Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

ДИПЛОМНА РОБОТА МАГІСТРА

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

на тему «Метод та засоби емулятора виявлення кібер-загроз типу «фішинг»

КвРКІ 015100. 16.13 ПЗ

Виконав: студент 2 курсу, група КІ2м-20-1 _____ Михасько Я.Ю.
Підпис Ініціали, прізвище

Керівник доктор техн. наук, професор _____ Лисенко С.М.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н., проф.
Т.О. Говорущенко
__ _____ 2022 р.

Хмельницький, 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень магістр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма освітньо-наукова програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 01 " 09 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Михасько Яні Юріївні
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) «Метод та засоби емулятора виявлення кібер-загроз типу «фішинг»

Керівник проекту (роботи) Лисенко С.М., д.т.н., професор
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 06.01.2022 р. № 1

2. Строк подання студентом проекту (роботи) на кафедру 03.05.2022 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження методів та засобів виявлення фішингових атак

Модель процесу функціонування емулятора виявлення атак типу фішинг

Удосконалений метод виявлення кібер-загроз типу «фішинг»

Програно-апаратні засоби емулятора виявлення кібер-загроз типу «фішинг»

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «_06_» _____09_____ 2021р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики ДРМ з керівником	05.09.2021	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	05.10.2021	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	05.11.2021	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	05.12.2021	виконано
5	Робота над науковою статтею	05.01.2022	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.02.2022	виконано
7	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	05.04.2022	виконано
8	Оформлення пояснювальної записки згідно вимог	15.04.2022	виконано
9	Попередній захист ДРМ	18.04.2022	виконано
10	Захист ДРМ на засіданні ЕК	До 10.05.2022	

Студент _____ Я.Ю.Михасько
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Лисенко С.М.
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: «Метод та засоби емулятора виявлення кібер-загроз типу «фішинг»

Автор роботи: Михасько Я.Ю.

Керівник роботи: Лисенко С.М., професор кафедри КІІС

Пояснювальна записка: 85 с., 16 рис., 11 табл., 2 дод., 25 джерел.

ПРОГРАМНО-ТЕХНІЧНИЙ ЗАСІБ ВИЯВЛЕННЯ АТАК ТИПУ ФІШИНГ, ФІШИНГ, ЕМУЛЯТОР, КІБЕР-ЗАГРОЗА, ПКВМ.

Об’єктом дослідження є процес виявлення атак типу фішинг на основі застосування емулятора.

Предметом дослідження є моделі, методи та програмно-технічні засобами виявлення атак типу фішинг на основі застосування емулятора.

Метою дипломної роботи є підвищення достовірності та виявлення атак типу фішинг на основі застосування емулятора.

Для розв’язання поставлених задач використовувалися основні положення системного аналізу, методів аналізу даних, теорії дискретної математики, методів машинного навчання, теорії комп’ютерних мереж та систем.

Наукова новизна отриманих результатів:

− набули подальшого розвитку програмно-технічні засоби захисту інформації від атак типу фішинг, які забезпечують виявлення атак типу «фішинг» із застосуванням емулятора з високою достовірністю.

− удосконалено метод виявлення кібер-загроз типу «фішинг» на основі застосування емулятора, який на відміну від відомих розглядає деталі модифікацій фішингових атак та їх еволюцію з часом, і який забезпечує підвищення достовірності виявлення таких атак.

В результаті здійсненого проєктування емулятора стало можливим виконати експериментальні дослідження з окрема: отримати результати аналізу засобами емулятора на основі баз даних фішингових сайтів.

Практична значимість отриманих результатів полягає у розроблеці програмно-технічних засобів виявлення кібер-загроз типу «фішинг», які забезпечуватимуть захист інформації з високою достовірністю.

Дослідження, представлені у кваліфікаційній роботі, проводились в рамках держбюджетної НДР Хмельницького національного університету 1Б-2021 «Самоорганізована розподілена система виявлення зловмисного програмного забезпечення в комп’ютерних мережах» (ДР № 0121U109936) 2021-2022 рр. За темою дипломної роботи опубліковано статтю на тему «Дослідження методів виявлення кібератак в комп’ютерних мережах як засобів до побудови резильєнтних до вторгнень ІТ-інфраструктур» в фаховому журналі «Комп’ютерні системи та інформаційні технології» №3 за 2021 рік [1], а також опубліковано тези у Всеукраїнській науково-практичній конференції Актуальні Проблеми Комп’ютерних Наук (АПКН-2021) [2]. Було взято участь у Всеукраїнській науково-практичній конференції Актуальні Проблеми Комп’ютерних Наук.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ЕВФА – емулятора виявлення кібер-загроз типу «фішинг»

ЗПЗ – зловмисне програмне забезпечення

ФА – фішингова атака

КБ – кіберезпека

ЖЦ – життєвий цикл

ПК – персональний комп’ютер

ІС – інформаційна система

ВСТУП

Інтернет надає людям і компаніям новий спосіб зв’язку. Це спричинило зміни в способі життя людей, коли, наприклад, онлайн-покупки та онлайн-освіта почали замінювати магазини та школи. Це також спричинило зміни в компаніях, які тепер зосереджуються на створенні нових Інтернет-продуктів та послуг, щоб домінувати на ринку. Однак цей зсув дає злочинцям можливість розпочинати нові види злочинів за допомогою комп’ютерів і мереж, відомих як кіберзлочинність.

Фішингова атака – це тип кіберзлочину, при якому конфіденційна інформація людей викрадається через веб-сайт, який імітує інший, законний веб-сайт. Незважаючи на величезні зусилля науковців і промисловості останніх років, дослідження протидії фішингу все ще стикаються з кількома проблемами.

Так звані "фішингові атаки" (phishing attacks) - це тип атаки, під час яких зловмисники використовують легітимний веб-сайт, щоб викрасти конфіденційну інформацію у кінцевих користувачів. Фішингові атаки є однією з важливих загроз для приватних осіб та корпорацій у сучасному Інтернеті. Ця проблема активно досліджується як науковцями колами, так і індустрією антивірусних засобів портягом останніх кількох років.

Спроби надати ефективні антифішингові рішення мали два основні підходи: (1) ідентифікувати фішингову атаку шляхом порівняння її подібності з цільовим сайтом, та (2) полягає у розгляді внутрішніх характеристик нападів.

В дослідженні пропонується розглянути цю проблему під новим кутом зору – а саме спробою розроблення емулятора виявлення кіберзагроз типу «фішинг».

В основі виявлення пропонується здійснювати виявлення атак із застосування апаратно-технічного засобу, який буде здатним не лише використовувати внутрішні характеристики атаки та порівнювати схожість між атаками та цільовими сайтами, а також аналізуватиме поведінку атаки.

Запропонований метод здійснює глибокий аналіз того, як зловмисники створюють фішингові атаки. Аналіз ґрунтується на тому, що більшість фішингових атак є дублікатами або квазідублікатами колишніх атак.

Враховуючи, що фішингові атаки не будуються з нуля, в роботі залучено емулятор атак, який дозволяє продукувати висновок про присутність атаки на основі кластеризації для оцінки подібності між атаками.

Метод також ґрунтується на залученні процедури аналізу еволюції потенційних фішингових атак шляхом відстежування змін з плином часу.

Метою методу є отримання результату щодо наявності множини поведінкових змін зловмисного характеру, а також аналіз кількості ітерацій атак.

Іншим аспектом запропонованого методу є виявлення та аналіз фішингових атак на стороні клієнта.

З цією метою емулятор виявлення атак повинен бути розміщений у сервері мережі.

Для цього здійснюється статичний аналіз вихідного коду «наборів фішингу», а також аналіз та відстеження вкраденої інформації.

Оскільки більшість фішингових атак використовують електронну пошту як засіб вилучення вкраденої інформації, в дослідженні запропоновано модель глибокого навчання для виявлення цих повідомлень у мережевому трафіку. Цей підхід може бути використаний, щоб легко виявити, що фішингова атака розміщена у великій комп’ютерній мережі.

Метою кваліфікаційної роботи магістра роботи є підвищення достовірності та виявлення атак типу фішинг на основі застосування емулятора.

Поставлена мета досягається розв’язанням наступних задач:

1. Дослідити методи та засоби виявлення фішингових атак, а саме проаналізовано суть поняття фішингових атак.

2. Дослідити переваги та недоліки методів виявлення фішингових атак. Дослідити життєвий цикл фішингових атак. Проаналізувати основні механізми виявлення фішингових сайтів, виявлення фішингу за допомогою чорних списків, виявлення фішингу шляхом порівняння з ціллю, виявлення фішингу шляхом перегляду внутрішніх характеристик.

3. Розробити моделі процесу функціонування емулятора виявлення атак типу фішинг, зокрема представлені моделі фішингових атак, подано процес виявлення фішингових копій і варіацій, спроектувати відбиток фішингової сторінки: вектор тегів.

4. Провести моделювання схожості фішингових сторінок: пропорційна відстань, наведено алгоритм кластеризації.

5. розробити удосконалений метод виявлення кіберзагроз типу «фішинг».

6. Здійснити реалізацію удосконаленого методу виявлення кіберзагроз типу «фішинг»у вигляді емулятора - програмно-апаратного засобу.

Об'єкт дослідження – процес виявлення атак типу фішинг на основі застосування емулятора.

Предмет дослідження – моделі, методи та програмно-технічні засобами виявлення атак типу фішинг на основі застосування емулятора.

Методи дослідження. Для розв’язання поставлених задач використовуються основні положення системного аналізу, методів аналізу даних, теорії дискретної математики, методів машинного навчання, теорії комп’ютерних мереж та систем.

Наукова новизна отриманих результатів:

1. Удосконалено метод виявлення кібер-загроз типу «фішинг» на основі застосування емулятора, який на відміну від відомих розглядає деталі модифікацій фішингових атак та їх еволюцію з часом, і який забезпечує підвищення достовірності виявлення таких атак.

2. Набули подальшого розвитку програмно-технічні засоби захисту інформації від атак типу фішинг, які забезпечують виявлення атак типу «фішинг» із застосуванням емулятора з високою достовірністю.

Практична цінність. В результаті виконаного наукового дослідження було розроблено програмно-технічні засоби виявлення кібер-загроз типу «фішинг», які забезпечуватимуть захист інформації з високою достовірністю.

Зв’язок роботи з науковими програмами, планами, темами. Дослідження, представлені у кваліфікаційній роботі, проводились в рамках держбюджетної

НДР Хмельницького національного університету 1Б-2021 «Самоорганізована розподілена система виявлення зловмисного програмного забезпечення в комп’ютерних мережах» (ДР № 0121U109936) 2021-2022 рр.

За темою дипломної роботи опубліковано статтю на тему «Дослідження методів виявлення кібератак в комп’ютерних мережах як засобів до побудови резильєнтних до вторгнень ІТ-інфраструктур» в фаховому журналі «Комп’ютерні системи та інформаційні технології» №3 за 2021 рік [1], а також опубліковано тези у Всеукраїнській науково-практичній конференції Актуальні Проблеми Комп’ютерних Наук (АПКН-2021) [2]. Було взято участь у Всеукраїнській науково-практичній конференції Актуальні Проблеми Комп’ютерних Наук.

1 ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ ВИЯВЛЕННЯ ФІШИНГОВИХ АТАК

1.1 Що таке фішингові атаки

Так звані «фішингові атаки» — це атаки, під час яких фішингові сайти маскуються під легітимні веб-сайти з метою крадіжки конфіденційної інформації [3]. Це відбувається, коли зловмисники надсилають SMS або електронні листи зі шкідливими URL-адресами, обманюючи жертв, щоб відкрити шкідливе посилання, яке спрямовує їх на підроблений веб-сайт із зовнішнім виглядом дуже схожим на законний веб-сайт. Після того, як жертви обманом надають свою конфіденційну інформацію, зловмисник може використати цю інформацію, щоб запустити атаки соціальної інженерії або вкрасти іншу інформацію, пов’язану з обліковим записом жертви. На рисунку 1.1 показано фішингову атаку, яка імітує сторінку входу в Gmail. Зловмисник імітує кожну деталь легітимного, наприклад логотип, фавікон і навіть зовнішні посилання, що ускладнює візуальне відрізнення від легітимного [4].

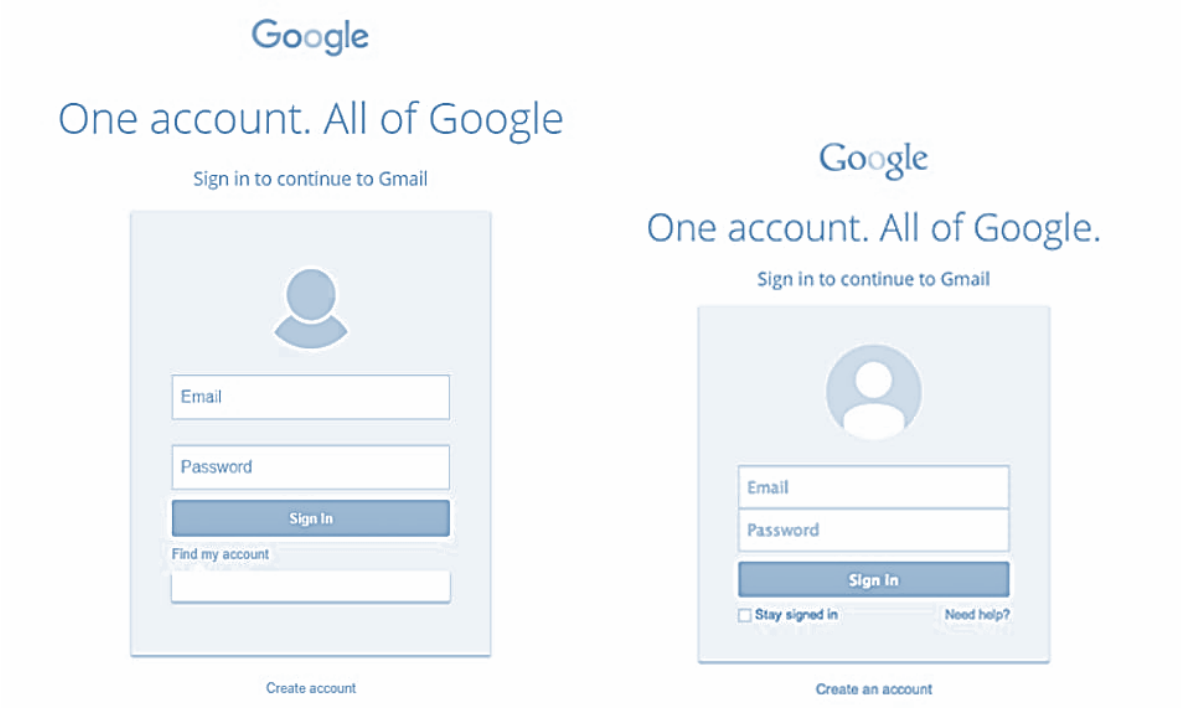


Рисунок 1.1 – Приклад фішингу: а) фішингова атака, яка імітує сторінку входу в Gmail; б) легітимна сторінка входу в Gmail

Відповідно до цільового діапазону атак, фішингові атаки можна розділити на дві категорії: цільовий фішинг і клон-фішинг. Цільовий фішинг – це атака, спрямована на особу. Щоб підвищити ймовірність успіху та довіри, зловмисники зазвичай збирають та використовують особисту інформацію деяких членів організації. Наприклад, зловмисник, який видає себе за адміністратора, надсилає фішингові листи учасникам. І навпаки, клон-фішинг, як основна форма фішингових атак, є атакою без спеціального вибору цілей атаки. У клон-фішингу, щоб зробити атаку більш ефективною, зловмисник постійно вдосконалює технологію обфускації, щоб збільшити схожість між підробленим сайтом і його цільовим сайтом [5].

1.2 Дослідження методів виявлення фішингових атак

Сьогодні існує велика кількість науково-практичних рішень щодо вирішення задачі виявлення фішингових атак.

В [6] вказують на те, що в підпільній економіці кіберзлочинності видною моделлю стає кримінальне програмне забезпечення як послуга, яка включає три ролі: виробників, рекламодавців і покупців, як показано на рисунку 1.2. Спочатку виробник проектує та створює повну структуру злочинного ПЗ, а потім повідомляє про це рекламодавця. Потім рекламодавець публікує злочинну програму на підпільних форумах і серверах Інтернет-релейних чатів (IRC). Нарешті, покупець замовляє рекламоване кримінальне програмне забезпечення та використовує його для здійснення кібератак. Іноді продюсером і рекламодавцем може бути одна і та ж людина. У цьому випадку виробник і покупець торгують безпосередньо напряму.

Зокрема, виробник приховує власну адресу електронної пошти в коді. Після того, як придбаний фішинговий сайт успішно обманює жертву, щоб надати конфіденційну інформацію, копія вкраденої інформації, надіслана покупцеві, також буде надіслана виробнику [7].

1.2.1 Життєвий цикл фішингових атак

Загалом життєвий цикл фішингової атаки складається з трьох періодів: період підготовки, активний період і період неактивності. Фішингова атака починається з підготовчого періоду і стає активною, коли фішингове посилання публікується в соціальних мережах або надсилається електронною поштою. Як тільки атака буде виявлена або заблокована, вона переходить до періоду неактивності. Однак це не означає, що атака припинена. Зловмисник може відновити атаку після внесення ряду змін [8].

1.2.2 Підготовчий період

Під час підготовки фішингової атаки зловмиснику необхідно пройти три етапи: підготувати фішинговий сервер, розгорнути фішингову атаку та опублікувати фішингову атаку.

Підготувати фішинговий сервер. На цьому етапі зловмисник або орендує хмарний хост (шкідлий сервер), або зламає вразливий сервер (скомпрометований сервер), щоб використовувати його як фішинговий сервер. Шкідливі сервери зазвичай надходять від постачальників хмарного хостингу, які пропонують нижчу плату за підписку або безкоштовні пробні періоди. Скомпрометований сервер є законним хостом, таким як особистий блог або комерційний веб-сайт, але він вразливий, оскільки не оновлює виправлення вчасно або використовує старішу версію платформи. Щоб знайти вразливих хостів, зловмисник використовує різні інструменти для сканування своєї жертви в Інтернеті. Один із підходів описано [9], які запитують пошукові системи Інтернету за допомогою добре розроблених пошукових термінів під назвою «Google Dorks». Google Dorks використовує розширені пошукові оператори пошукових систем, такі як "inurl" або "intitle", для отримання вразливих

Портали веб-сайтів. Потім зловмисник зламує хости, щоб отримати контроль і права доступу.

Розгортання фішингової атаки. Під час фази розгортання атаки зловмисник завантажує та розгортає фішингову атаку на фішинг-сервері та завершує відповідну конфігурацію, готуючись до атаки. Наприклад, зловмисник запускає веб-сервіси та служби SMTP, щоб забезпечити доступ до Інтернету та доставку електронної пошти. Для зловмисників, з невеликими знаннями в Інтернеті, вони можуть вибрати оплату конкретної підпільної організації, яка керує великою кількістю фішингових серверів, щоб допомогти їм завершити цей етап. Вартість послуги залежить від масштабу розгорнутих серверів і кількості запущених фішингових атак.

Здійснення фішингової атаки. Щоб «опублікувати» атаку, зловмисник надсилає жертвам повідомлення з фішинговими URL-адресами електронною поштою або через соціальну мережу та заманює їх відкрити фішингове посилання. Щоб підвищити довіру до посилання, зловмисник використовує різні методи обфускації, щоб приховати фактичне фішингове посилання. Наприклад, під час атаки омографа зловмисник створює доменне ім’я, схоже на добре відоме законне доменне ім’я, замінюючи деякі символи омографами (наприклад, G00gle.com, App0le.com) [10]. Служби скорочення URL-адрес, які надають користувачам коротшу URL-адресу для представлення оригінальної, також використовуються зловмисники, щоб приховати фактичне фішингове посилання.

1.2.3 Період дії

На цьому етапі жертва, яка натиснула фішингові посилання, спрямовується на фішинговий сайт, і її спонукають надати свою особисту інформацію. Згодом викрадена інформація або стає товаром на підпільному ринку, або безпосередньо використовується для викрадення майна жертви (наприклад, викрадення вмісту банківського рахунку). Варто зазначити, що вміст кожного відвідування фішингового веб-сайту може відрізнятися. У деяких просунутих фішингових атаках зловмисник використовує такі методи, як одноразовий доступ або відображення кількох сторінок. Це дозволяє

зловмиснику перенаправляти сканери безпеки компанії або жертв, які знову відвідують сайт, на законний сайт або фіктивну сторінку. В [11] показано, що такий підхід може допомогти продовжити тривалість атак.

1.2.4 Період неактивності

Після того, як фішинг-атака ідентифікована пильною жертвою або виявлена антифішинговими продуктами, її інформація синхронізується з організацією безпеки мережі та додається до її чорного списку. Потім чорний список синхронізується з антифішинговими продуктами та браузерами, щоб запобігти поширенню атаки. На даний момент фішингова атака перейшла в період неактивності. Однак це не означає закінчення атаки; зловмисник все ще може перезапустити атаку, перейшовши на інший хост або змінивши URL-адресу.

І промисловість, і науковці доклали багато зусиль для дослідження боротьби з фішингом і винайшли різні антифішингові рішення. Наприклад, такі браузери, як Google Chrome, FireFox і Safari, використовують безпечний перегляд Google, щоб надати своїм користувачам вбудований сервіс для запобігання фішинговим атакам[1]. Браузери Microsoft Internet Explorer і Edge також містять подібний вбудований механізм захисту під назвою SmartScreen. Однак зростання фішингових атак ніколи не припинялось і навіть не сповільнювалося. Згідно зі звітом Anti Phishing Working Group (APWG)[2], кількість зареєстрованих фішингових атак зросла на 340% з 2007 по 2021 рік. Це свідчить про те, що проблема фішингових атак не вирішена повністю. Мало придатних для використання набору даних. Хоча науковці запропонували різні дослідження проти фішингу, лише деякі з них опублікували свої набори даних, використані в експериментах. Навіть для цих опублікованих наборів даних надаються лише фішингові URL-адреси, і майже всі ці URL-адреси більше не дійсні на момент публікації. Це робить складним для подальших досліджень порівнювати їхні методи з попередньою роботою. Як наслідок, більшість

антифішингових досліджень повідомляють про ідеальні результати за власним набором даних, що ускладнює оцінку ефективності цих методів.

Новий напрямок досліджень необхідний для виявлення фішингових атак. У останніх антифішингових дослідженнях спостерігається тенденція до того, що нові дослідження або поєднують деякі нові функції з функціями попередньої роботи, або змішують різні функції, відмінні від інших. Однак вони не надають жодної оцінки впливу своїх нових функцій або комбінації функцій на виявлення фішингових атак. Тому важко оцінити цінність їхньої праці.

Сучасні дослідження протидії фішингу ведуться переважно у двох напрямках. Перший підхід полягає в тому, щоб знайти схожість між фішинговим сайтом і легальним сайтом, який він імітує. Другий підхід полягає в тому, щоб спробувати визначити внутрішні характеристики фішингових сайтів для тренування моделі машинного навчання для визначення фішингових та легальних сайтів. Однак ці підходи стикаються з низкою труднощів, щоб отримати уявлення про фішингові атаки.

Відстеження фішингової атаки. Хоча деякі загальнодоступні бази даних, наприклад, PhishTank[3] та VirusTotal[4], надають довгостроковий історичний архів фішингових атак, вони містять лише обмежену інформацію, таку як фішингова URL-адреса та дата повідомлення. Крім того, час існування фішингових атак дуже короткий, в середньому менше 15 годин[5]. Тому більшість опублікованих фішингових URL-адрес неактивні або недоступні. З наведених вище причин немає жодної структури чи методу для довгострокового моніторингу розвитку та модифікації фішингових атак. Нинішній аналіз розвитку фішингу зосереджується на простих статистичних даних, таких як кількість і ціль атак, а не на внутрішніх змінах.

Зупинка поширення фішингових атак. Нині багато антифішингових продуктів використовують чорні списки для стримування поширення фішингових атак. Але навіть після блокування фішингової атаки зловмисник все ще може повторно запустити ту ж фішингову атаку на іншому хості або з іншою URL-адресою, що може просто взламати наявні чорні списки на основі URL-адрес або домену.

Заздалегідь виявляти фішингові атаки. Затримка між ініціюванням і виявленням атаки, навіть лише кілька годин, все ще дає зловмисникам шанс успішно обдурити жертв. Потрібне краще рішення, щоб зупинити атаку, як тільки атака стане активною.

1.3 Дослідження методів виявлення фішингових атак

Існує значна частина наукової роботи, присвячена автоматичному виявленню та аналізу фішингових атак, головним чином у трьох напрямках: виявлення фішингових сайтів, виявлення фішингової електронної пошти та аналіз фішингових комплектів.

1.3.1 Виявлення фішингових сайтів

Оскільки фішингова атака працює лише тоді, коли жертви потрапляють на фішинговий сайт, загальною та інтуїтивно зрозумілою стратегією захисту є виявлення та блокування цих зловмисних сайтів. Більшість опублікованої літератури в цьому напрямку можна віднести до однієї з наступних категорій: виявлення фішингу за допомогою чорних списків, виявлення фішингу шляхом порівняння з цілями та виявлення фішингу за допомогою аналізу внутрішніх характеристик [12].

1.3.2 Виявлення фішингу за допомогою чорних списків

Чорний список – це механізм контролю доступу, який блокує перераховані елементи для доступу до захищених ресурсів. Щоб запобігти фішинговим атакам, основні браузери мають інтегровані динамічні чорні списки. Наприклад, Firefox, Safari та Chrome мають чорний список Google, Google Safe Browsing, щоб забезпечити користувачам захист від фішингових атак. Далі обговорюємо, як працюють чорні списки фішингу та проблеми, з якими стикаються чорні списки.

Як працюють чорні списки фішингу?

Колекція URL-адрес. На цьому кроці постачальник чорного списку збирає фішингові URL-адреси з різних джерел даних. Сюди входять виявлені фішингові сторінки або електронні листи від компаній з інтернет-безпеки, а також атаки, повідомлені системам спільноти (наприклад, PhishTank).

Перевірка URL-адреси. Щоб зменшити кількість помилкових спрацьовувань, може знадобитися перевірка зібраних URL-адрес вручну. Рецензенти позначають зібрані URL-адреси, перевіряючи додаткову інформацію, таку як репутація домену та архів[2] сторінок.

Оновлення чорного списку. Як тільки підтверджена URL-адреса підтверджується як фішинг, вона додається до центральної бази даних, а потім синхронізується з клієнтськими браузерами та партнерами, які підписалися на джерело чорного списку [13].

1.3.3 Неефективність чорних списків фішингу

На практиці проблема фішингових чорних списків полягає в тому, що вони не можуть запобігти фішинговим атакам в режимі реального часу. Іншими словами, існує затримка між ініціюванням фішингової атаки та моментом її додавання до чорного списку. В [14] показано, що 47%-83% фішингових атак потрапляли в чорний список через 12 годин.

Результати з [15] показали, що більшість фішингових атак (62%) потрапляли в чорний список у середньому через 20 днів після встановлення. Ця затримка спричинена кожним кроком у процесі компіляції та публікації, як показано на рисунку 2.2. Наприклад, коли постачальник чорного списку збирає фішингову URL-адресу, атака може бути активною деякий час. Оскільки процеси перевірки та оновлення зазвичай займають більше часу, затримка ще більше.

Затримка в чорних списках фішингу дає зловмисникам час, щоб обдурити жертв і зібрати вкрадену інформацію. Тому чорні списки не можуть ефективно

запобігти фішинговим атакам, а можуть блокувати лише ті атаки, які були активні деякий час.

1.3.4. Виявлення фішингу шляхом порівняння з ціллю

Суть фішингових атак полягає в тому, що, імітуючи законний сайт, зловмисники можуть обманом змусити жертв ввести свою конфіденційну інформацію. Тому, якщо сайт схожий на інший законний сайт, він, швидше за все, є фішинговим. З цієї точки зору деякі методи пропонують порівняти схожість між фішинговими та легальними сайтами для виявлення фішингових атак. Ці дослідження зосереджені на вирішенні двох основних проблем. Однією з проблем є визначення мети фішингу. Інша проблема полягає у виборі відповідних показників для вимірювання подібності [16].

1.3.5 Подібність макета сторінки

DOM — це інтерфейс програмування сторінки HTML, який представляє кожен компонент сторінки як вузол у деревоподібній структурі. Зв'язок між компонентами представляється як зв'язок між батьківським і дочірнім вузлом. Таким чином, деякі дослідження намагаються використовувати DOM для аналізу подібності макета сторінки.

В [17] представлено розширення для браузера, що використовує подібність дерева DOM для виявлення фішингових атак. Спочатку вони зберігають зіставлення між конфіденційною інформацією користувача та законними сайтами, які використовують цю інформацію. Коли та сама конфіденційна інформація повторно використовується на іншому сайті, розширення перевіряє, чи DOM нового сайту схожий на пов’язаний. Якщо два DOM схожі, новий сайт позначається як фішинг. У своєму експерименті вони повідомляють лише про хибнопозитивні та хибнонегативні результати за різних порогових значень, але вони не згадують, скільки легальних сайтів і фішингових сайтів використовується для тестування, що ускладнює оцінку

ефективності їх методу. Інша проблема в цьому дослідженні полягає в тому, що умова виявлення «подібний DOM» може бути просто зламана зловмисниками на практиці. Наприклад, як показано на рисунку 1.2, деякі веб-сайти можуть мати кілька записів на сторінці входу. Ліва – це звичайна сторінка входу у Facebook, яку використовують більшість користувачів. Праворуч це ще один запис для входу. Таким чином, розширення браузера може зберігати лише DOM загального запису. Тому, якщо зловмисник створить атаку, яка імітує запис входу праворуч, запропонований метод буде невдалим. Зловмисник також може створювати фішингові атаки, які мають схожий вигляд, але складаються з абсолютно іншого DOM. Наприклад, зловмисник може зберігати більшість компонентів як зображення та вставляти їх як фон фішингової сторінки.

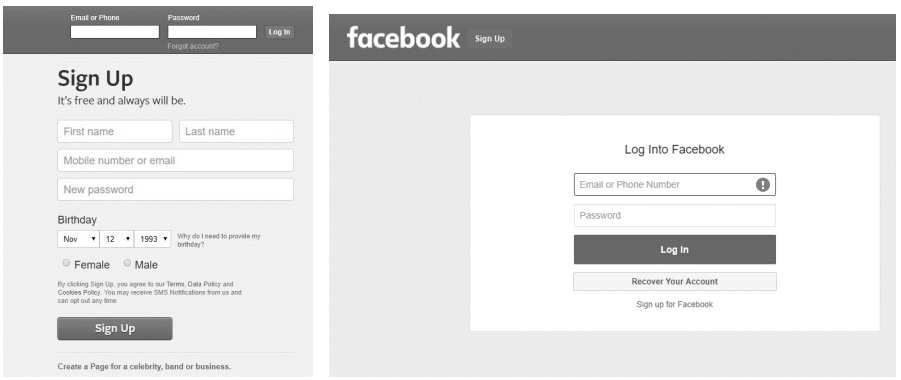


Рисунок 1.2 – Кілька входів у Facebook

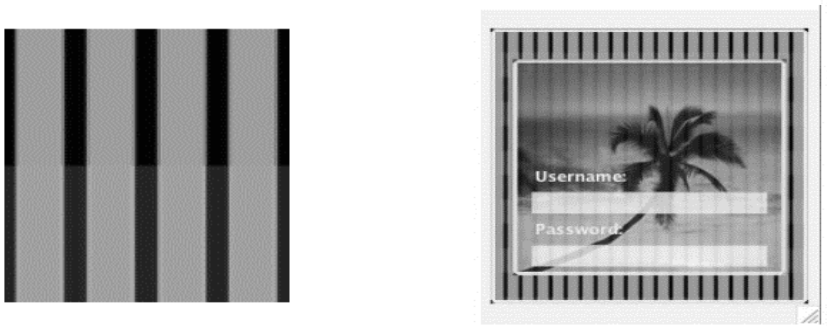
Щоб подолати цю проблему, в [18] за пропоновано використовувати функції просторового макета для порівняння схожості макета сторінки. Спочатку вони розбивають сторінку на кілька компонентів, а потім виділяють просторові характеристики кожного компонента, такі як положення та розмір. Ці просторові особливості можуть допомогти відловити сторінки з частково схожими компонентами. Щоб виявити фішингову сторінку, просторові характеристики підозрілої сторінки порівнюються з «доброякісним» набором даних, який складається з просторових об’єктів, отриманих із законних сайтів. Якщо подібність просторових ознак виходить за межі попередньо визначеного порогу, це ідентифікується як фішинг. Вони повідомляють про результат 93,3% точності та 91,9% відкликання на наборі даних із 100 фішингових і 100

легальних сайтів. Хоча цей підхід покращує показники, що використовуються для порівняння макета сторінки, і не вимагає конфіденційної інформації користувача для отримання цільового сайту, він все ще страждає від проблеми, подібної до роботи Розіелло: важко виявити фішингову сторінку, яка використовує більшість компонентів цільовий сайт як фонові зображення.

1.3.6. Візуальна схожість сторінки

Оскільки використання схожості макетів сторінок неефективно для виявлення фішингу, деякі дослідження намагаються застосувати технології візуального порівняння, щоб знайти схожість між фішинговими атаками та цільовими сайтами.

В [19] запропоновано схему «Динамічні обкладинки безпеки», яка не дозволяє зловмисникові імітувати сайт, вбудовуючи на сторінку візуальну ідентичність. Метою цього методу є встановлення сертифіката безпеки між користувачем і надійним веб-сайтом, який третій стороні важко імітувати. Іншими словами, користувач і надійний веб-сайт ділиться секретом як доказом безпечного зв’язку, який не може бути передбачений або відомий жодній третій стороні. Зокрема, користувач спочатку надсилає маркер безпеки на надійний веб-сайт, а потім сервер генерує хеш на основі цього токена за допомогою хеш-функції, відомої обом сторонам. Це хеш-значення пізніше перевіряється шляхом порівняння з обчисленим клієнтом хеш-значенням. Щоб спростити процес перевірки, вони пропонують технологію, яка називається візуальним хешуванням, візуальне представлення хеш-значення. Сторінка, повернута з надійного сервера, використовує цей візуальний хеш як фонове зображення, як показано на рисунку 1.3. Якщо сторінка, що перевіряється, не містить цього візуального хеша або містить невідповідний хеш, вона ідентифікується як фішинг.



a) b)

Рисунок 1.3 – Приклад веб-сторінки, захищеної візуальним хешем [19]: (a) згенерований візуальний хеш; (b) перевірена сторінка, розбита візуальним хешем

Одна з проблем цього дослідження полягає в тому, що автори не застосовують жодного формального експерименту до свого методу. Натомість вони обговорюють лише ряд проектів для майбутніх тестів. Тому іншим важко оцінити їхній метод. Більше того, цей метод має ряд обмежень, які ускладнюють його застосування на практиці. По-перше, це вимагає додаткової модифікації на стороні сервера для підтримки перевірки. Більше того, для завершення перевірки потрібно кілька людських втручань. Користувач не тільки ризикує витоку маркера безпеки (подумайте, чи надсилає користувач маркер безпеки на фішинговий сайт), але також несе відповідальність за ідентифікацію індикатора безпеки (візуальний хеш).

Щоб зменшити втручання користувача, деякі дослідження пропонують використовувати попередню колекцію доброякісних веб-сайтів [20] або ідентифікувати цільові сайти для фішингу за допомогою аналізу візуальної інформації. Ці дослідження зосереджені на пошуку належних візуальних функцій і показників схожості, які можуть відображати схожість фішингового сайту з цільовим сайтом.

В [21] запропоновано метод, який використовує відстань землеройників (EMD) та візуальні підписи скріншоту сторінки з низькою роздільною здатністю для оцінки візуальної подібності. Вони нормалізують скріншот сторінки у квадратне зображення фіксованого розміру (10*10). Для представлення

візуальних ознак масштабованого зображення пропонується вектор ознак, який складається з центроїда розподілу позицій і частоти кожного кольору. Щоб знайти оптимальний поріг EMD для виявлення фішингових атак, вони проводять експерименти з набором перевірки та вибирають поріг, який дає найкращий результат виявлення. Якщо відстань EMD між підозрілою сторінкою та будь-яким із сайтів у захищеній колекції сайтів менше цього оптимального порогу, це позначається як фішинг. Вони повідомляють про результат 99,87% точності для дуже незбалансованого набору даних, який складається з 10 272 легітимних сторінок, але лише 9 фішингових сторінок. Однак, стиснувши знімок екрана сторінки до такого маленького ескізу, більшість інформації на сторінці буде втрачено. Тому, якщо модель застосована до великого набору фішингових даних, може бути багато помилкових результатів.

В [22] подано простий метод отримання візуальних функцій знімка екрана сторінки. У своїй роботі вони застосовують звичайний компресор, такий як gzip або bzip2, щоб стиснути скріншот сторінки. Нормалізована відстань стиснення (NCD) використовується для вимірювання подібності між стиснутими зображеннями. Результат 95% точності і менше 1,7% помилкових спрацьовувань повідомляється на наборі даних із 120 легітимних сторінок і 320 фішингових сайтів. Проблема такого підходу полягає в тому, що ефективність виявлення сильно залежить від вибору компресора, але автори не дають чітких відповідей про те, як вибрати «оптимальний» компресор.

Замість використання стиснутої візуальної функції знімка екрана, деякі дослідження повідомляють про хороші результати, використовуючи часткову візуальну інформацію як ідентифікатор виявлення, таку як фавікон [23]. У своїй роботі автори спершу витягують фавікон із DOM підозрілої сторінки. Потім він порівнюється з значком і логотипом цільових сайтів для фішингу. Щоб зменшити кількість помилкових спрацьовувань, інформація про домен і рейтинг сторінки пропонуються як додаткові індикатори для виявлення фішингових атак. Вони повідомляють про результат 99,5% справжніх позитивних результатів і 0,15% хибних позитивних результатів на наборі даних з 3642

фішингових сторінок і 19 585 легітимних сторінок. Однак цей підхід працює лише для атак із використанням іконок.

1.3.7 Подібність ключових слів сторінки

Усі перераховані вище методи вимагають ручного збору потенційних цілей фішингу через відсутність можливості автоматично ідентифікувати ціль фішингу. Щоб подолати це обмеження, деякі дослідження використовують пошукову систему для пошуку цілей фішингу. Метою цього дослідження є вилучення ключових слів із підозрілої сторінки та пошук їх у пошуковій системі. Якщо домен підозрілої сторінки не відповідає жодній із сторінок, проіндексованих пошуковою системою, це ідентифікується як фішинг.

В [24] подано текстовий метод «Cantina» з використанням частоти термінів та інверсної частоти документів (TF-IDF) для визначення ключових слів сторінки. Ключові слова, виділені алгоритмом TF-IDF, потім передаються пошуковій системі для запитів на сторінки, що містять подібні ключові слова. Вони повідомляють про 89% справжніх позитивних і 1% помилкових результатів у базі даних із 100 фішингових і 100 легальних сайтів.

