

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«На правах рукопису»

УДК 004.056

«До захисту допущено»

В.о. завідувача кафедри

_____ М.В.Грайворонський

“___” грудня 2019 р.

Магістерська дисертація
на здобуття ступеня магістра

зі спеціальності: 125 Кібербезпека

на тему: _____ *Методи виявлення шкідливих Javascript-сценаріїв* _____

Виконав: студент II курсу, групи ФБ-81мп
(шифр групи)

_____ *Самойленко Роман Юрійович* _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник _____ *доцент, к.ф.-м.н. Грайворонський М.В.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____
(підпис)

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
 Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
 Спеціальність (спеціалізація) – 125 Кібербезпека («Системи і технології кібербезпеки»)

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ М.В.Грайворонський
 (підпис)

«__» _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Самойленко Роман Юрійович

(прізвище, ім'я, по батькові)

1. Тема дисертації Методи виявлення шкідливих Javascript-сценаріїв

науковий керівник дисертації Грайворонський Микола Владленович,
доцент, к.ф.-м.н.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «15» 11 2019 р. № 3927

2. Термін подання студентом дисертації 10.12.2019 р.

3. Об'єкт дослідження Виявлення шкідливих Javascript-сценаріїв

4. Вихідні дані підвищення точності визначення шкідливості Javascript на більш ніж 15%, зниження кількості помилок першого і другого родів.

5. Перелік завдань, які потрібно розробити 1. Зібрати інформацію про шкідливе програмне забезпечення, створене за допомогою Javascript. 2. Розглянути літературні джерела за темою. 3. Визначити напрям дослідження та сформулювати проблему. 4. Провести детальний аналіз вибраного напрямку. 5. На основі отриманих даних розробити допоміжні програмні засоби. 6. Описати вирішення проблеми. 7. Розробити стартап-проект, який дозволить реалізувати вирішення проблеми в якості конкурентноспроможного продукту. 8. Оформити дисертацію.

6. Орієнтовний перелік ілюстративного матеріалу Рисунок 1.1 - Популярність Javascript відносно інших мов програмування по роках. 1.5 Зміни векторів атак по роках. 2.1.1 Приклад вигляду "points-to" графів, основаних на коді. 2.1.2 Приклад вигляду абстрактного синтаксичного дерева. 3.1 Схема системи класифікації.

7. Орієнтовний перелік публікацій _____

8. Дата видачі завдання 01.05.2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Збір інформації	01.05.2019 – 01.06.2019	
2	Вивчення літературних джерел	01.06.2019 – 01.07.2019	
3	Визначення напрямку дослідження та пошук проблеми. Аналіз вибраного напрямку.	01.07.2019 – 01.08.2019	
4	Розробка програмних засобів	01.08.2019 - 01.09.2019	
5	Вирішення проблеми	01.10.2019 – 01.11.2019	
6	Розробка стартап-проекту	01.11.2019 — 20.11.2019	
7	Оформлення магістерської дисертації	20.11.2019 — 05.12.2019	
8	Подання дисертації на передзахист	06.12.2019	

Студент

(підпис)

(ініціали, прізвище)

Науковий керівник дисертації

(підпис)

(ініціали, прізвище)

РЕФЕРАТ

Обсяг роботи 82 сторінки, 7 ілюстрації, 27 таблиць, 1 додатки, 20 джерело літератури.

Об'єктом дослідження є шкідливі Javascript-сценарії.

Предметом дослідження є методи виявлення шкідливих Javascript-сценаріїв.

Детектування, методи виявлення, класифікація, javascript, шкідливий програмний засіб, статичний аналіз, машинне навчання.

ABSTRACT

Size of work: 82 pages, 7 pics., 27 tables, 1 applications, 20 sources.

Object of study is the process of detecting malicious Javascript scenarios.

Subject of study is the methods for detecting malicious Javascript scenarios.

