bee Ee Khoo. 2016. Copy-move forgery detection using on locality sensitive hashing and k-means clustering. In Information Science and Applications (ICISA) 2016. Springer, 663–672.

7. SitiHajarAminahAli,SeiichiOzawa,TaoBan,JunjiNakazato,andJumpei Shimamura.2016.AneuralnetworkmodelfordetectingDDoSattacks using darknet traffic features. In 2016 International Joint Conference on Neural Networks (IJCNN). IEEE, 2979–2985.

8. TurkyNAlotaiby,AlanoudAlhakbani,NujoodAlwhibi,GasebAlotaibi,andSalehAAlshebeili.2019.LocalitySensitiveHashingforECG-based Subject Identification. In 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA). IEEE, 1–4.

9. Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. 2014. Beyond locality-sensitive hashing. In Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms. SIAM, 1018–1028.

10. MariaAstefanoaei,PaulCesaretti,PanagiotaKatsikouli,MayankGoswami,andRikSarkar.2018.Multi-resolutionsketchesandlocalitysensitive hashing for fast

trajectory processing. In Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 279–288.

11. Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2017. ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In International Conference on Similarity Search and Applications. Springer, 34–49.

12. Mozhgan Azimpourkivi, Umut Topkara, and Bogdan Carbunar. 2017. A secure mobile authentication alternative to biometrics. In Proceedings of the 33rd Annual Computer Security Applications Conference. 28–41.

13. MosesSCharikar.2002.Similarityestimationtechniquesfromroundingal gorithms.InProceedingsofthethiry-fourthannualACMsymposiumon Theory of computing. 380–388.

14. Edgar Chávez, Gonzalo Navarro, RicardoBaeza-Yates, and José Luis Marroquín. 2001. Search in ginmetric spaces. ACM computing surveys (CSUR) 33, 3 (2001), 273–321.

15. Chyouhwa Chen, Shi-Jinn Horng, and Chin-PinHuang. 2011. Locality sensitivehashingforsampling-basedalgorithmsinassociationrulemining. Expert Systems with Applications 38, 10 (2011), 12388–12397.

16. Dandan Chen. 2016. Structural Nonparallel Support Vector Machine Based on LSH for Large-Scale Prediction. In 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW). IEEE, 839–846.

17. Lianhua Chi and Xingquan Zhu. 2017. Hashing techniques: A survey and taxonomy. ACM Computing Surveys (CSUR) 50, 1 (2017), 1–36.

18. Michael Cochez, Vagan Terziyan, and Vadim Ermolayev. 2017. Large scale knowledge matching with balanced efficiency-effectiveness using lsh forest. In Transactions on Computational Collective Intelligence XXVI. Springer, 46–66.

19. Rodolfo da Silva Villaca, Luciano Bernardes de Paula, Rafael Pasquini, and Mauricio Ferreira Magalhaes. 2013. A similarity search system

based on the hamming distance of social profiles. In 2013 IEEE Seventh International Conference on Semantic Computing. IEEE, 90–93.

20. Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. 2011. Fast locality-sensitive hashing. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. 1073–1081.

21. Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In Proceedings of the twentieth annual symposium on Computational geometry. 253–262.

22. Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. 2019. Learning Space Partitions for Nearest Neighbor Search. arXiv preprint arXiv:1901.08544 (2019).

23. 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.

24. Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. Digital investigation, 3, pp. 91-97.

25. Protect your users properly today with Spamhaus ZEN + DBL + RPZ. [Електронний ресурс]. - Режим доступу: https://www.spamhaus.org

26. Кондратюк А.В., Кметик В.В. Виявлення програм-вимагачів. Матеріали науково-практичного симпозіуму «Захист інформації», Тернопіль, 2023. – С. 100-101.

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

ДОДАТОК А

Копії публікацій

ГРОМАДСЬКЕ ОБЄДНАННЯ
«КІБЕРБЕЗПЕКА І АВТОМАТИЗАЦІЯ»

Матеріали
науково-практичного симпозіуму
«ЗАХИСТ ІНФОРМАЦІЇ»

30 листопада 2023
Тернопіль

УДК 004.056.5

КОНДРАТЮК А.В., КМЕТИК В.В.
Західноукраїнський національний університет

ВИЯВЛЕННЯ ПРОГРАМ-ВИМАГАЧІВ

Вступ. Історія програм-вимагачів сягає кінця 1990-х років, коли їх передбачили як потенційну загрозу, яка агресивно використовує криптографію. У цьому відношенні атаки програм-вимагачів можна розділити на дві категорії: крипто-вимагачі та програми-вимагачі з блокуванням. Крипто-вимагач характеризується своєю здатністю шифрувати файли, таким чином унеможливлює жертвам доступ до файлів, без ключа дешифрування. Крім того, програми-вимагачі блокують та/або вимикають деякі служби на комп'ютері жертви. У той час як захисники виявляють уразливості нульового дня, суб'єкти загрози зайняті пошуком нових способів проникнення в цільові системи, залишаючись непоміченими днями, тижнями чи навіть місяцями [1].