В [25] представлено метод «Cantina», поєднуючи методи обробки природної мови. У своїй роботі вони пропонують витягти ключові слова із заголовка сторінки та тексту авторських прав, які більше репрезентують цільовий бренд. Щоб ефективно витягти назву бренду, вони визначають 11 регулярних виразів, орієнтованих на різні варіанти форматів авторських прав. Щоб охопити конкретний випадок, коли назва бренду не включена в назву чи авторські права, але існує в інших об’єктах DOM, вони застосовують розпізнавання названих сутностей, щоб знайти назву бренду в текстовому вмісті. Розпізнавання іменованих сутностей — це завдання класифікації в рамках системи машинного навчання, яка шукає і групує імена об’єктів у неструктурованому тексті в деякі заздалегідь визначені категорії, наприклад, осіб та організацій. У цій роботі для ідентифікації брендів веб-сайтів використовується Stanford Named Entity Recognizer. Запропонований підхід дає 90,06% справжніх позитивних результатів і 1,95% хибних позитивних

результатів у наборі даних із 7906 фішингових сторінок і 3543 легітимних сторінок.

Замість вилучення ключових слів із текстового вмісту деякі дослідження використовують інші функції для отримання ключових слів сторінки. В [26] представлено рішення за допомогою логотипу сайту. Спочатку вони використовують техніку сегментації зображень, щоб витягти логотип веб-сайту зі сторінки, яка перевіряється. Потім вони використовують пошук зображень Google, щоб отримати ключові слова, пов'язані з логотипом сайту. Вони можуть отримати істинну позитивну частоту 95,8% і нульову хибнопозитивну частоту, використовуючи набір даних із 400 фішингових сторінок і 50 легітимних сторінок. Недоліком цього методу є те, що ефективність виявлення в основному залежить від здатності розпізнавання зображень пошукової системи та якості сегментації зображення логотипу сайту. У своєму експерименті вони показали, що неправильна сегментація логотипу значно знижує точність виявлення до 40%.

1.3.8 Подібність вбудованих посилань

Іншим напрямком пошуку схожості між фішинговими атаками та цільовими сайтами є перевірка гіперпосилань, вбудованих на сторінку.

В [27] пропонують метод на основі графіків із використанням вбудованих посилань для ідентифікації мети фішингу. Враховуючи підозрілу сторінку, вони спочатку створюють набір пов'язаних сторінок під назвою «паразитична спільнота», яка складається з усіх сторінок, на які переходять вбудовані посилання (прямі посилання), і сторінок з подібними ключовими словами, отриманих шляхом запиту пошукової системи (непрямі посилання). Потім відправляється сканер для встановлення зв'язків для паразитарної спільноти. Зокрема, сканер перевіряє вбудовані посилання кожної сторінки у паразитарній спільноті та додає направлене ребро, якщо сторінка переходить з іншої сторінки через вбудоване посилання. Нарешті, між підозрілою сторінкою та її паразитичною спільнотою створюється орієнтований граф. Сторінка з

найсильнішим з’єднанням із підозрілою сторінкою визначається як мета фішингу. У їхній роботі 99,67% справжніх позитивних і 0,5% хибнопозитивних зареєстровано в базі даних з 3374 фішингових і 1200 легальних сайтів.

В [28] розглянуто гіперпосилання між фішинговими та цільовими сайтами з іншої точки зору. Замість того, щоб отримати сайт із найсильнішим зв’язком із підозрілою сторінкою, вони дивляться на перетин між прямими та непрямими посиланнями. Потім IP-адреси загальних посилань порівнюються з підозрілою сторінкою. Якщо підозріла сторінка не відповідає жодному IP-адресу, її позначають як фішинг. Вони повідомляють про рівень точності 98,95% і рівень помилкових спрацьовувань у 1,2% у базі даних з 10 005 фішингових і 1000 легальних сайтів.

1.3.9 Виявлення фішингу шляхом перегляду внутрішніх характеристик

У зв’язку з широким використанням методів машинного навчання деякі дослідження намагаються використовувати машинне навчання для виявлення внутрішніх характеристик фішингових атак. Метою цих підходів є навчання бінарного класифікатора шляхом вивчення відношень між ознаками даних та основною правдою (фішинговою чи легітимною). Ефективність прогнозування залежить від двох факторів: вибору функції та вибору моделі.

У таблиці 1.1 наведено підсумок досліджень, які застосовують алгоритми машинного навчання для виявлення фішингових сторінок. Роботи [29] та [30] досягли двох найкращих результатів із застосуванням SVM та байєсівської мережі. Однак це не означає, що ці два алгоритми кращі за інші через використання різних наборів функцій і наборів даних. Порівняння алгоритмів за однакових умов виконується лише в роботах [31, 32]; однак ці два дослідження дають різні висновки. Байєсівський мережевий класифікатор перемагає всі інші класифікатори в своїх експериментах, тоді як в [33] показують, що AdaBoost дає найкращі результати. Можна помітити, що через відсутність загальних наборів даних та наборів функцій важко зробити висновок, який алгоритм машинного навчання найбільш підходить для виявлення фішингових сторінок. Тому

алгоритми слід порівнювати в процесі вибору моделі, але в більшості досліджень відсутній цей необхідний крок.

Таблиця 1.1 – Алгоритми машинного навчання, які використовуються для виявлення фішингових сторінок

Робота	Алгоритм машинного навчання	# функцій	Результат	Набір даних
[14]	Дерево рішень	18	83.09% СП і 0.43% ПП	680 фішингових і 4,149 легальних сторінок
[15]	Всі 6 алгоритмів	15	99.63% СП і 0.41% ПП	8,118 фішингових і 4,883 легальних сторінок
[16]	SVM	15	99.65% СП і 0.42% ПП	1,764 фішингових і 700 легальних сторінок
[17]	ANN	17	92.18% точність	800 фішингових і 600 легальних сторінок
[18]	SVM	7	84% точність	279 фішингових і 100 легальних сторінок
[19]	Всі 6 алгоритмів	9	86.14% СП і 13.83% ПП	1,500 фішингових і 1,500 легальних сторінок

Наприклад, в [34] пропонують шість функцій на основі URL-адрес для виявлення ненормальних URL-адрес: 1) URL-адреса використовує IP-адресу як ім'я хоста. 2) URL-адреса містить назву бренду. Щоб виявити назви брендів в URL-адресах, вони застосовують регулярний вираз для вилучення літер і цифр із доменного імені, а потім порівнюють отриманий рядок зі списком попередньо скомпільованих назв брендів. 3) Кількість крапок в URL-адресах. 4)

Неправильне розташування домену верхнього рівня (ДВР) або назви бренду. Деякі фішингові URL-адреси безпосередньо використовують законне ім’я хосту як субдомен. Отже, у цих URL-адресах використовується більше точок, ніж у законних, і ДВР або назва бренду з’являються в ненормальному місці в імені домену (наприклад, http://netflix.com-securebillingupdate. sign.newandsupport.com). 5) У імені домену з’являються підозрілі символи (наприклад, @ і −). 6) URL містить конфіденційні слова. В [35] Повідомляється, що фішингові атаки, як правило, створюють URL-адреси з використанням обмеженого набору слів. Вони складають список конфіденційних слів, які зазвичай з’являються у фішингових URL-адресах, таких як вхід і реєстрація. Сян та ін. потім використав кількість цих конфіденційних слів у URL-адресі як ідентифікатор фішингової атаки. На додаток до вищезазначених шести функцій, в роботі [36] довжина URL-адреси також пропонується як функція на основі URL-адреси для виявлення фішингу, оскільки фішингові URL-адреси зазвичай довші за законні.

В [37] представлено лексичну особливість фішингових URL-адрес. У цій роботі вони витягують маркер рядка з URL-адреси. Маркер рядка визначається як рядок, чутливий до регістру, що складається з символів A-Z, a-z, 0-9, підкреслення (_) і дефіса (-). Будь-які інші символи використовуються як роздільники маркерів. Потім вони підраховують кількість кожного маркера, який з’являється у фішингових URL-адресах і законних URL-адресах, позначених як X_t і X_n. Оцінка ймовірності фішингу для токена рядка визначається як $\frac{X_t}{X_t+X_n}$. Нарешті, ймовірність того, що URL буде виявлено як фішинг, є середньою ймовірністю всіх маркерів у URL-адресі.

Функції на основі елементів HTML. Інша категорія функцій для виявлення фішингових атак базується на елементах HTML на сторінці. В [38] зроблено висновок, що фішингові атаки, як правило, використовують більш специфічні елементи HTML, такі як форми, вбудовані посилання, елементи введення та Javascript, ніж законні веб-сайти. Вони підраховують кількість цих

елементів, які з’являються в DOM, і використовують частоту кожного елемента як числову ознаку.

В [39] порівняно елементи HTML з Веб-індентичністю, яка є абстракцією права власності на сторінку. Щоб отримати веб-ідентифікацію, вони спочатку витягують слова із вмісту сторінки та ранжирують їх відповідно до частоти їх появи. Як веб-ідентифікатор вибирається слово з найвищою частотою. Потім вони перевіряють, чи відображається веб-ідентифікація в URL-адресі, обробнику форми чи файлі cookie. Бінарні ознаки генеруються з кожного порівняння і в кінцевому підсумку об’єднуються разом як вектор ознак.

В [40] розглянуто характеристики елемента форми на фішинговій сторінці. По-перше, вони виявили, що багато фішингових атак надсилають форми в різні домени. Крім того, фішингові атаки рідко використовують https для шифрування запитів на подачу. Останнє, але не менш важливе, структура елементів фішингової форми дуже проста, вона складається лише з елементів введення та текстів (наприклад, імені користувача та пароля).

В [41] аналізують веб-технології, які використовуються на фішинговій сторінці. Вони повідомляють, що низка атак використовує спливаючі вікна для подання облікових даних. Зловживають кілька перенаправлень або наведення миші, щоб приховати реальні фішингові посилання. Вони також повідомляють, що приблизно в 1,6% фішингових атак вимикається функція клацання правою кнопкою миші, щоб користувачі не могли переглядати DOM фішингових сторінок у браузері.

Веб-функції. Веб-функції – це функції, які збираються з Інтернету, а не з самої сторінки, наприклад, вік домену, рейтинг сторінки та рейтинг домену. Оскільки тривалість фішингових атак дуже коротка, вік домену набагато менший за легітимний, а рейтинг сторінки нижчий. Деякі дослідження також застосовують метод отримання споріднених доменних імен за ключовими словами до підходів на основі машинного навчання [42]. Зокрема, двійкова функція встановлюється на 1, якщо домен, який перевіряється, не відповідає жодній зі сторінок, повернутих пошуковою системою з подібними ключовими словами, інакше вона встановлюється на 0.

Функція на основі безпеки. Оскільки більшість законних сайтів, зокрема веб-сайти електронної комерції, використовують SSL для підвищення безпеки веб-сайту, деякі дослідження [43] показують, що якщо сайт не використовує SSL або сертифікат SSL не виданий довіреною компанією, він є підозрілим.

1.4 Висновки та постановка задачі

У цьому розділі розглянуто питання виявлення фішингових сторінок, виявлення фішингових електронних листів та аналіз фішингових атак.

Виявлено, що наявні фішингові чорні списки мають проблему в тому, що вони не можуть ефективно запобігти раннім атакам і блокують лише випадки атак, а не реальні атаки. Пори те, що певні методи та засоби можуть компенсувати цей недолік, дослідження виявили рад недоліків, зокрема виявлення фішингових атак погано забезпечують доступ до веб-сервісів а ткож набору даних, що унеможливлює для інших відтворити їхні результати та порівняти їх із власним методом.

Таким чином, актуальною є науково-практична задача удосконалення методу виявлення кіберзагроз типу «фішинг» з метою підвищення достовірності та ефективності виявлення атак типу «фішинг».

2 МОДЕЛЬ ПРОЦЕСУ ФУНКЦІОНУВАННЯ ЕМУЛЯТОРА ВИЯВЛЕННЯ АТАК ТИПУ ФІШИНГ

2.1 Моделі фішингових атак

З метою вирішення поставлених завдань необхідним є побудова моделі атак типу фішинг, а також моделі процесу функціонування емулятора виявлення атак типу фішинг.

Основою виявлення факту функціонування атаки типу фішинг є порівняння подібності відомих фішингових атак. Оскільки, більшість нових атак є варіантами відомих, це показує, що виявлення таких реплік є головними кроками вирішення задачі.

Представимо модель атаки типу фішинг з підміною сторінки у вигляді кортежу:

$$M_1 = \langle A_1, S_1, L_1, F_1, U_1, H_1, V_1 \rangle, \tag{2.1}$$

де $A_1 = \{a_j\}$ – множина API функції, що необхідно відслідковувати та збирати фішинговим функціоналом;

$A_t = \{a_{t_j}\}$ - відслідковувана API функція, призначена для перехоплення введених користувачем на веб-сервісі даних засобами клавіатури;

$A_s = \{a_{s_j}\}$ - відслідковувана API функції, призначена для перехоплення введених користувачем на веб-сервісі даних засобами знімку екрану;

$A_l = \{a_{l_j}\}$ - відслідковувана API функції, призначена для перехоплення введених користувачем на веб-сервісі даних засобами, $A_t, A_l, A_s \in A_1$;

$U_1 = \{u_j\}$ – множина API функції для призначена для перехоплення отриманих в результаті запиту з веб-сервісу даних;

$H_1 = \{h_j\}$ – множина API функції для акумулювання отриманих зловмисниками даних користувача;

$V_1 = \{v_j\}$ – множина API функції підміни сторінки зловмисником, V_{em} - засобами команд, отриманих з електронної пошти, V_{FTP} - засобами команд, отриманих з FTP, V_{int} - засобами команд, отриманих з мережі Інтернет, $V_{em}, V_{FTP}, V_{\text{int}} \in V_1$;

S_1 –інфікована сторінки фішигновою атакою;

$L_1 = \{l_j\}_{j=1}^{6}$ – етапи життєвого циклу фішингового ЗПЗ, що здійснює атаки типу фішинг з підміною сторінки;

$F_1 = \{f_j\}$ – множина API функцій фішингового ЗПЗ, що відповідають за реалізацію відповідного етапу життєвого циклу ЗПЗ;

API функція інфікування веб-сервісу $l_1^1 \Rightarrow \text{Y} \xrightarrow{f_1^1} \{c_{\text{in}} | c_{\text{in}} \in C\}$, де Y – множина зловмисних API функцій, призначених для імплементації функціоналу фішингового ЗПЗ, C – множина інфікованих в мережі КС, c_{in} – інфіковані комп'ютерні системи;

API функції, призначені для здійснення відслідковування даних веб-сервісу $l_2^1 \Rightarrow U_1 \xrightarrow{f_2^1} \{a | a \in A_1\}$;

API функція для акумуляції даних сторінок веб-сервісу, що відслідковувались $l_3^1 \Rightarrow H_1 \xrightarrow{f_3^1} \{a | a \in A_1\}$;

функції акумуляції даних щодо веб-сервісу в пам’ять інфікованої КС користувачів $l_4^1 \Rightarrow S_1 \cup \{a | a \in A_1\} \xrightarrow{f_4^1} S_1'$;

API функції, призначені для здійснення команд передачі акумульованих даних щодо інфікованого веб-сервісу засобами мережу Інтернет $l_5^1 \Rightarrow V_1 \xrightarrow{f_5^1} S_1'$;

API функції, призначені для вилучення акумульованих даних щодо інфікованого веб-сервісу з пам’яті КС $l_6^1 \Rightarrow S_1 \setminus S_1' \xrightarrow{f_6^1} S_1$.

Представимо модель фішингової атаки, що здійснює зміну методу ідентифікації у веб-сервісі вигляді кортежу:

$$M_2 = \langle B_2, S_2, L_2, F_2, W_2, Q_2 \rangle, \tag{2.2}$$

де $B_2 = \{b_j\}$ – множина ЗПЗ, яке запускає фішингова атака;

$B_{AV} = \{b_{AV_j}\}$ - модуль виявлення антивірусного ПЗ;

$B_W = \{b_{W_j}\}$ - ЗПЗ типу worm;

$B_C = \{b_{C_j}\}$ - ЗПЗ типу компаньйон;

$B_M = \{b_{M_j}\}$ - ЗПЗ типу модифікатор;

$B_P = \{b_{P_j}\}$ - ЗПЗ типу поліморфне ПЗ;

$B_S = \{b_{S_j}\}$ - ЗПЗ типу script;

$B_{st} = \{b_{st_j}\}$ - ЗПЗ типу stelth, $B_{AV}, B_W, B_C, B_M, B_P, B_S, B_{st} \in B_2$;

$W_2 = \{w_j\}$ – множина механізмів антивиявлення щодо ЗПЗ;

$Q_2 = \{q_j\}$ – множина API функції, призначених для інсталяції та заміну завантажуваних сторінок веб-сервісів;

S_2 – масив пам'яті, виділений в інфікованій КС користувачів;

$L_2 = \{l_j\}_{j=1}^{4}$ – етапи життєвого циклу фішингового ЗПЗ, що здійснює зміну методу ідентифікації у веб-сервісі;

$F_2 = \{f_j\}$ – множина API функцій фішингового ЗПЗ, що реалізують етапи здійснення зміни методу ідентифікації у веб-сервісі;

функція інфікування веб-сервісів у КС $l_1^2 \Rightarrow \mathrm{Y} \xrightarrow{f_1^2} \{c_{\text{in}} | c_{\text{in}} \in C\}$, де Y – множина зловмисних API функції, які реалізовані з метою виконання зміни методу ідентифікації у веб-сервісі;

API функції, призначені для здійснення виокремлення відповідних модулів зміни методу ідентифікації у веб-сервісі у КС користувачів $l_2^2 \Rightarrow Q_2 \xrightarrow{f_2^2} \{b | b \in B_2\}$;

API функції, призначені для здійснення заміну методу ідентифікації у веб-сервісі в КС користувача $l_3^2 \Rightarrow S_2 \cup B_2 \xrightarrow{f_3^2} S_2'$;

API функції, призначені для здійснення оновлення версійності ЗПЗ в інфікованій КС користувача $l_4^2 \Rightarrow B_2 \times B_2' \xrightarrow{f_4^2} B_2'$.

Представимо модель атаки типу фішинг, що здійснює копіювання веб-сервісу у вигляді кортежу:

$$M_3 = \langle D_3, F_3, V, \mu_3, \_, \_, \_ \rangle, \quad (2.3)$$

де $D_3 = \{d_j\}_{j=1}$ – множина файлів операційної системи, що включає в себе містять реєстраційні дані;

$D_l = \{d_j\}$ - збережені логіни користувачів;

$D_p = \{d_j\}$ - збережені паролі користувачів, $D_l, D_p \in D_3$;

$H_3 = \{h_j\}$ – множина API функцій для перенесення сторінок веб-сервісу з користувацькими даними;

$V_3 = \{v_j\}$ – множина API функцій, на які покладено встановлення зв'язку з КС зловмисника, V_{em} - засобами електронних поштових скриньок, V_{FTP} - засобами протоколу передачі даних FTP, V_{int} - засобами мережі Інтернет, $V_{em}, V_{FTP}, V_{\text{int}} \in V_3$;

μ_3 – файли користувачів інфікованої КС;

$L_3 = \{l_j\}_{j=1}^{5}$ – етапи життєвого циклу фішингового ЗПЗ, що здійснює копіювання веб-сервісу;

$F_3 = \{f_j\}$ – множина API функцій фішингового ЗПЗ, що відповідає за реалізацію певної фази життєвого циклу;

API функції, що інфікують КС $l_1^3 \Rightarrow \mathrm{Y} \xrightarrow{f_1^3} \{c_{\text{in}} | c_{\text{in}} \in C\}$, де Y – множина зловмисних API функцій, включених в тіло фішингового ЗПЗ, C – множина

атакованих КС в мережі, c_{in} – інфікована в мережі КС;

функція , призначена для перенесення реєстраційних даних користувача інфікованої КС $H_2^2 \Rightarrow \quad _3 \xrightarrow{f_2^3} d \in \{ D \in \quad _3 \}$;

функція акумуляції даних користувача , що користувався веб-сервісом $l_3^3 \Rightarrow \mu_3 \cup \{ d | d \in D_3 \} \xrightarrow{f_3^3} \mu_3^{'}$;

функція запуску команд відпраки акумульованих даних на КС зловмисників, які здійснили фішинг атаку $l_4^3 \Rightarrow V_3 \xrightarrow{f_4^3} \mu_3^{'}$;

API функція знищення акумульованих даних щодо користувачів веб-сервісу $l_5^3 \Rightarrow \mu_3 \setminus \mu_3^{'} \xrightarrow{f_5^3} \mu_3$.

2.2 Виявлення фішингових копій і варіацій

Розглянемо математичний апарат, який буде застосовано для здійснення процесу виявлення фішингових атак.

2.2.1 Відбиток фішингової сторінки: вектор тегів

Веб-сторінка відображається як деревовидна структура даних, яка називається DOM.

Таким чином, легко виміряти схожість сторінок, порівнюючи DOM.

Однією з поширених метрик DOM є відстань редагування дерева (TED), яка представляє мінімальну вартість операції, яка перетворює один DOM в інший.

Однак порівняння на основі TED є витратним у часі, а «сучасний» алгоритм AP-TED [44] має часову складність O(n3) і складність пам’яті O(m∗n), де n і m є розміри двох DOM, які порівнюються.

Для вирішення задачі можливим є спрощене представлення сторінок, яке легше порівнювати.

Відомо, що використання методу «виявлення майже дублікатів фішингу на основі хешування», який у їхньому випадку виявляє до 72,67% їх бази даних фішингових сайтів є репліками принаймні з одного іншого сайту в тому самому бази даних.

Цей метод «на основі хешу» бере HTML фішингової сторінки та видаляє з нього всі пробіли. Він також видаляє всі значення за замовчуванням у кожному полі INPUT, замінюючи ці значення порожніми рядками.

Отриманий HTML потім хешується за допомогою SHA-1.

Також модливим є застосування засобу SimHash [45] для виявлення дублікатів сторінок для сканування веб-сторінок, де можливим є перетворення кожної веб-сторінки в набір функцій, позначених як вектор ознак, а потім перетворюють вектор функцій у 64-бітний SimHash.

Відстань Хеммінга між двома SimHash використовується для оцінки подібності двох сторінок (чим менше значення, тим більше схожі сторінки).

Однак жоден із перерахованих вище методів не стосується порівняння подібності фішингових атак.

Метод на основі хешування дуже суворий і не відповідає сторінкам, які мають дуже невеликі варіації одна від одної.

Метод на основі SimHash працює лише на сторінці з великою кількістю тексту, але фішингові сторінки зазвичай містять дуже мало тексту або навіть зовсім не містять (наприклад, мають лише зображення). Тому потрібне представлення сторінки, яке здатне швидко порівняти подібність DOM.

В даному дослідженні пропонується здійснювати обчислення засобами залучення векторів тегів фішингової сторінки.

Спочатку здійснюється визначення множини тегів HTML.

Тому в роботі використовується повний набір елементів HTML, наданий Консорціумом1 World Wide Web, з якого було видалено деякі з найбільш поширених тегів, такі як <body>, <head> і <html>.

У підсумку отримується список із 107 тегів, які можна отримати з веб-сайту2.

Потім здійснюється призначення довільного, але фіксованого порядку тегів у корпусі та визначаємо «вектор» розміру корпусу.

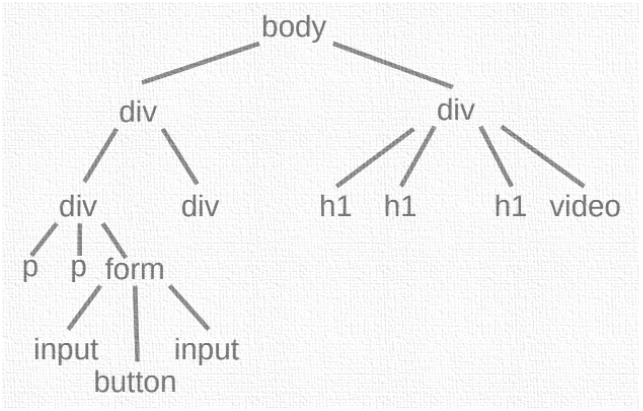
Для кожної DOM здійснюється обчислення відповідного вектору, підраховуючи, скільки разів кожен тег корпусу з’являється в DOM.

Як приклад розглянемо рисунок 2.1, де представлено два простих DOM. Якщо корпус складається з html тегів

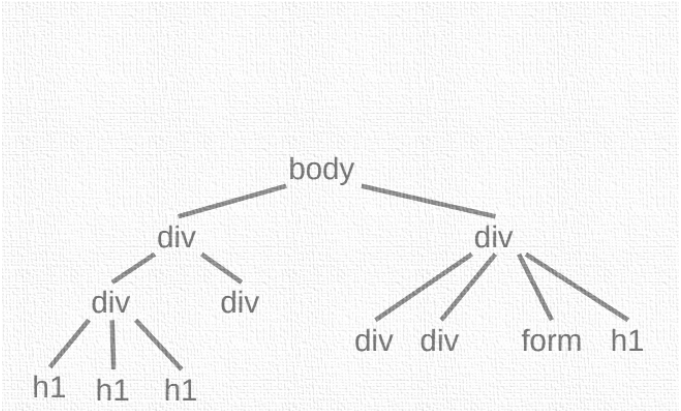
<form> <b> <p> <h1> <button> <video> <input> <iframe> і ,

у такому порядку, потім вектор тегів для сторінки p1 дорівнює

<1, 0 ,2 ,3, 1, 1, 2, 0, 4>.



а)



b)

Рисунок 2.1 - Вектори тегів вектор тегів для сторінки p2 дорівнює <1, 0 ,0 ,4, 0, 0, 0, 0, 6>:

(a) DOM сторінки p_1

(b) DOM сторінки p_2

2.2.2 Моделювання схожості фішингових сторінок: пропорційна відстань

Існують дві загальні метрики вимірювання подібності векторів: евклідова відстань і косинусна подібність, але ці показники не застосовуються до векторів тегів.

Проблема з евклідовою відстанню полягає в тому, що вона не нормалізована і має тенденцію давати більші відстані для векторів з більшою кількістю тегів.

Таким чином, зловмисники можуть легко збільшити відстань, додавши багато фіктивних тегів (наприклад,), не впливаючи на зовнішній вигляд сторінки.

Косинусна подібність оцінює кут між векторами, незалежно від масштабу значень, що на практиці спричинить багато помилкових спрацьовувань.

Щоб подолати наведені вище обмеження, використовуємо те, що називається пропорційною відстанню, щоб виміряти подібність між векторами тегів.

Необхідна зауважити, що інші показники, такі як нормалізована квадратна евклідова відстань, могли бути використані, ймовірно, з порівнянними результатами.

В роботі застосовано пропорційну відстань, оскільки вона швидко обчислюється і дає хороші результати.

Пропорційна відстань - це проста метрика різниці, в якій здійснюється підрахунок кількості елементів корпусу, які з'являються різну кількість разів на обох сторінках (іншими словами, кількість індексів векторів тегів, які мають різне значення).

Цей простий підрахунок матиме тенденцію генерувати більші відмінності для сторінок, які є відносно складними і які використовують велику різноманітність тегів із корпусу, тоді як сторінки, які використовують лише кілька тегів, матимуть менші відмінності.

Отже, ділимо початкове число на кількість тегів корпусу, які з'являються хоча б на одній із двох сторінок.

Нехай k1 і k2 — два ненульові вектори тегів над одним корпусом розміру n.

Формально необхідно здійснити визначення пропорційної відстані P (k1, k2) між k1 і k2 наступним чином:

$$P\left(k_1\ i\ k_2\right) = \frac{\sum_{i=1}^{n} A(k_1[i], k_2[i])}{\sum_{i=1}^{n} B(k_1[i], k_2[i])} \qquad (2.1)$$

де $A(x, y) = \{1$ якщо $x \neq y$ 0 інакше та

$B(x, y) = \{1$ якщо $x \neq 0$ АБО $y \neq 0$ 0 інакше

Як приклад розглянемо два вектори з рисунку 2.1: сім із дев'яти можливих тегів корпусу використовуються принаймні одним із двох векторів, і лише перший, <form>, з'являється однакову кількість разів в обох векторів, тому P (k1, k2) = 6/7 = 0,85, що вказує на велику різницю між двома сторінками.

2.3 Алгоритм кластеризації

Мета кластеризації - згрупувати в одному кластері досліджувані сторінки, які мають відносно подібні вектори тегів, оскільки вони вважаються схожими одна на одну.

Таким чином, даний сайт можна дещо модифікувати велику кількість разів і повторно публікувати після кожної зміни.

Як висновок, ймовірно, можна розглядати довгі «рядки» модифікацій, у яких кожен новий екземпляр досить схожий на екземпляр, на якому він заснований, але через деякий час отримуємо екземпляри, які досить віддалені один від одного.

Таким чином, буде спостерігатися уникнення використання алгоритму кластеризації на основі центроїда, такого як t-середніх.

Замість цього використовується ієрархічний підхід до кластеризації для групування векторів, які мають принаймні один інший вектор, відносно подібний у кластері.

Це має перевагу в тому, що завжди виробляється однаковий вихід для того самого входу, незалежно від порядку, в якому вектори додаються до моделі.

Ці нові сайти можна додавати в модель жадібним способом, без необхідності перераховувати модель.

Необхідно зауважити, що коли до набору додається новий сайт, два кластери можуть бути об’єднані разом.

У роботі було використано простий жадібний алгоритм із загальною складністю O(n2) для n векторів, як показано в Алгоритмі 2.1.

Алгоритм спочатку обчислює пропорційну відстань кожної пари векторів тегів і створює набір ребер (рядки 5-7).

Потім набір ребер сортується в порядку зростання відстані (рядок 8). Нарешті, він починає об’єднувати вектори від найменшої відстані до більшої, поки не буде досягнуто заданий поріг (рядки 9-13).

Алгоритм 2.1 Алгоритм кластеризації

Вхідні дані: набір векторів тегів, які групуються $K = \{k_1, k_2, ..., k_n\}$ і межа H

Вихідні дані: набір кластерів заданих векторів тегів Cl

1: procedure Clustering(*Vector K* , *Double H*)

2: *ES* ← *{}*

3: *n* ← size of *K*

4: $Cl \leftarrow \{\{ k_1\}, \{ k_2\}, ..., \{ k_n\}\}$ *l*> Спочатку кожен вектор утворює кластер

5: for $i \leftarrow 0$ to $n - 2$ do *l*> Ініціалізація інформації ребра

6: for $j \leftarrow i + 1$ to $n - 1$ do

7: Add [*ki*, *kj* , *P* (*ki*, *kj*)] into *ES*

8: *SES* ← Sort(*ES*) *l*> Сортування набору ребер у порядку зростання відстані

9: for *ki, kj, p* ∈ *SES* do *l*> Об'єднення кластерів, поки не досягнеться межа

10: if $p > H$ then

11: break

12: if isSameCluster(*ki, kj*) = *false* then

13: MergeCluster(ki, kj , Cl)

return Cl

2.3.1 Поєднання кластеризації та оптимального порога

Як зазвичай з алгоритмами кластеризації, метод заснований на деякій межі H.

Загальне та інтуїтивне визначення межі кластеризації - це межа, яка дає кластери, які є компактними і далекими один від одного.

Метою даного процесу є визначення зв'язку кластеризації, використовуючи середню пропорційну відстань векторів всередині кластера.

Далі виконується усереднення цього значення і ділення його на найменшу пропорційну відстань між двома векторами в будь-яких двох досліджуваних кластерах.

Чисельник оцінює компактність кластерів (чим компактніші кластери, тим менше значення), а знаменник оцінює, наскільки віддалені кластери: чим далі один від одного кластери, тим більше значення.

Таким чином, менше значення для сполучення кластеризації краще.

Формально, задавши Clt ⊆ Cl підмножину набору кластерів Cl, що має більше одного елемента, визначається зв'язок кластеризації таким чином: для

даного кластера Cli ∈ Clt, що має ni > 1 елементів, визначаємо компактність множини кластери Clt наступним чином:

$$Comp\left(Cl_t\right) = \frac{1}{\left|Cl_t\right|} \sum_{Cl_i \in Cl_t} (InterComp(Cl_i)) \quad (2.2)$$

де $InterComp\left(Cl_i\right) = \frac{2}{n_i\left(n_i - 1\right)}$,

$$\sum_{x \in Cl_i, y \in Cl_i, x \neq y} P(x, y)$$

Потім необхідно здійснити визначення мінімальної відстані у наборі кластерів Cl:

$$Min(Cl) = min_{Cl_i, Cl_j \in Cl, Cl_i \neq Cl_j} \quad InterMin(Cl_i, Cl_j), \quad (2.3)$$

де $InterMin\left(Cl_i, Cl_j\right) = min_{x \in Cl_i, x \in Cl_j} P(x, y)$.

Нарешті, зв'язок кластеризації Cl визначається як:

$$CCl(Cl) = \frac{Comp(Cl_t)}{Min(Cl)} \quad (2.4)$$

Щоб вибрати оптимальну межу кластеризації для заданого набору векторів, здійснюється розрахунок зв'язку кластеризації з різними межами. Оптимальною вважається та, яка мінімізує зв'язок кластеризації.

2.3.2 Модель процесу виявлення фішингу

Щоб визначити, чи є новий вектор тегів фішинговим варіантом, вектор порівнюється з усіма відомими атаками в фішинговій базі даних.

Якщо відстань менша за оптимальний поріг, це вважається фішинговою атакою.

Проблема цього підходу полягає в тому, що його непросто масштабувати, оскільки в міру додавання в базу даних все більше і більше екземплярів фішингу час порівняння буде лінійно збільшуватися.

Щоб вирішити цю проблему, можливим є використання переваг результату кластеризації, щоб визначити межу кластерів.

Враховуючи кластер векторів Cl, можна визначити новий вектор pc, який називається прототипом кластера, де кожен елемент являє собою набір значень, що з'являються в цій позиції.

Наприклад, розглянемо кластер із трьох векторів

<<0,1,34,2,4>,
<2,1,36,3,4>,
<2,1,37,3,4>>,

його прототип є

<<0,2>, <1>,<34, 36, 37>, <2,3> ,<4>>.

Щоб порівняти вектор і прототип, здійснимо перевизначення P у формулі наступним чином:

$$P(k,\ pc\ ) = \left[\frac{\sum_{i=1}^{n} minA(k[i],pc[i])}{\sum_{i=1}^{n} maxB(k[i],pc[i])},\ \frac{\sum_{i=1}^{n} maxA(k[i],pc[i])}{\sum_{i=1}^{n} minB(k[i],pc[i])} \right] (2.5)$$

де $minA(e, E) = \{1$ якщо $e \notin E$ 0 інакше та max
$A(e, E) = \{1$ якщо $E \backslash e \neq \emptyset$ 0 інакше ,

$minB(e,E) = \{1$ якщо $\exists_z \in (E \cup e)\ z \neq 0\ 0$ інакше та $maxB(e,E) = \{1$ якщо $(E \cup e) \backslash 0 \neq \emptyset\ 0$

Тепер є можливим використовувати прототип кластера для визначення меж кластера.

Спочатку обчислюємо діапазон значень A і B.

Розглянемо, що в конкретній позиції i, якщо значення (k[i]) у векторі k не з'являється в наборі значень (pc[i]) прототипу кластера, мінімум A, minA, дорівнює 1, інакше він дорівнює 0.

Навпаки, якщо pc[i] містить будь-яке значення, відмінне від k[i], максимум A дорівнює 1.

Аналогічно, мінімум B, minB, дорівнює 1, якщо в об'єднанні pc[i] і k[i] існує ненульове значення; інакше це 0.

Максимум (maxB) B дорівнює 1, якщо хоча б один із pc[i] і k[i] не містить 0.

Нарешті, отримано можливість підсумувати значення для всіх позицій і отримаємо кінцевий діапазон значень Pproto.

Якщо мінімум Pproto більший за поріг, вектор повинен бути поза кластером; якщо максимум Pproto менший за поріг, вектор безперечно знаходиться в кластері.

Наступним роком є необхідність порівняння коженого вектора у кластері лише в тому випадку, якщо він не належить до двох наведених вище випадків. Повний алгоритм показаний в Алгоритмі 2.2.

Алгоритм 2.2 – Алгоритм виявлення варіацій

Вхідні дані: набір кластерів відомих фішингових атак Cl, оптимальна межа H і вектор, що перевіряється k

Вихід: чи є вектор різновидом відомих фішингових кластерів

1: procedure isVariation(ClusterSet Cl, Double H, Vector k)

2: PC ← fetchProtoType(Cl)

3: for pci ∈ PC do

4: minA, maxA, minB, maxB ← 0

5: for ej ∈ pci do

6: if elem(t, j) ∈/ ej then

7: minA++, maxA++

Отримати прототип кластера

Перевірте кожен кластер

Перевірте кожен елемент (тег) прототипу

Оновити minA та maxA

8: else if len(ej) /= 1 then

9: maxA++

10: if 0 ∈/ ej or elem(t, j) /= 0 then

11: minB++, maxB++

Оновити minB та maxB

12: else if len(ej) /= 1 then

13: maxB++

14: minP ← minA/maxB

15: maxP ← maxA/minB

16: if minP > H then return false

17: else if maxP ≤ H then return true

18: else if isInCluster(t, Cli) = true then return true

return false

2.4 Удосконалення процесу виявлення шляхом покращення процедури здійснення кластеризації виявлення фішингової атаки

2.4.1 Зважена пропорційна різниця

При застосування вищеописаних моделей опрацювання даних було виявлено, що пропорційна відстань P, може не підкреслювати «кількість»

відмінностей між кожним тегом HTML, а просто фокусується на тому, чи однакова кількість тегів.

Наприклад, вектор

$$k1 = \{1, 2, 5, 6\} \text{ і } k2 = \{109, 2, 5, 6\}$$

мають однакову відстань до вектора

$$k3 = \{2, 2, 5, 6\},$$

тобто , P(k1, k3) = P(k2, k3).

Для дослідження необхідно врахувати той факт, що k2 більше відрізняється від k3, ніж k1.

Тому необхідним є визначення нової відстані, так званої зваженою пропорційної різниці (WPD), щоб порівняти подібність векторів тегів.

Замість того, щоб використовувати відстань Хеммінга як чисельник, було використано суму зважених різниць (WD), визначену за такою формулою:

$$WD\left(k_1, k_2\right) = \sum_{i=1}^{n} \frac{|k_1[i] - k_2[i]|}{max(k_1[i], k_2[i])}. \quad (2.6)$$

У P значення A для даного тега є булевим (0 або 1).

Для тегів, які використовуються в обох векторах, WD знаходиться в діапазоні [0, 1).

Чим більше різниця між кількістю тегів, тим більше WD.

Тоді виконаємо визначення значення параметра S наступним чином:

$$S\left(k_1, k_2\right) = \sum_{i=1}^{n} EQU(k_1[i], k_2[i]) \quad (2.7)$$

де $EQU\left(k_1[i], k_2[i]\right) = \{1$ якщо $k_1[i] = k_2[i],\ k_1[i] \neq 0\ \ 0$ інакше .

Нарешті, зважена пропорційна різниця (WPD) визначається наступним чином:

$$WPD\left(k_1, k_2\right) = \frac{WD\left(k_1,k_2\right)}{WD\left(k_1,k_2\right)+S\left(k_1,k_2\right)} \quad (2.8)$$

Необхідно зауважити, що зважена пропорційна різниця може ускладнити для зловмисників збільшення схожості.

Зловмисникам потрібно внести більше змін для кількох тегів, щоб уникнути виявлення як варіант відомих атак.