Detection methods, classification, javascript, malicious javascript, static analysis, machine training.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів.....	8
Вступ.....	9
1 Шкідливі Javascript-сценарії.....	12
1.1 Основні відомості про Javascript.....	12
1.2 Безпека у Javascript.....	16
1.3 Класифікація шкідливих Javascript-сценаріїв.....	18
1.3.1 3 точки зору загроз.....	18
1.3.2 3 точки зору атак.....	19
1.4 Способи захисту	19
1.5 Сучасні тенденції	21
1.6 Формування проблеми.....	22
Висновки до розділу 1.....	23
2 Аналіз методів виявлення шкідливих Javascript-сценаріїв.....	24
2.1 Основні відомості про методи виявлення.....	24
2.2 Вибір методів для вдосконалення.....	29
2.3 Створення системи виявлення.....	29
2.3.1 Опис.....	31
2.3.2 Етап видалення файлів JavaScript.....	31
2.3.3 Етап видалення ознак.....	31
2.3.4 Етап підготовки даних.....	32
2.4 Огляд допоміжних та існуючих рішень.....	32
2.5 Огляд класифікаторів.....	34
2.6 Обфускація.....	38
2.7 Проблема методів виявлення шкідливих Javascript-сценаріїв.....	40
Висновки до розділу 2.....	41
3 Удосконалення методів виявлення шкідливих Javascript-сценаріїв.....	42
3.1 Вирішення проблеми.....	42
3.2 Розрахунки точності роботи.....	45
3.3 Порівняння точності роботи.....	46

	7
Висновки до розділу 3.....	48
4 Стартап проект.....	49
4.1 Опис ідеї проекту.....	51
4.2 Технологічний аудит ідеї проекту.....	51
4.3 Розроблення ринкової можливості стартап-проекту.....	51
4.4 Розроблення ринкової стратегії стартап-проекту.....	59
4.5 Розроблення маркетингової програми стартап-проекту.....	62
Висновки до розділу 4.....	65
Висновки.....	66
Перелік джерел посилань.....	67
Додаток А.....	69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JS – Javascript;
HTML – HyperText Markup Language;
CSS – Cascading Style Sheets;
URL – Uniform Resource Locator;
ML – Machine Learning;
DOM – Document Object Model;
HTTP - HyperText Transfer Protocol;
SOP – Same Origin Policy;
API – Application Programming Interface;
ПЗ – програмне забезпечення;
ПК – персональний комп'ютер;
ВМ – віртуальна машина

ВСТУП

JavaScript - це динамічна мова програмування, що разом з HTML та CSS використовується на стороні клієнта для керування поведінкою веб-сторінок. Вона використовується переважною більшістю сучасних веб-сайтів та підтримується всіма сучасними веб-браузерами. Однак, останніми роками мова JS стала дуже зручною мовою для створення веб-атак. Шкідливі скрипти можна вбудувати у веб-сторінку і вони будуть працювати кожний раз, коли сторінка завантажується в браузері. Такі скрипти можуть ухилятися від засобів безпеки таких як файерволи, антивірусне програмне забезпечення, тощо. JS дозволяє творцям веб-сайтів запускати будь-який код під час його відвідування користувачем. Кіберзлочинці регулярно маніпулюють кодом на незліченній кількості веб-сайтів, щоб змусити їх виконувати зловмисні функції. Для програмістів JavaScript — це така мова програмування, неправильне використання якої легко може призвести до створення великої кількості бекдорів.

Сучасні користувачі не уявляють свого життя без мережі Інтернет. Онлайн банкінг, перегляд мультимедіа, соціальні мережі — ці залежності сучасних користувачів дуже допомагають кіберзлочинцям обманювати їх. За допомогою шкідливого JS Інтернет-нападники дуже часто переспрямовують користувачів на зкомпрометовані веб-сайти та таким чином обманом отримують потрібну їм інформацію.

Нові веб-атаки з використанням JS відбуваються щодня, а їх загальна кількість стрімко збільшується з кожним роком. Через зростаючі ризики та збитки це змушує бізнес, громадськість та приватних осіб ставитись до питань безпеки більш серйозно. Саме тому виявлення шкідливих Javascript-сценаріїв є важливим та актуальним завданням.

Наразі точному визначенню того, чи є JS-сценарій шкідливий заважає багато факторів. Найголовніші з них — постійна зміна способів розповсюдження шкідливого ПЗ та їх постійне маскування, зміна зовнішнього вигляду шкідливого коду. Кіберзлочинці добре приховують зловмисні функції коду і деколи дуже

важко правильно відповісти на питання, чи є код шкідливим. Ця невизначенність породжує велику кількість помилок, як першого так і другого родів, а отже знижує загальну точність виявлення.