Програми вимагачі характеризується своєю схильністю до розвитку як інтенсивності, так і стратегії атаки. Отже, це вимагає, щоб фахівці з безпеки докладали більше зусиль для пошуку ефективних рішень для боротьби з атаками програм вимагачів.

Мета: Проаналізувати ознаки програм-вимагачів, які дозволять запобігти поширенню атаки**.**

1. Ознаки атаки програм-вимагачів

Атаки програм-вимагачів зростають. Згідно з останніми дослідженнями, кількість атак програм-вимагачів майже подвоїлася в першій половині 2022 року, причому Сполучені Штати є найбільш цільовою країною, на яку припадає приблизно 55% заражень.

Програмне забезпечення-вимагач – це форма зловмисного програмного забезпечення, що постійно розвивається, призначена для шифрування файлів на пристрої, роблячи будь-які файли та системи, які від них залежать, непридатними. Потім зловмисники вимагають викуп в обмін на розшифровку [1].

Останні штами програм-вимагачів використовують техніку «подвійного вимагання», яка передбачає отримання копії даних жертви перед початком процесу шифрування. Потім зловмисники погрожуватимуть оприлюднити дані, якщо жертва відмовиться платити викуп.

Поширені атаки програм-вимагачів. Більшість атак програм-вимагачів здійснюються електронною поштою за допомогою певних методів соціальної інженерії, таких як фішинг. Зазвичай зловмисник маскується під довірену особу, щоб обманом змусити жертву завантажити шкідливу програму.Однак існує багато інших способів зараження.

Наприклад, у деяких випадках жертва буде перенаправлена на шкідливий веб-сайт, який запропонує встановити програму-вимагач. Крім того, веб-сайт може представити підроблений екран входу, який зловмисник використовуватиме для збору облікових даних, а потім використає ці облікові дані для здійснення

атаки з цільової організації.

Деякі форми програм-вимагачів вбудовані в програми та плагіни, які жертви встановлюють, вважаючи, що їм довіряють. Хоча рідше, бувають випадки, коли програма-вимагач зберігається на знімному носії, який автоматично запускається, коли жертва підключає диск до свого пристрою. В даний час спостерігається збільшення кількості атак програм-вимагачів, які використовують протокол віддаленого робочого столу (RDP) для виконання атаки.

Щоб зашифрувати 10 000 файлів середньому варіанту програми-вимагача, потрібно приблизно 4,5 хвилини. Природно, різні компанії зберігатимуть різну кількість файлів, тому важко точно передбачити, скільки часу знадобиться для повного розгортання атаки програм-вимагачів. Однак, якщо припустити, що компанії мають правильні рішення, навіть невеликого часового вікна може бути достатньо, щоб зупинити атаку.

Звичайно, щоб запобігти поширенню атаки програм-вимагачів, є ознаки, на які потрібно звернути увагу, зокрема [2]:

- сплеск активності диска, оскільки сценарій програми-вимагача шукає та шифрує файли у вашій системі;
- низька продуктивність системи, оскільки сценарій використовує системні ресурси для виконання пошуку та шифрування файлів;
- створення нових облікових записів, особливо привілейованих;
- підозрілий вхідний і вихідний мережевий трафік, оскільки сценарій програми-вимагача взаємодіє з сервером командування та керування (C&C);
- встановлення несанкціонованого програмного забезпечення, оскільки зловмисники встановлюють різні інструменти, наприклад Mimikatz, щоб допомогти їм використовувати вразливості та виконувати інші відповідні завдання;
- системи безпеки втручаються, щоб перешкодити моніторингу;
- резервні копії підробляють, намагаючись перешкодити жертві відновити свої файли;
- у мережі скануються порти, що свідчить про те, що зловмисники намагаються перейти з однієї системи в іншу;
- програми більше не працюють, оскільки файли, від яких вони залежать, шифруються.

У разі атаки програм-вимагачів важливо від’єднати заражені системи від мережі. Також важливо мати план відновлення після атаки програм-вимагачів, включаючи регулярне резервне копіювання та стратегію відновлення.

Висновки.

Атаки програм-вимагачів є серйозною загрозою для компаній будь-якого розміру. Вони можуть призвести до втрати цінних даних, простою та втрати прибутку.