2.4.2 Алгоритм виявлення фішингових варіацій для WPD

Хоча зважена пропорційна різниця WPD є покращеною версією пропорційної відстані P, існує ситуація, коли стає неможливим безпосередньо використовувати алгоритм виявлення, оскільки визначення чисельника та знаменника на відстані було повністю змінено.

В такому випадку можливим є використання прототипу кластера, щоб знайти межу.

Розглядаючи визначення WPD для вирішення вказаної проблеми, можна перетворити його наступним чином:

$$WPD\left(k_1, k_2\right) = \frac{WD\left(k_1,k_2\right)}{WD\left(k_1,k_2\right)+S\left(k_1,k_2\right)} = \frac{1}{1+\frac{S\left(k_1,k_2\right)}{WD\left(k_1,k_2\right)}} \quad (2.9)$$

Так само, якщо відомо мінімум і максимум S і WD, то можна отримати діапазон значень WPD(k, pc):

$$\left[\frac{1}{1+\frac{maxS(k,pc)}{minWD(k,pc)}}, \frac{1}{1+\frac{minS(k,pc)}{maxWD(k,pc)}}\right],$$

де k — вектор, що перевіряється,

pc — прототип фішингового кластера.

Щоб обчислити minS(k, pc), maxS(k, pc), minWD(k, pc) і maxWD(k, pc), знаходимо відповідний мінімум і максимум у кожній позиції та, нарешті, складаємо їх.

Формально,

$$WPD_{proto}(k,\ pc) \in \left[\frac{1}{1+\frac{\sum_{i=1}^{n} maxS(k[i],pc[i])}{\sum_{i=1}^{n} minWD(k[i],pc[i])}}, \frac{1}{1+\frac{\sum_{i=1}^{n} minS(k[i],pc[i])}{\sum_{i=1}^{n} maxWD(k[i],pc[i])}} \right]$$

де

$$inWPD(e,E) = min(\frac{|e-r|}{(e,r)} | \forall r \in E),$$

$$maxWPD(e,E) = (\forall r \in E) \qquad (2.10)$$

$minS(e,E) = \{1$ якщо $e \neq 0$ ТА $E = \{e\}$ 0 інакше $maxS$

$(e,E) = \{1$ якщо $e \neq 0$ ТА $e \in E$ 0 інакше

Також необхідно зауважити, що для обчислення діапазону значень WD(e, E), потрібно знати всі значення в кожній позиції в прототипі.

З цією метою можна вносити наступні зміни в прототип.

Для цього включаємо всі значення та сортуємо їх у порядку зростання, відзначаючи як

$E = \{e_1, e2, ..., e_n\}$ де $e_1 <= e_2 <= ... <= e_n$.

Максимум WD(e, E) є найбільшим із

$$\frac{|e-e_1|}{(e,e_1)} \text{ та } \frac{|e-e_n|}{(e,e_n)}$$

тоді як мінімум є найменшим із

$$\frac{|e-e_l|}{(e,e_l)} \text{ та } \frac{|e-e_r|}{(e,e_r)}$$

де e_l і e_r — дві найближчі до e, що можна зробити, застосувавши двійковий пошук зі складністю O(log(n)).

Для S(e, E), якщо набір значень E має лише ненульовий елемент e, його мінімальний minS(e, E) дорівнює 1.

Алгоритм 2.3 – Алгоритм виявлення варіацій для WPD

Вхідні дані: набір кластерів відомих фішингових атак Cl, оптимальна межа H і вектор, що перевіряється k

Вихід: чи є вектор різновидом відомих фішингових кластерів

1: procedure isVariation(ClusterSet Cl, Double H, Vector k)
2: PT ← fetchProtoType(Cl) l>Отримати прототип кластера
3: for pci ∈ PT do l> Перевірити кожен кластер
4: minWD, maxWD, minS, maxS ← 0
5: for ej ∈ pci do l> Перевірити кожен елемент (тег) прототипу
6: minWDj ← 1
7: maxWDj ← 0
8: for r ∈ ej do l> Оновити minWD та maxWD
9: $tmp \leftarrow \frac{|elem(k,j)-r|}{max(elem(k,j),r)}$
10: if tmp > maxWDj then
11: maxWDj ← tmp
12: if tmp < minWDj then
13: minWDj ← tmp
14: minWD+ = minWDj , maxWD+ = maxWDj
15: if elem(k,j) ∈ ej and elem(k, j) /= 0 then l> Оновити minS та maxS
16: maxS++
17: if len(ej) = 1 then
18: minS++
19: minWPD ← $\frac{1}{1+\frac{maxS}{minWD}}$
20: maxWPD ← $\frac{1}{1+\frac{maxS}{minWD}}$
21: if minWPD > H then return false
22: else if maxWPD ≤ H then return true
23: else if isInCluster(t, Ci) = true then return true
return false

Його максимальний max(e, E) дорівнює 1, коли ненульовий елемент e з’являється в E.

Повний алгоритм, який використовується для виявлення фішингових варіацій для WPD, показаний вище.

2.4.3 З'єднання внутрішньокластерних векторів

Існують принаймні дві поширені моделі, які широко використовуються, коли справа доходить до внутрішньокластерних з’єднань:

1. Одинарне зв’язування, де кожен вузол всередині кластера під’єднаний щонайменше до одного батьківського, створюючи мінімальне охоплююче дерево над елементами кластера.
2. Повне зв'язування, де створюється повний граф між усіма елементами кластерів.
3. Однак виявлено, що жодна з цих двох моделей не може точно відобразити еволюцію векторів фішингу всередині кластера.

Хороша модель повинна зберігати зв’язок між елементами серії модифікацій, зроблених для даної атаки.

Деякі з цих елементів можуть виявитися досить віддаленими один від одного після довгої серії модифікацій.

Також вона також повинна відображати той факт, що деякі з елементів знаходяться на дуже невеликій відстані один від одного всередині кластера.

Ця ідея проілюстрована на рисунку 2.3.

Вектори a, b, c і d близькі один до одного, а це означає, що між цими чотирма векторами є незначні відмінності.

З іншого боку, вектор e, незважаючи на те, що все ще є частиною того самого кластера, насправді відносно «далекий» від перших чотирьох векторів і пов’язаний з ними лише через довгу серію невеликих варіацій.

Щоб зафіксувати ці серії модифікацій, внесених до фішингових атак всередині кластера, пропонується використовувати модель напівповного зв’язку (SCL).

Зокрема, для будь-якої пари векторів тегів k_i та k_j в одному кластері, де i ≠ j, маємо ребро E (k_i, k_j) ∈ SCL тоді й тільки тоді, коли WPD(k_1, k_2) ≤ OPT , де OPT є оптимальним порогом для кластерів векторів тегів.

Простий спосіб побачити цю модель полягає в тому, що всередині кластера "подібні" вектори пов'язані між собою.

Ця модель є проміжною моделлю між опорним деревом і повний графік.

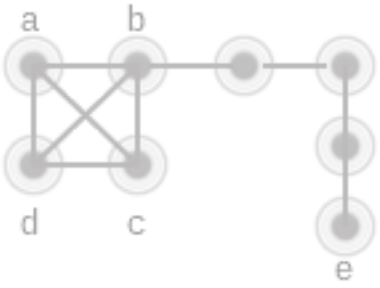


Рисунок 2.3 – Ілюстрація графіка напівповного зв’язку

2.4.4 Зв'язок напівповної кластеризації зв'язків

Розглянемо процес визначення зв’язку кластеризації для напівповного графа зв’язків.

Необхідно визначити Min(Cl_i, Cl_j) як мінімальну відстань між двома кластерами, яка визначається як мінімальна відстань, яку можна знайти між двома векторами, одним у Cl_i і одним у Cl_j.

Тобто:

$$Min\left(Cl_i, Cl_j\right) = \left(\left\{\forall x \in Cl_i, \forall y \in Cl_j\right\}\right). \qquad (2.11)$$

Оскільки в роботі було використано модель SCL для захоплення з'єднань всередині кластерів векторів тегів.

Таким чином, можна визначити зв’язок кластеризації SCL за допомогою наступної формули, яка обчислює середню відстань всередині SCL і ділить її на відстань між кластерами.

Таким чином, включаємо лише ті кластери, які мають більше одного елемента.

$$\frac{\frac{1}{t}\sum_{i=1}^{t}\frac{1}{|E_i|}\sum_{j=1}^{|E_i|}\frac{1}{|E_i|}\left\{E_j(x,y)\in SCL_i\right\}}{\{i\neq j, 1\leq i,j\leq t\}} \qquad (2.12)$$

де t - кількість кластерів, що мають більше одного елемента, E (x, y) - це ребро між x і y в графі SCL, Cl_i - і-тий кластер з більш ніж одним елементом, SCL_i - це SCL для Cl_i і $|E_i|$ – кількість ребер у SCL_i.

2.5 Висновок

В даному розділі подано основні аспекти моделі процесу функціонування емулятора виявлення атак типу фішинг, зокрема представлені моделі фішингових атак, подано процес виявлення фішингових копій і варіацій, спроєктовано відбиток фішингової сторінки: вектор тегів.

Також в розділі проведено моделювання схожості фішингових сторінок: пропорційна відстань, наведено алгоритм кластеризації.

Розроблені моделі стануть основою удосконаленого методу виявлення кіберзагроз типу «фішинг».

3 УДОСКОНАЛЕНИЙ МЕТОД ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

3.1 Основи удосконаленого методу виявлення кіберзагроз типу «фішинг»

На основі описаних моделей в другому розділі у дослідженні пропонується удосконалений метод виявлення кіберзагроз типу «фішинг», який розглядає деталі модифікацій та їх еволюцію з часом.

Метод ґрунтується на основі графіка, який використовується для відстеження та аналізу фішингових модифікацій та розвитку.

Відомо, що більшість фішингових атак повторно запускаються після деяких модифікацій.

Щоб проаналізувати розвиток фішингових атак, було побудовано модель SCL для кожного векторного кластера тегів.

Наступним кроком є додавання додаткової інформації, яка використовується для аналізу фішингових модифікацій, і складається графік модифікації фішингової атаки.

Кожен вузол представляє унікальний вектор тегів, а мітка вузла показує кількість екземплярів фішингової атаки, що використовують вказаний для опрацювання вектор.

Спрямоване ребро E(x, y) фіксує еволюцію від вектора x до вектора y, тобто модифікацію відповідної атаки, яка перетворює початкову атаку (яка має вектор x) у дещо іншу атаку (яка має вектор y).

Текст на краю містить деталі змін.

Наприклад, ребро з міткою «div:+2, input:-3» слід інтерпретувати як означаючи, що два теги div були додані до атаки, а три теги введення було видалено під час створення нового варіанту атаки.

Напрямок краю визначається датою звіту двох пов'язаних векторів; край перетікає від попередньої атаки до пізнішої.

Оскільки один вектор може мати кілька екземплярів атаки, вважаємо, що «повідомлена дата» вектора – це дата, коли сталася перша атака, яка викликала цей вектор.

Як наслідок цього визначення, вихідний вузол графіка, тобто вузол, який має нульовий ступінь, є найбільш раннім зареєстрованим екземпляром атаки в джерелі даних із цієї серії модифікацій.

Виокремимо ці вузли (рис. 3.1.).

Вузол, про варіанти атак які повідомлялося раніше, мають позитивний градус і показані синім кольором на графіку.

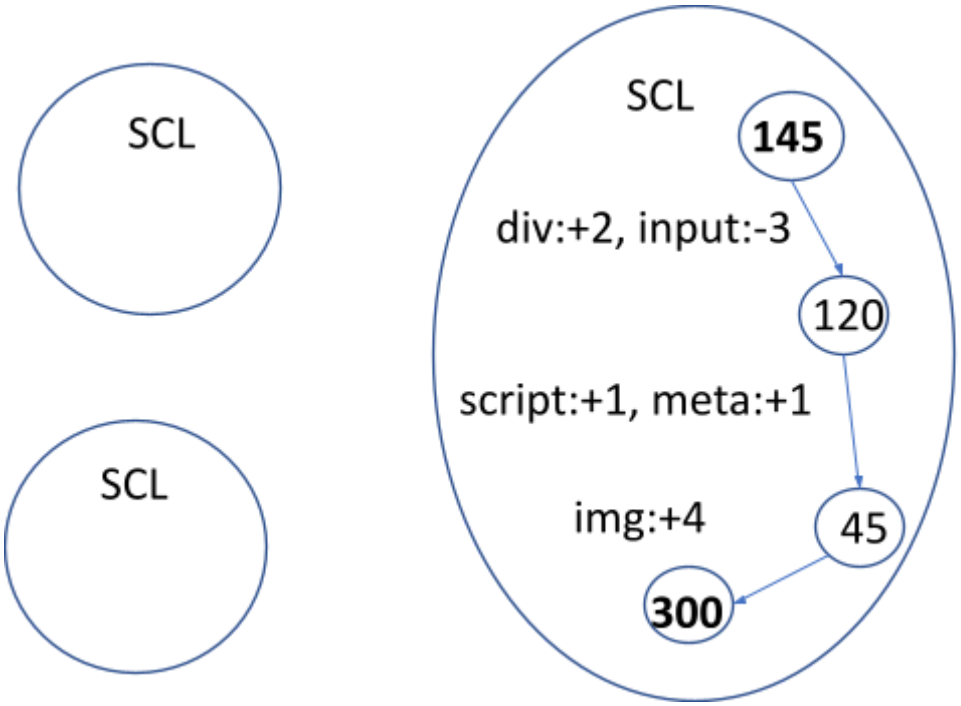


Рисунок 3.1 - Приклад модифікації фішингової атаки

3.2 Побудова вектору ознак

Орієнтація ребер у графі модифікації фішингової атаки визначається датою звіту, коли DOM створюють вектори тегів, від попереднього до наступного.

Вважатимемо вектор тегів нульового ступеня в графі модифікації фішингової атаки головним вектором.

Головний вектор представляє одну з початкових версій атаки в базі даних.

Звичайно, кожен кластер містить принаймні один головний вектор (найбільш ранній вектор у цьому кластері), але він може мати кілька, якщо відстань між векторами занадто велика, щоб їх можна було з’єднати в графі модифікації фішингової атаки.

Наявність кількох головних векторів у кластері означає, що деякі атаки були модифіковані одночасно, або не вистачало проміжних кроків у базі даних.

Кожен неголовний вектор може бути досягнутий принаймні з одного з головних векторів у кластері.

Ці головні вектори дають уявлення про початкові атаки, а неголовні вектори дають уявлення про те, як вони еволюціонували з часом.

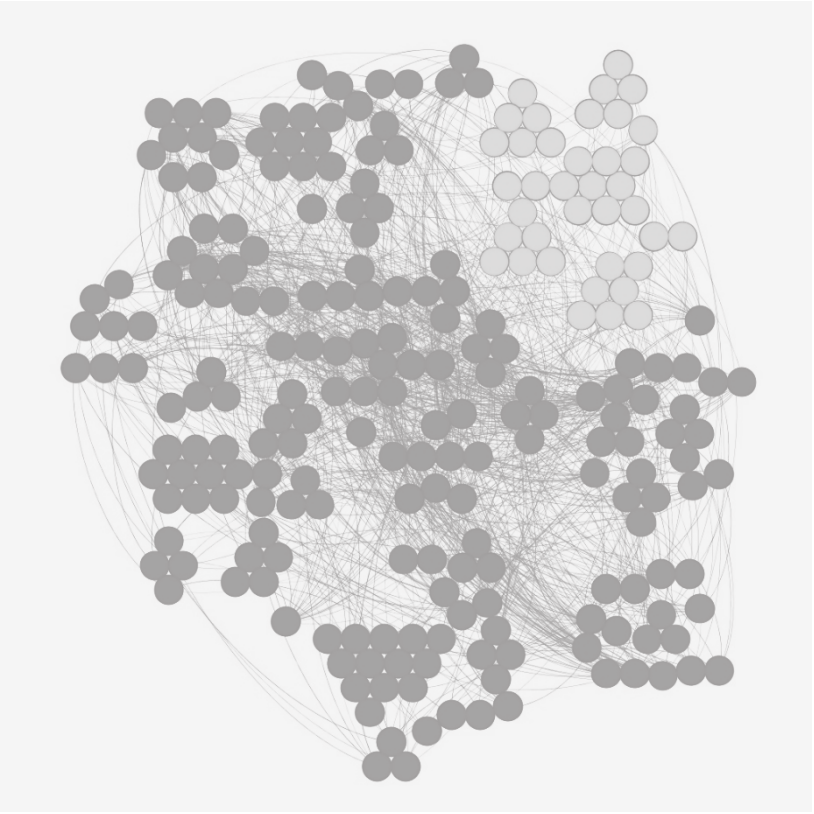
На рисунку 3.2 показано модифікації фішингових атак двох найбільших кластерів у базі даних (головні вектори зеленим, неголовні вектори синім кольором).

Видно, що головних векторів набагато менше, ніж неголовних, що вказує на те, що більшість атак у цих кластерах розвинулися з вихідних векторів.

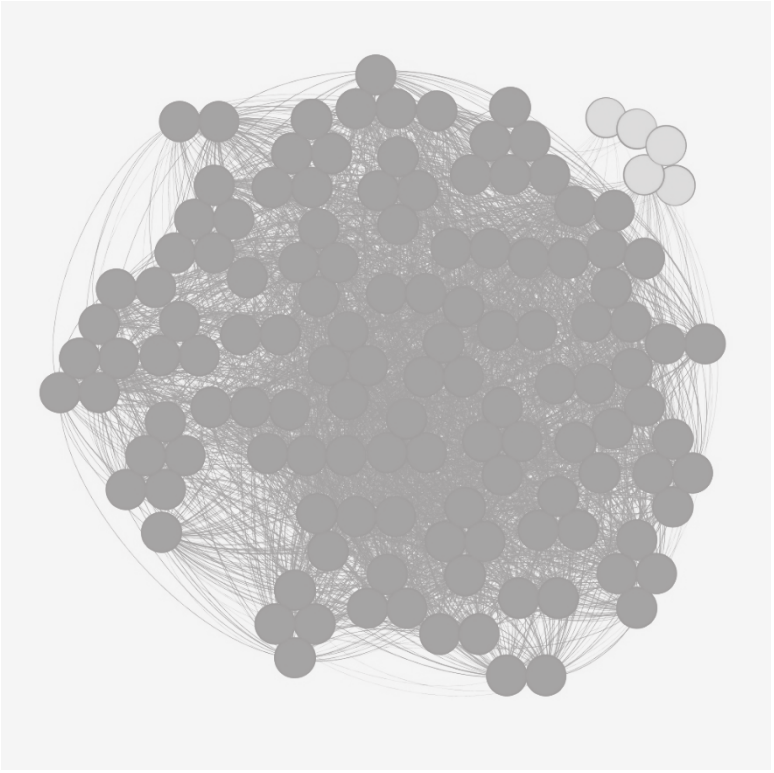
3.3 Побудова змінених тегів

Як пояснювалося раніше, кожен неголовний вектор v має принаймні один спрямований шлях у графі модифікації фішингової атаки від головного вектора до v.

Визначаємо шлях еволюції v (EP_v) як спрямований шлях від головного вектора до v для при якому сума зважених пропорційних відстаней ребер уздовж шляху мінімальна.



а)



б)

Рисунок 3.2: Графіки модифікації фішингових атак двох найбільших кластерів:

(a) Граф модифікації кластера 0

(b) Граф модифікації кластера 1

Іншими словами, EP_v – це спрямований шлях від одного з головних векторів до v, для якого величина перетворення є найменшою.

Для неголовного вектора v та його шляху еволюції $EP_v = [k0, k1, \ldots, kt-1, kt = v]$, що складається з t векторів, маємо такі визначення:

1. Відстань шляху ($PathDis_v$) — це сума зважених пропорційних відстаней ребер уздовж шляху еволюції EP_v. Він являє собою оцінку «розміру» різниці між v та його головним вектором.

$$PathDis_v = \sum_{i=0}^{t-1} \left(WPD\left(k_i, k_{i+1}\right) \right). \quad (3.1)$$

2. Відстань еволюції (ED_v) — це середньозважена пропорційна відстань ребер вздовж шляху еволюції EP_v. Він являє собою середню «кільку» різниці в кожній модифікації. Формально,

$$ED_v = \frac{WPD_V}{t}. \quad (3.2)$$

3. Тривалість життя варіації (VL_v) – це різниця у часі між датою звіту v та звітною датою його головного вектора. Він представляє повний період часу, протягом якого ця атака була активно змінена. Якщо $K_{report}(k_i)$ є звітною датою вектора k_i, то маємо:

$$VL_v = K_{report}\left(k_t\right) - K_{report}\left(k_0\right). \quad (3.3)$$

4. Інтервал оновлення (UI_v) – це середнє значення різниці в часі між послідовними векторами на шляху еволюції EP_v. Він показує, як часто впроваджуються модифікації. Формально,

$$UI_V = \frac{VL_V}{t}. \quad (3.4)$$

Шлях еволюції надає інструмент для відстеження змін між варіантом і його «джерелом» атакою.

Щоб проаналізувати зміни, знайдені на шляхах еволюції, існують такі визначення:

1. Модифіковані теги (MT) – це набір тегів, які використовуються будь-де на краю набору шляхів еволюції. Це теги, які були додані або видалені для зміни атак.

2. Підмножини тегів модифікації (MTS) – це всі підмножини набору тегів, які використовуються принаймні на одному краю набору шляхів еволюції.

Тому виключаємо синглтони з MTS, а розглядаємо лише підмножини щонайменше двох тегів.

Наприклад, якщо графік модифікації фішингової атаки має лише два ребра, одне з мітками {div:+1, a:+6}, а інше – {input:+3, a:+5, h2:+1} , множина MT дорівнює { , <a>, <input>, <h2>} і також маємо п'ять підмножин в MTS, а саме { , <a>}, {<input>, <a>, <h2>}, {<input>, <a>}, {<input>, <h2>} і {<a>, <h2>}.

3.4 Експериментальні дослідження застосування запропонованого методу

3.4.1 Застосовані база даних

Для проведення експериментальних досліджень було використано три джерела, щоб зібрати базу даних фішингових сайтів, яка включає керований спільнотою портал PhishTank4, хмарну платформу обміну інформацією про загрози IBM X-Force5 та платформу аналізу розвідки фішингу OpenPhish6.

В результаті цього зібрали список з 23 554 «перевірених» фішингових сайтів, щодня завантажуючи архіви з 1 січня 2020 року по 1 січня 2022 року.

Також було записано дату подання кожної URL-адреси.

Для кожної з цих URL-адрес автоматично отримуємо DOM, надсилаючи веб-сканер за URL-адресою.

Збираємо IP-адресу веб-сервера.

Необхідно зауважити, що в деяких випадках початкова URL-адреса повертає переспрямування на іншу URL-адресу, і в цьому випадку рекурсивно виконуємо переспрямування, поки не досягнемо фактичного ймовірного фішингового сайту.

Повідомлені дані – це дані, зібрані після виконання необхідних для виявлення ФА перенаправлень.

Щоб порівняти результати алгоритму кластеризації на фішингових і легальних сайтах, зібрали базу даних нефішингових веб-сайтів. Для цього використовуємо портал Alexa, який відстежує веб-трафік і надає списки найпопулярніших веб-сайтів. В ході цього отримуємо 12 522 законних сайтів, у тому числі 9 737 URL-адрес із серії безкоштовних «500 найпопулярніших» сайтів Alexa за країнами [46] та ще 15 063 URL-адреси, випадково вибраних із 100 000 найкращих до 460 697 веб-сайтів Alexa.

3.4.2 Результати кластеризації

Результати моделі кластеризації наведені в таблиці 3.1.

Різниця між двома моделями полягає в тому, що остання покращена модель використовує для побудови кластерів зважену пропорційну різницю (WPD) і напівповне зв’язування (SCL).

Виявивши, що набір даних із 23 554 екземплярів фішингу генерує лише 22 785 різних векторів, тоді як 12 522 законних сайтів генерують майже таку ж кількість векторів. Використовуючи пропорційну відстань (P), 90 551 екземпляр фішингу потрапляє на 5 729 кластерів, які містять більше одного екземпляра (відзначаємо ці кластери як «позначені» кластери в таблиці), що означає, що 91,86% всієї фішингової бази даних складається з повторюваних атак протягом періоду спостереження. У покращеній кластерній моделі, яка використовує WPD і SCL, ловимо більше варіантів фішингу (92,91%), але отримуємо менше позначених кластерів.

Таблиця 3.1 – Результати кластеризації

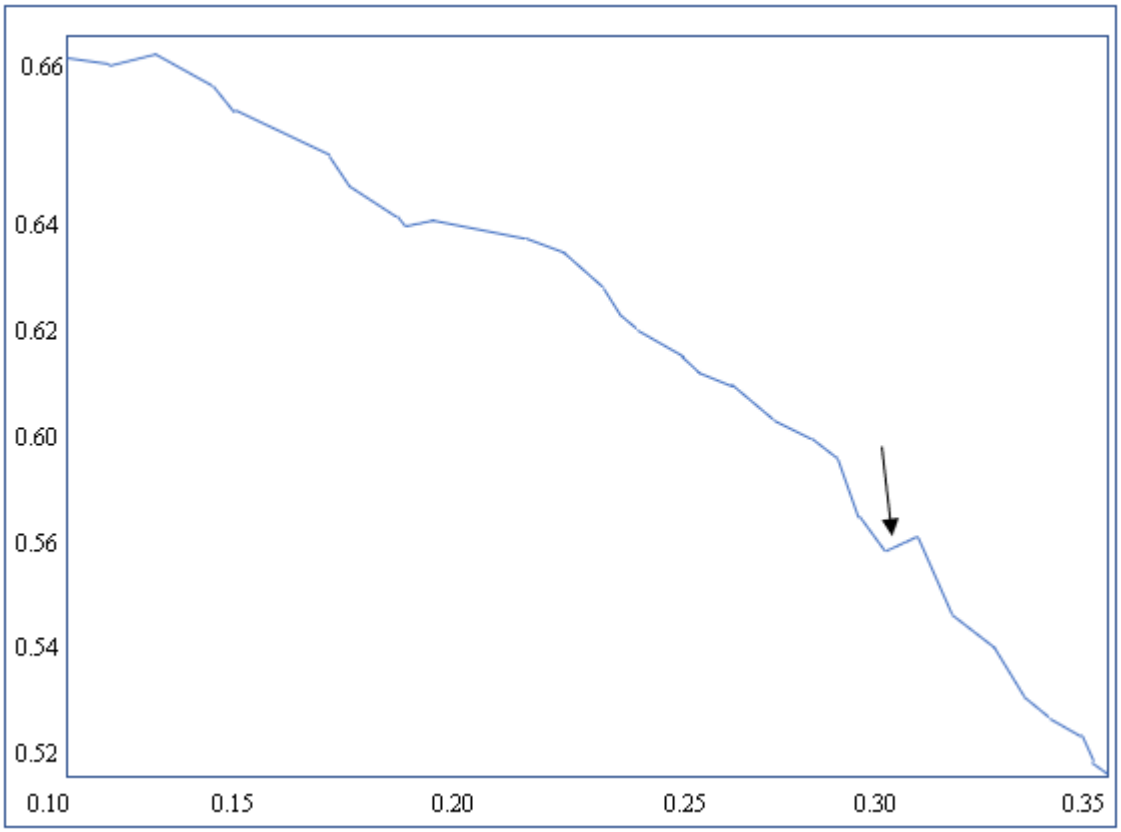
Кількість випадків фішингу	34 551	
Кількість векторів фішингу	22,785	
Кількість легальних сайтів	24,800	
Кількість законних векторів	11 539	
	Модель з P	Модель з WPD та SCL
Оптимальний поріг	0.33	0.26
Кількість позначених кластерів	5,729	5,334
Кількість випадків фішингу в позначених кластерах	90,551 (91.86%)	91,580(92.91%)
Кількість легальних сайтів у фішингових кластерах	128 (0.52%)	161 (0.65%)

На рисунку 3.3 показано, як обрано оптимальний поріг для обох кластерних моделей.

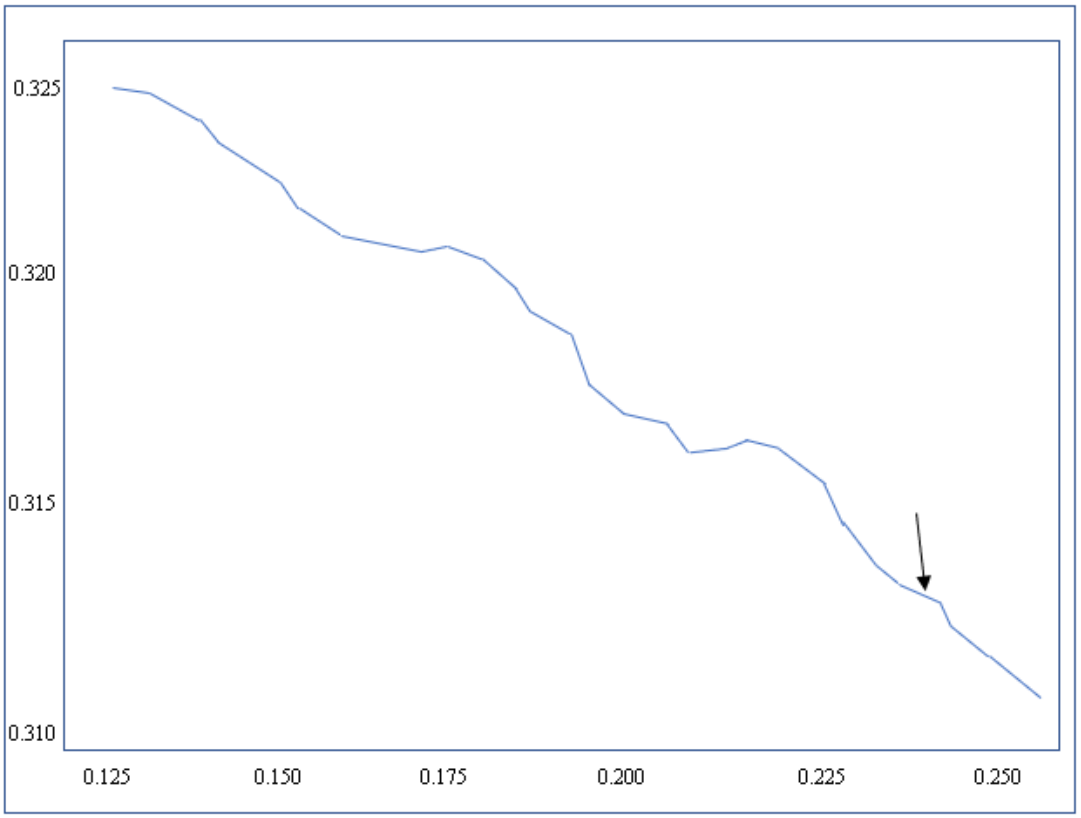
В результаті видно, що обидві моделі повертають оптимальний поріг у точці з найменшим значенням зв’язку залученої кластеризації.

Оскільки WPD завжди дає менше значення, ніж P, вдосконалена модель кластеризації має нижчий оптимальний поріг 0,26.

У базі даних із 12 522 легітимних сайтів лише 128 із цих сайтів потрапляють в один із фішингових кластерів, якщо використовується P як показник подібності, що становить лише 0,52%.



(a) Оптимальний поріг моделі з використанням P



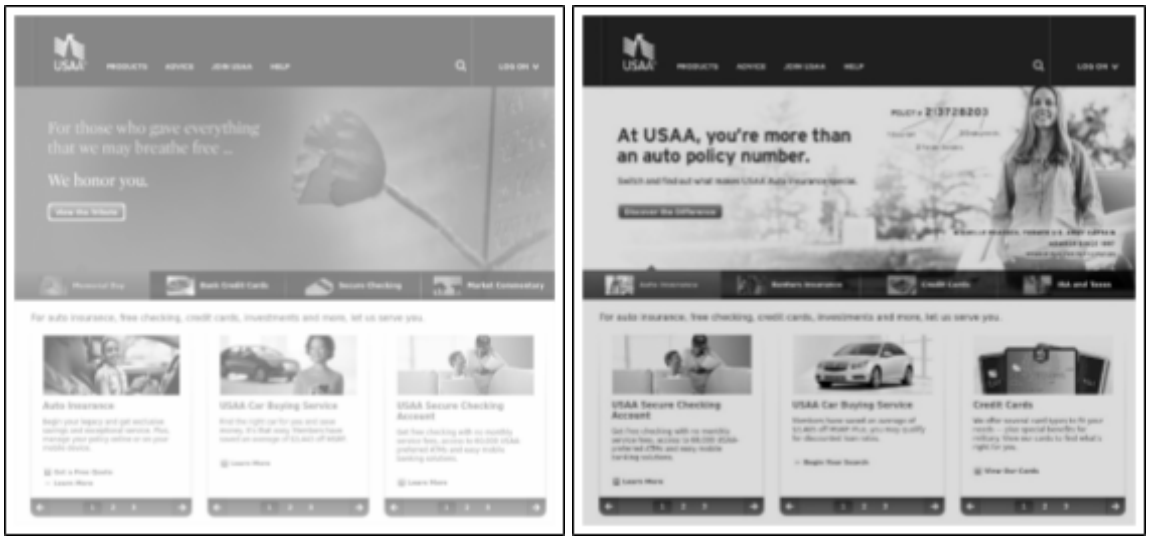
(b) Оптимальний поріг моделі з використанням WPD і SCL

Рисунок 3.3 – Вибір оптимального порогу

У вдосконаленій кластерній моделі хибнопозитивні результати покращені, але все ще залишаються дуже низькими – 0,65%.

Основною причиною таких помилкових результатів є те, що деякі фішингові сайти є копіями законного сайту (див., наприклад, рисунок 3.4 (a) і (b)).

Це показує, що ФА не схильні безпосередньо копіювати сайт, на який вони атакують. Причина цього, ймовірно, полягає в тому, що це особливо полегшує виявлення та фіксацію атак.



(a) Законний сайт (b) Фішинговий сайт

Рисунок 3.4 – Приклад хибнопозитивного результату

3.4.3 Порівняння кластеризації

Результати показують, що вдосконалена модель кластеризації виявляє більше варіантів фішингу, ніж оригінальна модель із використанням P.

Щоб краще зрозуміти деталі цих додаткових захоплених випадків, розглянемо відмінності результатів кластеризації між двома моделями.

Таблиця 3.2 – Порівняння результатів здійснення кластеризації

	Оригінальна модель P	Покращена модель WPD та SCL)
кількість одиночних атак	8020	6991
кількість атак, варіанти яких модель не може виявити, але може виявити інша модель	1052	23
Відстань до найближчого сусіда	0.27-0.32	0.26-0.30
Оптимальний поріг	0.33	0.26

Щоб спростити опис, називаємо атаки, які не входять до позначених кластерів, як «одиночні атаки».

Результат порівняння наведено в таблиці 3.2.

Існує 1052 одиночних атак, варіанти яких оригінальна модель не може виявити, але покращена модель може.

Більшість цих атак одного типу змінюють декілька тегів, наприклад , <p>, що збільшує P між векторами. Але різниця в змінах дуже мала, що мало впливає на WPD, тому вдосконалена модель все ще може ідентифікувати їх як фішингові варіанти. Є 23 одиночні атаки, визначені покращеною моделлю, але не оригінальною моделлю. Для цих випадків P до найближчого сусіда коливається від 0,27 до 0,32, а WPD до найближчого сусіда коливається від 0,26 до 0,30. Як видно, ці екземпляри відносяться до векторів, близьких до порогової межі. За допомогою перевірки вручну виявили, що більшість випадків є помилковими, отриманими від оригінальної моделі, і лише 5 випадків є фактичними варіантами фішингу.

3.4.4 Ефективність визначення фішингових варіацій

Підхід до виявлення фішингу на основі кластеризації вимагає порівняння між даним вектором і кожним відомим вектором фішингу. Щоб підвищити продуктивність, пропонуємо прототип кластера, який використовує результати кластеризації для визначення меж кластера. Основні результати порівняння продуктивності між використанням і невикористанням прототипу наведені в таблиці 3.3.

Спочатку збираємо збалансований тестовий набір із 2282 векторів фішингу (10% від усього набору фішингових даних) і законних векторів того ж розміру, які випадковим чином відбираються з фішингового та законного набору даних. Решта 20503 вектори фішингу використовуються як фішингові кластери, що порівнюються. Було б цікаво дізнатися, яка ця різниця в часі, оскільки більшість векторів, які будуть порівнюватися під час розгортання, будуть законними. Щоб усунути похибку вимірювань, проводимо 10 експериментів для кожного порівняння та повідомляємо середнє значення вимірювань.

Під час порівняння для моделі кластеризації з використанням P, 1097 тестових векторів виявлено як фішингові варіації, що потребує для завершення порівняння без використання прототипу 2919 секунд.

Завдяки використанню прототипу час порівняння скорочується вдвічі.

Це вказує на те, що межа кластера, отримана прототипами кластера, значно зменшує кількість порівнянь з векторами.

Якщо подивитись на покращену модель кластеризації з використанням WPD і SCL, 1264 тестових вектора ідентифікуються як схожі на відомі атаки.

Ця модель займає більше часу, ніж модель з використанням P, оскільки обчислення WPD складніше, ніж P.

Якщо використовується прототип, час порівняння зменшується до 71%, а для перевірки вектора потрібно в середньому 756 мс (відповідно 315 мс для одного прикладу).

Створення прототипів кластерів для понад 11 000 кластерів займає лише близько 2 секунд, що показує, що оновлення прототипу є дуже ефективним, коли такий підхід виявлення фішингу використовується з потоковим введенням.

Таблиця 3.3 – Порівняння ефективності виявлення фішингових варіацій

Кількість тестових векторів фішингу	2,282	
кількість тестових легітимних векторів	2,282	
кількість тестових екземплярів	10,941	
кількість векторів фішингу в кластерах	20,503	
	Модель з PD	Модель з WPD та SCL
кількість фішингових кластерів	12,583	11,320
кількість тестових векторів у фішингових кластерах	1,097	1,264
Час створення прототипу	2 с	
Час для порівняння без прототипу	2,919 с	4,884 с
Час для порівняння з прототипом	1,461 с	3,451 с
Час на вектор без прототипу	640 мс	1070 мс
Час на вектор із прототипом	320 мс	756 мс
Час на екземпляр без прототипу	267 мс	446 мс
Час на екземпляр з прототипом	134 мс	315 мс

Як видно, завдяки порівнянню між оригінальною моделлю кластеризації та вдосконаленою моделлю кластеризації, остання модель більш ефективна для виявлення фішингових варіацій.

3.4.5 Цілі фішингу

У таблиці 3.4 показано цільові бренди для 10 найбільших кластерів, упорядкованих за кількістю екземплярів атаки.

За винятком сьомого кластера, всі інші кластери мають чітко визначені цілі.

Сьомий кластер націлений на більшість великих соціальних мереж і провайдерів електронної пошти, таких як Youtube і Gmail.

Один із прикладів атаки в цьому кластері показаний на рисунку 3.5(a). У десятому кластері знаходимо атаку, яка використовує той самий шаблон, але націлений на різні бренди, як показано на рисунку 3.5(b) і рисунку 3.5(c).

Єдина зміна між двома атаками – це текст і зображення на сторінці: вектори їх тегів ідентичні.