У цій магістерській дисертації проводиться дослідження, мета якого - підвищити точність виявлення шкідливих JS-сценаріїв та удосконалити існуючі методи виявлення.

Актуальність роботи. Останнім часом шкідливі Javascript-сценарії набувають все більшого розповсюдження. Оскільки Javascript переважно працює на стороні клієнта, збитків можуть зазнати не тільки компанії, а і звичайні користувачі. Саме через це дуже важливо вміти ефективно виявляти зловмисні Javascript-сценарії та запобігати їх роботі.

Мета і завдання дослідження. Розробка класифікації методів виявлення шкідливих Javascript-сценаріїв. Підвищення точності роботи існуючих методів.

Об'єкт дослідження. Виявлення шкідливих Javascript-сценаріїв.

Предмет дослідження. Методи виявлення шкідливих Javascript-сценаріїв.

Методи дослідження. *Аналіз.* Використовувався під час вивчення та усвідомлення нової інформації. *Класифікація.* Використовувався по ходу всієї роботи для визначення шкідливості Javascript-сценарію. *Порівняння.* Використовувався під час визначення точності роботи покращених методів у порівнянні з незміненими методами. *Вимірювання.* Використовувався під час підрахунку точної кількості правильної та неправильної класифікації зразків Javascript-коду.

Наукова новизна одержаних результатів. Удосконалено точність роботи існуючих методи виявлення шкідливих Javascript-сценаріїв.

Практичне значення одержаних результатів. Опираючись на результати даної роботи, підприємства, які займаються розробкою та підтримкою систем виявленням та запобіганням вторгнення, можуть підвищити ефективність своєї роботи при аналізі шкідливих Javascript-сценаріїв.

Ключові слова. Детектування, методи виявлення, класифікація, javascript, шкідливий програмний засіб, статичний аналіз, машинне навчання.

1 ШКІДЛИВІ JAVASCRIPT-СЦЕНАРІЇ

1.1 Основні відомості про Javascript

Javascript — мультипарадигмена, об'єктно-орієнтована, прототипна мова програмування. Це багатоцільова мова і використовується в дуже різних задачах:

- серверне програмування;
- створення мобільних аплікацій;
- створення прикладного програмного забезпечення;
- надання веб-сторінкам інтерактивності;
- тощо.

JavaScript є скрізь. Вже сьомий рік поспіль ця мова займає першу сходинку серед найбільш використовуваних мов програмування. Близько 67.8% розробників застосовують її у 2019 році. Підйом JS до найпопулярнішої у світі мови програмування відбувався синхронно із зростанням та розвитком мережі Інтернет вцілому. На рисунку 1.1 вказані позиції Javascript за популярністю по роках.

Колись мова JS була створена через необхідність, а зараз без неї не можуть уявити Інтернет. Вона використовується для створення близько 95,2% (або 1,52 мільярдів) веб-сайтів, включаючи найбільші у світі, такі як Facebook та YouTube. Без JS сучасні користувачі не мали б безлічі корисних сервісів, наприклад таких як Google Maps.

Зазвичай Javascript використовується у якості мови для програмного доступу до застосунків. Найбільш широко використовується у браузерях як мова сценаріїв(скриптів) на стороні клієнта, що надає можливість взаємодіяти з користувачем, обмінюватись даними з сервером, змінювати зовнішній вигляд та поведінку веб-сторінок.

JavaScript - це передусім клієнтська мова, тобто він працює на вашому комп'ютері у вашому браузері. Однак останнім часом впровадження Node.js дозволило JavaScript виконувати код і на серверах.