Однак є кроки, які компанії можуть вжити, щоб захистити себе від атак програм-вимагачів, зокрема навчати співробітників, підтримувати актуальне програмне забезпечення та використовувати надійні паролі.

Перелік використаних джерел.

1. Ransomware attack: What is it and how does it work? [Електронний ресурс]. - Режим доступу: https://nordvpn.com/uk/blog/what-is-ransomware/

2. StopRansomware Guide. [Електронний ресурс]. - Режим доступу: https://www.cisa.gov/stopransomware

УДК 004.056.5

КОСТЮК О. В.

Західноукраїнський національний університет

КЛЮЧОВІ АСПЕКТИ ТА ПЕРЕВАГИ ЦЕНТРАЛІЗОВАНОГО ЗБОРУ ТА ЗБЕРЕЖЕННЯ ЖУРНАЛІВ РОБОТИ ІНФРАСТРУКТУРИ

Вступ. У сучасному світі мережева інфраструктура стає все більш складною і розгалуженою, обслуговуючи потреби підприємств і організацій будь-якого масштабу. Кожного дня зростає кількість пристроїв, програм та мережевих послуг. З огляду на цю динаміку, важливо забезпечити ефективний моніторинг та управління інфраструктурою, щоб вчасно виявляти проблеми, забезпечувати безпеку та оптимізувати роботу [1].

Один із важливих аспектів сучасного моніторингу мережі - це збір та аналіз журналів подій (логів) з різних джерел в одній централізованій системі. Цей підхід дозволяє отримати повну картину подій, що відбуваються у мережі та системі, а також ефективно реагувати на проблеми та загрози [2].

Необхідність централізованого збору журналів стає більш актуальною в умовах зростаючої складності інформаційних систем та загроз цифровій безпеці. Розглянемо цю тему детальніше і визначимо, як цей підхід може сприяти покращенню ефективності моніторингу та безпеки мережі.

Мета. Дослідження переваги централізованого збору журналів у мережевому середовищі.

1. Огляд централізованого збору логів елементів інфраструктури

Збір логів - це процес перехоплення, обробки та зберігання журналів подій з мереж, інфраструктур та додатків. Ці журнали містять дані про помилки, зміни конфігурацій та інші важливі події. Процес включає збір логів з різних джерел, таких як мережеві пристрої та сервери, а також їх агрегацію й обробку для вилучення корисної інформації. Логи зберігаються у централізованій базі даних, деякі з них архівуються для виконання регуляторних вимог.

Різні програмні засоби використовуються для збору логів. До них належать ELK Stack для зберігання, пошуку, обробки та візуалізації логів, OpenSearch - безкоштовний аналог ELK з функціоналом аналізу в реальному часі, Elastic Beats для легковагового збору логів, та Syslog - стандартний протокол для UNIX-подібних систем і мережевих пристроїв.



ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В. ЧОРНОВОЛА

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ (КБКІТ – 2023)

науково-практична конференція
молодих вчених, аспірантів та студентів

29–31 серпня 2023
Тернопіль

СЕКЦІЯ 3

КРИПТОГРАФІЧНІ МЕТОДИ ЗАХИСТУ ІНФОРМАЦІЇ

УДК 004.056.5

Смірнов Д.С., Іваніцкий Н.Ю., Кондратюк А.В.

Західноукраїнський національний університет

ВИЯВЛЕННЯ НЕВІДОМИХ ШКІДЛИВИХ ПРОГРАМ ЗА ДОПОМОГОЮ SSDEEP

Вступ. Для фільтрації постійно зростаючих обсягів цифрових даних необхідні автоматизовані системи. Процес фільтрації зазвичай використовується для виявлення підозрілих файлів або для фільтрації безпечних файлів операційної системи. Зазвичай для цього використовуються попередньо обчислені бази даних криптографічних хеш-значень і порівнюються хеші наявних файлів із файлами в базі даних. Ідентифікація та фільтрація точних дублікатів зменшує кількість даних, які потрібно досліджувати вручну. Зведення до мінімуму сторонніх даних або виділення підозрілих і релевантних даних є необхідною умовою проведення ефективного розслідування та реагування на інциденти. Однак криптографічні хеш-функції обмежені ідентифікацією ідентичних копій; відредаговані версії документів або оновлене програмне забезпечення неможливо ідентифікувати. Різні версії коду програмного забезпечення, змінені документи чи вбудовані об'єкти, швидше за все, не відповідатимуть жодному з попередньо обчислених хеш-значень. Подібним чином зловмисне програмне забезпечення, яке використовує самомодифікацію для зміни свого коду при кожному зараженні, залишається невиявленим. Це також дозволяє зловмисникам протидіяти процесу фільтрації, вносячи зміни лише в один біт до відомих файлів. Отже, криптографічне хешування не підходить для ідентифікації подібних файлів.