Причина формування кластерів при атаках, спрямованих на один і той же бренд, полягає в тому, що для націлювання кожного бренду використовуються різні шаблони.

Таблиця 3.4 – Цільова марка 10 найкращих кластерів

Кластериний індекс	Фірмове найменування цілі фішингу	Кількість сторінок
1	Gmail	11391
2	PayPal	4493
3	Google Docs	2016
4	MailBox	1933
5	DropBox	1619
6	Yahoo	1536
7	Multiple	1421
8	PayPal	1332
9	Free Mobile	1229
10	OneDrive/GoogleDrive	1191

3.4.6 Життєвий цикл фішингових веб-сервісів

Один кластер представляє одну атаку, яка складається з кількох екземплярів атаки (фактичні кампанії розсилки, які намагаються залучити жертв на сайт).

Тут досліджується тривалість життя цих атак у базі даних. Тривалість життя визначається як тривалість між першим і останнім спостережуваним випадком атаки.

Оскільки одиночні кластери містять лише один екземпляр, аналіз зосереджено на позначених кластерах. Основні результати наведені на рисунку 3.6: близько 40% позначених кластерів активні менше одного місяця. Однак тривалі атаки можуть залишатися активними від 2 місяців до всіх 24 місяців спостереження. Загалом середня тривалість життя кластера в базі даних становить 128 днів.

З іншого боку, якщо подивитись на фактичні екземпляри атак, маємо бімодальний розподіл: 40% кластерів, які активні менше одного місяця, становлять лише близько 6% випадків атаки (зазначається, що тут розглядається відсоток екземплярів усього набору фішингових даних, а не лише екземплярів у позначених кластерах), тоді як понад 40% екземплярів атаки знаходяться на іншому кінці спектру, у кластерах тривалістю 22 місяці та більше. Це показує, що, всупереч тому, що вважалося раніше, фактичні атаки можуть тривати тривалий час через багато короткочасних екземплярів атак (екземпляри атак блокуються приблизно через 10 годин відповідно до [7]), і більшість випадків є частиною тривалих нападів.

На рисунку 3.7 показано, як часто спостерігаються екземпляри атак у 5334 позначених кластерах бази даних.

В результаті бачимо, що майже половина позначених кластерів у середньому більше одного екземпляра атаки на два тижні, що містить майже 80% екземплярів. Існує також невелика кількість кластерів з тривалим періодом перезапуску, деякі навіть охоплюють більше одного року. Це вказує на те, що більшість атак залишаються активними протягом тривалого часу через часте

перезапуск нових екземплярів, а деякі атаки знову запускаються після тривалого періоду інкубації.

Щоб зрозуміти, як ці атаки розвиваються та виживають такий довгий час, розглянемо розподіл векторів, IP-адрес та доменних імен другого рівня9 у часі. На рисунку 3.8 показано це для двох зразкових кластерів: одного з найтриваліших у базі даних (711 днів) і одного з найбільшою кількістю випадків атак (4493 фішингові атаки). На цьому рисунку числа є сукупними, і повторне використання вектора/домену/IP позначається іншою крапкою на рівні, на якому спостерігався цей вектор/домен/IP. Як можна побачити тут, зловмисники мають тенденцію змінювати домени найчастіше, причому IP-адреси на другому місці, але фактичні вектори змінюються не так сильно. Кластер з найбільшою тривалістю життя дає нам цікавий випадок, що напад залишається активним протягом тривалого часу, супроводжуючись кількома інкубаційними періодами. Якщо подивитись на повний набір позначених кластерів, то середнє співвідношення IP/векторів серед кластерів становить 1,91 (максимум – 103), а середнє співвідношення доменів/векторів – 2,05 (максимум – 105). Це доводить, що зловмисники під час повторного запуску атаки шукають інший домен для атаки і дуже часто інший хост для неї, але рідко витрачають час на модифікацію самого сайту. Це говорить про те, що для зловмисників дорожче модифікувати атаку, не кажучи вже про створення нової, порівняно з використанням нового домену або нового хоста.

Один із важливих моментів цього аналізу полягає в тому, що дана фішинг-атака може мати тривалий термін служби через тисячі «екземплярів» цієї атаки. За таких умов не дивно, що спроба захисту від реальних випадків атаки не є хорошою ефективною стратегією. Це програшна гра, оскільки зловмисники можуть просто запускати нові екземпляри за дуже низькою ціною. Що може бути дорожчим для зловмисників, так це створення нових атак. Таким чином, прикро, що більшість методів профілактики, які використовуються сьогодні, дійсно засновані на виявленні екземплярів. Це також створює ілюзію того, що профілактика ефективна, тоді як вона ефективна лише для усунення короткочасних нападів. Набагато ефективнішою стратегією є націлювання на

саму атаку. У цьому контексті метод кластеризації є кроком у правильному напрямку: після виявлення екземпляра створюється кластер для фактичної атаки, і будуть виявлені нові майбутні екземпляри. Це говорить про те, що веб-браузери слід розширити за допомогою таких рішень, які були виявлені під час роботи, щоб вони були більш ефективними у запобіганні фішингу.

3.5 Дослідження методу виявлення фішингових веб-сервісів

Щоб зрозуміти, чому виявлений під час роботи інструмент буде відігравати важливу роль у стратегії захисту від фішингу, незважаючи на те, що він виявляє не нові фішингові сайти, а скоріше нові ітерації відомих, потрібно розглянути, як типова реалістична корпоративна глибока стратегія боротьби з фішингом працює. У такому контексті постійний потік десятків тисяч, якщо не мільйонів щоденних URL-адрес має оброблятися, щоб оцінити фішингові атаки та внести їх у чорний список майже в реальному часі. Зазвичай це робиться в два етапи: перший рівень швидкої фільтрації застосовується для видалення переважної більшості URL-адрес, які є доброякісними.

Отриманий набагато менший набір потенційних фішингових сайтів потім аналізується за допомогою трудомістких методів. Другий підхід повільний, часто вимагає фактичного відображення веб-сайтів у браузері, а іноді навіть вимагає втручання людини. Обробка сайту на цьому етапі займає від секунд до хвилин. Таким чином, кількість сайтів, які досягають цього другого кроку, має велике значення: щоб захист був ефективним, ця кількість має бути якомога меншою.

Результати досліджень показують, що щонайменше 90% атак є повтореннями попередніх атак. Це означає, що на другому етапі вище переважна більшість часу витрачається на повторне підтвердження відомої атаки, тоді як за допомогою такої техніки, яка була виявлені під час роботи, можна автоматично знайти багато, якщо не всі, з цих повторів майже без хибнопозитивних результатів. Більше того, метод кластеризації може швидко оцінити сайт: за допомогою неоптимізованого коду в базі даних із майже 100

000 URL-адрес обробляємо сайт у середньому за 134 мс (відповідно 332 мс, якщо використовуємо WPD).

Таким чином, виявлений під час роботи інструмент є ефективним і важливим проміжним кроком у стратегії захисту: перелік потенційних атак, які були позначені першим кроком, потім аналізується за методом, який використовуємо в роботі. Згідно з результатами, приблизно 90% фактичних атак у цьому списку буде автоматично видалено (і відповідні вектори негайно додані до кластерів). Потім значно скорочений список передається на наступний і останній крок, де нові фішингові атаки оцінюються за допомогою більш повільних методів. Коли нова атака підтверджується на цьому останньому кроці, ця інформація негайно повертається в систему, і створюється відповідний кластер. З цього моменту всі вхідні варіанти цієї нової атаки будуть автоматично видалені і не досягнуть останнього кроку. Додавання цього проміжного кроку може стати відмінністю між системою, яка може впоратися з потоком вхідних атак, який бачимо сьогодні, і системою, яка не може цього зробити.

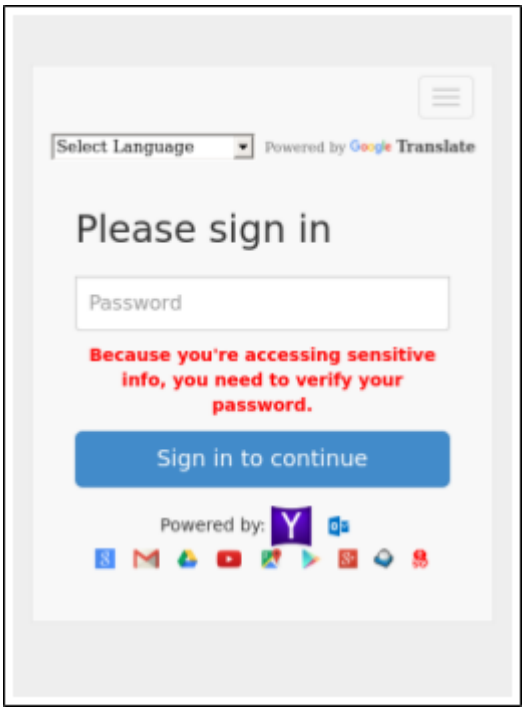
Зловмисники можуть спробувати протидіяти інструменту, ускладнивши виявлення повторної атаки. У цьому відношенні нинішній підхід не важко перемогти, хоча зараз він добре працює. Однак фактичний метод, який використовується для виявлення повторного подання, можна адаптувати, і виявлення подібності між DOM є активним дослідженням. Навіть якщо відсоток спійманих повторних атак зменшиться, виявлені все одно будуть чистим прибутком для захисту, і, отже, цей крок виявлення має бути частиною стратегії, доки зловмисники повторюють атаки. Кінцева мета цього кроку полягає в тому, щоб не допустити, щоб зловмисники покладалися на існуючі атаки, і змусити їх щоразу створювати нові, підвищуючи їх вартість і знижуючи їх ефективність.

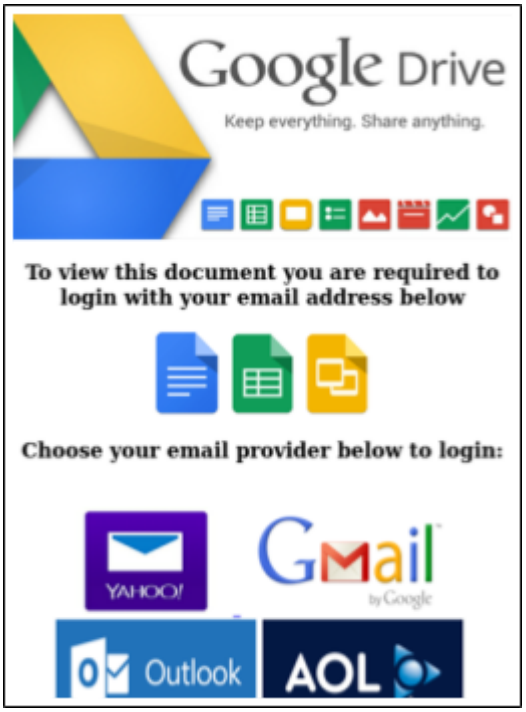
3.6 Обмеження застосування методу

Хоча результати показують, що підхід добре працює при виявленні фішингових варіацій, існують деякі обмеження, які можна було б покращити в подальшій роботі.

По-перше, зловмисник може зламати підхід, вводячи багато фіктивних тегів (наприклад, і <p>), щоб зменшити подібність між атаками. Рішенням цієї потенційної атаки є встановлення ваги для кожного тегу. Для фіктивних тегів потрібно встановити меншу вагу, тому вплив змін на цей тег мало впливає на остаточні показники схожості. Звичайно, завдання цього рішення полягає в тому, щоб знайти «хороші» ваги для кожного тегу. Можемо навчити модель машинного навчання встановлювати ці ваги, використовуючи екземпляри потенційних атак.

Іншим обмеженням підходу є те, що він не може виявити «нові» атаки, які не були помічені моделлю. Одним із рішень для цього є необхідність додати ще одну модель виявлення, яка розглядає характеристики фішингових атак для подальшої перевірки. Як тільки «нову» атаку виявляє додаткова модель, вона додається до бази даних, тому серія варіантів цієї атаки буде заблокована підходом на основі кластеризації.





(a) Один із прикладів атаки в кластері 7 (b) Атака класа 10 на Google Drive

(c) Атака в кластері 10, спрямована на OneDrive

Рисунок 3.5 – Приклади 10 найкращих кластерів

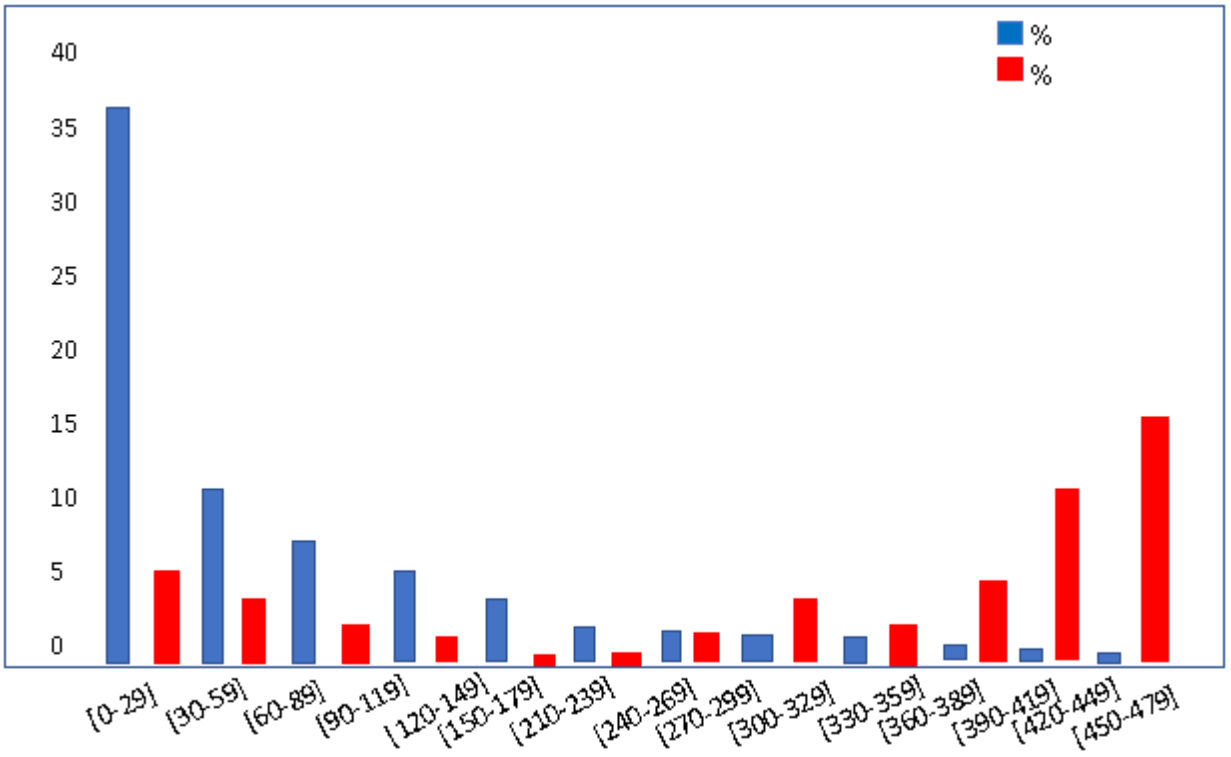


Рисунок 3.6 – Життєвий цикл кластерів.

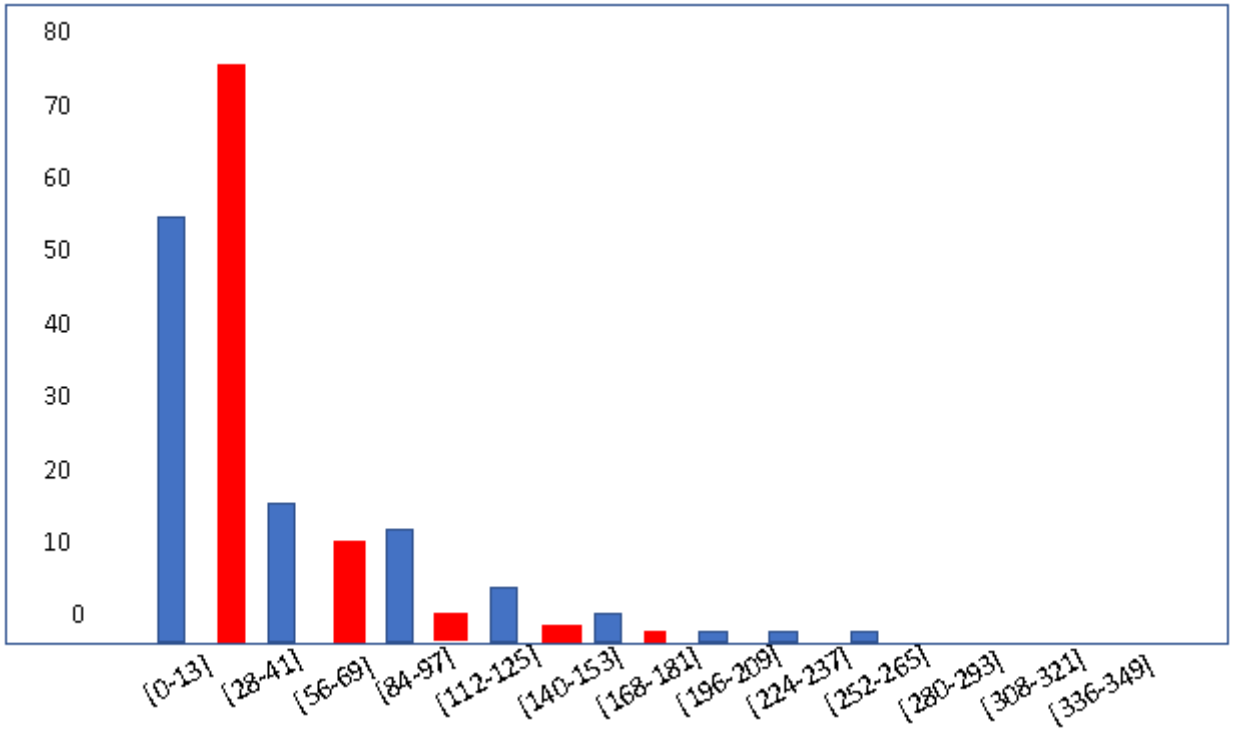


Рисунок 3.7 – Середній час між атаками.

3.7 Висновок

В розділі представлено удосконалений метод виявлення кіберзагроз типу «фішинг», який розглядає деталі модифікацій та їх еволюцію з часом, який ґрунтується на основі графіка, який використовується для відстеження та аналізу фішингових модифікацій та розвитку.

Було зоблено висновок, що звичайна поведінка зловмисників, які використовують фішингові сайти, полягає в тому, щоб неодноразово повторно публікувати свої атаки, використовуючи різні доменні імена, різні хости, а іноді й деякі незначні модифікації, ймовірно, як спосіб обійти дуже короткий середній час життя даного фішингового сайту.

Це дає можливість виявити ці «репліки», порівнюючи нові екземпляри атаки з раніше відомими.

Було здійснено розроблення удосконалено методу на основі кластеризації, який обчислює вектор, який підраховує кількість разів, коли HTML-теги використовуються в DOM сайтів.

Було визначено відстань та її покращену версію між цими векторами та використали ці відстані для об’єднання сайтів з відстанню, нижчою за заданий поріг.

Метод об’єднує в той самий кластер вектори, які мають принаймні один інший вектор із кластера, ближче за поріг. Таким чином, ці кластери не базуються на центроїдних, а вміщують довгі рядки послідовних модифікацій. Метод є детермінованим, що означає, що кластери можна ітеративно будувати з часом.

В результаті роботи тести показали, що методом буде виявлено велику кількість нещодавно виявлених екземплярів фішингових атак: близько 90% фішингових сайтів, про які повідомлялося, насправді є копіями вже відомих фішингових сайтів, і інструмент може негайно позначити їх.

Більше того, рівень хибнопозитивних результатів дуже низький – 0,65%. Це показує, що метод, представлений тут, є ефективним інструментом для додавання до стратегії захисту від фішингу як етапу попередньої обробки для відсіювання великої кількості нових екземплярів фішингових атак, що значно зменшує навантаження на трудомісткі серверні інструменти в завдання пошуку та підтвердження нових атак.

Цей інструмент призначений як доповнення до такого захисного механізму, а не заміна існуючих засобів. Однак це є необхідним удосконаленням, щоб запобігти тому, щоб тривалий етап виявлення нових атак був переповненим величезною кількістю нових випадків атак, зафіксованих сьогодні.

Хоча метод кластеризації є досить ефективним, можливим є ситуація, коли зловмисники намагатимуться перемогти його, ретельніше модифікуючи свої екземпляри атак. Загальну схему захисту просто потрібно буде адаптувати за допомогою більш розумних способів виявлення різних випадків однієї і тієї ж атаки.

Новизною методу є порівняння атаки з іншими відомими атаками, а не з цільовим сайтом.

Оскільки метод виявлення фішингу на основі кластеризації вимагає порівняння з кожною відомою атакою, час виявлення буде лінійно збільшуватися, оскільки до бази даних додається все більше і більше екземплярів атак. Це ускладнює застосування підходу до великомасштабної бази даних. Щоб подолати вищевказане обмеження, пропонуємо прототип кластера, який використовує переваги результатів кластеризації для визначення меж досліджуваного кластера.

В результаті показано, що використання прототипу кластера дозволяє скоротити час обробки до половини.

Отже , як бачимо, що тривалі атаки виживають у сучасних методах запобігання, повторно запускаючи дуже схожі атаки на нові домени та на нові сервери, можливо, після деяких незначних змін DOM.

Сучасні методи запобігання спрямовані на виявлення та занесення в чорний список випадків атак, а не на самі атаки. Це програшна стратегія, від якої зловмисники легко ухиляються, і стратегія, яка створює помилкове відчуття ефективності.

Відомо, що стратегії захисту від фішингу повинні бути зосереджені на фактичних атаках за екземплярами атаки, змушуючи зловмисників вкладати більше коштів між кожною ітерацією своїх атак. Після ідентифікації екземпляра атаки всю атаку можна нейтралізувати.

Аналіз результатів роботи запропонованого удосконаленого методу показав, що більшість екземплярів атак є похідними від невеликого набору «головних» атак, і лише кілька послідовних версій розгортаються.

Це вказує на те, що зловмисники не прагнуть оновлювати та покращувати базову лінію своїх атак, а натомість продовжують переробляти з тієї ж базової версії.

Також виявлено, що кожна фішингова атака має тенденцію модифікуватися самостійно, незалежно від інших атак; кожен кластер атак використовує свій власний шаблон сторінки і вдосконалюється без загального плану атак.

4 ПРОГРАНО-АПАРАТНІ ЗАСОБИ ЕМУЛЯТОРА ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

4.1 Емулятор виявлення кібер-загроз типу «фішинг»

Запропонований удосконалений метод виявлення кіберзагроз типу «фішинг» було реалізовано у вигляді емулятора - програмно-апаратного засобу, який ґрунтується на використанні програмованій користувачем вентильній матриці, ПКВМ (FPGA).

В дослідженні було залучено DE1-SoC Development Kit, який представляє надійну платформу апаратного проектування, побудовану на основі FPGA Altera System-on-Chip (SoC), яка поєднує в собі новітні двоядерні вбудовані ядра Cortex-A9 з провідною в галузі програмованою логікою для максимальної гнучкості дизайну [56].

FPGA дозволяє використовувати потужність величезної можливості переналаштування в поєднанні з високопродуктивною системою процесора з низьким енергоспоживанням.

SoC від Altera інтегрує жорстку процесорну систему на основі ARM (HPS), що складається з процесора, периферійних пристроїв та інтерфейсів пам’яті, безперешкодно пов’язаних із структурою FPGA за допомогою високопропускної магістралі міжз’єднання.

Плата розробника DE1-SoC оснащена високошвидкісною DDR3 пам’ять, відео- та аудіо-можливості, мережа Ethernet та багато іншого, що обіцяє багато захоплюючих додатків.

FPGA складається з трьох основних програмованих елементів: неперемикаючих програмованих логічних комірок (PLB), осередків вводу-виводу (комірки вводу-виводу) і внутрішніх з'єднань.

FPGA є функціональним елементом, який використовується для побудови логіки користувача, а BVV забезпечує з'єднання між контактами корпусу та внутрішніми сигнальними лініями.

Програмовані внутрішні комунікаційні ресурси забезпечують контроль над тим, як входи та виходи блоків PLB та I/O підключаються до відповідних мереж. Усі трасові канали мають однакову ширину (однакова кількість провідників). Більшість блоків BVV можна встановити в один ряд (висота) або один стовпець (ширина) арматури.

Логічний блок класичного FPGA складається з таблиці пошуку на 4 входи та тригера.

FPGA Altera System-on-Chip використовує 6-вхідних пошукових таблиць у високопродуктивних частинах схем, що пояснює підвищення продуктивності роботи спроєктованого програмно-апаратного емулятора виявлення кібер-загроз типу «фішинг».

4.2 Архітектура та програмне забезпечення емулятора виявлення кібер-загроз типу «фішинг»

Для вирішення задачі проєктування та реалізації емулятора виявлення кібер-загроз типу «фішинг» було побудованого його архітектуру (дивись рисунок 4.1).

Наявні у структурі тумблери на перетині вертикального і горизонтального каналів дозволили використати таку архітектуру для кожного провідника, що міститься в блоці комутатора, а також три програмовані перемикачі, що дозволили йому підключатися до інших трьох провідників у сусідніх сегментах каналу.

Модель або топологія комутатора, що використовується в цій архітектурі, є топологією перемикання плоского або домену. У цій топології провідники підключаються лише до провідників у сусідніх каналах, і т. д.

Оскільки траси FPGA Altera System-on-Chip є несегментованими, тобто кожен сегмент провідника з'єднує лише один логічний блок з блоком комутатора. Довші сліди завдяки огинанню програмованих перемикачів у блоці перемикача. Щоб збільшити швидкість з'єднань всередині системи, архітектура

FPGA Altera System-on-Chip використовує більш тривалі з'єднання трасування між логічними блоками.

Для програмної підтримки функціонування емулятора виявлення кібер-загроз типу «фішинг» було розроблено програмне забезпечення, що базується на HDL і HLS на основі C для FPGA (див. Додаток А).

Було здійснено високорівневий синтез проектування FPGA в порівнянні з введенням проектування з використанням мов рівня передачі реєстрів (RTL). Застосовані потоки C-to-RTL є вихідним кодом для синтезу високого рівня (HLS).

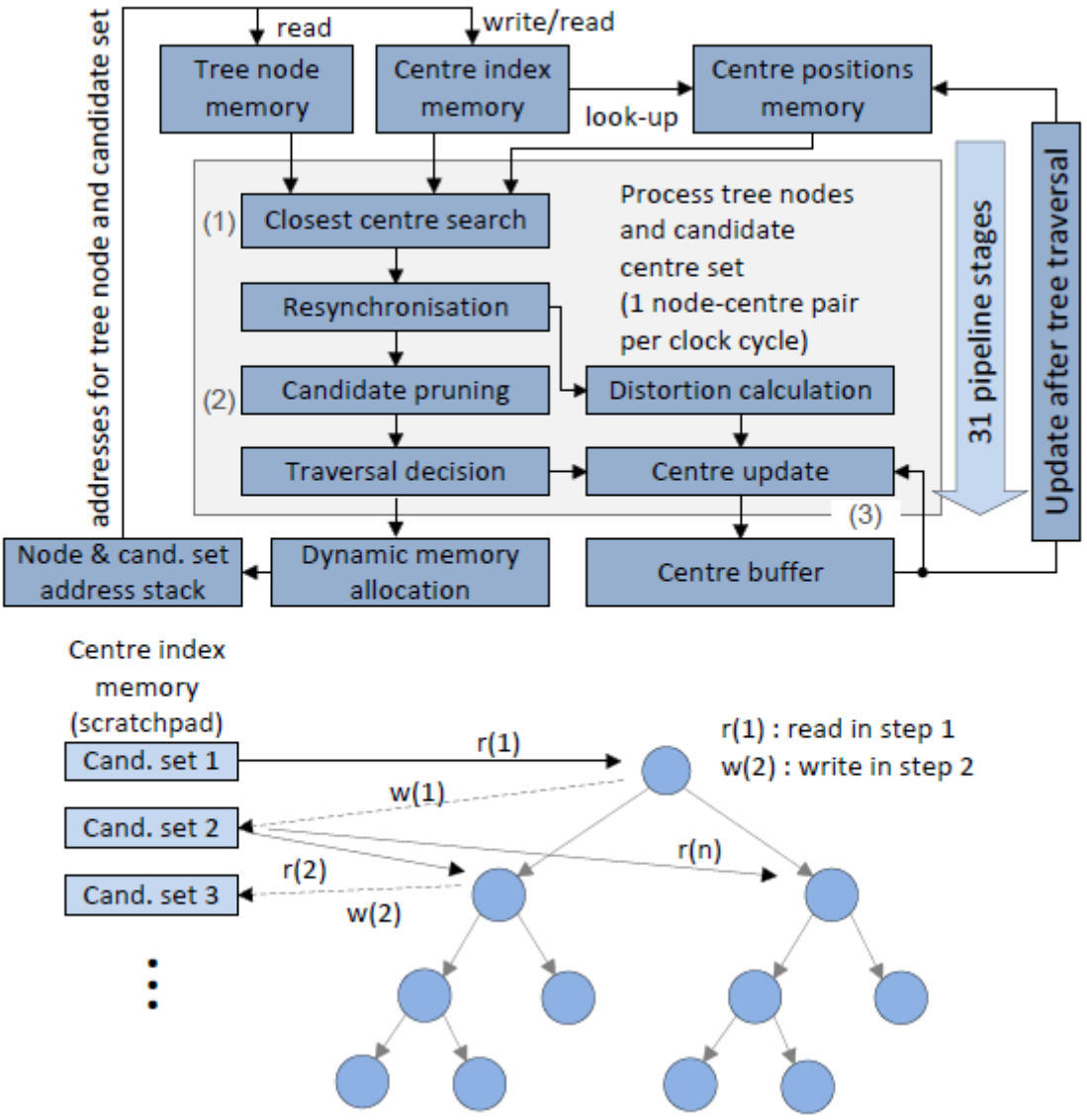


Рисунок 4.1 – Архітектура емулятора виявлення кібер-загроз типу «фішинг» [50]

4.3 Результати аналізу засобами емулятора

4.3.1 База даних фішингових сайтів

Для експериментальних досліджень реалізованого емулятора було зібрано базу даних про фішинг, а також URL-адреси екземплярів фішингових атак із [47] та платформи аналізу безпеки підприємства [48].

Загалом назбирали 54 575 «перевірених» фішингових сайтів шляхом отримання щоденного архіву з PhishTank з 1 січня 2020 року по 31 жовтня 2021 року.

Для кожного фішингового сайту: отримали DOM, першу URL-адресу (повідомлену), кінцеву URL-адресу (яка відрізняється від першої URL-адреси лише тоді, коли зловмисник використав переспрямування) і знімок екрана кінцевої сторінки.

4.3.2 Вектори та результати кластеризації засобами емулятора

Щоб обчислити набір векторів тегів, використовуємо той самий корпус із 107 унікальних тегів.

Потім підраховуємо кількість зустрічей кожного тегу в кожній DOM і використовуємо ці числа для створення цілих векторів із 107 об’єктів.

В результаті отримуємо 7 600 унікальних векторів тегів із DOM 13 689 екземплярів фішингових атак

Таблиця 4.1 – Результати векторної та кластеризації

Оптимальний поріг	0.25
кількість векторів	7,600
кількість кластерів із кількома елементами (позначено прапорцем)	941
кількість одноелементних кластерів	3,869
кількість фішингових сайтів у позначених	12,443 (90.9%)

кластерах	

Далі застосовуємо вдосконалену модель кластеризації, для створення фішингових кластерів.

В результаті було отримано оптимальний поріг 0,25 і закінчуємо 941 позначеним кластером, який охоплює 12,443 (90,9%) випадків фішингу.

4.3.3 Виявлення джерела змін засобами емулятора

Одне з можливих пояснень модифікацій, полягає в тому, що атаку фактично змінює не зловмисник, а сервер хостингу, який автоматично вводить деякий HTML на сторінки, наприклад деякі посилання відстеження Google Analytics, деякі плагіни WordPress або деякі інші бібліотеки Javascript [49].

Оскільки дана атака буде розміщена на ряді серверів, ці зміни будуть неправильно інтерпретовані як модифікації самої атаки.

Щоб переконатися в цьому, порівнюємо DOM фішингових атак з DOM домашніх сторінок сервера, на якому розміщені ці атаки.

Далі здійснюється видалення всіх «пробілів» (включаючи \t \r \n \f\v) з обох DOM, а потім витягуємо вміст, загальний між двома DOM.

Цей вміст міг надходити з сервера хостингу, а не від самої атаки.

Розробляємо це для всіх екземплярів атаки в базі даних, для яких можна отримати доступ до домашньої сторінки хосту та має інший вектор тегів від атаки.

Потім збираємо DOM 14 584 таких домашніх сторінок. З них 2566 мають загальний вміст із розміщеними атаками.

Більш уважний розгляд тегів, залучених до цього поширеного вмісту, показує, що тег <meta> бере участь у 2280 із цих випадків, що не дивно, оскільки <meta> використовується для такої інформації, як кодування, розмір сторінки тощо, інформація, яка зазвичай встановлюється сервером хостингу.

Тег <script> є дуже віддаленою секундою, наявною лише в 96 випадках досліджень.

Це показує, що тег <meta> є єдиним тегом, для якого сервер хостингу дійсно може вплинути на результати. Тому вирішується взагалі видалити цей тег із векторів тегів.

Повторюючи експеримент без тега <meta>, знаходимо той самий оптимальний поріг (0,25) і в кінцевому підсумку отримуємо 8290 векторів тегів, розподілених між 913 позначеними кластерами (кластер із щонайменше 2 векторами) і 3912 одновекторними кластерами.

4.3.4 Вибір зразка кластерів

Застосовуємо графік модифікації фішингової атаки, до 913 позначених кластерів.

Далі помічаємо, що в графі є кілька кластерів з дуже малою кількістю ребер, а це означає, що для цих кластерів база даних не містить багато варіантів відповідних атак.

У таблиці 4.2 показано детальний розподіл розмірів графіків модифікації фішингової атаки.

Лише 46,88% кластерів мають графік модифікації фішингової атаки з двома або більше ребрами, але вони містять більше 75% випадків фішингової атаки.

Для дослідження обираємо кластери з графіком модифікації фішинг-атак із 30 або більше ребрами, оскільки вони фіксують більшість випадків фішингової атаки (52%) і містять достатню кількість варіацій атак для аналізу їхньої еволюції з часом.

4.3.5 Аналіз основних векторів засобами емулятора

У таблиці 4.3 наведено огляд векторів провідних/неголовних для всіх 62 кластерів: загалом існує 190 (10,47%) головних векторів, що охоплюють близько 35% випадків атаки.

Це показує, що головні вектори часто використовуються для повторного запуску атак.

Більше того, 34 кластери (54,84%) мають два або більше головних векторів, що свідчить про декілька початкових версій атаки, які пізніше були об’єднані за допомогою серії оновлень.

Таблиця 4.2 – Кількість ребер і розподіл сторінок серед графіків модифікації фішингових атак

кількість ребер у кластері	кількість охоплених кластерів (% від загальної кількості)	кількість охоплених сторінок (% від загальної кількості)	кількість покритих ребер
2	428 (46.88%)	41,229 (75.55%)	18,636
3	394 (43.15%)	40,579 (74.35%)	18,568
4	258 (28.26%)	38,059 (69.74%)	18,160
5	243 (26.62%)	37,321 (68.38%)	18,100
10	150 (16.43%)	34,539 (63.29%)	17,504
15	107 (11.72%)	31,638 (57.97%)	17,043
20	88 (9.64%)	30,797 (56.43%)	16,732
30	62 (6.79%)	28,801 (52.77%)	16,113
40	47 (5.15%)	26,298 (48.19%)	15,591
50	42 (4.60%)	25,306 (46.37%)	15,381

Вручну перевіривши DOM головних векторів, виявилось, що головні вектори можна згрупувати в три категорії:

1. Різні початкові версії атаки.

2. Різні кроки однієї атаки.

3. Копії різних версій цільового сайту.

Таблиця 4.3 – Кількість основних/неголовних векторів у кластерах

кількість кластерів	62
кількість векторів	1814
кількість випадків атаки	28,455
кількість основних векторів	190 (10.47%)
кількість екземплярів атаки в основних векторах	9,855 (34.22%)
кількість кластерів із двома або більше основними векторами	34 (54.84%)
кількість кластерів лише з одним основним вектором	28 (45.16%)

Різні початкові версії атаки зловмисників із достатніми змінами, щоб перевищити відстань за поріг. Може статися, що ціль модифікована або випущено кілька нових функцій одночасно.

На рисунку 4.2а показано такий приклад.

Різні кроки однієї атаки.

Деякі атаки проходять кілька етапів, коли збирають додаткову інформацію від жертви.

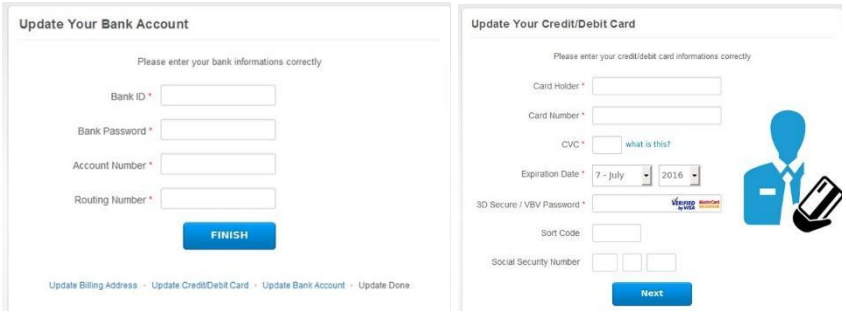
Наприклад, на рисунку 4.2б перший крок використовується для збору інформації для входу, а якщо вона надається, слідує другий крок, на якому запитуються дані кредитної картки.

Ці різні кроки розпізнаються як такі, що належать до однієї атаки, але різниця надто велика для порога, а також немає спрямованого шляху в графі SCL.

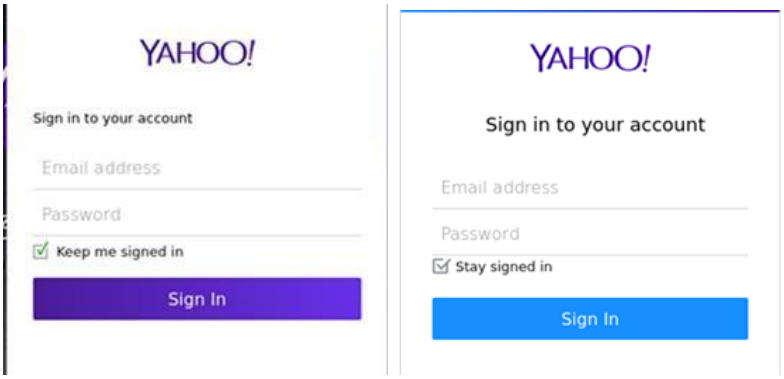
Копії різних версій цільового сайту.

Як показано на рисунку 4.2в, іноді головні вектори є копіями цільових сайтів, знятих у різний час.

а)



б)



в)

Рисунок 4.3 – Приклади основних векторів

а) різні версії, розроблені фішерами;

б) різні кроки одних і тих самих атак

в) різні версії, скопійовані з легальних сайтів (сторінка входу в Yahoo, приблизно 2020 і 2021 рр.)