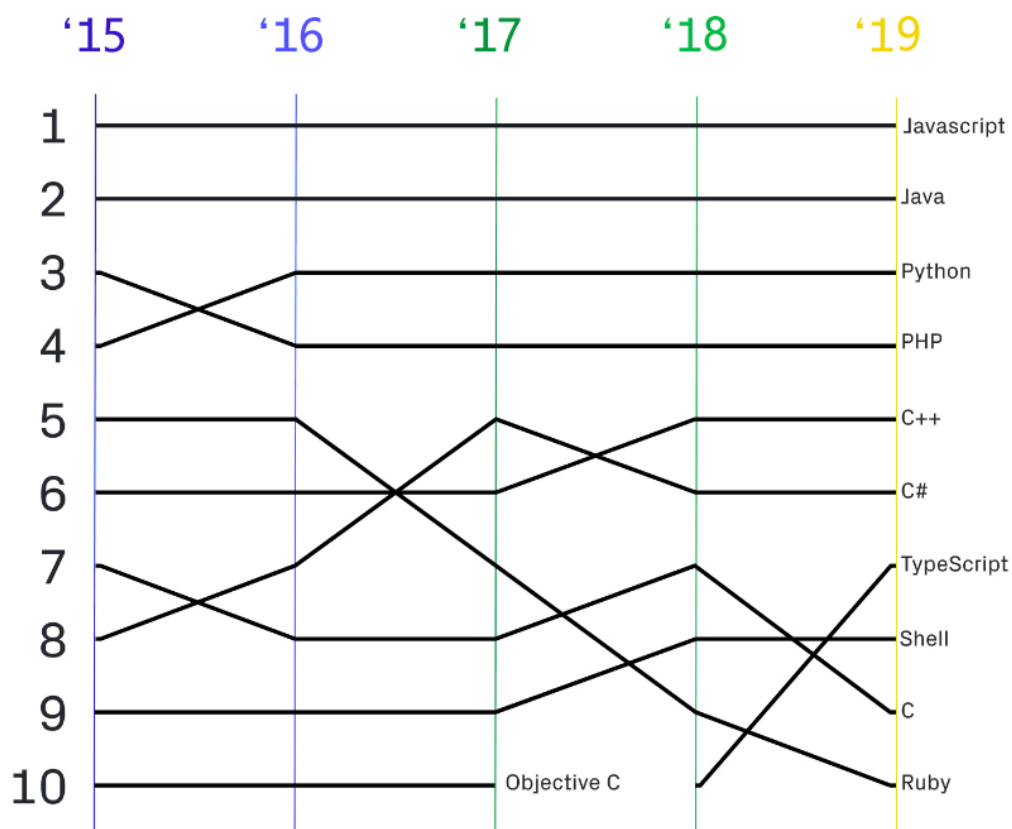


Рисунок 1.1 - Популярність Javascript відносно інших мов програмування по роках

З моменту випуску JavaScript перевершив за популярністю такі популярні мови як Java, Flash та інші. Її легко вивчити, вона має відкриту до пропозицій та доброзичливу спільноту і, що найголовніше, дозволяє розробникам швидко створювати корисні додатки з аудиторією в мільйони користувачів.

Початок 1990-х був важливим часом для Інтернету. Ключові гравці Інтернет-технологій, такі як Netscape та Microsoft, опинилися в розпалі війн браузерів, і Netscape Navigator та Internet Explorer від Microsoft йшли нарівні. У вересні 1995 року програміст Netscape на ім'я Брандан Ейх розробив нову скриптову мову лише за 10 днів. Спочатку вона була названа Mocha, але швидко стала відома як LiveScript, а пізніше JavaScript.

У 1997 році завдяки швидкому зростанню JavaScript, стало зрозуміло, що хтось повинен керувати розробкою мови та належним чином підтримувати її. Netscape вирішив передати завдання створення нової мовної специфікації Європейській асоціації виробників комп'ютерів (ECMA) - органу, заснованому з

метою стандартизації обчислень. Специфікації ECMA були позначені як ECMA-262, а мови ECMAScript включав в себе JavaScript, JScript та ActionScript.

2005 рік виявився проривним для JavaScript. Документ, опублікований Джессі Джеймсом Гарреттом, представив Аїах, революційний набір технологій, що включав JavaScript. Аїах значно покращив користувацький досвід, дозволяючи користуватися веб-сторінками так, наче вони є програмами для настільних ПК. Це зробило JS популярною професійною мовою програмування. Цей документ вважається наріжним каменем спільноти JavaScript. У той час у JavaScript було багато проблем, включаючи її багатослівний характер під час виконання простих речей та проблеми несумісності між браузерами. У відповідь на це спільнота випустила jQuery та низку інших фреймворків.