Мета: дослідити переваги та можливості нечіткого хешування

1. Нечіткі хеші

Для ідентифікації даних використовуються криптографічні хеші. У цифровій криміналістиці вони служать засобом ідентифікації файлів. Вони використовуються для відповіді на запитання: «Чи точно файл X збігається з файлом Y?» Зміна в одному біті файлу призведе до створення абсолютно нового криптографічного хешу для цього файлу. Одними з найбільш часто використовуваних криптографічних хешів є MD5, SHA-256 і SHA-512.

Для виявлення подібних файлів використовують хешування на основі подібності, одним із прикладів є SSDeep. SSDeep – це техніка часткового хешування, що запускається контекстом (CTPH), щоб ідентифікувати файли з попередніми файлами, тобто знайти повторюваний вміст. SSDEEP можна ефективно використовувати для порівняння пошуку подібності вмісту з існуючою загальновідомою базою даних зловмисного програмного забезпечення. В більшості антивірусів провайдери використовують цю нечітку техніку для пошуку нового шкідливого програмного забезпечення. SSDeep обчислюється шляхом запуску хеш-функції для сегментів файлу фіксованого розміру. В

результаті створюється спеціальний хеш, який відповідає на запитання: «Чи схожий файл X на файл Y?». Тому невеликі зміни у файлі лише трохи змінять хеш SSDeep, який він створює.

Розглянемо приклад. Ось два повідомлення з різницею лише в 1 байт: Повідомлення 1. Spam Detection Algorithms Based on Hash Similarity Повідомлення 2. Spam Detection Algorithms Based on Hash SimilaritY. Обчислимо хеш функції SHA256 використовуючи бібліотеку Openssl:

%echo -n "Spam Detection Algorithms Based on Hash Similarity" | openssl dgst-sha256
7d6ac100b53635babc40a19888ba3fdaecfd516f4b5b498f6810339c9e8f6e7d
%echo -n "Spam Detection Algorithms Based on Hash SimilaritY" | openssl dgst -sha256
511a21ad202878764322b07e72050d60b3ba6159a954e26c9078fac4ab053bfc

Як бачимо, для кожного повідомлення створюється інший хеш SHA-256.

Тепер обчислимо на SSDeep хеші цих повідомлень:

% ssdeep -r *
ssdeep,1.1--blocksize:hash:hash,filename
3:XIbAmG3uWGW/X/y:XI0+E//y,"/file1.txt"
3:XIbAmG3uWGW/X/S:XI0+E//S,"/file2.txt"

Отже, хеш SSDeep майже ідентичний, за винятком різниці в одному символі. Як було сказано раніше, криптографічні хеші можуть допомогти однозначно ідентифікувати зловмисне програмне забезпечення. Якщо ми виявимо підозрілий файл, ми можемо запитати наші програми аналізу загроз, щоб дізнатися, чи знають вони про його хеш SHA256, і ми можемо отримати глибокі результати щодо зловмисного програмного забезпечення. Однак, якщо зловмисне програмне забезпечення, яке ми досліджуємо, було модифіковане, наприклад, зробило найменшу зміну рядка, тоді метод його пошуку за допомогою хешу SHA256 не дасnm жодних результатів, якщо тільки ця модифікація не відома програмі аналізу загроз, яку ми використовуємо.SSDeep працює за допомогою одного методу: спочатку порівнює вміст двох файлів і створює унікальні хеші для обох, а потім порівнює ці хеші, щоб оцінити будь-яку схожість між ними та знайти будь-які незначні зміни, які наявні між файлами. Завдяки цьому процесу SSDeep визначає файли, які мають схожі характеристики, хоча вони незалежно обробляються кількома програмами чи службами.

Нечітке хешування швидко стало одним із популярних методів цифрової криміналістики та аналізу шкідливих програм, особливо серед аналітиків. Одним із відомих інструментів, який використовується для аналізу нечіткого хешування, є ssdeep, який дозволяє швидко й ефективно порівнювати файли на схожість.

Висновки. В роботі показано переваги та можливості нечіткого хешування, за допомогою SSDeep, для виявлення варіантів шкідливого програмного забезпечення або перевірки того, чи два файли мають спільний код.

Перелік використаних джерел.

1. SSDEEP Hash – Threat Detection with Fuzzy Techniques. [Електронний ресурс].- Режим доступу: https://www.socinvestigation.com/ssdeep-hash-threat-detection-with-fuzzy-techniques/