Цільовий сайт було змінено, щоб відповідні екземпляри атаки спочатку не збігалися.

Також можливо, що в деяких випадках у базі даних відсутня ще більш рання версія атаки, яка б дала початковий та єдиний головний вектор.

4.3.6 Аналіз історії змін

Таблиця 4.4 надає середні значення атрибутів, для всіх шляхів еволюції у вибраних 62 кластерах.

Щоб обчислити ці значення, не включаємо шляхи еволюції, які включені в інші, більш тривалі шляхи.

Результати показують, що загалом атаки змінюються лише раз на шість місяців (186 днів) і що зміни зазвичай не є різкими (середній WPD між цими змінами становить 0,111).

Також показано, що середня відстань шляху мала, всього 0,1719. Отже, середня довжина шляхів еволюції становить лише 0,1719/0,111 < 2, менше двох ребер.

Це вказує на те, що зловмисники зазвичай не підтримують довгі шляхи еволюції, щоб створити безліч варіацій з часом. Натомість мають тенденцію знову і знову створювати нові варіації з одних і тих самих основних векторів.

Також вважаємо, що кожна варіація, як правило, залишається активною протягом тривалого часу, приблизно дев’ять місяців (267 днів).

Таблиця 4.4 – Аналіз шляхів еволюції в базі даних

кількість шляхів еволюції	1,230
Середня відстань шляху	0.1719
Середня відстань еволюції	0.111
Середня тривалість життя варіацій	267 днів
Середній інтервал оновлення	186 днів

З результатів видно, що більшість модифікацій фішингових атак походять від невеликого набору основних версій.

Кожна з цих модифікацій, як правило, повторно використовується як є протягом тривалого періоду часу.

4.3.7 Типи модифікацій, які спостерігаються під час фішингових атак

Щоб зрозуміти модифікації, які відбуваються на шляху еволюції, проаналізуємо загальні модифікації серед кластерів.

10 найпоширеніших модифікованих тегів (MT) і кількість кластерів, у яких вони з'являються, є <script> (57), (53), <img> (52), <a> (51), <input> (50),
 (48), <link> (47), (47), <p> (41) і <style> (40). 10 найпоширеніших підмножин тегів модифікації (MTS) серед вибраних 62 кластерів наведено в таблиці 4.5.

Таблиця 4.5 – 10 найпоширеніших MTS в базі даних

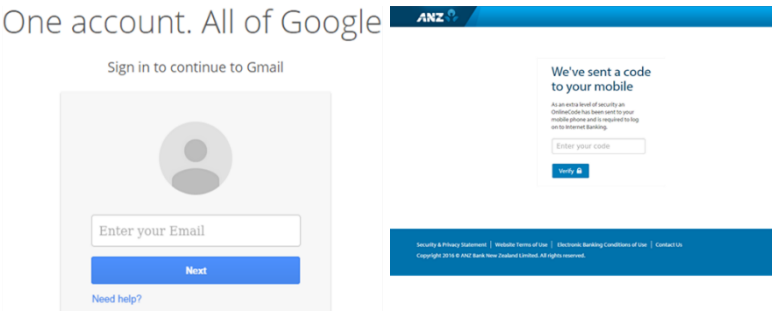
MTS	Кількість кластерів	%	Кількість сторінок	%
{a, div}	45	72.58%	403	24.82%
{div, img}	44	70.97%	286	17.61%
{div, script}	44	70.97%	403	24.82%
{div, span}	40	64.52%	264	16.26%
{br, div}	39	62.90%	215	13.24%
{img, script}	39	62.90%	199	12.25%
{a, img}	37	59.68%	235	14.47%
{link, script}	37	59.68%	215	13.24%
{script, span}	35	56.45%	174	10.71%
{input, span}	35	56.45%	161	9.91%

Виявивши, що окрім тегів , і
, які використовуються для пробілів або контейнерів, і функціональних тегів <script> і <link>, які

використовуються для додавання сценаріїв і ресурсів, фішери використовують лише три теги в топ-10 MTC: <a>, <img> і <input>. На рисунку 4.3 показано два приклади (візуальних) модифікацій, які мають лише один тег, який фактично оновлюється.

На рисунку 4.3а для зміни цілі додано один тег <img>.

На рисунку 4.3б фішинг-атака облікових даних електронної пошти перетворюється на фішингову сторінку податкової декларації шляхом зміни фонових зображень і додавання 31 тегу <input>.



(а)

б)

Рисунок 4.3 – Модифікація атак шляхом зміни одного тегу:

а) один тег <img> додається між лівою та правою атакою

б) між лівим і правим додано 31 тег <input>, і фонове зображення змінено

Також важливо зауважити, що, незважаючи на дуже малу кількість тегів, які використовуються для виконання модифікацій, жоден з топових MTC не використовується більш ніж на 25% ребер.

Щоб краще зрозуміти, наскільки поширеною чи незвичайною є кожна комбінація MTS, обчислюємо індекс Жаккара: для кожної пари кластерів обчислюємо кількість 10 провідних MTS (відповідно верхніх 10 MT), загальних для обох кластерів, поділених на кількість топ-10 MTC (відповідно топ-10 MT), включених в будь-який кластер.

На рисунку 4.4 показано розподіл отриманих значень.

Зробивши висновок , розподіл індексів Жаккара для пар з топ-10 MT охоплює відносно широкий діапазон, від 0,1 до 0,7.

Це вказує на те, що різні кластери використовують ті самі теги для створення варіацій, наприклад або <input>.

З іншого боку, розподіл індексів Жаккара для пар MTC дуже різний: більшість індексів менші за 0,3, а переважна більшість (майже 80%) менші за 0,1.

Ці результати показують, що навіть якщо насправді використовується дуже мало тегів, коли атаки змінюються, комбінація використовуваних тегів, як правило, є унікальною для атаки.

Іншими словами, атаки розвиваються незалежно одна від одної, і модифікації вносяться з причин, які є специфічними для кожної атаки, а не як глобальне оновлення, здійснене для цілого ряду атак.

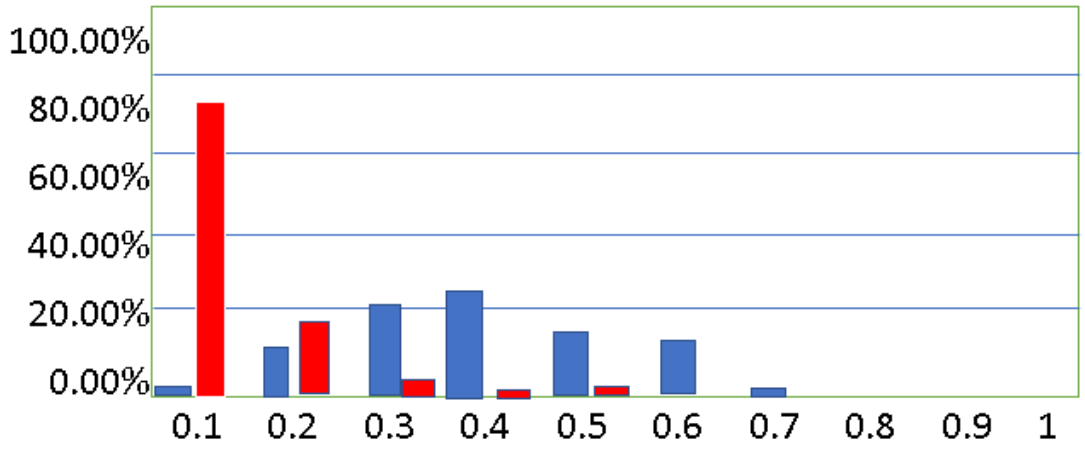


Рисунок 4.4 – Гістограма індексу Жакара для 10 найвищих MT і MTS

4.4 Висновок

В даному розділі було реалізовано удосконалений метод виявлення кіберзагроз типу «фішинг»у вигляді емулятора - програмно-апаратного засобу, який ґрунтується на використанні програмованій користувачем вентильній матриці, ПКВМ (FPGA).

В дослідженні було залучено DE1-SoC Development Kit, який представляє надійну платформу апаратного проектування, побудовану на основі FPGA Altera System-on-Chip (SoC). Також для вирішення задачі проєктування та реалізації емулятора виявлення кібер-загроз типу «фішинг» було побудованого його архітектуру.

В результаті здійсненого проєктування емулятора стало можливим виконати експериментальні дослідження з окрема: отримати результати аналізу засобами емулятора на основі баз даних фішингових сайтів, побудувати вектори та отримати результати кластеризації, здійснити виявлення джерела змін, виконати вибір зразка кластерів, здійснити аналіз отриманих результатів, які досягають 90 % достовірності.

ВИСНОВКИ

У першому розділі проведено дослідження методів та засобів виявлення фішингових атак, а саме проаналізовано суть поняття фішингових атак.

Також в розділі було досліджено переваги та недоліки методів виявлення фішингових атак. Досліджено життєвий цикл фішингових атак. Проаналізовано основні механізми виявлення фішингових сайтів, виявлення фішингу за допомогою чорних списків, виявлення фішингу шляхом порівняння з ціллю, виявлення фішингу шляхом перегляду внутрішніх характеристик. В розділі представлено постановку задачі.

У другому розділі було подано основні аспекти моделі процесу функціонування емулятора виявлення атак типу фішинг, зокрема представлені моделі фішингових атак, подано процес виявлення фішингових копій і варіацій, спроєктовано відбиток фішингової сторінки: вектор тегів.

Також в розділі проведено моделювання схожості фішингових сторінок: пропорційна відстань, наведено алгоритм кластеризації.

У третьому розділі представлено удосконалений метод виявлення кіберзагроз типу «фішинг», який розглядає деталі модифікацій та їх еволюцію з часом, який ґрунтується на основі графіка, який використовується для відстеження та аналізу фішингових модифікацій та розвитку.

В четвертому розділі представлено реалізацію удосконаленого методу виявлення кіберзагроз типу «фішинг»у вигляді емулятора - програмно-апаратного засобу, який ґрунтується на використанні програмованій користувачем вентильній матриці, ПКВМ (FPGA). В дослідженні було залучено DE1-SoC Development Kit, який представляє надійну платформу апаратного проектування, побудовану на основі FPGA Altera System-on-Chip (SoC). Також для вирішення задачі проєктування та реалізації емулятора виявлення кібер-загроз типу «фішинг» було побудованого його архітектуру. В результаті здійсненого проєктування емулятора стало можливим виконати експериментальні дослідження з окрема: отримати результати аналізу засобами емулятора на основі баз даних фішингових сайтів, побудувати вектори та

отримати результати кластеризації, здійснити виявлення джерела змін, виконати вибір зразка кластерів, здійснити аналіз отриманих результатів, достовірність яких досягає 90%.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Лисенко Сергій, Сокальський Дмитро, Михасько Яна. Дослідження методів виявлення кібератак в комп'ютерних мережах як засобів до побудови резильєнтних до вторгнень ІТ-інфраструктур. *Computer Systems and Information Technologies*. 2021. № 3. с. 31-35

2. Сокальський Д. О., Лисенко С. М. Метод розробки емулятора виявлення кібер-загроз типу «фішинг». *Збірник наукових праць XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021»*. Хмельницький, 2021. с. 156-158.

3. Lysenko, S., Bobrovnikova, K., Savenko, O., Shchuka, R. Technique for Cyberattacks Detection Based on DNS Traffic Analysis, *CEUR-WS*, Vol 2732. ISSN: 1613–0073. 2020. pp. 171-182

4. Bobrovnikova, Kira, Sergii Lysenko, Piotr Gaj, Dmytro Denysiuk "Technique for IoT Cyberattacks Detection Based on DNS Traffic Analysis.*CEUR-WS*.Vol.2623. ISSN: 1613–00732020, pp. 208-218.

5. Sergii Lysenko, Kira Bobrovnikova, Peter Popov, Viacheslav Kharchenko, Dmytro Medzatyi. Spyware Detection Technique Based on Reinforcement Learning. *CEUR-WS*.Vol. 2623 ISSN: 1613–0073 (2020), pp. 307-316

6. Lysenko, S., Bobrovnikova, K., Matiukh, S., Hurman, I., Savenko, O.Detection of the botnets' low-rate DDoS attacks based on self-similarity. *International Journal of Electrical and Computer Engineering*, ISSN 2088-8708, 2020, 10(4), pp. 3651-3659.

7. Savenko, O., Sachenko, A., Lysenko, S., Markowsky, G., Vasylkiv, N. Botnet detection approach based on the distributed systems. *International Journal of Computing*, ISSN 1727-6209, 2020, 19(2), pp. 190-198.

8. S. P. Ripa, F. Islam and M. Arifuzzaman, The Emergence Threat of Phishing Attack and The Detection Techniques Using Machine Learning Models, *International Conference on Automation, Control and Mechatronics for Industry 4.0 (ACMI)*, 2021, pp. 1-6, doi: 10.1109/ACMI53878.2021.9528204.

9. T. Nathezhtha, D. Sangeetha and V. Vaidehi, WC-PAD: Web Crawling

based Phishing Attack Detection, *International Carnahan Conference on Security Technology (ICCST)*, 2019, pp. 1-6, doi: 10.1109/CCST.2019.8888416.

10. S. Patil and S. Dhage, A Methodical Overview on Phishing Detection along with an Organized Way to Construct an Anti-Phishing Framework, *5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, 2019, pp. 588-593, doi: 10.1109/ICACCS.2019.8728356.

11. M. D. Bhagwat, P. H. Patil and T. S. Vishawanath, A Methodical Overview on Detection, Identification and Proactive Prevention of Phishing Websites, *Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, 2021, pp. 1505-1508, doi: 10.1109/ICICV50876.2021.9388441.

12. M. Baykara and Z. Z. Gürel, Detection of phishing attacks, *6th International Symposium on Digital Forensic and Security (ISDFS)*, 2018, pp. 1-5, doi: 10.1109/ISDFS.2018.8355389.

13. G. -G. Geng, Z. -W. Yan, Y. Zeng and X. -B. Jin, RRPhish: Anti-phishing via mining brand resources request, *IEEE International Conference on Consumer Electronics (ICCE*), 2018, pp. 1-2, doi: 10.1109/ICCE.2018.8326085.

14. S. Parekh, D. Parikh, S. Kotak and S. Sankhe, A New Method for Detection of Phishing Websites: URL Detection, *Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018, pp. 949-952, doi: 10.1109/ICICCT.2018.8473085.

15. Saeed Abu-Nimeh, Dario Nappa, Xinlei Wang, and Suku Nair. A comparison of machine learning techniques for phishing detection. *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*. ACM. 2017. pages 60–69.

16. J. -H. Li and S. -D. Wang, PhishBox: An Approach for Phishing Validation and Detection, *IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech),* 2017, pp. 557-564, doi: 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.101.

17. C. E. Shyni, A. D. Sundar and G. S. E. Ebby, Phishing Detection in Websites using Parse Tree Validation. *Recent Advances on Engineering, Technology and Computational Sciences (RAETCS)*, 2018, pp. 1-4, doi: 10.1109/RAETCS.2018.8443961.

18. F. Al-Obeidat and E-SM El-Alfy. Hybrid multicriteria fuzzy classification of network traffic patterns, anomalies, and protocols. Personal and Ubiquitous Computing, 2017. pages 1–15.

19. Paul D. Allison. Convergence failures in logistic regression. *SAS Global Forum*. 2018. volume 360. pages 1–11.

20. Riyad Alshammari and A. Nur Zincir-Heywood. Machine learning based encrypted traffic classification: Identifying ssh and skype. *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. IEEE. 2019. pages 1–8.

21. Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Georgios Sakkis, Constantine D. Spyropoulos, and Panagiotis Stamatopoulos. Learning to fi spam e-mail: A comparison of a naive bayesian and a memory-based approach. arXiv preprint cs/0009009. 2010.

22. Anti-Phishing Working Group. Global Phishing Report 2H 2014. URL: http://docs.apwg. org/reports/APWG_Global_Phishing_Report_2H_2014.pdf. (дата звернення: 25.04.2022).

23. Anti-Phishing Working Group. Global Phishing Survey: Trends and Domain Name Use in 2016. http://docs.apwg.org/reports/APWG_Global_Phishing_Report_ 2015-2016.pdf. 2017. (дата звернення: 25.04.2022).

24. Simone Aonzo, Alessio Merlo, Giulio Tavella, and Yanick Fratantonio. Phishing attacks on modern android. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18*, ACM. 2018.

25. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

26. Mohammad Behdad, Luigi Barone, Mohammed Bennamoun, and Tim

French. Natureinspired techniques in the context of fraud detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2012. Vol. 42(6). pp. 1273–1290.

27. Yoshua Bengio, R´ejean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*. 2003. 3(Feb). pp. 1137–1155.

28. Giovanni Bottazzi, Emiliano Casalicchio, Davide Cingolani, Fabio Marturana, and Marco Piu. Mp-shield: a framework for phishing detection in mobile devices. In Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM). 2015 IEEE International Conference on. IEEE. 2015. pages 1977– 1983.

29. Madhusudhanan Chandrasekaran, Krishnan Narayanan, and Shambhu Upadhyaya. Phishing email detection based on structural properties. In NYS cyber security conference, Albany, New York, 2006. volume 3.

30. Ee Hung Chang, Kang Leng Chiew, San Nah Sze, and Wei King Tiong. Phishing detection via identification of website identity. In 2013 International Conference on IT Convergence and Security. *ICITCS 2013*. IEEE. 2013. pages 1–4.

31. Moses S. Charikar. Similarity estimation techniques from rounding algorithms. *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM. 2002. pages 380–388.

32. Teh-Chung Chen, Scott Dick, and James Miller. Detecting visually similar web pages: Application to phishing detection. *ACM Trans. Internet Technol*. June 2010. Vol. 10(2). pp. 1–5. 38.

33. Kyunghyun Cho, Bart Van Merri¨enboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078. 2014.

34. Igino Corona, Battista Biggio, Matteo Contini, Luca Piras, Roberto Corda, Mauro Mereu, Guido Mureddu, Davide Ariu, and Fabio Roli. Deltaphish: Detecting phishing webpages in compromised websites. *In European Symposium on*

Research in Computer Security, Springer. 2017. pages 370–388.

35. Marco Cova, Christopher Kruegel, and Giovanni Vigna. There Is No Free Phish: An Analysis of "Free" and Live Phishing Kits. *In 2nd Conference on USENIX Workshop on Offensive Technologies (WOOT),* San Jose, CA, 2008. volume 8. pages 1–8.

36. Elisabeth Crawford, Irena Koprinska, and Jon Patrick. Phrases and feature selection in e-mail classification. In *ADCS*. 2004. pages 59–62.

37. Bin Cui, Anirban Mondal, Jialie Shen, Gao Cong, and Kian-Lee Tan. On effective e-mail classification via neural networks. *In International Conference on Database and Expert Systems Applications, Springer*. 2005. pages 85–94.

38. Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. *Proceedings of the 2005 symposium on Usable privacy and security*. ACM. 2005. pages 77–88.

39. Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. *Proceedings of the 16th international conference on World Wide Web*. ACM. 2007. pages 649– 656.

40. Anthony Y. Fu, Liu Wenyin, and Xiaotie Deng. Detecting phishing web pages with visual similarity assessment based on earth mover's distance (emd). *IEEE transactions on dependable and secure computing*. 2006. Vol. 3(4) pp. 301–311.

41. Evgeniy Gabrilovich and Alex Gontmakher. The homograph attack. *Communications of the ACM*. 2002. Vol. 45(2). pp. 128.

42. Sujata Garera, Niels Provos, Monica Chew, and Aviel D. Rubin. A framework for detection and measurement of phishing attacks. *Proceedings of the 2007 ACM workshop on Recurring Malcode - WORM '07*. ACM. New York, NY, 2007. pages 1–8.

43. Guang Gang Geng, Xiao Dong Lee, Wei Wang, and Shian Shyong Tseng. Favicon - a clue to phishing sites detection. *eCrime Researchers Summit (eCRS)*. Sept 2013. 2013. pages 1–10.

44. R. Gowtham and Ilango Krishnamurthi. A comprehensive and efficacious architecture for detecting phishing webpages. *Computers & Security*, 40:23–37, 2014.

45. Isredza Rahmi A. Hamid and Jemal Abawajy. Hybrid feature selection for phishing email detection. *International Conference on Algorithms and Architectures for Parallel Processing, Springer,* 2011. pages 266–275.

46. Xiao Han, Nizar Kheir, and Davide Balzarotti. Phisheye: Live monitoring of sandboxed phishing kits. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM. 2016. pages 1402–1413.

47. Sepp Hochreiter and Ju¨rgen Schmidhuber. Long short-term memory. *Neural computation*. 1997. Vol. 9(8). pp. 1735–1780.

48. Martin Hus´ak, Milan C˘ erma´k, Toma´˘s Jirs´ık, and Pavel C˘ eleda. Https traffic analysis and client identifi using passive ssl/tls fi ting. *EURASIP Journal on Information Security 2016*. 2016. Vol. (1). pp. 6.

49. Danesh Irani, Steve Webb, Jonathon Giffin, and Calton Pu. Evolutionary study of phishing. In *ECrime Researchers Summit*, IEEE. 2008. 2008. pages 1–10.

50. [36] Mahmoud Khonji, Youssef Iraqi, and Andrew Jones. Lexical url analysis for discrim- inating phishing and legitimate websites. In Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference, pages 109–115. ACM, 2011.

51. Sophie Le Page, Guy-Vincent Jourdan, Gregor V. Bochmann, Jason Flood, and Iosif- Viorel Onut. Using url shorteners to compare phishing and malware attacks. *APWG Symposium on Electronic Crime Research (eCrime). IEEE 2018*. 2018. pages 1–13.

52. Sophie Le Page, Guy-Vincent Jourdan, Gregor V Bochmann, Iosif-Viorel Onut, and Jason Flood. Domain classifier: Compromised machines versus malicious registrations. *International Conference on Web Engineering*, *Springer*, 2019. pages 265–279.

53. Wenyin Liu, Gang Liu, Bite Qiu, and Xiaojun Quan. Antiphishing through Phishing Target Discovery. *IEEE Internet Computing*. 2012. Vol. 16(2). pp. 52–61.

54. Christian Ludl, Sean McAllister, Engin Kirda, and Christopher Kruegel. On the effectiveness of techniques to detect phishing sites. *International Conference*

on Detection of Intrusions and Malware, and Vulnerability Assessment, *Springer,* 2017. pages 20–39.

55. Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. *Proceedings of the 16th international conference on World Wide Web*, ACM. 2017. pp.141–150.

56. DE1-SoC Development Kit - hardware design platform built around the Altera System-on-Chip (SoC) FPGA. URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl ?Language=English&CategoryNo=167&No=836

Додаток А

(обов’язковий)

Код системного програмного забезпечення емулятора виявлення кібер-загроз типу «фішинг»

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;
use work.lloyds_algorithm_pkg.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity memory_mgmt is
    port (
        clk : in std_logic;
        sclr : in std_logic;
        rd : in std_logic;
        rd_node_addr : in node_address_type;
        k : in centre_index_type;
        wr_init_node : in std_logic;
        wr_node_address_init : in node_address_type;
        wr_node_data_init : in node_data_type;
        wr_init_pos : in std_logic;
        wr_centre_list_pos_address_init : in centre_index_type;
        wr_centre_list_pos_data_init : in data_type;
        valid : out std_logic_vector(0 to PARALLEL_UNITS-1);
        rd_node_data : out par_node_data_type;
        rd_centre_list_pos_data : out par_data_type
    );
end memory_mgmt;
```

```
architecture Behavioral of memory_mgmt is

    constant MEM_LAT : integer := 2;

    type rd_state_type is (idle, reading_centre_list);

        type pos_addr_delay_type is array(1 to PARALLEL_UNITS-1) of
centre_index_type;

    component node_memory
        port (
            clka : IN STD_LOGIC;
            wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            addra : IN STD_LOGIC_VECTOR(NODE_POINTER_BITWIDTH-1 DOWNTO 0);
            dina : IN STD_LOGIC_VECTOR(D*COORD_BITWIDTH-1 DOWNTO 0);
            clkb : IN STD_LOGIC;
            addrb : IN STD_LOGIC_VECTOR(NODE_POINTER_BITWIDTH-1 DOWNTO 0);
            doutb : OUT STD_LOGIC_VECTOR(D*COORD_BITWIDTH-1 DOWNTO 0)
        );
    end component;

    component centre_positions_memory_top
        port (
            clk : in std_logic;
            wea : in std_logic_vector(0 to PARALLEL_UNITS-1);
            addra : in par_centre_index_type;
            dina : in par_data_type;
            addrb : in par_centre_index_type;
            doutb : out par_data_type
        );
    end component;

    signal rd_state : rd_state_type;
    signal rd_counter_done : std_logic;
    signal rd_counter : centre_index_type;

    signal reading_centres : std_logic;
    signal delay_line : std_logic_vector(0 to MEM_LAT+PARALLEL_UNITS-1-1);

    signal rd_k_reg : centre_index_type;
    signal rd_node_address_reg : node_address_type;

    signal wr_node_reg : std_logic;
```

```
        signal wr_node_address_reg : node_address_type;
        signal wr_node_data_reg : node_data_type;
         signal tmp_wr_node_address : std_logic_vector(NODE_POINTER_BITWIDTH-1 downto
0);
        signal tmp_wr_node_data_in : std_logic_vector(D*COORD_BITWIDTH-1 downto 0);
        signal tmp_rd_node_data_out : std_logic_vector(D*COORD_BITWIDTH-1 downto 0);

         signal tmp_rd_node_address : std_logic_vector(NODE_POINTER_BITWIDTH-1 downto
0);

        signal wr_pos_reg : std_logic;
        signal wr_pos_address_reg : centre_index_type;
        signal wr_pos_data_reg : data_type;
         signal tmp_wr_centre_list_pos_data_init : std_logic_vector(D*COORD_BITWIDTH-1
downto 0);
        signal tmp_centre_pos_out : std_logic_vector(D*COORD_BITWIDTH-1 downto 0);

        signal pos_wea : std_logic_vector(0 to PARALLEL_UNITS-1);
        signal pos_wr_address : par_centre_index_type;
        signal pos_wr_data : par_data_type;
        signal pos_rd_address : par_centre_index_type;
        signal pos_rd_data : par_data_type;
        signal pos_addr_delay : pos_addr_delay_type;

    begin

        --writing to memories

        -- delay buffer wr input by one cycle due to state machine
        wr_input_reg_proc : process(clk)
        begin
            if rising_edge(clk) then
                if sclr = '1' then
                    wr_node_reg <= '0';
                    wr_pos_reg <= '0';
                else
                    wr_node_reg <= wr_init_node;
                    wr_pos_reg <= wr_init_pos;
                end if;

                wr_node_address_reg <= wr_node_address_init;
                wr_node_data_reg <= wr_node_data_init;
```

```
        wr_pos_address_reg <= wr_centre_list_pos_address_init;
        wr_pos_data_reg <= wr_centre_list_pos_data_init;

    end if;
end process wr_input_reg_proc;

tmp_wr_node_address <= std_logic_vector(wr_node_address_reg);
tmp_wr_node_data_in <= nodedata_2_stdlogic(wr_node_data_reg);

tmp_wr_centre_list_pos_data_init <= datapoint_2_stdlogic(wr_pos_data_reg);



-- reading memories

fsm_proc : process(clk)
begin
    if rising_edge(clk) then

        if sclr = '1' then
            rd_state <= idle;
        elsif rd_state = idle AND rd = '1' then
            rd_state <= reading_centre_list;
        elsif rd_state = reading_centre_list AND rd_counter_done = '1' then
            rd_state <= idle;
        end if;

    end if;
end process fsm_proc;

counter_proc : process(clk)
begin
    if rising_edge(clk) then

        if sclr = '1' then
            rd_counter <= (others => '0');
        else
            if rd_state <= idle then
                rd_counter <= (others => '0');
            else
                rd_counter <= rd_counter+1;
            end if;
        end if;
```

```
            end if;
        end process counter_proc;

            rd_counter_done <= '1' WHEN rd_counter = rd_k_reg AND rd_state =
reading_centre_list ELSE '0';

        addr_reg_proc : process(clk)
        begin
            if rising_edge(clk) then
                if rd = '1' then
                    rd_node_address_reg <= rd_node_addr;
                    rd_k_reg <= k;
                end if;
            end if;
        end process addr_reg_proc;


        tmp_rd_node_address <= std_logic_vector(rd_node_address_reg);


        node_memory_inst : node_memory
            port  map(
                clka => clk,
                wea(0) => wr_node_reg,
                addra => tmp_wr_node_address,
                dina => tmp_wr_node_data_in,
                clkb => clk,
                addrb => tmp_rd_node_address,
                doutb => tmp_rd_node_data_out
            );

        G_PAR_0 : for I in 0 to PARALLEL_UNITS-1 generate
            rd_node_data(I) <= stdlogic_2_nodedata(tmp_rd_node_data_out);
        end generate G_PAR_0;


        G_PAR_1 : for I in 0 to PARALLEL_UNITS-1 generate
            pos_wea(I) <= wr_pos_reg;
            pos_wr_address(I) <= wr_pos_address_reg;
            pos_wr_data(I) <= wr_pos_data_reg;
        end generate G_PAR_1;
```

```
            G_PAR_1_1 : if PARALLEL_UNITS > 1 generate
                pos_addr_delay_proc : process(clk)
                begin
                    if rising_edge(clk) then
                        pos_addr_delay(1) <= rd_counter;
                            pos_addr_delay(2 to PARALLEL_UNITS-1) <= pos_addr_delay(1 to
PARALLEL_UNITS-2);
                    end if;
                end process pos_addr_delay_proc;
            end generate G_PAR_1_1;

            pos_rd_address(0) <= rd_counter;
            G_PAR_2 : for I in 1 to PARALLEL_UNITS-1 generate
                pos_rd_address(I) <= pos_addr_delay(I);
            end generate G_PAR_2;

            centre_positions_memory_top_inst : centre_positions_memory_top
                port map (
                    clk => clk,
                    wea => pos_wea,
                    addra => pos_wr_address,
                    dina => pos_wr_data,
                    addrb => pos_rd_address,
                    doutb => pos_rd_data
                );



            reading_centres <= '1' WHEN rd_state = reading_centre_list ELSE '0';

            dely_line_proc : process(clk)
            begin
                if rising_edge(clk) then
                    if sclr = '1' then
                        delay_line <= (others => '0');
                    else
                        delay_line(0) <= reading_centres;
                           delay_line(1 to MEM_LAT+PARALLEL_UNITS-1-1) <= delay_line(0 to
MEM_LAT+PARALLEL_UNITS-1-2);
                    end if;
                end if;
            end process dely_line_proc;

            G_PAR_3 : for I in 0 to PARALLEL_UNITS-1 generate
```

```
        valid(I) <= delay_line(MEM_LAT+I-1);
        rd_centre_list_pos_data(I) <= pos_rd_data(I);
    end generate G_PAR_3;

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.all;
use ieee.math_real.all;



-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;



package lloyds_algorithm_pkg is

    constant N_MAX : integer := 32768;
    constant D : integer := 3;
    constant K_MAX : integer := 256;
    constant L_MAX : integer := 6;

    constant PARALLEL_UNITS : integer := 1;
    constant USE_DSP_FOR_ADD : boolean := false;

    constant COORD_BITWIDTH : integer := 16;
    constant COORD_BITWIDTH_EXT : integer := 32;
    constant INDEX_BITWIDTH : integer := integer(ceil(log2(real(K_MAX))));
    constant NODE_POINTER_BITWIDTH : integer := integer(ceil(log2(real(N_MAX))));

    constant MUL_FRACTIONAL_BITS : integer := 6;
    constant MUL_INTEGER_BITS : integer := 12;
    constant MUL_BITWIDTH : integer := MUL_INTEGER_BITS+MUL_FRACTIONAL_BITS;
    constant MUL_CORE_LATENCY : integer := 3;

    constant SYNTHESIS : boolean := false;

    subtype coord_type is std_logic_vector(COORD_BITWIDTH-1 downto 0);
```

```
        subtype centre_index_type is unsigned(INDEX_BITWIDTH-1 downto 0);
        subtype node_index_type is unsigned(NODE_POINTER_BITWIDTH-1 downto 0);
         subtype node_address_type is std_logic_vector(NODE_POINTER_BITWIDTH-1 downto
0);


        type data_type is array(0 to D-1) of coord_type;

        subtype coord_type_ext is std_logic_vector(COORD_BITWIDTH_EXT-1 downto 0);
        type data_type_ext is array(0 to D-1) of coord_type_ext;

        type node_data_type is
            record
                position : data_type;
            end record;
        -- size of node_data_type : D*COORD_BITWIDTH


        -- array types for parallelism
              type   par_centre_index_type   is   array(0   to   PARALLEL_UNITS-1)   of
centre_index_type;
        type par_coord_type is array(0 to PARALLEL_UNITS-1) of coord_type;
        type par_coord_type_ext is array(0 to PARALLEL_UNITS-1) of coord_type_ext;
        type par_data_type is array(0 to PARALLEL_UNITS-1) of data_type;
        type par_data_type_ext is array(0 to PARALLEL_UNITS-1) of data_type_ext;
        type par_node_data_type is array(0 to PARALLEL_UNITS-1) of node_data_type;

        -- helper functions
        function stdlogic_2_datapoint(c : std_logic_vector) return data_type;
        function stdlogic_2_datapoint_ext(c : std_logic_vector) return data_type_ext;
        function datapoint_2_stdlogic(c : data_type) return std_logic_vector;
        function datapoint_ext_2_stdlogic(c : data_type_ext) return std_logic_vector;
        function nodedata_2_stdlogic(n : node_data_type) return std_logic_vector;
        function stdlogic_2_nodedata(n : std_logic_vector) return node_data_type;
        function saturate(val : std_logic_vector) return std_logic_vector;
             function   sext(val   :   std_logic_vector;   length   :   integer)   return
std_logic_vector;
             function   zext(val   :   std_logic_vector;   length   :   integer)   return
std_logic_vector;
        function conv_ext_2_normal(val : data_type_ext) return data_type;
        function conv_normal_2_ext(val : data_type) return data_type_ext;

    end lloyds_algorithm_pkg;
```

```
package body lloyds_algorithm_pkg is



    function datapoint_2_stdlogic(c : data_type) return std_logic_vector is
        variable result : std_logic_vector(D*COORD_BITWIDTH-1 downto 0);
    begin
        for I in 0 to D-1 loop
                    result((I+1)*COORD_BITWIDTH-1  downto  I*COORD_BITWIDTH)  :=
std_logic_vector(c(I));
        end loop;
        return result;
    end datapoint_2_stdlogic;

     function datapoint_ext_2_stdlogic(c : data_type_ext) return std_logic_vector
is
        variable result : std_logic_vector(D*COORD_BITWIDTH_EXT-1 downto 0);
    begin
        for I in 0 to D-1 loop
               result((I+1)*COORD_BITWIDTH_EXT-1 downto I*COORD_BITWIDTH_EXT) :=
std_logic_vector(c(I));
        end loop;
        return result;
    end datapoint_ext_2_stdlogic;



    function stdlogic_2_datapoint(c : std_logic_vector) return data_type is
        variable result : data_type;
    begin
        for I in 0 to D-1 loop
            result(I) := c((I+1)*COORD_BITWIDTH-1 downto I*COORD_BITWIDTH);
        end loop;
        return result;
    end stdlogic_2_datapoint;

     function stdlogic_2_datapoint_ext(c : std_logic_vector) return data_type_ext
is
        variable result : data_type_ext;
    begin
        for I in 0 to D-1 loop
```

```
                                    result(I)  :=  c((I+1)*COORD_BITWIDTH_EXT-1  downto
I*COORD_BITWIDTH_EXT);
                end loop;
                return result;
            end stdlogic_2_datapoint_ext;




            function nodedata_2_stdlogic(n : node_data_type) return std_logic_vector is
                variable result : std_logic_vector(D*COORD_BITWIDTH-1 downto 0);
            begin

                result := datapoint_2_stdlogic(n.position);
                return result;

            end nodedata_2_stdlogic;



            function stdlogic_2_nodedata(n : std_logic_vector) return node_data_type is
                variable result : node_data_type;
            begin

                result.position := stdlogic_2_datapoint(n);

                return result;

            end stdlogic_2_nodedata;


            function saturate(val : std_logic_vector) return std_logic_vector is
                variable val_msb : std_logic;
                                                        variable      comp      :
std_logic_vector((val'length-MUL_INTEGER_BITS-MUL_FRACTIONAL_BITS)-1 downto 0);
                variable result : std_logic_vector(MUL_INTEGER_BITS+MUL_FRACTIONAL_BITS-1
downto 0);
            begin

                if MUL_INTEGER_BITS+MUL_FRACTIONAL_BITS < val'length then

                    val_msb := val(val'length-1);
```

```
                for I in (val'length-MUL_INTEGER_BITS-MUL_FRACTIONAL_BITS)-1 downto 0
loop
                    comp(I) := val_msb;
                end loop;

                 if val(val'length-2 downto MUL_INTEGER_BITS+MUL_FRACTIONAL_BITS-1) =
comp then
                    result := val(MUL_INTEGER_BITS+MUL_FRACTIONAL_BITS-1 downto 0);
                else
                    result(MUL_INTEGER_BITS+MUL_FRACTIONAL_BITS-1) := val_msb;
                          result(MUL_INTEGER_BITS+MUL_FRACTIONAL_BITS-2 downto 0) :=
(others => NOT(val_msb));
                end if;

            else

                result := sext(val,MUL_INTEGER_BITS+MUL_FRACTIONAL_BITS);

            end if;

            return result;

        end saturate;



             function  sext(val  :  std_logic_vector;  length  :  integer)  return
std_logic_vector is
            variable val_msb : std_logic;
            variable result : std_logic_vector(length-1 downto 0);
        begin
            val_msb := val(val'length-1);
            result(val'length-1 downto 0) := val;
            result(length-1 downto val'length) := (others => val_msb);
            return result;
        end sext;


             function  zext(val  :  std_logic_vector;  length  :  integer)  return
std_logic_vector is
            variable result : std_logic_vector(length-1 downto 0);
        begin
            result(val'length-1 downto 0) := val;
```

```
        result(length-1 downto val'length) := (others => '0');
        return result;
    end zext;


    function conv_ext_2_normal(val : data_type_ext) return data_type is
        variable result : data_type;
    begin
        for I in 0 to D-1 loop
            result(I) := val(I)(COORD_BITWIDTH-1 downto 0);
        end loop;
        return result;
    end conv_ext_2_normal;


    function conv_normal_2_ext(val : data_type) return data_type_ext is
        variable result : data_type_ext;
    begin
        for I in 0 to D-1 loop
            result(I) := sext(val(I),COORD_BITWIDTH_EXT);
        end loop;
        return result;
    end conv_normal_2_ext;

end package body;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;
use work.lloyds_algorithm_pkg.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity lloyds_algorithm_core is
    port (
```