Також створено низку фреймворків і бібліотек JavaScript, таких як Ember, Angular, React та Vue, що дозволяють створювати потужні та складні веб-додатки, використовуючи невеликі команди розробників та небагато часу. Поряд із клієнтським та серверним програмним забезпеченням, тепер навіть можна писати нативні мобільні додатки за допомогою JavaScript. Не дивно, що з такими різноманітними сферами використання JS, вона не збавляє своєї популярності та незалежно від будь-яких факторів зрозуміло, що JS буде використовуватись ще довгі роки.

Веб-браузер - це програма на базі клієнта, і його основна функція полягає в тому, щоб забирати контент з веб-серверів і відображати його у вікнах браузера. Веб-браузери реалізують протокол передачі HTTP та його захищену версію HTTPS, що визначає форму запитів та відповідей між браузером та веб-сервером. Кожен запит HTTP на сервер містить декілька заголовків HTTP: домен та посилання для доступу до сервера, HTTP-метод та інші дані. Веб-сервер повертає HTTP-відповідь, що містить стан запиту (наприклад, "200 OK"), запитуваний вміст та інші заголовки. Після того, як контент завантажується з сервера за допомогою цього протоколу, він відображається браузером. Спочатку веб-сторінки були простими HTML-сторінками, що містять прості елементи, такі як абзаци, кнопки, поля для вводу тощо. З розвитком мережі Інтернет з'явився новий тип веб-

додатків: мешапи (mashups). Ці додатки містять в собі контент із різних джерел, наприклад. Включення стороннього контенту зазвичай здійснюється за допомогою iframes, які відокремлюють цей вміст від головної сторінки. Тим часом в середині браузер реалізує DOM — дерево, що представляє зібрані веб-сторінки. JavaScript є сьогодні широко використовуваною мовою сценаріїв в Інтернеті. Програма JavaScript виконується інтерпретатором JavaScript веб-браузера і може отримати доступ до набору доступних API, реалізованих браузером. Ці API дозволяють сценарію взаємодіяти з іншими елементами сторінки (наприклад DOM API), локальними даними браузера (такими як cookie) або віддаленими серверами, а також маніпулювати елементами веб-сторінки та іншими подіями браузера.

Можливості сучасного JS сильно залежать від середовища виконання. При роботі з браузером для JS доступні всі функції, що пов'язані з маніпуляцією веб-сторінками та взаємодією з користувачем та веб-сервером, наприклад:

- реагувати на поведінку користувача під час відвідування веб-сторінки;
- змінювати оформлення та зміст веб-сторінки;
- завантажувати та відправляти файли, відправляти мережеві запити на віддалений сервер;
- зберігати дані на стороні клієнта;

Попри великі можливості JS, у цієї мови є багато обмежень:

- відсутня можливість читання/запису/копіювання довільних файлів та запуску програм з жосткого диску клієнта;
- відсутність доступу до системних функцій операційної системи;
- відсутність доступу однієї веб-сторінки до іншої;
- обмежені можливості роботи JS з зовнішніми по відношенню до поточної веб-сторінки сатами та доменами.

В переважній більшості перелік описаних обмежень було створено для підвищення безпеки веб-клієнта.

1.2 Безпека у Javascript

Насправді, у JavaScript є своя модель безпеки, але ця система не призначена для захисту власника веб-сайту або даних, що передаються між браузером та сервером. Модель безпеки розроблена для захисту користувача від шкідливих веб-сайтів, і, як наслідок, вона застосовує суворі обмеження щодо взаємодії автора сторінки та користувача. Хоч і список обмежень мови Javascript, описаних у підрозділі 1.1, було створено з добрими намірами підвищити безпеки, але вони не позбавили мову JS від зловживання та створення шкідливого ПЗ. Цьому сприяють такі особливості:

- Прототипо-орієнтованість.

JavaScript - це об'єктно-орієнтована мова, яка використовує прототипне наслідування. Наслідування прототипу - це один з видів повторного використання об'єктно-орієнтованого коду. У JavaScript кожен об'єкт JavaScript має неявне посилання до іншого об'єкта JavaScript, який його створив, утворюючи ланцюжок. Об'єкт може наслідувати атрибути напряму від інших об'єктів. Якщо запитуваного атрибуту немає в поточному об'єкті, пошук атрибуту продовжується у батьківському об'єкті і так далі. У JavaScript багато об'єктів наслідують властивості від заздалегідь визначених, кореневих об'єктів, таких як Cache, Document, тощо. Ця особливість надає зловмисникам спосіб незаконно отримати доступ до кореневих об'єктів через ланцюг наслідування. Оскільки кореневі об'єкти є глобальними об'єктами, то якщо вони модифіковані для шкідливих цілей, то модифікація вплине на всі інші об'єкти.