```
            clk : in std_logic;
            sclr : in std_logic;
            start : in std_logic;
            -- initial parameters
            n : in node_index_type;
            k : in centre_index_type;
            -- init node and centre memory
            wr_init_node : in std_logic;
            wr_node_address_init : in node_address_type;
            wr_node_data_init : in node_data_type;
            wr_init_pos : in std_logic;
            wr_centre_list_pos_address_init : in centre_index_type;
            wr_centre_list_pos_data_init : in data_type;
            -- access centre buffer
            rdo_centre_buffer : in std_logic;
            centre_buffer_addr : in centre_index_type;
            valid : out std_logic;
            wgtCent_out : out data_type_ext;
            sum_sq_out : out coord_type_ext;
            count_out : out coord_type;
            -- processing done
            rdy : out std_logic
        );
    end lloyds_algorithm_core;

    architecture Behavioral of lloyds_algorithm_core is

        type state_type is (idle, init, processing_phase, done);
        type schedule_state_type is (free, busy, wait_cycle);

                    type    par_element_type_ext    is    array(0    to    D-1)    of
std_logic_vector(PARALLEL_UNITS*COORD_BITWIDTH_EXT-1 downto 0);
                  type    par_element_type_ext_sum    is    array(0    to    D-1)    of
std_logic_vector(COORD_BITWIDTH_EXT+integer(ceil(log2(real(PARALLEL_UNITS))))-1   downto
0);

        component memory_mgmt
            port (
                clk : in std_logic;
                sclr : in std_logic;
                rd : in std_logic;
                rd_node_addr : in node_address_type;
                k : in centre_index_type;
```

```
        wr_init_node : in std_logic;
        wr_node_address_init : in node_address_type;
        wr_node_data_init : in node_data_type;
        wr_init_pos : in std_logic;
        wr_centre_list_pos_address_init : in centre_index_type;
        wr_centre_list_pos_data_init : in data_type;
        valid : out std_logic_vector(0 to PARALLEL_UNITS-1);
        rd_node_data : out par_node_data_type;
        rd_centre_list_pos_data : out par_data_type
    );
end component;

component process_node is
    port (
        clk : in std_logic;
        sclr : in std_logic;
        nd : in std_logic;
        u_in : in node_data_type;
        centre_positions_in : in data_type;
        rdy : out std_logic;
        final_index_out : out centre_index_type;
        sum_sq_out : out coord_type_ext;
        u_out : out node_data_type
    );
end component;

component centre_buffer_mgmt
    port (
        clk : in std_logic;
        sclr : in std_logic;
        init : in std_logic;
        addr_in_init : in centre_index_type;
        nd : in std_logic;
        request_rdo : in std_logic;
        addr_in : in centre_index_type;
        wgtCent_in : in data_type_ext;
        sum_sq_in : in coord_type_ext;
        count_in : in coord_type;
        valid : out std_logic;
        wgtCent_out : out data_type_ext;
        sum_sq_out : out coord_type_ext;
        count_out : out coord_type
    );
```

```
            end component;

            component adder_tree
                generic (
                    USE_DSP_FOR_ADD : boolean := true;
                    NUMBER_OF_INPUTS : integer := 4;
                    INPUT_BITWIDTH : integer := 16
                );
                port (
                    clk : in std_logic;
                    sclr : in std_logic;
                    nd : in std_logic;
                    sub :  in std_logic;
                     input_string : in std_logic_vector(NUMBER_OF_INPUTS*INPUT_BITWIDTH-1
downto 0);
                    rdy : out std_logic;
                                                                    output     :     out
std_logic_vector(INPUT_BITWIDTH+integer(ceil(log2(real(NUMBER_OF_INPUTS))))-1 downto 0)
                );
            end component;

            -- fsm
            signal state : state_type;
            signal start_processing : std_logic;
            signal first_output : std_logic;
            signal processing_done_counter : node_index_type;
            signal processing_done : std_logic;
            signal processing_done_reg : std_logic;

            -- scheduler
            signal schedule_state : schedule_state_type;
            signal schedule_counter : centre_index_type;
            signal schedule_node_counter : node_index_type;
            signal schedule_node_counter_reg : node_index_type;
            signal schedule_counter_done : std_logic;
            signal schedule_first : std_logic;
            signal schedule_next : std_logic;
            signal schedule_par_not_yet_matched : std_logic;

            -- memory mgmt
            signal memory_mgmt_rd : std_logic;
            signal memory_data_valid : std_logic_vector(0 to PARALLEL_UNITS-1);
            signal rd_node_addr : node_address_type;
```

```
        signal rd_k : centre_index_type;
        signal rd_node_data : par_node_data_type;
        signal rd_centre_positions : par_data_type;

        -- process_node
        signal pn_final_index_out : par_centre_index_type;
        signal pn_sum_sq_out : par_coord_type_ext;
        signal pn_rdy : std_logic_vector(0 to PARALLEL_UNITS-1);
        signal pn_u_out : par_node_data_type;

        -- centre buffer mgmt
        signal tmp_addr : par_centre_index_type;
        signal centre_buffer_valid : std_logic_vector(0 to PARALLEL_UNITS-1);
        signal centre_buffer_wgtCent : par_data_type_ext;
        signal centre_buffer_sum_sq : par_coord_type_ext;
        signal centre_buffer_count : par_coord_type;

        -- adder tree
                                        signal         at_input_string_count         :
std_logic_vector(PARALLEL_UNITS*COORD_BITWIDTH-1 downto 0);
        signal at_count_rdy : std_logic;
                                                signal          at_count_out          :
std_logic_vector(COORD_BITWIDTH+integer(ceil(log2(real(PARALLEL_UNITS))))-1 downto 0);
        signal at_input_string_wgtCent : par_element_type_ext;
        signal at_wgtCent_rdy : std_logic;
        signal at_wgtCent_out : par_element_type_ext_sum;
                                       signal         at_input_string_sum_sq         :
std_logic_vector(PARALLEL_UNITS*COORD_BITWIDTH_EXT-1 downto 0);
        signal at_sum_sq_rdy : std_logic;
                                                signal          at_sum_sq_out          :
std_logic_vector(COORD_BITWIDTH_EXT+integer(ceil(log2(real(PARALLEL_UNITS))))-1   downto
0);

        -- output
        signal tmp_valid : std_logic;
        signal tmp_count_out : coord_type;
        signal tmp_wgtCent_out : data_type_ext;
        signal tmp_sum_sq_out : coord_type_ext;

        -- stats not synthesised
        signal cycle_count_enable : std_logic;
        signal first_start : std_logic := '0';
        signal cycle_count : unsigned(31 downto 0);
```

```
    begin

        G0_SYNTH : if SYNTHESIS = false generate
            -- some statistics
            stats_proc : process(clk)
            begin
                if rising_edge(clk) then

                    if sclr = '1' then
                        cycle_count_enable <= '0';
                      elsif state = processing_phase AND processing_done_counter /= n
then
                        cycle_count_enable <= '1';
                    elsif processing_done_counter = n then
                        cycle_count_enable <= '0';
                    end if;

                    if start = '1' then
                        first_start <= '1'; -- latch the first start assertion
                    end if;

                    if first_start = '0' then
                        cycle_count <= (others => '0');
                    else -- count cycles for all iterations
                        cycle_count <= cycle_count+1;
                    end if;

                end if;
            end process stats_proc;
        end generate G0_SYNTH;

        fsm_proc : process(clk)
        begin
            if rising_edge(clk) then
                if sclr = '1' then
    --                state <= idle;
    --            elsif state = idle AND wr_init_node = '1' then
                    state <= init;
                elsif state = init AND start = '1' then
                    state <= processing_phase;
                   elsif state = processing_phase AND processing_done_reg = '1' AND
schedule_next = '1' then
```

```
            state <= done;
        elsif state = done then
            state <= init;
        end if;
    end if;
end process fsm_proc;

start_processing <= '1' WHEN state = init AND start = '1' ELSE '0';


-- scheduler (get next node from node memory)
scheduler_proc : process(clk)
begin
    if rising_edge(clk) then

        if sclr = '1' then
            schedule_state <= free;
        elsif schedule_state = free AND schedule_first = '1' then
            schedule_state <= busy;
        elsif schedule_state = busy AND schedule_counter_done = '1' then
            schedule_state <= free;
        end if;

        if sclr = '1' OR schedule_state = free then
            schedule_counter <= to_unsigned(0,INDEX_BITWIDTH);
        elsif schedule_state = busy then
            schedule_counter <= schedule_counter+1;
        end if;

        if sclr = '1' then
            schedule_node_counter <= (others => '0');
            processing_done_reg <= '0';
        else
            if schedule_next = '1' then
                schedule_node_counter <= schedule_node_counter+1;
            end if;

            if processing_done = '1' then
                processing_done_reg <= '1';
            end if;
        end if;

    end if;
```

```
            end process scheduler_proc;

            schedule_first <= '1' WHEN schedule_state = free AND state = processing_phase
ELSE '0';
                    schedule_next   <=   '1'   WHEN   schedule_state   =   busy   AND
schedule_par_not_yet_matched = '1' ELSE '0';

            schedule_counter_done <= '1' WHEN schedule_counter = k ELSE '0';

                    schedule_par_not_yet_matched   <=   '1'   WHEN   schedule_counter   <
to_unsigned(PARALLEL_UNITS,INDEX_BITWIDTH) ELSE '0';
                processing_done  <=  '1'  WHEN  schedule_node_counter  =  n  AND  state  =
processing_phase ELSE '0';

            memory_mgmt_rd <= schedule_next;
            rd_node_addr <= std_logic_vector(schedule_node_counter);
            rd_k <= k;

            memory_mgmt_inst : memory_mgmt
                port map (
                    clk => clk,
                    sclr => sclr,
                    rd => memory_mgmt_rd,
                    rd_node_addr => rd_node_addr,
                    k => rd_k,
                    wr_init_node => wr_init_node,
                    wr_node_address_init => wr_node_address_init,
                    wr_node_data_init => wr_node_data_init,
                    wr_init_pos => wr_init_pos,
                    wr_centre_list_pos_address_init => wr_centre_list_pos_address_init,
                    wr_centre_list_pos_data_init => wr_centre_list_pos_data_init,
                    valid => memory_data_valid,
                    rd_node_data => rd_node_data,
                    rd_centre_list_pos_data => rd_centre_positions
                );

            G_PAR_1 : for I in 0 to PARALLEL_UNITS-1 generate

                process_node_inst : process_node
                    port map(
                        clk => clk,
                        sclr => sclr,
                        nd => memory_data_valid(I),
```

```
                    u_in => rd_node_data(I),
                    centre_positions_in => rd_centre_positions(I),
                    rdy => pn_rdy(I),
                    final_index_out => pn_final_index_out(I),
                    sum_sq_out => pn_sum_sq_out(I),
                    u_out => pn_u_out(I)
                );

        end generate G_PAR_1;


        G_PAR_2 : for I in 0 to PARALLEL_UNITS-1 generate

              tmp_addr(I) <= pn_final_index_out(I) WHEN rdo_centre_buffer = '0' ELSE
centre_buffer_addr;

            centre_buffer_mgmt_inst : centre_buffer_mgmt
                port map (
                    clk => clk,
                    sclr => sclr,
                    nd => pn_rdy(I),
                    init => wr_init_pos,
                    addr_in_init => wr_centre_list_pos_address_init,
                    request_rdo => rdo_centre_buffer,
                    addr_in => tmp_addr(I),
                    wgtCent_in => conv_normal_2_ext(pn_u_out(I).position),
                    sum_sq_in => pn_sum_sq_out(I),
                    count_in => std_logic_vector(to_unsigned(1,COORD_BITWIDTH)),
                    valid => centre_buffer_valid(I),
                    wgtCent_out => centre_buffer_wgtCent(I),
                    sum_sq_out => centre_buffer_sum_sq(I),
                    count_out => centre_buffer_count(I)
                );

             at_input_string_count((I+1)*COORD_BITWIDTH-1 downto I*COORD_BITWIDTH) <=
centre_buffer_count(I);
                            at_input_string_sum_sq((I+1)*COORD_BITWIDTH_EXT-1   downto
I*COORD_BITWIDTH_EXT) <= centre_buffer_sum_sq(I);

            G_PAR_2_1 : for J in 0 to D-1 generate
                         at_input_string_wgtCent(J)((I+1)*COORD_BITWIDTH_EXT-1  downto
I*COORD_BITWIDTH_EXT) <= centre_buffer_wgtCent(I)(J);
            end generate G_PAR_2_1;
```

```
        end generate G_PAR_2;



        G_PAR_3 : if PARALLEL_UNITS > 1 generate

            adder_tree_inst_count : adder_tree
                generic map (
                    USE_DSP_FOR_ADD => USE_DSP_FOR_ADD,
                    NUMBER_OF_INPUTS => PARALLEL_UNITS,
                    INPUT_BITWIDTH => COORD_BITWIDTH
                )
                port map(
                    clk => clk,
                    sclr => sclr,
                    nd => centre_buffer_valid(0),
                    sub => '0',
                    input_string => at_input_string_count,
                    rdy => at_count_rdy,
                    output => at_count_out
                );

            G_PAR_3_1 : for J in 0 to D-1 generate
                adder_tree_inst_wgtCent : adder_tree
                    generic map (
                        USE_DSP_FOR_ADD => USE_DSP_FOR_ADD,
                        NUMBER_OF_INPUTS => PARALLEL_UNITS,
                        INPUT_BITWIDTH => COORD_BITWIDTH_EXT
                    )
                    port map(
                        clk => clk,
                        sclr => sclr,
                        nd => centre_buffer_valid(0),
                        sub => '0',
                        input_string => at_input_string_wgtCent(J),
                        rdy => at_wgtCent_rdy,
                        output => at_wgtCent_out(J)
                    );
                  tmp_wgtCent_out(J) <= at_wgtCent_out(J)(COORD_BITWIDTH_EXT-1 downto
0);
             end generate G_PAR_3_1;
```

```
            adder_tree_inst_sum_sq : adder_tree
                    generic map (
                        USE_DSP_FOR_ADD => USE_DSP_FOR_ADD,
                        NUMBER_OF_INPUTS => PARALLEL_UNITS,
                        INPUT_BITWIDTH => COORD_BITWIDTH_EXT
                    )
                    port map(
                        clk => clk,
                        sclr => sclr,
                        nd => centre_buffer_valid(0),
                        sub => '0',
                        input_string => at_input_string_sum_sq,
                        rdy => at_sum_sq_rdy,
                        output => at_sum_sq_out
                    );

            tmp_valid <= at_count_rdy;
            tmp_count_out <= at_count_out(COORD_BITWIDTH-1 downto 0);
            tmp_sum_sq_out <= at_sum_sq_out(COORD_BITWIDTH_EXT-1 downto 0);

        end generate G_PAR_3;

        G_PAR_4 : if PARALLEL_UNITS = 1 generate

            tmp_valid <= centre_buffer_valid(0);
            tmp_count_out <= centre_buffer_count(0);
            tmp_wgtCent_out <= centre_buffer_wgtCent(0);
            tmp_sum_sq_out <= centre_buffer_sum_sq(0);

        end generate G_PAR_4;


        processing_done_counter_proc : process(clk)
        begin
            if rising_edge(clk) then
                if sclr = '1' then
                    first_output <= '0';
                                                        processing_done_counter   <=
to_unsigned(PARALLEL_UNITS-1,NODE_POINTER_BITWIDTH);
                elsif pn_rdy(PARALLEL_UNITS-1) = '1' then
                    first_output <= '1';
                    if first_output = '1' then
```

```
                                                                processing_done_counter   <=
processing_done_counter+to_unsigned(PARALLEL_UNITS,NODE_POINTER_BITWIDTH);
                        end if;
                    end if;
                end if;
            end process processing_done_counter_proc;


            valid           <= tmp_valid;
            wgtCent_out     <= tmp_wgtCent_out;
            sum_sq_out      <= tmp_sum_sq_out;
            count_out       <= tmp_count_out;

            rdy             <= '1' WHEN processing_done_counter >= n ELSE '0';


        end Behavioral;

        library IEEE;
        use IEEE.STD_LOGIC_1164.ALL;
        USE ieee.numeric_std.all;
        use ieee.math_real.all;
        use work.lloyds_algorithm_pkg.all;

        -- Uncomment the following library declaration if using
        -- arithmetic functions with Signed or Unsigned values
        --use IEEE.NUMERIC_STD.ALL;

        -- Uncomment the following library declaration if instantiating
        -- any Xilinx primitives in this code.
        --library UNISIM;
        --use UNISIM.VComponents.all;

        entity dot_product is
            generic (
                SCALE_MUL_RESULT : integer := 0
            );
            port (
                clk : in std_logic;
                sclr : in std_logic;
                nd : in std_logic;
                point_1 : in data_type;
                point_2 : in data_type;
```

```
            result : out coord_type_ext;
            rdy : out std_logic
        );
    end dot_product;

    architecture Behavioral of dot_product is

        constant LAYERS_TREE_ADDER : integer := integer(ceil(log2(real(D))));

                    type mul_res_array_type is array(0 to D-1) of
std_logic_vector(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1-1 downto 0);

          --type tree_adder_res_array_type is array(0 to LAYERS_TREE_ADDER-1, 0 to
D/2-1) of std_logic_vector(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE_ADDER-1 downto
0);

        component addorsub
            generic (
                USE_DSP : boolean := true;
                A_BITWIDTH : integer := 16;
                B_BITWIDTH : integer := 16;
                RES_BITWIDTH : integer := 16
            );
            port (
                clk : in std_logic;
                sclr : in std_logic;
                nd : in std_logic;
                sub : in std_logic;
                a : in std_logic_vector(A_BITWIDTH-1 downto 0);
                b : in std_logic_vector(B_BITWIDTH-1 downto 0);
                res : out std_logic_vector(RES_BITWIDTH-1 downto 0);
                rdy : out std_logic
            );
        end component;

        component madd
            generic (
                MUL_LATENCY : integer := 3;
                A_BITWIDTH : integer := 16;
                B_BITWIDTH : integer := 16;
                INCLUDE_ADD : boolean := false;
                C_BITWIDTH : integer := 16;
                RES_BITWIDTH : integer := 16
```

```
        );
        port (
            clk : in std_logic;
            sclr : in std_logic;
            nd : in std_logic;
            a : in std_logic_vector(A_BITWIDTH-1 downto 0);
            b : in std_logic_vector(B_BITWIDTH-1 downto 0);
            c : in std_logic_vector(C_BITWIDTH-1 downto 0);
            res : out std_logic_vector(RES_BITWIDTH-1 downto 0);
            rdy : out std_logic
        );
    end component;

    component adder_tree
        generic (
            USE_DSP_FOR_ADD : boolean := true;
            NUMBER_OF_INPUTS : integer := 4;
            INPUT_BITWIDTH : integer := 16
        );
        port (
            clk : in std_logic;
            sclr : in std_logic;
            nd : in std_logic;
            sub :  in std_logic;
             input_string : in std_logic_vector(NUMBER_OF_INPUTS*INPUT_BITWIDTH-1
downto 0);
            rdy : out std_logic;
                                                                output     :     out
std_logic_vector(INPUT_BITWIDTH+integer(ceil(log2(real(NUMBER_OF_INPUTS))))-1 downto 0)
        );
    end component;

    signal tmp_mul_res : mul_res_array_type;
                                signal        tmp_tree_adder_input_string        :
std_logic_vector(D*(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1)-1 downto 0);

    --signal tmp_tree_adder_res : tree_adder_res_array_type;
                                        signal         tmp_tree_adder_res         :
std_logic_vector(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE_ADDER-1 downto 0);

    signal const_0 : std_logic_vector(MUL_BITWIDTH-1 downto 0);

    signal tmp_mul_rdy : std_logic;
```

```
    signal delay_line_tree_adder : std_logic_vector(0 to 2*LAYERS_TREE_ADDER-1);
    signal tmp_final_res : coord_type_ext;

begin

    const_0 <= (others => '0');

    G1: for I in 0 to D-1 generate

        G_FIRST: if I = 0 generate

            madd_inst : madd
                generic map(
                    MUL_LATENCY => MUL_CORE_LATENCY,
                    A_BITWIDTH => MUL_BITWIDTH,
                    B_BITWIDTH => MUL_BITWIDTH,
                    INCLUDE_ADD => false,
                    C_BITWIDTH => MUL_BITWIDTH,
                    RES_BITWIDTH => 2*MUL_BITWIDTH-SCALE_MUL_RESULT+1
                )
                port map (
                    clk => clk,
                    sclr => sclr,
                    nd => nd,
                    a => saturate(point_1(I)),
                    b => saturate(point_2(I)),
                    c => const_0,
                    res => tmp_mul_res(I),
                    rdy => tmp_mul_rdy
                );
        end generate G_FIRST;

        G_OTHER: if I > 0 generate

            madd_inst : madd
                generic map(
                    A_BITWIDTH => MUL_BITWIDTH,
                    B_BITWIDTH => MUL_BITWIDTH,
                    C_BITWIDTH => MUL_BITWIDTH,
                    RES_BITWIDTH => 2*MUL_BITWIDTH-SCALE_MUL_RESULT+1
                )
                port map (
```

```
                        clk => clk,
                        sclr => sclr,
                        nd => nd,
                        a => saturate(point_1(I)),
                        b => saturate(point_2(I)),
                        c => const_0,
                        res => tmp_mul_res(I),
                        rdy => open
                    );
            end generate G_OTHER;

        end generate G1;


        G2 : for I in 0 to D-1 generate
              tmp_tree_adder_input_string((I+1)*(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1)-1
downto I*(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1)) <= tmp_mul_res(I);
        end generate G2;

        adder_tree_inst : adder_tree
            generic map (
                USE_DSP_FOR_ADD => USE_DSP_FOR_ADD,
                NUMBER_OF_INPUTS => D,
                INPUT_BITWIDTH => 2*MUL_BITWIDTH-SCALE_MUL_RESULT+1
            )
            port map (
                clk => clk,
                sclr => sclr,
                nd => tmp_mul_rdy,
                sub => '0',
                input_string => tmp_tree_adder_input_string,
                rdy => rdy,
                output => tmp_tree_adder_res
            );

                                        G3:        if        COORD_BITWIDTH_EXT        <=
2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE_ADDER generate
                                                                --tmp_final_res       <=
tmp_tree_adder_res(LAYERS_TREE_ADDER-1,0)(COORD_BITWIDTH_EXT-1 downto 0);
            tmp_final_res <= tmp_tree_adder_res(COORD_BITWIDTH_EXT-1 downto 0);
        end generate G3;
```

```
                                        G4:        if        COORD_BITWIDTH_EXT        >
2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE_ADDER generate
                  --tmp_final_res(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE_ADDER-1
downto                                        0)                                        <=
tmp_tree_adder_res(LAYERS_TREE_ADDER-1,0)(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE
_ADDER-1 downto 0);
                                          --tmp_final_res(COORD_BITWIDTH_EXT-1    downto
2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE_ADDER)        <=        (others        =>
tmp_tree_adder_res(LAYERS_TREE_ADDER-1,0)(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE
_ADDER-1));
                     tmp_final_res(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE_ADDER-1
downto  0)  <=  tmp_tree_adder_res(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE_ADDER-1
downto 0);
                                           tmp_final_res(COORD_BITWIDTH_EXT-1    downto
2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE_ADDER)        <=        (others        =>
tmp_tree_adder_res(2*MUL_BITWIDTH-SCALE_MUL_RESULT+1+LAYERS_TREE_ADDER-1));
          end generate G4;


          --rdy <= delay_line_tree_adder(2*LAYERS_TREE_ADDER-1);
          result <= tmp_final_res;


      end Behavioral;

      library IEEE;
      use IEEE.STD_LOGIC_1164.ALL;
      use IEEE.NUMERIC_STD.ALL;
      use ieee.math_real.all;
      use work.lloyds_algorithm_pkg.all;

      -- Uncomment the following library declaration if using
      -- arithmetic functions with Signed or Unsigned values
      --use IEEE.NUMERIC_STD.ALL;

      -- Uncomment the following library declaration if instantiating
      -- any Xilinx primitives in this code.
      --library UNISIM;
      --use UNISIM.VComponents.all;

      entity divider_top is
          generic (
              ROUND : boolean := false
```

```
        );
        port (
            clk : in std_logic;
            sclr : in std_logic;
            nd : in std_logic;
            dividend : in data_type_ext;
            divisor : in coord_type;
            rdy : out std_logic;
            quotient : out data_type;
            divide_by_zero : out std_logic
        );
    end divider_top;

    architecture Behavioral of divider_top is

        constant QUOTIENT_BITWIDTH : integer := COORD_BITWIDTH_EXT;
        constant FRACTIONAL_BITWIDTH : integer := COORD_BITWIDTH_EXT-COORD_BITWIDTH;

                    type divider_result_type is array(0 to D-1) of
std_logic_vector(QUOTIENT_BITWIDTH+FRACTIONAL_BITWIDTH-1 downto 0);
        type quotient_type is array(0 to D-1) of std_logic_vector(QUOTIENT_BITWIDTH-1
downto 0);


        component divider
            port (
                aclk : IN std_logic;
                s_axis_divisor_tvalid : IN std_logic;
                --s_axis_divisor_tready : OUT std_logic;
                   s_axis_divisor_tdata : IN std_logic_vector(COORD_BITWIDTH-1 DOWNTO
0);
                s_axis_dividend_tvalid : IN std_logic;
                --s_axis_dividend_tready : OUT std_logic;
                     s_axis_dividend_tdata : IN std_logic_vector(COORD_BITWIDTH_EXT-1
DOWNTO 0);
                m_axis_dout_tvalid : OUT std_logic;
                                                        m_axis_dout_tdata : OUT
std_logic_vector(QUOTIENT_BITWIDTH+FRACTIONAL_BITWIDTH-1 DOWNTO 0)
                --m_axis_dout_tuser : OUT STD_LOGIC_VECTOR(0 DOWNTO 0)
            );
        end component;

        component divider_v3_0
```

```
            port (
                clk : IN std_logic;
                sclr : IN std_logic;
                s_axis_divisor_tvalid : IN std_logic;
                divisor : IN std_logic_vector(COORD_BITWIDTH-1 DOWNTO 0);
                dividend : IN std_logic_vector(COORD_BITWIDTH_EXT-1 DOWNTO 0);
                quotient: OUT std_logic_vector(QUOTIENT_BITWIDTH-1 DOWNTO 0);
                fractional : OUT std_logic_vector(FRACTIONAL_BITWIDTH-1 downto 0)
            );
        end component;

        component dsp_round
            generic (
                BITWIDTH_IN : integer := 32;
                BITWIDTH_OUT : integer := 32
            );
            port (
                sclr             : in std_logic;
                nd               : in std_logic;
                AB_IN            : in std_logic_vector (BITWIDTH_IN-1 downto 0);
                CARRYIN_IN       : in std_logic;
                CLK_IN           : in std_logic;
                C_IN             : in std_logic_vector (BITWIDTH_IN-1 downto 0);
                P_OUT            : out std_logic_vector (BITWIDTH_OUT-1 downto 0);
                rdy              : out std_logic
            );
        end component;

        signal tmp_divisor : coord_type;
        signal divider_result : divider_result_type;
        signal tmp_quotient : quotient_type;
        signal divider_valid : std_logic_vector(0 to D-1);
        signal tmp_divide_by_zero : std_logic_vector(0 to D-1);

        signal round_valid : std_logic_vector(0 to D-1);
        signal c_in_const : std_logic_vector(COORD_BITWIDTH_EXT-1 downto 0);

    begin

                                        c_in_const(QUOTIENT_BITWIDTH-1         downto
QUOTIENT_BITWIDTH-FRACTIONAL_BITWIDTH-1) <= (others => '0');
            c_in_const(QUOTIENT_BITWIDTH-FRACTIONAL_BITWIDTH-2 downto 0) <= (others =>
'1');
```

```
          tmp_divisor <= std_logic_vector(to_unsigned(1,COORD_BITWIDTH)) WHEN divisor =
std_logic_vector(to_unsigned(0,COORD_BITWIDTH)) ELSE divisor;

          G_DIV : for I in 0 to D-1 generate
              divider_inst : divider
                  port map (
                      aclk => clk,
                      s_axis_divisor_tvalid => nd,
                      --s_axis_divisor_tready => open,
                      s_axis_divisor_tdata => tmp_divisor,
                      s_axis_dividend_tvalid => nd,
                      --s_axis_dividend_tready => open,
                      s_axis_dividend_tdata => dividend(I),
                      m_axis_dout_tvalid => divider_valid(I),
                      m_axis_dout_tdata => divider_result(I)
                      --m_axis_dout_tuser(0) => tmp_divide_by_zero(I)
                  );

                                                              tmp_quotient(I)       <=
divider_result(I)(QUOTIENT_BITWIDTH+FRACTIONAL_BITWIDTH-1 downto FRACTIONAL_BITWIDTH);

              G_ROUND : if ROUND = true generate
                  dsp_round_inst : dsp_round
                      generic map (
                          BITWIDTH_IN => COORD_BITWIDTH_EXT,
                          BITWIDTH_OUT => COORD_BITWIDTH
                      )
                      port map (
                          sclr => sclr,
                          nd => divider_valid(I),
                          AB_IN => divider_result(I)(COORD_BITWIDTH_EXT-1 downto 0),
                              CARRYIN_IN => divider_result(I)(COORD_BITWIDTH_EXT-1), --
round towards zero
                          CLK_IN => clk,
                          C_IN => c_in_const,
                          P_OUT => quotient(I),
                          rdy => round_valid(I)
                      );
              end generate G_ROUND;

              G_NO_ROUND : if ROUND = false generate
```

```
                        quotient(I)  <=  divider_result(I)(QUOTIENT_BITWIDTH-1  downto
FRACTIONAL_BITWIDTH);
            end generate G_NO_ROUND;

        end generate G_DIV;

        G_NO_ROUND_1 : if ROUND = false generate
            rdy <= divider_valid(0);
        end generate G_NO_ROUND_1;

        G_ROUND_1 : if ROUND = true generate
            rdy <= round_valid(0);
        end generate G_ROUND_1;

        divide_by_zero <= '0';--tmp_divide_by_zero(0);


    end Behavioral;


    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    USE ieee.numeric_std.all;
    use ieee.math_real.all;
    use work.lloyds_algorithm_pkg.all;

    -- Uncomment the following library declaration if using
    -- arithmetic functions with Signed or Unsigned values
    --use IEEE.NUMERIC_STD.ALL;

    -- Uncomment the following library declaration if instantiating
    -- any Xilinx primitives in this code.
    --library UNISIM;
    --use UNISIM.VComponents.all;

    entity compute_distance_top is
        port (
            clk : in std_logic;
            sclr : in std_logic;
            nd : in std_logic;
            point_1 : in data_type;
            point_2 : in data_type;
            distance : out coord_type_ext;
```

```
            point_1_out : out data_type;
            point_2_out : out data_type;
            rdy : out std_logic
        );
    end compute_distance_top;

    architecture Behavioral of compute_distance_top is

                            constant      LAT_DOT_PRODUCT      :      integer      :=
MUL_CORE_LATENCY+2*integer(ceil(log2(real(D))));
        constant LAT_SUB : integer := 2;
        constant LATENCY : integer := LAT_DOT_PRODUCT+LAT_SUB;

        type data_delay_type is array(0 to LATENCY-1) of data_type;

        component addorsub
            generic (
                USE_DSP : boolean := true;
                A_BITWIDTH : integer := 16;
                B_BITWIDTH : integer := 16;
                RES_BITWIDTH : integer := 16
            );
            port (
                clk : in std_logic;
                sclr : in std_logic;
                nd : in std_logic;
                sub : in std_logic;
                a : in std_logic_vector(A_BITWIDTH-1 downto 0);
                b : in std_logic_vector(B_BITWIDTH-1 downto 0);
                res : out std_logic_vector(RES_BITWIDTH-1 downto 0);
                rdy : out std_logic
            );
        end component;

        component dot_product
            generic (
                SCALE_MUL_RESULT : integer := 0
            );
            port (
                clk : in std_logic;
                sclr : in std_logic;
                nd : in std_logic;
                point_1 : in data_type;
```

```
            point_2 : in data_type;
            result : out coord_type_ext;
            rdy : out std_logic
        );
    end component;

    signal tmp_diff : data_type;

    signal tmp_sub_rdy : std_logic;
    signal tmp_dot_product_rdy : std_logic;
    signal tmp_dot_product_result : coord_type_ext;

    signal data_delay_1 : data_delay_type;
    signal data_delay_2 : data_delay_type;

begin

    G1: for I in 0 to D-1 generate

        G_FIRST: if I = 0 generate
            addorsub_inst : addorsub
                generic map (
                    USE_DSP => USE_DSP_FOR_ADD,
                    A_BITWIDTH => COORD_BITWIDTH,
                    B_BITWIDTH => COORD_BITWIDTH,
                    RES_BITWIDTH => COORD_BITWIDTH
                )
                port map (
                    clk => clk,
                    sclr => sclr,
                    nd => nd,
                    sub => '1',
                    a => point_1(I),
                    b => point_2(I),
                    res => tmp_diff(I),
                    rdy => tmp_sub_rdy
                );


        end generate G_FIRST;

        G_OTHER: if I > 0 generate
            addorsub_inst : addorsub
```

```
            generic map (
                USE_DSP => USE_DSP_FOR_ADD,
                A_BITWIDTH => COORD_BITWIDTH,
                B_BITWIDTH => COORD_BITWIDTH,
                RES_BITWIDTH => COORD_BITWIDTH
            )
            port map (
                clk => clk,
                sclr => sclr,
                nd => nd,
                sub => '1',
                a => point_1(I),
                b => point_2(I),
                res => tmp_diff(I),
                rdy => open
            );

    end generate G_OTHER;

end generate G1;

dot_product_inst : dot_product
    generic map (
        SCALE_MUL_RESULT => MUL_FRACTIONAL_BITS
    )
    port map (
        clk => clk,
        sclr => sclr,
        nd => tmp_sub_rdy,
        point_1 => tmp_diff,
        point_2 => tmp_diff,
        result => tmp_dot_product_result,
        rdy => tmp_dot_product_rdy
    );

-- feed point_2 from input of this unit to output
data_delay_proc : process(clk)
begin
    if rising_edge(clk) then
        data_delay_1(0) <= point_1;
        data_delay_1(1 to LATENCY-1) <= data_delay_1(0 to LATENCY-2);

        data_delay_2(0) <= point_2;
```

```
                data_delay_2(1 to LATENCY-1) <= data_delay_2(0 to LATENCY-2);

            end if;
        end process data_delay_proc;


        rdy <= tmp_dot_product_rdy;
        distance <= tmp_dot_product_result;
        point_1_out <= data_delay_1(LATENCY-1);
        point_2_out <= data_delay_2(LATENCY-1);

    end Behavioral;

    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use IEEE.NUMERIC_STD.ALL;
    use ieee.math_real.all;

    -- Uncomment the following library declaration if using
    -- arithmetic functions with Signed or Unsigned values
    --use IEEE.NUMERIC_STD.ALL;

    -- Uncomment the following library declaration if instantiating
    -- any Xilinx primitives in this code.
    --library UNISIM;
    --use UNISIM.VComponents.all;

    entity adder_tree is
        generic (
            USE_DSP_FOR_ADD : boolean := true;
            NUMBER_OF_INPUTS : integer := 4;
            INPUT_BITWIDTH : integer := 16
        );
        port (
            clk : in std_logic;
            sclr : in std_logic;
            nd : in std_logic;
            sub :  in std_logic;
                input_string :  in  std_logic_vector(NUMBER_OF_INPUTS*INPUT_BITWIDTH-1
downto 0);
            rdy : out std_logic;
                                                                output       :       out
std_logic_vector(INPUT_BITWIDTH+integer(ceil(log2(real(NUMBER_OF_INPUTS))))-1 downto 0)
```

```
        );
    end adder_tree;

    architecture Behavioral of adder_tree is

                            constant      LAYERS_TREE_ADDER      :      integer      :=
integer(ceil(log2(real(NUMBER_OF_INPUTS))));
        constant INT_BITWIDTH : integer := INPUT_BITWIDTH+LAYERS_TREE_ADDER;

        constant SINGLE_ADDER_LAT : integer := 2;
        constant TREE_ADDER_LAT : integer := SINGLE_ADDER_LAT*LAYERS_TREE_ADDER;

              type   input_array_type   is   array(0   to   NUMBER_OF_INPUTS-1)   of
std_logic_vector(INPUT_BITWIDTH-1 downto 0);
           type  tree_adder_res_array_type  is  array(0  to  LAYERS_TREE_ADDER-1,  0  to
integer(ceil(real(NUMBER_OF_INPUTS)/real(2)))-1)   of   std_logic_vector(INT_BITWIDTH-1
downto 0);

              type   data_delay_type   is   array(0   to   SINGLE_ADDER_LAT-1)   of
std_logic_vector(INT_BITWIDTH-1 downto 0);
            type  data_delay_array_type  is  array(0  to  LAYERS_TREE_ADDER-1-1)  of
data_delay_type;


        component addorsub
            generic (
                USE_DSP : boolean := true;
                A_BITWIDTH : integer := 16;
                B_BITWIDTH : integer := 16;
                RES_BITWIDTH : integer := 16
            );
            port (
                clk : in std_logic;
                sclr : in std_logic;
                nd : in std_logic;
                sub : in std_logic;
                a : in std_logic_vector(A_BITWIDTH-1 downto 0);
                b : in std_logic_vector(B_BITWIDTH-1 downto 0);
                res : out std_logic_vector(RES_BITWIDTH-1 downto 0);
                rdy : out std_logic
            );
        end component;
```

```
        signal input_array : input_array_type;
        signal data_delay_array : data_delay_array_type;
        signal ctrl_delay_line : std_logic_vector(0 to TREE_ADDER_LAT-1);

        signal tmp_tree_adder_res : tree_adder_res_array_type;

    begin

        G0: for I in 0 to NUMBER_OF_INPUTS-1 generate
                    input_array(I)  <=  input_string((I+1)*INPUT_BITWIDTH-1  downto
I*INPUT_BITWIDTH);
        end generate G0;

        G2: for I in 0 to LAYERS_TREE_ADDER-1 generate

            G_TOP_LAYER: if I = 0 generate
                G2_2: for J in 0 to (NUMBER_OF_INPUTS/(2**(I+1)))-1 generate
                    addorsub_inst_2 : addorsub
                        generic map (
                            USE_DSP => USE_DSP_FOR_ADD,
                            A_BITWIDTH => INPUT_BITWIDTH,
                            B_BITWIDTH => INPUT_BITWIDTH,
                            RES_BITWIDTH => INT_BITWIDTH
                        )
                        port map (
                            clk => clk,
                            sclr => sclr,
                            nd => '1',
                            sub => sub,
                            a => input_array(2*J),
                            b => input_array(2*J+1),
                            res => tmp_tree_adder_res(I,J),
                            rdy => open
                        );
                end generate G2_2;

                                        G2_3:   if   NUMBER_OF_INPUTS/(2**(I+1))   <
integer(ceil(real(NUMBER_OF_INPUTS)/real(2**(I+1)))) generate
                    data_delay_array_proc : process(clk)
                    begin
                        if rising_edge(clk) then
                                data_delay_array(I)(0)(INPUT_BITWIDTH-1 downto 0) <=
input_array(NUMBER_OF_INPUTS/(2**(I+1))*2);
```

```
                                        data_delay_array(I)(0)(INT_BITWIDTH-1 downto
INPUT_BITWIDTH)                    <=                    (others                    =>
input_array(NUMBER_OF_INPUTS/(2**(I+1))*2)(INPUT_BITWIDTH-1));
                                    data_delay_array(I)(1 to SINGLE_ADDER_LAT-1) <=
data_delay_array(I)(0 to SINGLE_ADDER_LAT-2);
                        end if;
                    end process data_delay_array_proc;
                                tmp_tree_adder_res(I,NUMBER_OF_INPUTS/(2**(I+1))) <=
data_delay_array(I)(SINGLE_ADDER_LAT-1);
                end generate G2_3;
            end generate G_TOP_LAYER;

            G_OTHER_LAYER: if I > 0 AND I < LAYERS_TREE_ADDER-1 generate
                                                G2_2:    for    J    in    0    to
((NUMBER_OF_INPUTS+NUMBER_OF_INPUTS-(NUMBER_OF_INPUTS/(2**I))*(2**I))/(2**(I+1)))-1
generate
                    addorsub_inst_2 : addorsub
                        generic map (
                            USE_DSP => USE_DSP_FOR_ADD,
                            A_BITWIDTH => INT_BITWIDTH,
                            B_BITWIDTH => INT_BITWIDTH,
                            RES_BITWIDTH => INT_BITWIDTH
                        )
                        port map (
                            clk => clk,
                            sclr => sclr,
                            nd => '1',
                            sub => sub,
                            a => tmp_tree_adder_res(I-1,2*J),
                            b => tmp_tree_adder_res(I-1,2*J+1),
                            res => tmp_tree_adder_res(I,J),
                            rdy => open
                        );
                end generate G2_2;

                                                                    G2_3:      if
((NUMBER_OF_INPUTS+NUMBER_OF_INPUTS-(NUMBER_OF_INPUTS/(2**I))*(2**I))/(2**(I+1)))      <
integer(ceil(real(NUMBER_OF_INPUTS)/real(2**(I+1)))) generate
                    data_delay_array_proc : process(clk)
                    begin
                        if rising_edge(clk) then
                                                            data_delay_array(I)(0)  <=
tmp_tree_adder_res(I-1,NUMBER_OF_INPUTS/(2**(I+1))*2);
```

```
                                        data_delay_array(I)(1 to SINGLE_ADDER_LAT-1) <=
data_delay_array(I)(0 to SINGLE_ADDER_LAT-2);
                            end if;
                        end process data_delay_array_proc;
                                    tmp_tree_adder_res(I,NUMBER_OF_INPUTS/(2**(I+1))) <=
data_delay_array(I)(SINGLE_ADDER_LAT-1);
                    end generate G2_3;
                end generate G_OTHER_LAYER;

                G_BOTTOM_LAYER: if I = LAYERS_TREE_ADDER-1 AND LAYERS_TREE_ADDER > 1
generate
                    G2_2: for J in 0 to 0 generate
                        addorsub_inst_2 : addorsub
                            generic map (
                                USE_DSP => USE_DSP_FOR_ADD,
                                A_BITWIDTH => INT_BITWIDTH,
                                B_BITWIDTH => INT_BITWIDTH,
                                RES_BITWIDTH => INT_BITWIDTH
                            )
                            port map (
                                clk => clk,
                                sclr => sclr,
                                nd => '1',
                                sub => sub,
                                a => tmp_tree_adder_res(I-1,2*J),
                                b => tmp_tree_adder_res(I-1,2*J+1),
                                res => tmp_tree_adder_res(I,J),
                                rdy => open
                            );
                    end generate G2_2;
                end generate G_BOTTOM_LAYER;

            end generate G2;


            ctrl_delay_line_proc : process(clk)
            begin
                if rising_edge(clk) then
                    if sclr = '1' then
                        ctrl_delay_line <= (others => '0');
                    else
                        ctrl_delay_line(0) <= nd;
```

```
                        ctrl_delay_line(1 to TREE_ADDER_LAT-1) <= ctrl_delay_line(0 to
TREE_ADDER_LAT-2);
                    end if;
                end if;
            end process ctrl_delay_line_proc;

            rdy <= ctrl_delay_line(TREE_ADDER_LAT-1);
            output <= tmp_tree_adder_res(LAYERS_TREE_ADDER-1,0);

        end Behavioral;


        library IEEE;
        use IEEE.STD_LOGIC_1164.ALL;
        USE ieee.numeric_std.all;

        -- Uncomment the following library declaration if using
        -- arithmetic functions with Signed or Unsigned values
        --use IEEE.NUMERIC_STD.ALL;

        -- Uncomment the following library declaration if instantiating
        -- any Xilinx primitives in this code.
        library UNISIM;
        use UNISIM.VComponents.all;

        -- computes a+b or a-b

        entity addorsub is
            generic (
                USE_DSP : boolean := true;
                A_BITWIDTH : integer := 16;
                B_BITWIDTH : integer := 16;
                RES_BITWIDTH : integer := 16
            );
            port (
                clk : in std_logic;
                sclr : in std_logic;
                nd : in std_logic;
                sub : in std_logic;
                a : in std_logic_vector(A_BITWIDTH-1 downto 0);
                b : in std_logic_vector(B_BITWIDTH-1 downto 0);
                res : out std_logic_vector(RES_BITWIDTH-1 downto 0);
                rdy : out std_logic
```

```
        );
    end addorsub;

    architecture Behavioral of addorsub is

        function my_max(v1 : integer; v2 : integer) return integer is
            variable result : integer;
        begin
            if v1 < v2 then
                result := v2;
            else
                result := v1;
            end if;
            return result;
        end my_max;

             function sext(val : std_logic_vector; length : integer) return
std_logic_vector is
            variable val_msb : std_logic;
            variable result : std_logic_vector(length-1 downto 0);
        begin
            val_msb := val(val'length-1);
            result(val'length-1 downto 0) := val;
            result(length-1 downto val'length) := (others => val_msb);
            return result;
        end sext;

        constant INT_BITWIDTH : integer := A_BITWIDTH+1;
        constant LATENCY : integer := 2;

        signal GND_ALUMODE      : std_logic;
        signal GND_BUS_3        : std_logic_vector (2 downto 0);
        signal GND_BUS_18       : std_logic_vector (17 downto 0);
        signal GND_BUS_30       : std_logic_vector (29 downto 0);
        signal GND_BUS_48       : std_logic_vector (47 downto 0);
        signal GND_OPMODE       : std_logic;
        signal VCC_OPMODE       : std_logic;

        signal AB_IN : std_logic_vector(47 downto 0);
        signal C_IN : std_logic_vector(47 downto 0);
        signal P_OUT : std_logic_vector(47 downto 0);

        signal delay_line       : std_logic_vector(0 to LATENCY-1);
```

```
    signal a_reg : std_logic_vector(A_BITWIDTH-1 downto 0);
    signal b_reg : std_logic_vector(B_BITWIDTH-1 downto 0);
    signal p_reg : std_logic_vector(47 downto 0);
    signal tmp_sum : signed(INT_BITWIDTH-1 downto 0);

begin

    G_DSP : if USE_DSP = true generate

        GND_ALUMODE <= '0';
        GND_BUS_3(2 downto 0) <= "000";
        GND_BUS_18(17 downto 0) <= "000000000000000000";
        GND_BUS_30(29 downto 0) <= "000000000000000000000000000000";
        GND_BUS_48(47 downto 0) <=
              "000000000000000000000000000000000000000000000000";
        GND_OPMODE <= '0';
        VCC_OPMODE <= '1';

        C_IN(47 downto A_BITWIDTH) <= (others=> a(A_BITWIDTH-1));
        C_IN(A_BITWIDTH-1 downto 0) <= a;

        AB_IN(47 downto B_BITWIDTH) <= (others=> b(B_BITWIDTH-1));
        AB_IN(B_BITWIDTH-1 downto 0) <= b;


        DSP48E_INST : DSP48E
        generic map( ACASCREG => 1,
                 ALUMODEREG => 0,
                 AREG => 1,
                 AUTORESET_PATTERN_DETECT => FALSE,
                 AUTORESET_PATTERN_DETECT_OPTINV => "MATCH",
                 A_INPUT => "DIRECT",
                 BCASCREG => 1,
                 BREG => 1,
                 B_INPUT => "DIRECT",
                 CARRYINREG => 1,
                 CARRYINSELREG => 0,
                 CREG => 1,
                 MASK => x"3FFFFFFFFFFF",
                 MREG => 1,
                 MULTCARRYINREG => 1,
                 OPMODEREG => 0,
```

```
                        PATTERN => x"000000000000", -- pattern to all zeros
                        PREG => 1,
                        SEL_MASK => "MASK",
                        SEL_PATTERN => "PATTERN",
                              SEL_ROUNDING_MASK => "MODE1", -- set to MODE1 instead of
SEL_MASK to overrule SEL_MASK
                        USE_MULT => "NONE",
                        USE_PATTERN_DETECT => "PATDET", -- enable pattern detect
                        USE_SIMD => "ONE48")
                  port map (A(29 downto 0)=>AB_IN(47 downto 18),
                        ACIN(29 downto 0)=>GND_BUS_30(29 downto 0),
                        ALUMODE(3)=>GND_ALUMODE,
                        ALUMODE(2)=>GND_ALUMODE,
                        ALUMODE(1)=>sub,
                        ALUMODE(0)=>sub,
                        B(17 downto 0)=>AB_IN(17 downto 0),
                        BCIN(17 downto 0)=>GND_BUS_18(17 downto 0),
                        C(47 downto 0)=>C_IN(47 downto 0),
                        CARRYCASCIN=>GND_ALUMODE,
                        CARRYIN=>GND_ALUMODE,
                        CARRYINSEL(2 downto 0)=>GND_BUS_3(2 downto 0),
                        CEALUMODE=>VCC_OPMODE,
                        CEA1=>nd,
                        CEA2=>VCC_OPMODE,
                        CEB1=>nd,
                        CEB2=>VCC_OPMODE,
                        CEC=>nd,
                        CECARRYIN=>VCC_OPMODE,
                        CECTRL=>VCC_OPMODE,
                        CEM=>VCC_OPMODE,
                        CEMULTCARRYIN=>VCC_OPMODE,
                        CEP=>VCC_OPMODE,
                        CLK=>clk,
                        MULTSIGNIN=>GND_ALUMODE,
                        OPMODE(6)=>GND_OPMODE,
                        OPMODE(5)=>VCC_OPMODE,
                        OPMODE(4)=>VCC_OPMODE,
                        OPMODE(3)=>GND_OPMODE,
                        OPMODE(2)=>GND_OPMODE,
                        OPMODE(1)=>VCC_OPMODE,
                        OPMODE(0)=>VCC_OPMODE,
                        PCIN(47 downto 0)=>GND_BUS_48(47 downto 0),
                        RSTA=>GND_ALUMODE,
```

```
                        RSTALLCARRYIN=>GND_ALUMODE,
                        RSTALUMODE=>GND_ALUMODE,
                        RSTB=>GND_ALUMODE,
                        RSTC=>GND_ALUMODE,
                        RSTCTRL=>GND_ALUMODE,
                        RSTM=>GND_ALUMODE,
                        RSTP=>GND_ALUMODE,
                        ACOUT=>open,
                        BCOUT=>open,
                        CARRYCASCOUT=>open,
                        CARRYOUT=>open,
                        MULTSIGNOUT=>open,
                        OVERFLOW=>open,
                        P(47 downto 0)=>P_OUT(47 downto 0),
                        PATTERNBDETECT=>open,
                        PATTERNDETECT=>open,
                        PCOUT=>open,
                        UNDERFLOW=>open);

            -- trunc LSB (safe)
            --res <= P_OUT(INT_BITWIDTH-1 downto INT_BITWIDTH-RES_BITWIDTH);
            -- trunc MSB (unsafe)
            res <= P_OUT(RES_BITWIDTH-1 downto 0);

        end generate G_DSP;

        G_NODSP : if USE_DSP = false generate

            reg_proc : process(clk)
            begin
                if rising_edge(clk) then
                    a_reg <= a;
                    b_reg <= b;
                    p_reg <= sext(std_logic_vector(tmp_sum),48);
                end if;
            end process reg_proc;

            add_sub : process(a_reg,b_reg,sub)
            begin
                if sub ='0' then
                                                                    tmp_sum    <=
signed(sext(a_reg,INT_BITWIDTH))+signed(sext(b_reg,INT_BITWIDTH));
                else
```

```
                                                                    tmp_sum     <=
signed(sext(a_reg,INT_BITWIDTH))-signed(sext(b_reg,INT_BITWIDTH));
                    end if;
                end process add_sub;

                res <= p_reg(RES_BITWIDTH-1 downto 0);

            end generate G_NODSP;



            -- compensate for downconverter latency
            delay_line_proc : process(clk)
            begin
            if rising_edge(clk) then
                if sclr = '1' then
                    delay_line <= (others => '0');
                else
                    delay_line(0) <= nd;
                    delay_line(1 to LATENCY-1) <= delay_line(0 to LATENCY-2);
                end if;
            end if;
            end process delay_line_proc;

            rdy <= delay_line(LATENCY-1);