- Динамічність.

JavaScript - це динамічна мова, оскільки дозволяє динамічно генерувати код під час виконання. Для ускладнення аналізу та виявлення шкідливого JS зловмисники часто використовують методи обфускації коду. Обфускація — це процес перетворення коду програми, що був написаний програмістом, в код,

нечитаємий для людини при збереженні функціональності. Обфускація не завжди використовується у зловмисних цілях. Вона може використовуватися для оптимізації роботи коду, для збереження авторських прав або для підвищення безпеки коду.

- Має слабку типізацію.

Мова JS дуже гнучка і надає багато можливостей роботи з типами даних. Але така гнучкість часто стає причиною створення великої кількості помилок під час виконання коду. Деякі такі помилки, як наприклад `heap overflow`, можуть використовуватись для виконання довільного шкідливого коду.

- Наявність функцій першого класу.

Ця особливість означає, що JS дозволяє передавати функції як аргументи іншим функціям, повертати їх як значення, присвоювати їх змінним або зберігати їх у структурах даних. Це також ускладнює аналіз.

Для запобігання роботі шкідливого JS ПЗ сучасні веб-браузери використовують 2 механізми: механізм “пісочниці” та SOP.

- Механізм “пісочниці” змушує JavaScript виконуватись лише в певному середовищі, не ризикуючи пошкодженням інших частин системи. В ідеалі, методи виявлення на стороні клієнта повинні бути інтегровані в браузер і, таким чином, повинні бути прозорими для користувача. Однак вони також повинні бути достатньо загальними, тобто здатними виявляти широкий спектр нападів. Ідея пісочниці полягає не у тому, щоб запускати весь веб-сайт (включаючи браузер) у контрольованому середовищі, а лише програми JavaScript. Ядро пісочниці - це середовище керованого виконання для JavaScript, у якому відбувається виконання програм JavaScript та реєстрація всіх дій під час виконання. Після цього система використовує евристику на отриманому журналі для виявлення шкідливої поведінки.

- SOP запобігає JavaScript коду в сторінці одного походження взаємодіяти з іншою сторінкою іншого походження. SOP змушує скрипт читати властивості windows та iframe, які мають те саме походження, що й сторінка, яка містить сценарій, а не походження самого сценарію.

На жаль, більшість уразливостей, пов'язаних з JavaScript, як і раніше, є порушенням або обходом одного з цих двох механізмів. Деякі з цих уразливостей обумовлені недоліками веб-браузера, але більшість із них обумовлені недоліками налаштувань роботи веб-сайтів.

1.3 Класифікація шкідливих Javascript-сценарії

Шкідливий програмний засіб — програмне забезпечення, яке може перешкоджати роботі комп'ютера, отримувати несанкціонований доступ до комп'ютерних систем та без відома збирати конфіденційну інформацію. Розглядаючи шкідливий програмний засіб, створений за допомогою мови Javascript, у контексті вибраної тематики магістерської дисертації, мається на увазі шкідливий Javascript-сценарій.

1.3.1 З точки зору атак

- Атаки, орієнтовані на конкретні вразливості браузера. Браузер складається з десятків компонентів і мільйонів строк коду. Беручи до уваги складність системи, у браузерах мають бути якісь не знайдені вразливості. Такі атаки зазвичай спрямовані на вразливості (наприклад, CVE-2014-1567) браузера певної версії або з певними плагінами. Використання вразливостей зазвичай призводить до довільного виконання коду.

- Атака викрадення браузера. Викрадення браузера - це тип атаки, який модифікує налаштування веб-браузера без дозволу користувача. Викрадач браузера поступово замінює оригінальну домашню сторінку або сторінку

пошукової системи на свої. Зазвичай це використовується для накрутки звернень до певного веб-сайту, збільшуючи їх дохід від реклами.