        end Behavioral;
```

Додаток Б

(обов’язковий)

Копія публікації

INTERNATIONAL SCIENTIFIC JOURNAL
«COMPUTER SYSTEMS AND INFORMATION TECHNOLOGIES»

UDC 004.93
DOI: 10.31891/CSIT-2021-5-4

SERGII LYSENKO, DMYTRO SOKALSKYI, IANA MYKHASKO
Khmelnytskyi National University, Khmelnytskyi, Ukraine

METHODS FOR CYBERATTACKS DETECTION IN THE COMPUTER NETWORKS AS A MEAN OF RESILIENT IT-INFRASTRUCTURE CONSTRUCTION: STATE-OF-ART

The paper presents a state-of-art of the methods for cyberattacks detection in the computer networks. The main accent was made on the concept of the resilience for the IT infrastructure. The concept of cyber resilience in the terms of cybersecurity was presented. The survey includes the set of approaches devoted to the problem of construction resilient infrastructures. All investigated approaches are aimed to construct and maintain infrastructure's resilience for cyberattacks resistance. Mentioned techniques and frameworks keep the main principles to assure resilience. To do this there exists some requirements to construct such infrastructure: IT infrastructure has to include the set ready to use measures of preparation concerning the possible cyber threats; it must include the set of special measures for the protection, as well as for cyberattacks detection; important issue and required is the possibility to respond the attack and to be able to absorb the negative attacks' impact; IT infrastructure must be as adaptive as it is possible, because today the dynamic of the attacks mutation is very high; IT infrastructure must be recoverable after the attacks were performed. In addition, the state-of-art found out that known approaches have domain-specific usage and it is important to develop new approaches and frameworks for the cyberattacks detection in the computer networks as a means of resilient IT-infrastructure construction.

Keywords: cyberattack, IT infrastructure, malware, computer systems, resilience, detection efficiency, network traffic

СЕРГІЙ ЛИСЕНКО, ДМИТРО СОКАЛЬСЬКИЙ, ЯНА МИХАСЬКО
Хмельницький національний університет

ДОСЛІДЖЕННЯ МЕТОДІВ ВИЯВЛЕННЯ КІБЕРАТАК В КОМП’ЮТЕРНИХ МЕРЕЖАХ ЯК ЗАСОБІВ ДО ПОБУДОВИ РЕЗИЛЬЄНТНИХ ДО ВТОРГНЕНЬ IT-ІНФРАСТРУКТУР

IT-інфраструктура стає все більш взаємопов’язаною, тоді як кіберзагрози для критичної інфраструктури стають все більш складними, від яких важко захиститися. Кібербезпека акцентує увагу на створенні засобів захисту, щоб запобігти втрати конфіденційності, цілісності та доступності цифрової інформації та систем, але останніми роками кібератаки продемонстрували, що жодна система не є непроникною. Кібер резильентність стала додатковим пріоритетом, який спрямований на те, щоб IT-інфраструктури могли підтримувати суттєві рівні продуктивності, навіть якщо можливості погіршуються внаслідок кібератаки. У статті представлено сучасні методи виявлення кібератак у комп’ютерних мережах. Основний акцент був зроблений на концепції стійкості для IT-інфраструктури. Було представлено поняття кіберстійкості з точки зору кібербезпеки. Дослідження включає комплекс підходів, присвячених проблемі побудови резильентних інфраструктур. Усі досліджувані підходи спрямовані на створення та підтримку резильентності інфраструктури до кібератак. Згадані методи та фреймворки зберігають основні принципи забезпечення резильентності. Для цього існують певні вимоги до побудови такої інфраструктури: IT-інфраструктура повинна включати набір готових до використання заходів підготовки щодо можливих кіберзагроз; має включати комплекс спеціальних заходів щодо захисту, а також виявлення кібератак; важливою і необхідною проблемою є можливість реагувати на атаку та мати можливість поглинати вплив негативних атак; IT-інфраструктура має бути максимально адаптивною, тому що сьогодні динаміка мутації атак дуже висока; IT-інфраструктура повинна бути відновлена після здійснення атак. Крім того, сучасний дослідження показало, що відомі підходи мають нішеве застосування і важливо розробляювати нові підходи та засоби для виявлення кібератак у комп’ютерних мережах як засобу побудови резильентних IT-інфраструктур.

Ключові слова: шкідливе програмне забезпечення, живучість, комп’ютерні системи, достовірність виявлення, кібератака, мережний трафік.

Introduction

IT infrastructures have deeply into all aspects of our life. The cyber-physical systems based on the different infrastructures are a new concept today and are the next generation of system that must secure and be ready to resist the cyberthreats today [1-9]. In recent years, network cyberattacks on the IT infrastructure of information systems is still growing [10-14]. Thus, in June 2010, the security experts have found a botnet attack “StuxNet”, which threated the industrial system, infected with computer systems and networks around the world [15].

Since the botnet malware StuxNet have appeared, a great variety of new cyber threads IT infrastructure had appeared [16-17]. Furthermore, new attacks involve new approaches and system information, that makes it possible to overcome the protection of IT infrastructure. That’s why there exists an important task to investigate, maintain, and develop new methods for cyberattacks detection in the computer networks as a mean of resilient IT-infrastructure construction.

This research is a state-of-art of the methods for cyberattacks detection in the computer networks, that present the set of approach devoted to the problem of construction of resilient IT-infrastructures.

Concept of cyber resilience

Nowadays, the problem of cyber defense is not only issue of protection of infrastructure. The very new concept is the resilience [14].The concept of resilience is applied to a lot of contexts as well as to IT infrastructures. For instance, in the ecology the resilient item is a kind of population property that is able to survive. This concept is used in economics, construction industry [15] etc.

In the term of cybersecurity and it usage for IT infrastructure, the concept of resilience is the special ability to execute the resistance, restoration and adaption in the situation of cyberattacks performing [18-20].

So, the importunate of the development of IT infrastructure resilience is very significant.

Techniques for resilient IT infrastructures construction

Today there is a huge number of approaches devoted to cybersecurity. But task of cyber resilience is not solved yet. Nevertheless, researches make attempts to bring new techniques to construct the IT infrastructures more resilient against the cyber threads.

In [21] the technique for resilient cyber-physical systems construction is proposed. It is devoted to the task of detection of the communication delays in the infrastructure networks in the situation of denial-of-service (DoS) attacks.

The approach previously was based on the usage of multiagent systems (MASs) in order to identify the DoS attacks. The next-gen approach is based on the distributed resilient technique, that involves as the mathematical tools technique a general heterogeneous linear multiagent systems. It enables the possibility to deal with the nonuniform communication delays.

The core of the approach is the usage of the kinds of observers, that are sampled-based. To make the approach more efficient the authors proposed to make the observers be adaptive and distributed. This idea was achieved by using a buffer mechanism, that made it possible to eliminate the heterogeneous behavior generated by communication delays. In the terms of the adopting of the adaptive distributed resilient observers, the techniques made it possible to develop the resilient mechanism able to resist the denial-of-service attacks. In addition, authors included a time-varying sampling period sequence in order to prevent the attack implementation and its detection by the sampling period of the possibly infected infrastructure.

For the purpose to verify the overall technique efficiency, using the provided resilient observers, a special system controller for detection was developed. Its implementation and the series experiments conducting has proved the effectiveness of the prosed technique for resilient cyber-physical systems.

An approach for resilient control of cyber-physical systems was proposed in [22]. It is a technique for the resilient Cyber-Physical Systems construction.

In the research the authors tried to simulate different situations that may cause with IT infrastructures during its functioning. The core of the idea is to develop the system that must support the correct operations set for the crucial functional elements notwithstanding providing the resistant misbehavior.

The technique uses a moving target defense paradigm. The main idea is to deal with the linear switching of state-space matrices. The approach involves both the physical and network layers concerning a control system presented in network.

The efficiency of the proposed technique was substantiated by the set of experiments. The results have showed that appliance of the technique for the systems had made it possible to maintain systems' stability. We also evaluate, via simulation, a step-by-step procedure that takes a transfer function, representing the dynamics of the physical process.

In addition, author proved that the involvement of the approach made it possible to develop the resilient IT infrastructure, where there is a topology of decentralized controllers.

In [23] an approach for the constructing of the resilient systems against the cyberattack is proposed. Authors presented that today there is a strong need to protect the infrastructures under the attacks. To do this the cybersecurity issue must be organized around the main terms of confidentiality, integrity and availability. In addition, the main problem and drawbacks of cybersecurity of the IT- infrastructures is its strong increasing. In this situation, the cybersecurity for the IT- infrastructures has become unable to take into account the huge aspects as the time dynamics of time, space bound behavior, rapid changes etc.

In the approach the authors proposed the analysis of construction aspects concerning the resilient systems under the cyberattack.

The main idea is that there is a need to make the cyberattack resilient missions, that will give the opportunities to achieve the completion of resilient mission goals. Also, it will give the possibility to provide the IT assets and the needed services, which will enable the support of the resilient actions concerning the attacked system.

The research presents also a set of architectural issues in order to construct proper cyberattack resilience missions. It involves the mission-centricity, survivability (using the adaptation procedure). In addition, it includes the mission C2, cybersecurity management mission to achieve the efficient mission execution.

To perform the evaluation of the effectiveness of the proposed approach, the authors have developed the resilient IT infrastructure, that includes two multi-agent systems. These systems are adaptive and have possibility to

interact. Furthermore, researchers presented a set of models and algorithms, defined for the proposed resilient system with the experimental results.

In [24] an approach for the resilient cyber-physical systems construction is presented. It includes the set of steps for the development of an updated state estimation process, that is aimed to increase the resilience of IT-infrastructures under the cyberattacks.

Authors of the technique proposed to involve the existing data, that are presented in the current state estimators. The main idea of the approach is to establishes the state estimation vulnerability concerning the cyberattacks that are able to execute the data injection target at evading detecting by the mean of the outlier routines, that are used in the situation of the estimation processes.

In addition, the researchers presented the set of architectural solutions based on usage of the emerging smart grid technologies.

In [25] an agent-based cyber control strategy design for resilient control systems was produced. It presents an approach directed into defense of the critical infrastructure.

Mentioned approach includes several sets:

1) development of the cyber security research built into the industrial control system environment. The system involves the set of mechanisms to present opened and closed loop, feedback, designs.

2) integration of the cyber-physical design. It has to ensure the possibility to utilize the system data for correct response on the physical system.

3) integration of the techniques that are able to address a new approach to distributed IT infrastructures. , which considers both the industrial process control dynamics for SES, as well as the influences of the benign and malicious human.

In [26] a technique for resilient cyber-physical systems construction was presented.

In the research the authors proposed several contributions to ensure cybersecurity and infrastructure resilience. To do this the framework WAMPAC was developed. It is the security frame work, tht makes it possible to provide the end-to-end attack-resilient IT infrastructure in the power grid.

The approach involves the cybersecurity issues:

1) framework maintains the infrastructure life cycle (such as risk assessment, cyberattacks prevention measures, cyberattacks detection procedures, cyberattacks mitigation measures);

2) framework involves a defense-in-depth principles that combines the combines the cyberattacks resilience at IT infrastructure and the software levels;

3) framework is able to detect the cyber–physical security anomalies.

The authors also have presented a set of cyberattack-resilient algorithms, such as anomaly detection and mitigation approaches. The paper includes the set of experiments that prove the efficiency of the frame work. To do this it presents some cases how to prevent, detect and mitigate the attacks.

In [27] research devoted to the theoretical and some practical issues concerning the resilience of IT infrastructures are presented.

The paper shows the vast importance of the resilience today, as it can increase the adversary's effort level for the needed for the malicious objectives achievement. In terms of resilient infrastructure, it must be highly resistant to malware activities and can be able to prevent the cyberattacks execution.

The author proposed to construct systems, that were able to evaluate resilience in the presence of cyberattacks. The research involves the game-based simulation framework, that demonstrates the process of attack as well as the defending procedure.

An approach also shows a set of simulations of sufficient fidelity. To do this the manuscript includes the description of complex heterogeneous simulations. The framework is modeling integration tool names SURE for infrastructure against the cyberattacks.

The inbuild modeling simulator uses the models, constructed with the use of a model-based integration tool the heterogeneous and distributed simulations. It is able to perform the construction of the rapid design, synthesis, and evaluation of experiments.

The main feature of the SURE framework is the possibility to make the transportation systems. In this case the framework is able to provide the needed domain-specific languages, specified models, tools for the translation of constructed models. The proposed framework has in-built simulation driver tool. It's main aim is to perform the establishment of a coherent experimentation environment.

Conclusions

The paper presents a state-of-art of the methods for cyberattacks detection in the computer networks. The main accent was made on the concept of the resilience for the IT infrastructure. The concept of cyber resilience in the terms of cybersecurity was presented. The survey includes the set of approaches devoted to the problem of construction resilient infrastructures. All investigated approaches are aimed to construct and maintain infrastructure's resilience for cyberattacks resistance. Mentioned techniques and frameworks keep the main principles to assure resilience. To do this there exists some requirements to construct such infrastructure:

1) IT infrastructure has to include the set ready to use measures of preparation concerning the possible cyber threats;

2) it must include the set of special measures for the protection, as well as for the cyberattacks detection;

3) important issue and required is the possibility to respond the attack and to be able to absorb the negative attacks' impact;

4) IT infrastructure must be as adaptive as it is possible, because today the dynamic of the attacks mutation is very high;

5) IT infrastructure must be recoverable after the attacks were performed.

In addition, the state-of-art found out that known approaches have domain-specific usage and it is important to develop new approaches and frameworks for the cyberattacks detection in the computer networks as a means of resilient IT-infrastructure construction.

References

1. McAfee Mobile Threat Report Q2, 2021. URL: https://www.mcafee.com/content/dam/cons umer/en-us/docs/2020-Mobile-Threat-Report.pdf. (accessed 10.10.2021).
2. S. Liu, B. Xu, S. Li and Y. Liu, Resilient control strategy of cyber-physical system under DoS attacks, *36th Chinese Control Conference (CCC)*, 2017, pp. 7760-7765, doi: 10.23919/ChiCC.2017.8028584.
3. X. Koutsoukos et al., SURE: A Modeling and Simulation Integration Platform for Evaluation of Secure and Resilient Cyber–Physical Systems, *Proceedings of the IEEE*, vol. 106, no. 1, pp. 93-112, Jan. 2018, doi: 10.1109/JPROC.2017.2731741.
4. A. M. Melin, E. M. Ferragut, J. A. Laska, D. L. Fugate and R. Kisner, A mathematical framework for the analysis of cyber-resilient control systems, *6th International Symposium on Resilient Control Systems (ISRCS)*, 2013, pp. 13-18, doi: 10.1109/ISRCS.2013.6623743.
5. L. Zhang, X. Chen, F. Kong and A. A. Cardenas, Real-Time Attack-Recovery for Cyber-Physical Systems Using Linear Approximations, *IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 205-217, doi: 10.1109/RTSS49844.2020.00028.
6. A. Lukina, A. Tiwari, S. A. Smolka, L. Esterle, J. Yang and R. Grosu, Resilient Control and Safety for Cyber-Physical Systems, *IEEE Workshop on Monitoring and Testing of Cyber-Physical Systems (MT-IT INFRASTRUCTURES)*, 2018, pp. 16-17, doi: 10.1109/MT-IT INFRASTRUCTURES.2018.00015.
7. C. Deng and C. Wen, MAS-Based Distributed Resilient Control for a Class of Cyber-Physical Systems With Communication Delays Under DoS Attacks, *IEEE Transactions on Cybernetics*, vol. 51, no. 5, pp. 2347-2358, May 2021, doi: 10.1109/TCYB.2020.2972686.
8. S. Sulochana and V. Manjula, Resilient system for secure sharing of information against false data injection attack, *International Conference on Information Communication and Embedded Systems* (ICICES), 2016, pp. 1-5, doi: 10.1109/ICICES.2016.7518943.
9. Lysenko, S., Bobrovnikova, K., Savenko, O., Shchuka, R. Technique for Cyberattacks Detection Based on DNS Traffic Analysis, *CEUR-WS*, Vol 2732. ISSN: 1613–0073. 2020. pp. 171-182
10. Bobrovnikova, Kira, Sergii Lysenko, Piotr Gaj, Dmytro Denysiuk "Technique for IoT Cyberattacks Detection Based on DNS Traffic Analysis.CEUR-WS.Vol.2623. ISSN: 1613–00732020, pp. 208-218.
11. Sergii Lysenko, Kira Bobrovnikova, Peter Popov, Viacheslav Kharchenko, Dmytro Medzatyi. Spyware Detection Technique Based on Reinforcement Learning. *CEUR-WS*.Vol. 2623 ISSN: 1613–0073 (2020), pp. 307-316
12. Lysenko, S., Bobrovnikova, K., Matiukh, S., Hurman, I., Savenko, O.Detection of the botnets' low-rate DDoS attacks based on self-similarity. *International Journal of Electrical and Computer Engineering*, ISSN 2088-8708, 2020, 10(4), pp. 3651-3659.
13. Savenko, O., Sachenko, A., Lysenko, S., Markowsky, G., Vasylkiv, N. Botnet detection approach based on the distributed systems. *International Journal of Computing*, ISSN 1727-6209, 2020, 19(2), pp. 190-198.
14. Sergii Lysenko, Kira Bobrovnikova, Peter Popov, Viacheslav Kharchenko, Dmytro Medzatyi. Spyware Detection Technique Based on Reinforcement Learning. CEUR-WS.Vol. 2623 ISSN: 1613–0073 (2020), pp. 307-316
15. S. Al-Rabiaah, The "Stuxnet" Virus of 2010 As an Example of A "APT" and Its "Recent" Variances, *21st Saudi Computer Society National Computer Conference (NCC)*, 2018, pp. 1-5, doi: 10.1109/NCG.2018.8593143.
16. Lysenko S.., Bobrovnikova Y., Kharchenko V. Methods of detecting botnets in computer systems. *Modern information systems*. 2019. T.3. №4. C.87-95.
17. Sergii Lysenko, Kira Bobrovnikova, Piotr Gaj, Tomas Sochor, and Iryna Forkun Resilient Computer Systems Development for Cyberattacks Resistance. *CEUR-WS*. 2021, 2853, pp. 353–361.
18. Cimellaro G. P., Dueñas-Osorio L., Reinhorn A.M. Introduction to special issue on resilience-based analysis and design of structures and infrastructure systems. Structural Engineering, 2016, No. 142(8), pp.1-5.
19. Linkov, I., Eisenberg, D. A., Plourde, K., Seager, T. P., Allen, J., Kott, A. Resilience metrics for cyber systems. Environment Systems and Decisions, 2013, No. 33(4), pp. 471-476.
20. Bodeau, D.J., Graubart, R. D. Cyber resiliency design principles: selective use throughout the lifecycle and in conjunction with related disciplines. MITRE Corp., Tech. Rep., 2017, p.98.
21. C. Deng, Distributed Resilient Control for Cyber-Physical Systems under Denial-of-Service Attacks, *23rd International Conference on Mechatronics Technology (ICMT)*, 2019, pp. 1-5, doi: 10.1109/ICMECT.2019.8932108.
22. M. Segovia-Ferreira, J. Rubio-Hernan, R. Cavalli and J. Garcia-Alfaro, Switched-Based Resilient Control of Cyber-Physical Systems, *IEEE Access*, vol. 8, pp. 212194-212208, 2020, doi: 10.1109/ACCESS.2020.3039879.
23. G. Jakobson, Mission-centricity in cyber security: Architecting cyberattack resilient missions, *5th International Conference on Cyber Conflict* (CYCON 2013), 2013, pp. 1-18.
24. S. Hopkins, E. Kalaimannan and C. S. John, Foundations for Research in Cyber-Physical System Cyber Resilience using State Estimation, *SoutheastCon*, 2020, pp. 1-2, doi: 10.1109/SoutheastCon44009.2020.9249745.
25. C. Rieger, Q. Zhu and T. Başar, Agent-based cyber control strategy design for resilient control systems: Concepts, architecture and methodologies, *2012 5th International Symposium on Resilient Control Systems*, 2012, pp. 40-47, doi: 10.1109/ISRCS.2012.6309291.
26. A. Ashok, M. Govindarasu and J. Wang, Cyber-Physical Attack-Resilient Wide-Area Monitoring, Protection, and Control for the Power Grid, *Proceedings of the IEEE*, vol. 105, no. 7, pp. 1389-1407, July 2017, doi: 10.1109/JPROC.2017.2686394.
27. X. Koutsoukos et al., SURE: A Modeling and Simulation Integration Platform for Evaluation of Secure and Resilient Cyber–Physical Systems, *Proceedings of the IEEE*, vol. 106, no. 1, pp. 93-112, Jan. 2018, doi: 10.1109/JPROC.2017.2731741.

Sergii Lysenko **Сергій Лисенко**	Doctor of Science, Professor of Computer Engineering & System Programming Department, Khmelnytskyi National University, Khmelnytskyi, Ukraine, e-mail: sirogyk@ukr.net, orcid.org/0000-0001-7243-8747, Scopus Author ID: 54420643500, ResearcherID: I-1728-2018 https://scholar.google.com.ua/citations?hl=uk&user=TuAfytwAAAAJ&view_op=list_works	доктор технічних наук, професор кафедри комп'ютерної інженерії та системного програмування, Хмельницький національний університет, Хмельницький, Україна
Dmytro Sokalskyi **Дмитро Сокальський**	Master student, Computer Engineering, Khmelnytskyi National University, Khmelnytskyi, Ukraine, e-mail: sokalskij7@gmail.com	студент, комп'ютерна інженерія, Хмельницький національний університет, Хельницький, Україна.
Iana Mykhasko **Яна Михасько**	Master student, Computer Engineering, Khmelnytskyi National University, Khmelnytskyi, Ukraine, e-mail: yashannyy@gmail.com	студентка, комп'ютерна інженерія, Хмельницький національний університет, Хельницький, Україна.

Додаток В

(обов’язковий)

Копія тез доповіді на Всеукраїнській науково-практичній конференції Актуальні Проблеми Комп’ютерних Наук (АПКН-2021)

УДК 004.7.056.5

Михасько Я. Ю., Лисенко С. М.

Хмельницький національний університет

МЕТОД РОЗРОБКИ ЕМУЛЯТОРА ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

Запропоновано метод розробки емулятора виявлення кібер-загроз типу «фішинг». Метод здійснює глибокий аналіз того, як зловмисники створюють фішингові атаки. Аналіз грунтується на залученні емулятора атак, який дозволяє продукувати висновок про присутність атаки на основі кластеризації для оцінки подібності між атаками. Метод також грунтується на залученні процедури аналізу еволюції потенційних фішингових атак шляхом відстежування змін з плином часу.

A method for developing an emulator for detecting cyber threats of the "phishing" type has been proposed. The method provides an in-depth analysis of how attackers create phishing attacks. The analysis is based on the involvement of an attack emulator, which allows to draw a conclusion about the presence of an attack based on clustering to assess the similarity between attacks. The method is also based on the use of a procedure to analyze the evolution of potential phishing attacks by tracking changes over time.

Так звані «фішингові атаки» (phishing attacks) – це тип атаки, під час яких зловмисники використовують легітимний веб-сайт щоб викрасти конфіденційну інформацію у кінцевих користувачів [1]. Фішингові атаки є однією з важливих загроз для приватних осіб та корпорацій у сучасному Інтернеті. Ця проблема активно досліджується як науковцями колами, так і індустрією антивірусних засобів портягом останніх кількох років [2].

Спроби надати ефективні антифішингові рішення мали два основні підходи: (1) ідентифікувати фішингову атаку шляхом порівняння її подібності з цільовим сайтом, та (2) полягає у розгляді внутрішніх характеристик нападів [3-4].

В дослідженні пропонується розглянемо цю проблему під новим кутом зору – а саме пробою розроблення емулятора виявлення кібер-загроз типу «фішинг».

В основі виявлення пропонується здійснювати виявлення атак із застосування апаратно-технічного засобу, який буде здатним не лише використовувати внутрішні характеристики атаки та порівнювати схожість між атаками та цільовими сайтами, а також аналізуватиме поведінку атаки [5-6].

Запропонований метод здійснює глибокий аналіз того, як зловмисники створюють фішингові атаки. Аналіз ґрунтується на тому, що більшість фішингових атак є дублікатами або квазідублікатами колишніх атак.

Враховуючи, що фішингові атаки не будуються з нуля, в роботі залучено емулятор атак, який дозволяє продукувати висновок про присутність атаки на основі кластеризації для оцінки подібності між атаками.

Метод також ґрунтується на залученні процедури аналізу еволюції потенційних фішингових атак шляхом відстежування змін з плином часу.

Метою методу є отримання результату щодо наявності множини поведінкових змін зловмисного характеру, а також аналіз кількості ітерацій атак.

Іншим аспектом запропонованого методу є виявлення та аналіз фішингових атак на стороні клієнта.

З цією метою емулятор виявлення атак повинен бути розміщений у сервері мережі.

Для цього здійснюється статичний аналіз вихідного коду «наборів фішингу», а також аналіз та відстеження вкраденої інформації.

Оскільки більшість фішингових атак використовують електронну пошту як засіб вилучення вкраденої інформації, в дослідженні запропоновано модель глибокого навчання для виявлення цих повідомлень у мережевому трафіку. Цей підхід може бути використаний, щоб легко виявити, що фішингова атака розміщена у великій комп'ютерній мережі.

Результати роботи емулятора виявлення атак типу фішинг подано в таблиці 1.

Отже, запропоновано метод розробки емулятора виявлення кібер-загроз типу «фішинг».

Результати залучення методу демонструють точність виявлення в межах 90%, а рівень хибних спрацювань в межах 3-5%.

Таблиця 1 – Результати роботи емулятора

Параметри	**Значення**
Кількість аналізованих атак емулятором	98,571
Кількість легальних сайтів	24,800
Поріг виявлення	0.33
Ефективність виявлення, %	91.34
Рівень хибних спрацювань, %	3.55

Перелік посилань

1. Лисенко С.М., Щука Р.В. Аналіз методів виявлення шкідливого програмного забезпечення в комп'ютерних системах Вісник Хмельницького національного університету. Технічні науки. 2020. №2. С. 43-49. 101-107.
2. M. Zabihimayvan and D. Doran, "Fuzzy Rough Set Feature Selection to Enhance Phishing Attack Detection," 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2019, pp. 1-6, doi: 10.1109/FUZZ-IEEE.2019.8858884.
3. T. Nathezhtha, D. Sangeetha and V. Vaidehi, "WC-PAD: Web Crawling based Phishing Attack Detection," 2019 International Carnahan Conference on Security Technology (ICCST), 2019, pp. 1-6, doi: 10.1109/CCST.2019.8888416.
4. S. Patil and S. Dhage, "A Methodical Overview on Phishing Detection along with an Organized Way to Construct an Anti-Phishing Framework," 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS), 2019, pp. 588-593, doi: 10.1109/ICACCS.2019.8728356.
5. M. D. Bhagwat, P. H. Patil and T. S. Vishawanath, "A Methodical Overview on Detection, Identification and Proactive Prevention of Phishing Websites," 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), 2021, pp. 1505-1508, doi: 10.1109/ICICV50876.2021.9388441.
6. Lysenko, S., Bobrovnikova, K., Matiukh, S., Hurman, I., Savenko, O.Detection of the botnets' low-rate DDoS attacks based on self-similarity. International Journal of Electrical and Computer Engineering, ISSN 2088-8708, 2020, 10(4), pp. 3651-3659.
7. Savenko, O., Sachenko, A., Lysenko, S., Markowsky, G., Vasylkiv, N. Botnet detection approach based on the distributed systems. International Journal of Computing, ISSN 1727-6209, 2020, 19(2), pp. 190-198.

Додаток Г

(обов’язковий)

Презентація доповіді

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Кафедра комп’ютерної інженерії та системного програмування

Михасько Яна Юріївна

Метод та засоби емулятора виявлення кібер-загроз типу «фішинг»

Науковий керівник – д.т.н. доц. Лисенко С.М.

Хмельницький - 2022

Мета і задачі дослідження

Метою роботи є підвищення достовірності та виявлення атак типу фішинг на основі застосування емулятора

Об'єкт дослідження – процес виявлення атак типу фішинг на основі застосування емулятора.

Предмет дослідження – моделі, методи та програмно-технічні засобами виявлення атак типу фішинг на основі застосування емулятора.

Мета і задачі дослідження

Поставлена мета досягається розв’язанням таких основних задач:

1. Дослідити методи та засоби виявлення фішингових атак
2. Дослідити переваги та недоліки методів виявлення фішингових атак. Дослідити життєвий цикл фішингових атак.
3. Розробити моделі процесу функціонування емулятора виявлення атак типу фішинг
4. Провести моделювання схожості фішингових сторінок
5. Розробити удосконалений метод виявлення кіберзагроз типу «фішинг».
6. Здійснити реалізацію удосконаленого методу виявлення кіберзагроз типу «фішинг»у вигляді емулятора - програмно-апаратного засобу.

Наукова новизна отриманих результатів

1. Удосконалено метод виявлення кібер-загроз типу «фішинг» на основі застосування емулятора, який на відміну від відомих розглядає деталі модифікацій фішингових атак та їх еволюцію з часом, і який забезпечує підвищення достовірності виявлення таких атак.

2. Набули подальшого розвитку програмно-технічні засоби захисту інформації від атак типу фішинг, які забезпечують виявлення атак типу «фішинг» із застосуванням емулятора з високою достовірністю.

Практичне значення отриманих результатів

В результаті виконаного наукового дослідження було розроблено програмно-технічні засоби виявлення кібер-загроз типу «фішинг», які забезпечуватимуть захист інформації з високою достовірністю.

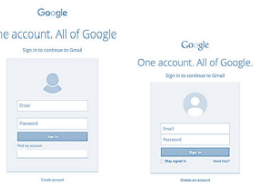
Актуальність дослідження

Фішингові атаки можуть бути у вигляді:

- кримінального програмного забезпечення як послуги та платіжних систем;
- фішингових сайтів
- шкідливого посилання яке публікується в соціальних мережах або надсилається електронною поштою;
- доменного імені, схожого на добре відоме законне доменне ім’я, замінюючи деякі символи омографами (наприклад, G00gle.com, App0le.com)

ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ ВИЯВЛЕННЯ ФІШИНГОВИХ АТАК

- Так звані «фішингові атаки» — це атаки, під час яких фішингові сайти маскуються під легітимні веб-сайти з метою крадіжки конфіденційної інформації.



Фішингову атака, яка імітує сторінку входу в Gmail

Відомі методи виявлення фішингу

1. Метод чорних списків
2. Виявлення фішингу шляхом порівняння з ціллю
3. Виявлення фішингу шляхом перегляду внутрішніх характеристик
4. Подібність ключових слів сторінки
5. Візуальна схожість сторінки

Недоліки відомих методів

В результаті досліджень було виявлено, що наявні фішингові чорні списки мають проблему в тому, що вони не можуть ефективно запобігти раннім атакам і блокують лише випадки атак, а не реальні атаки.

Попри те, що певні методи та засоби можуть компенсувати цей недолік, дослідження виявили рад недоліків, зокрема:

- виявлення фішингових атак погано забезпечують доступ до веб-сервісів,
- недосконалий доступ до набору даних, що унеможливлює для інших відтворити їхні результати та порівняти їх із власним методом.

МОДЕЛЬ ПРОЦЕСУ ФУНКЦІОНУВАННЯ ЕМУЛЯТОРА ВИЯВЛЕННЯ АТАК ТИПУ ФІШИНГ

З метою вирішення поставлених завдань було представлено та побудовано моделі атак типу фішинг, а також моделі процесу функціонування емулятора виявлення атак типу фішинг:

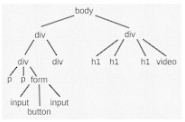
- Модель атаки типу фішинг з підміною сторінки: $M_1 = \langle A_1, S_1, L_1, F_1, U_1, H_1, V_1 \rangle$
- Модель фішингової атаки, що здійснює зміну методу ідентифікації у веб-сервісі: $M_2 = \langle B_2, S_2, L_2, F_2, W_2, Q_2 \rangle$
- Модель атаки типу фішинг, що здійснює копіювання веб-сервісу: $M_3 = \langle D_3, H_3, \mu_3, L_3, F_3, V_3 \rangle$

ВІДБИТОК ФІШИНГОВОЇ СТОРІНКИ: ВЕКТОР ТЕГІВ

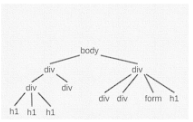
В дослідженні розглянуто здійснювання обчислень засобами залучення векторів тегів фішингової сторінки.

- здійснюється визначення множини тегів HTML
- отримується список із 107 тегів
- виконується призначення довільного, але фіксованого порядку тегів у корпусі та визначення «вектор» розміру корпусу

Вектори тегів вектор тегів для сторінки p2 дорівнює <1, 0 ,0 ,4, 0, 0, 0, 0, 6>



(a) DOM сторінки p_1



(b) DOM сторінки p_2

УДОСКОНАЛЕНИЙ МЕТОД ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

Метод грунтується на основі графіка, який використовується для відстеження та аналізу фішингових модифікацій та розвитку.

Етапи методу:

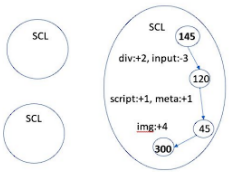
- Побудова вектору ознак
- Побудова змінених тегів

УДОСКОНАЛЕНИЙ МЕТОД ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

Для аналізу розвитку фішингових атак, були виконані наступні кроки

- побудовано модель SCL для кожного векторного кластера тегів
- додавання додаткової інформації, яка використовується для аналізу фішингових модифікацій,
- побудова графіку модифікації фішингової атаки.

Кожен вузол представляє унікальний вектор тегів, а мітка вузла показує кількість екземплярів фішингової атаки, що використовують вказаний для опрацювання вектор.

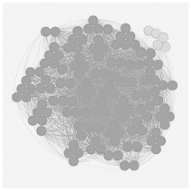


Приклад модифікації фішингової атаки

УДОСКОНАЛЕНИЙ МЕТОД ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

Графи модифікації фішингових атак двох найбільших кластерів

Граф модифікації кластера 0



Граф модифікації кластера 1

Експериментальні дослідження застосування запропонованого методу

1. Застосування баз даних

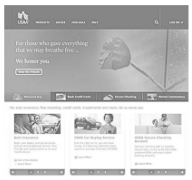
Було реалізовано:

- використання трьох джерел
- зібрано список з 23 554 «перевірених» фішингових сайтів
- автоматичне отримання DOM, надсилаючи веб-сканер за URL-адресою

2. Результати кластеризації

Кількість випадків фішингу	34 551	
Кількість векторів фішингу	22,785	
Кількість легальних сайтів	24,800	
Кількість законних векторів	11 559	
	Модель з P	Модель з WPD та SCL
Оптимальний поріг	0.33	0.26
Кількість позначених кластерів	5,729	5,334
Кількість випадків фішингу в позначених кластерах	90,551 (91.86%)	91,580(92.91%)
Кількість легальних сайтів у фішингових кластерах	128 (0.52%)	161 (0.65%)

Приклад хибнопозитивного результату



Законний сайт

Фішинговий сайт

3. Порівняння кластеризації

- Результати показують, що вдосконалена модель кластеризації виявляє більше варіантів фішингу, ніж оригінальна модель із використанням P.

	Оригінальна модель P	Покращена модель WPD та SCL)
кількість одиночних атак	8020	6991
кількість атак, варіанти яких модель не може виявити, але може виявити інша модель	1052	23
Відстань до найближчого сусіда	0.27-0.32	0.26-0.30
Оптимальний поріг	0.33	0.26

Порівняння результатів здійснення кластеризації

4. Ефективність визначення фішингових варіацій

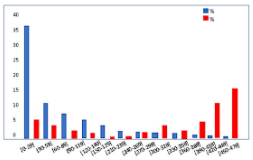
- щоб підвищити продуктивність, було використано прототип кластера, який використовує результати кластеризації для визначення меж кластера

5. Цілі фішингу

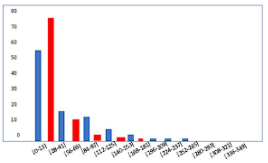
Кластерний індекс	Фірмове найменування цілі фішингу	Кількість сторінок
1	Gmail	11391
2	PayPal	4493
3	Google Docs	2016
4	MailBox	1933
5	DropBox	1619
6	Yahoo	1536
7	Multiple	1421
8	PayPal	1332
9	Free Mobile	1229
10	OneDrive/GoogleDrive	1191

Цільові бренди для 10 найбільших кластерів

6. Життєвий цикл фішингових веб-сервісів



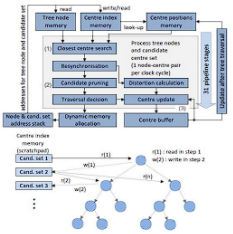
Життєвий цикл кластерів



Середній час між атаками

ПРОГРАНО-АПАРАТНІ ЗАСОБИ ЕМУЛЯТОРА ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

В дослідженні було залучено DE1-SoC Development Kit, який представляє надійну платформу апаратного проектування, побудовану на основі FPGA Altera System-on-Chip (SoC), яка поєднує в собі новітні двоядерні вбудовані ядра Cortex-A9



Архітектура емулятора виявлення кібер-загроз типу «фішинг»

ПРОГРАНО-АПАРАТНІ ЗАСОБИ ЕМУЛЯТОРА ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

Для експериментальних досліджень реалізованого емулятора було зібрано:

- базу даних про фішинг
- URL-адреси екземплярів фішингових атак із
- платформи аналізу безпеки підприємства

Для кожного фішингового сайту:

- отримали DOM, першу URL-адресу (повідомлену), кінцеву URL-адресу (яка відрізняється від першої URL-адреси лише тоді, коли зловмисник використав переспрямування) і знімок екрана кінцевої сторінки.

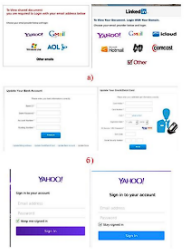
Аналіз основних векторів засобами емулятора

Вручну перевіривши DOM головних векторів, виявилось, що головні вектори можна згрупувати в три категорії:

1. Різні початкові версії атаки.

2. Різні кроки однієї атаки.

3. Копії різних версій цільового сайту.



Типи модифікацій, які спостерігаються під час фішингових атак

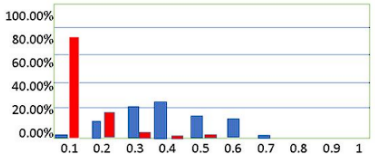
Щоб зрозуміти модифікації, які відбуваються на шляху еволюції, було проаналізовано загальні модифікації серед кластерів.

10 найпоширеніших модифікованих тегів (MT) і кількість кластерів, у яких вони з' являються, є <script> (57), (53), <img> (52), <a> (51), <input> (50),
 (48), <link> (47), (47), <p> (41) і <style> (40).

10 найпоширеніших підмножин тегів модифікації (MTS) серед вибраних 62 кластерів

MTS	Кількість кластерів	%	Кількість сторінок	%
{a, div}	45	72.58%	403	24.82%
{div, img}	44	70.97%	286	17.61%
{div, script}	44	70.97%	403	24.82%
{div, span}	40	64.52%	264	16.26%
{br, div}	39	62.90%	215	13.24%
{img, script}	39	62.90%	199	12.25%
{a, img}	37	59.68%	235	14.47%
{link, script}	37	59.68%	215	13.24%
{script, span}	35	56.45%	174	10.71%
{input, span}	35	56.45%	161	9.91%

Гістограма індексу Жакара для 10 найвищих MT і MTS



Публікації за матеріалами дипломної роботи

Дослідження, представлені у кваліфікаційній роботі, проводились в рамках держбюджетної НДР Хмельницького національного університету 1Б-2021 «Самоорганізована розподілена система виявлення зловмисного програмного забезпечення в комп'ютерних мережах» (ДР № 0121U109936) 2021-2022 рр.

За темою дипломної роботи опубліковано статтю на тему «Дослідження методів виявлення кібератак в комп'ютерних мережах як засобів до побудови резильєнтних до вторгнень IT-інфраструктур» в фаховому журналі «Комп'ютерні системи та інформаційні технології» №3 за 2021 рік [1], а також опубліковано тези у Всеукраїнській науково-практичній конференції Актуальні Проблеми Комп'ютерних Наук (АПКН-2021) [2]. Було взято участь у Всеукраїнській науково-практичній конференції Актуальні Проблеми Комп'ютерних Наук.

Висновки

У першому розділі проведено дослідження методів та засобів виявлення фішингових атак, а саме проаналізовано суть поняття фішингових атак.

Також в розділі було досліджено переваги та недоліки методів виявлення фішингових атак. Досліджено життєвий цикл фішингових атак. Проаналізовано основні механізми виявлення фішингових сайтів, виявлення фішингу за допомогою чорних списків, виявлення фішингу шляхом порівняння з ціллю, виявлення фішингу шляхом перегляду внутрішніх характеристик. В розділі представлено постановку задачі.

У другому розділі було подано основні аспекти моделі процесу функціонування емулятора виявлення атак типу фішинг, зокрема представлені моделі фішингових атак, подано процес виявлення фішингових копій і варіацій, спроєктовано відбиток фішингової сторінки: вектор тегів.

Висновки

Також в розділі проведено моделювання схожості фішингових сторінок: пропорційна відстань, наведено алгоритм кластеризації.

У третьому розділі представлено удосконалений метод виявлення кіберзагроз типу «фішинг», який розглядає деталі модифікацій та їх еволюцію з часом, який грунтується на основі графіка, який використовується для відстеження та аналізу фішингових модифікацій та розвитку.

Висновки

В четвертому розділі представлено реалізацію удосконаленого методу виявлення кіберзагроз типу «фішинг» у вигляді емулятора - програмно-апаратного засобу, який грунтується на використанні програмованій користувачем вентильній матриці, ПКВМ (FPGA). В дослідженні було залучено DE1-SoC Development Kit, який представляє надійну платформу апаратного проектування, побудовану на основі FPGA Altera System-on-Chip (SoC). Також для вирішення задачі проєктування та реалізації емулятора виявлення кібер-загроз типу «фішинг» було побудованого його архітектуру.

В результаті здійсненого проєктування емулятора стало можливим виконати експериментальні дослідження з окрема: отримати результати аналізу засобами емулятора на основі баз даних фішингових сайтів, побудувати вектори та отримати результати кластеризації, здійснити виявлення джерела змін, виконати вибір зразка кластерів, здійснити аналіз отриманих результатів, достовірність яких досягає 90%.

Додаток Г

(обов’язковий)

Презентація доповіді

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Кафедра комп’ютерної інженерії та системного програмування

Михасько Яна Юріївна

Метод та засоби емулятора виявлення кібер-загроз типу «фішинг»

Науковий керівник – д.т.н. доц. Лисенко С.М.

Хмельницький - 2022

Мета і задачі дослідження

Метою роботи є підвищення достовірності та виявлення атак типу фішинг на основі застосування емулятора

Об'єкт дослідження – процес виявлення атак типу фішинг на основі застосування емулятора.

Предмет дослідження – моделі, методи та програмно-технічні засобами виявлення атак типу фішинг на основі застосування емулятора.

Мета і задачі дослідження

Поставлена мета досягається розв'язанням таких основних задач:

1. Дослідити методи та засоби виявлення фішингових атак
2. Дослідити переваги та недоліки методів виявлення фішингових атак. Дослідити життєвий цикл фішингових атак.
3. Розробити моделі процесу функціонування емулятора виявлення атак типу фішинг
4. Провести моделювання схожості фішингових сторінок
5. Розробити удосконалений метод виявлення кіберзагроз типу «фішинг».
6. Здійснити реалізацію удосконаленого методу виявлення кіберзагроз типу «фішинг»у вигляді емулятора - програмно-апаратного засобу.

Наукова новизна отриманих результатів

1. Удосконалено метод виявлення кібер-загроз типу «фішинг» на основі застосування емулятора, який на відміну від відомих розглядає деталі модифікацій фішингових атак та їх еволюцію з часом, і який забезпечує підвищення достовірності виявлення таких атак.

2. Набули подальшого розвитку програмно-технічні засоби захисту інформації від атак типу фішинг, які забезпечують виявлення атак типу «фішинг» із застосуванням емулятора з високою достовірністю.

Практичне значення отриманих результатів

В результаті виконаного наукового дослідження було розроблено програмно-технічні засоби виявлення кібер-загроз типу «фішинг», які забезпечуватимуть захист інформації з високою достовірністю.

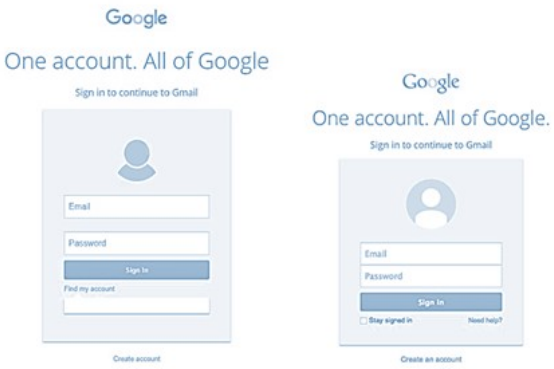
Актуальність дослідження

Фішингові атаки можуть бути у вигляді:

- кримінального програмного забезпечення як послуги та платіжних систем;
- фішингових сайтів
- шкідливого посилання яке публікується в соціальних мережах або надсилається електронною поштою;
- доменного імені, схожого на добре відоме законне доменне ім'я, замінюючи деякі символи омографами (наприклад, G00gle.com, App0le.com)

ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ ВИЯВЛЕННЯ ФІШИНГОВИХ АТАК

- Так звані «фішингові атаки» — це атаки, під час яких фішингові сайти маскуються під легітимні веб-сайти з метою крадіжки конфіденційної інформації.



Фішингову атака, яка імітує сторінку входу в Gmail

Відомі методи виявлення фішингу

1. Метод чорних списків
2. Виявлення фішингу шляхом порівняння з ціллю
3. Виявлення фішингу шляхом перегляду внутрішніх характеристик
4. Подібність ключових слів сторінки
5. Візуальна схожість сторінки

Недоліки відомих методів

В результаті досліджень було виявлено, що наявні фішингові чорні списки мають проблему в тому, що вони не можуть ефективно запобігти раннім атакам і блокують лише випадки атак, а не реальні атаки.

Попри те, що певні методи та засоби можуть компенсувати цей недолік, дослідження виявили рад недоліків, зокрема:

- виявлення фішингових атак погано забезпечують доступ до веб-сервісів,
- недосконалий доступ до набору даних, що унеможливлює для інших відтворити їхні результати та порівняти їх із власним методом.

МОДЕЛЬ ПРОЦЕСУ ФУНКЦІОНУВАННЯ ЕМУЛЯТОРА ВИЯВЛЕННЯ АТАК ТИПУ ФІШИНГ

З метою вирішення поставлених завдань було представлено та побудовано моделі атак типу фішинг, а також моделі процесу функціонування емулятора виявлення атак типу фішинг.

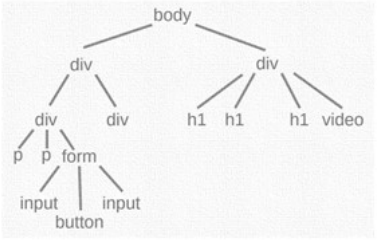
- Модель атаки типу фішинг з підміною сторінки: $M_1 = \langle A_1, S_1, L_1, F_1, U_1, H_1, V_1 \rangle$.
- Модель фішингової атаки, що здійснює зміну методу ідентифікації у веб-сервісі: $M_2 = \langle B_2, S_2, L_2, F_2, W_2, Q_2 \rangle$
- Модель атаки типу фішинг, що здійснює копіювання веб-сервісу: $M_3 = \langle D_3, H_3, \mu_3, L_3, F_3, V_3 \rangle$

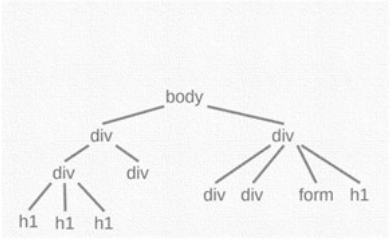
ВІДБИТОК ФІШИНГОВОЇ СТОРІНКИ: ВЕКТОР ТЕГІВ

В дослідженні розглянуто здійснювання обчислень засобами залучення векторів тегів фішингової сторінки.

- здійснюється визначення множини тегів HTML
- отримується список із 107 тегів
- виконується призначення довільного, але фіксованого порядку тегів у корпусі та визначення «вектор» розміру корпусу

Вектори тегів вектор тегів для сторінки p2 дорівнює <1, 0 ,0 ,4, 0, 0, 0, 0, 6>





(a) DOM сторінки p1 (b) DOM сторінки p2

УДОСКОНАЛЕНИЙ МЕТОД ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

Метод ґрунтується на основі графу, який використовується для відстеження та аналізу фішингових модифікацій та розвитку.

Етапи методу:

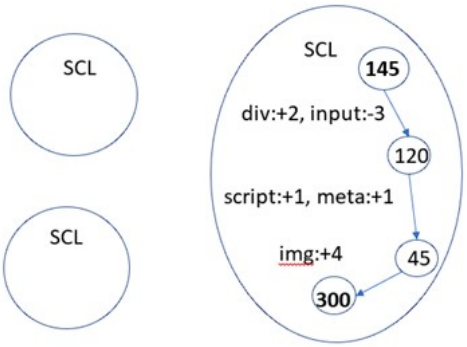
- Побудова вектору ознак
- Побудова змінених тегів

УДОСКОНАЛЕНИЙ МЕТОД ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

Для аналізу розвитку фішингових атак, були виконані наступні кроки

- побудовано модель SCL для кожного векторного кластера тегів
- додавання додаткової інформації, яка використовується для аналізу фішингових модифікацій,
- побудова графіку модифікації фішингової атаки.

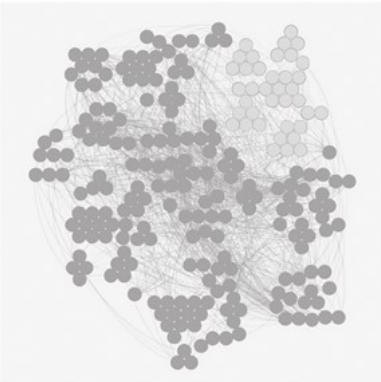
Кожен вузол представляє унікальний вектор тегів, а мітка вузла показує кількість екземплярів фішингової атаки, що використовують вказаний для опрацювання вектор



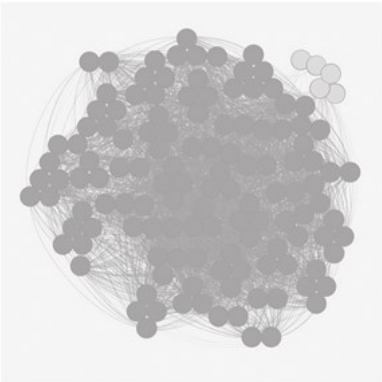
Приклад модифікації фішингової атаки

УДОСКОНАЛЕНИЙ МЕТОД ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

Графи модифікації фішингових атак двох найбільших кластерів



Граф модифікації кластера 0



Граф модифікації кластера 1

Експериментальні дослідження застосування запропонованого методу

1. Застосування баз даних

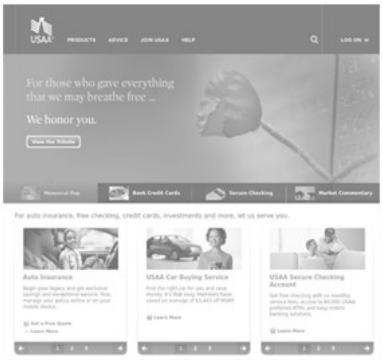
Було реалізовано:

- використання трьох джерел
- зібрано список з 23 554 «перевірених» фішингових сайтів
- автоматичне отримання DOM, надсилаючи веб-сканер за URL-адресою

2. Результати

Кількість випадків фішингу	34 551	
Кількість векторів фішингу	22,785	
Кількість легальних сайтів	24,800	
Кількість законних векторів	11 539	
	Модель з P	Модель з WPD та SCL
Оптимальний поріг	0.33	0.26
Кількість позначених кластерів	5,729	5,334
Кількість випадків фішингу в позначених кластерах	90,551 (91.86%)	91,580(92.91%)
Кількість легальних сайтів у фішингових кластерах	128 (0.52%)	161 (0.65%)

Приклад хибнопозитивного результату



Законний сайт

Фішинговий сайт

3. Порівняння кластеризації

- Результати показують, що вдосконалена модель кластеризації виявляє більше варіантів фішингу, ніж оригінальна модель із використанням P.

	Оригінальна модель P	Покращена модель WPD та SCL)
кількість одиночних атак	8020	6991
кількість атак, варіанти яких модель не може виявити, але може виявити інша модель	1052	23
Відстань до найближчого сусіда	0.27-0.32	0.26-0.30
Оптимальний поріг	0.33	0.26

Порівняння результатів здійснення кластеризації

4. Ефективність визначення фішингових варіацій

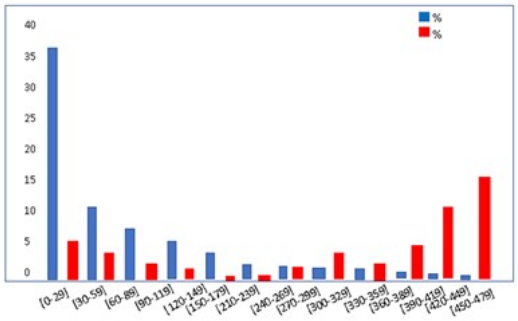
- щоб підвищити продуктивність, було використано прототип кластера, який використовує результати кластеризації для визначення меж кластера

5. Цілі фішингу

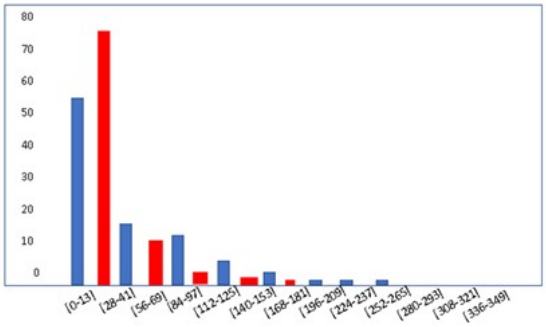
Кластерний індекс	Фірмове найменування цілі фішингу	Кількість сторінок
1	Gmail	11391
2	PayPal	4493
3	Google Docs	2016
4	MailBox	1933
5	DropBox	1619
6	Yahoo	1536
7	Multiple	1421
8	PayPal	1332
9	Free Mobile	1229
10	OneDrive/GoogleDrive	1191

Цільові бренди для 10 найбільших кластерів

6. Життєвий цикл фішингових веб-сервісів



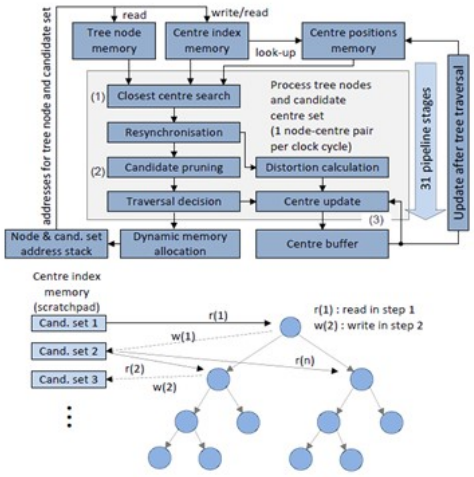
Життєвий цикл кластерів



Середній час між атаками

ПРОГРАНО-АПАРАТНІ ЗАСОБИ ЕМУЛЯТОРА ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

В дослідженні було залучено DE1-SoC Development Kit, який представляє надійну платформу апаратного проектування, побудовану на основі FPGA Altera System-on-Chip (SoC), яка поєднує в собі новітні двоядерні вбудовані ядра Cortex-A9



Архітектура емулятора виявлення кібер-загроз типу «фішинг»

ПРОГРАНО-АПАРАТНІ ЗАСОБИ ЕМУЛЯТОРА ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ ТИПУ «ФІШИНГ»

Для експериментальних досліджень реалізованого емулятора було зібрано:

- базу даних про фішинг
- URL-адреси екземплярів фішингових атак із
- платформи аналізу безпеки підприємства

Для кожного фішингового сайту:

- отримали DOM, першу URL-адресу (повідомлену), кінцеву URL-адресу (яка відрізняється від першої URL-адреси лише тоді, коли зловмисник використав переспрямування) і знімок екрана кінцевої сторінки.

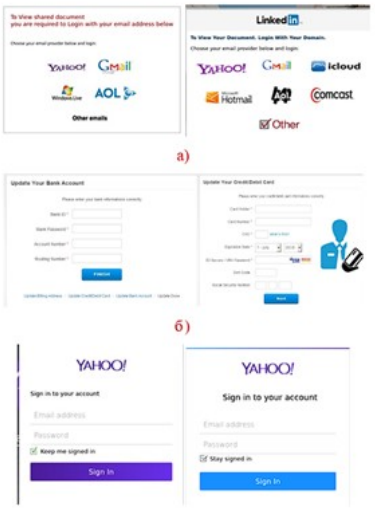
Аналіз основних векторів засобами емулятора

Вручну перевіривши DOM головних векторів, виявилось, що головні вектори можна згрупувати в три категорії:

1. Різні початкові версії атаки.

2. Різні кроки однієї атаки.

3. Копії різних версій цільового сайту.



Типи модифікацій, які спостерігаються під час фішингових атак

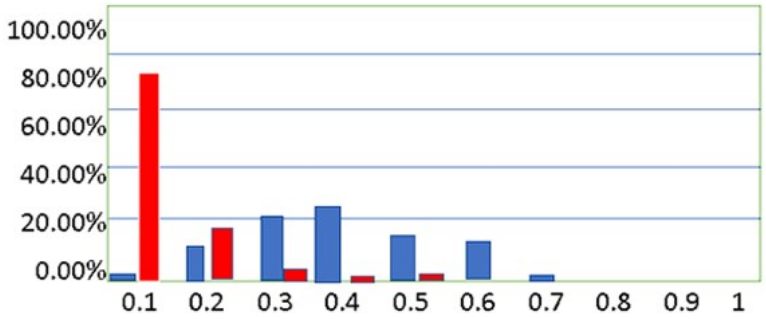
Щоб зрозуміти модифікації, які відбуваються на шляху еволюції, було проаналізовано загальні модифікації серед кластерів.

10 найпоширеніших модифікованих тегів (MT) і кількість кластерів, у яких вони з'являються, є <script> (57), (53), <img> (52), <a> (51), <input> (50),
 (48), <link> (47), (47), <p> (41) і <style> (40).

10 найпоширеніших підмножин тегів модифікації (MTS) серед вибраних 62 кластерів

MTS	Кількість кластерів	%	Кількість сторінок	%
{a, div}	45	72.58%	403	24.82%
{div, img}	44	70.97%	286	17.61%
{div, script}	44	70.97%	403	24.82%
{div, span}	40	64.52%	264	16.26%
{br, div}	39	62.90%	215	13.24%
{img, script}	39	62.90%	199	12.25%
{a, img}	37	59.68%	235	14.47%
{link, script}	37	59.68%	215	13.24%
{script, span}	35	56.45%	174	10.71%
{input, span}	35	56.45%	161	9.91%

Гістограма індексу Жакара для 10 найвищих MT і MTS



Публікації за матеріалами дипломної роботи

Дослідження, представлені у кваліфікаційній роботі, проводились в рамках держбюджетної НДР Хмельницького національного університету 1Б-2021 «Самоорганізована розподілена система виявлення зловмисного програмного забезпечення в комп'ютерних мережах» (ДР № 0121U109936) 2021-2022 рр.

За темою дипломної роботи опубліковано статтю на тему «Дослідження методів виявлення кібератак в комп'ютерних мережах як засобів до побудови резильентних до вторгнень IT-інфраструктур» в фаховому журналі «Комп'ютерні системи та інформаційні технології» №3 за 2021 рік [1], а також опубліковано тези у Всеукраїнській науково-практичній конференції Актуальні Проблеми Комп'ютерних Наук (АПКН-2021) [2]. Було взято участь у Всеукраїнській науково-практичній конференції Актуальні Проблеми Комп'ютерних Наук.

Висновки

У першому розділі проведено дослідження методів та засобів виявлення фішингових атак, а саме проаналізовано суть поняття фішингових атак.

Також в розділі було досліджено переваги та недоліки методів виявлення фішингових атак. Досліджено життєвий цикл фішингових атак. Проаналізовано основні механізми виявлення фішингових сайтів, виявлення фішингу за допомогою чорних списків, виявлення фішингу шляхом порівняння з ціллю, виявлення фішингу шляхом перегляду внутрішніх характеристик. В розділі представлено постановку задачі.

У другому розділі було подано основні аспекти моделі процесу функціонування емулятора виявлення атак типу фішинг, зокрема представлені моделі фішингових атак, подано процес виявлення фішингових копій і варіацій, спроєктовано відбиток фішингової сторінки: вектор тегів.

Висновки

Також в розділі проведено моделювання схожості фішингових сторінок: пропорційна відстань, наведено алгоритм кластеризації.

У третьому розділі представлено удосконалений метод виявлення кіберзагроз типу «фішинг», який розглядає деталі модифікацій та їх еволюцію з часом, який ґрунтується на основі графіка, який використовується для відстеження та аналізу фішингових модифікацій та розвитку.

Висновки

В четвертому розділі представлено реалізацію удосконаленого методу виявлення кіберзагроз типу «фішинг»у вигляді емулятора - програмно-апаратного засобу, який ґрунтується на використанні програмованій користувачем вентильній матриці, ПКВМ (FPGA). В дослідженні було залучено DE1-SoC Development Kit, який представляє надійну платформу апаратного проектування, побудовану на основі FPGA Altera System-on-Chip (SoC). Також для вирішення задачі проєктування та реалізації емулятора виявлення кібер-загроз типу «фішинг» було побудованого його архітектуру.

В результаті здійсненого проєктування емулятора стало можливим виконати експериментальні дослідження з окрема: отримати результати аналізу засобами емулятора на основі баз даних фішингових сайтів, побудувати вектори та отримати результати кластеризації, здійснити виявлення джерела змін, виконати вибір зразка кластерів, здійснити аналіз отриманих результатів, достовірність яких досягає 90%.

UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1010993695

Дата перевірки:
29.04.2022 08:36:40 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
29.04.2022 08:43:00 EEST

ID користувача:
100005591

Назва документа: **Метод та засоби емулятора виявлення кібер-загроз типу «фішинг»**

Кількість сторінок: **89** Кількість слів: **16563** Кількість символів: **117137** Розмір файлу: **6.00 MB** ID файлу: **1010898795**

1.55%
Схожість

Найбільша схожість: **0.52%** з джерелом з Бібліотеки (ID файлу: **1010865386**)

0.72% Джерела з Інтернету	58	Сторінка 91
1.01% Джерела з Бібліотеки	100	Сторінка 91

0.18% Цитат

Цитати	1	Сторінка 92

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	21

Fri Apr 29 07:47:44 EEST 2022, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 0.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 12%**

ID: 103208 Название: Метод та засоби емулятора виявлення кібер-загроз типу «фішинг» Добавлено в БД: 2022-04-29 Авторы: Михасько Я.Ю. Руководители: Лисенко С.М. Консультанты: Оппоненты:	Документ		Суммарное совпадение по Базе Данных	
	Символы	Лексемы	Символы	Лексемы
	104718	835	509 (0%)	8 (1%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

Дипломник: Михасько Яна Юріївна

Тема: Метод та засоби емулятора виявлення кібер-загроз типу «фішинг»

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг дипломної роботи:

Кількість листів креслень – ; кількість сторінок записки 85

1. Короткий зміст роботи та прийнятих рішень Представлена робота присвячена розробленню методу та засобів емулятора виявлення кібер-загроз типу «фішинг»

2. Висновок про відповідність роботи дипломному завданню Дипломна робота відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі проведено дослідження методів та засобів виявлення фішингових атак, а саме проаналізовано суть поняття фішингових атак. У другому розділі було подано основні аспекти моделі процесу функціонування емулятора. У третьому розділі представлено удосконалений метод виявлення кіберзагроз типу «фішинг», який розглядає деталі модифікацій та їх еволюцію з часом. У четвертому розділі представлено реалізацію удосконаленого методу виявлення кіберзагроз типу «фішинг»у вигляді емулятора

4. Позитивні сторони роботи: В результаті виконаного наукового дослідження було розроблено програмно-технічні засоби виявлення кібер-загроз типу «фішинг», які забезпечуватимуть захист інформації з високою достовірністю.

5. Негативні сторони роботи: Не в повній мірі здійснено аналіз DE1-SoC Development Kit FPGA Altera System-on-Chip (SoC).

6. Оцінка графічного оформлення та пояснювальної записки роботи: –

7. Відгук про роботу в цілому: В загальному робота виконана на достатньому рівні.

8. Інші зауваження: –

9. Оцінка дипломної роботи:

Розглянувши позитивні та негативні сторони представленої дипломної роботи вважаю, що робота заслуговує оцінки «відмінно» 4,75 (А)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Джулій В.М. , к.т.н., доцент, кафедри кібербезпеки Хмельницького національного університету

"27" квітня 2022р.

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Михасько Яна Юріївна

ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-20-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіатоповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

27.04.2022р

дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА СИСТЕМНОГО ПРОГРАМУВАННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Метод та засоби емулятора виявлення кібер-загроз типу «фішинг»

Автор: Михасько Яна Юріївна

Спеціальність: 123 – Комп'ютерна інженерія та програмування

Освітня програма: освітньо-наукова

Науковий керівник: Лисенко С.М., д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є незначними, законними і не є плагіатом, оскільки:

1) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10 джерелами з бібліотек та 15 джерелами з мережі Інтернет;
2) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 1.55% і адресується до 25 першоджерел, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ С.М. Лисенко

Гарант ОП _____ О. С. Савенко

Завідувач кафедри КІСП _____ Т. О. Говорущенко