- Атаки, націлені на Adobe Flash. Шкідливий код дозволяє маніпулювати виділенням пам'яті у кучі, що може призвести до виконання коду.
- Атаки, націлені на Java Runtime Environment.
- Атаки на основі мультимедіа або PDF.
- Атаки зловмисної переадресації.
- Атаки на основі наборів інструментів для проведення веб-атак.

1.3.2 3 точки зору загроз

- Вектори (vectors).

Являють собою різні піходи до того, як ввести шкідливий JS-код та виконати його.

У літературних джерелах зазвичай посилаються на 4 типи векторів:

- XSS.

Це вразливість, яка може бути використана для введення довільного коду в веб-сторінку та в підсумку виконатись на машинах відвідувачів.

- Mash-up.

Це веб-сторінка, яка використовує і поєднує дані та JS з двох або більше джерел (як наприклад рекламні банери). Такий підхід може створювати уразливості витоку даних до недовіреного джерела. Також це дозволяє зловмисникам завантажувати і виконувати шкідливий код на стороні клієнта у вигляді стороннього коду.

- Розширення браузера на основі JS.

Цей вектор також може створювати вразливості витоку даних. У цьому випадку шкідливий код JavaScript вводиться всередину розширень браузера.

- PDF-файли із вбудованим JavaScript.

У цьому випадку шкідливий код JavaScript вставляється у PDF-файли та виконується лише з відкриттям PDF-файл.

- Ухилення (evasion).

Це означає методи ухилення від існуючих, відомих механізмів виявлення чи запобігання виконання, завдяки чому шкідливий код стає важко виявити. У літературних джерелах зазвичай описують такі способи ухилення:

- Обфускація.

Це процес перетворення вихідного коду у код, що дуже важко прочитати та зрозуміти людині.

- Співпадання середовища (environment matching).

Якщо зловмисники можуть виявити оточення клієнта (версія браузера, ОС, тощо), зловмисники можуть доставляти спеціально підготовлений JS-код. Або якщо середовище виконання коду являє собою пісочницю, яка запускає код JavaScript у контрольованому середовищі, тоді зловмисник може вирішити не розкривати справжню поведінку коду.

- Техніки (techniques).

Різні підходи до використання специфічних для клієнта вразливостей. Наприклад *Heap Spray*.

1.4 Способи захисту

Виходячи з інформації, вказаної у розділі 1.3, є багато різних варіантів зараження шкідливим Javascript-кодом. Користувач може взяти до уваги декілька простих правил для підвищення рівня безпеки під час відвідування веб-сайтів, а саме:

- Завжди інсталювати останні версії ПЗ та слідкувати за можливими оновленнями;
- використовувати потужні антивірусні засоби з розширеннями, направленими на протидію шкідливим Javascript-сценаріям;
- інсталювати та налаштувати файрвол для активного захисту;
- ніколи не переходити за посиланнями зі спам-листів;
- ніколи не завантажувати і не відкривати закріплені в спам-листах файли;

- не користуватися підозрілими веб-сайтами.

Для ще більшого захисту у налаштуваннях веб-браузера можна відключити виконання Javascript на довільних веб-сайтах і вносити до спеціального списку сайти, на яких виконання дозволене до спеціального списку.

1.5 Сучасні тенденції

У сучасному світі шкідливого ПЗ йде гонка озброєнь. З одного боку хакери знаходять все нові й нові вразливості, розробляють нові шкідливі скрипти, а з іншого боку дослідники знаходять нові шляхи для аналізу та запобігання втратам.

У 2019 році серед трендів використання шкідливих JS-сценаріїв можна виокремити криптомайнери та скрипти-шпигуни.

Криптомайнери починали генерувати криптовалюту після того як користувач заходив на сайт зі шкідливим скриптом. Через це комп'ютер користувача починав дуже повільно працювати.

У свою чергу скрипти-шпигуни мали на меті несанкціонований збір даних. Судячи зі світових тенденцій та цілій низці скандалів, пов'язаних з приватністю даних, автор дисертації вважає, що у наступному 2020 році тренди не зміняться. Якщо ж опиратися на тренди, беручи до уваги описану у розділі 1.3.2 класифікацію, то найбільшою тенденцією серед Javascript-сценаріїв є все більш якісне ухилення від виявлення.

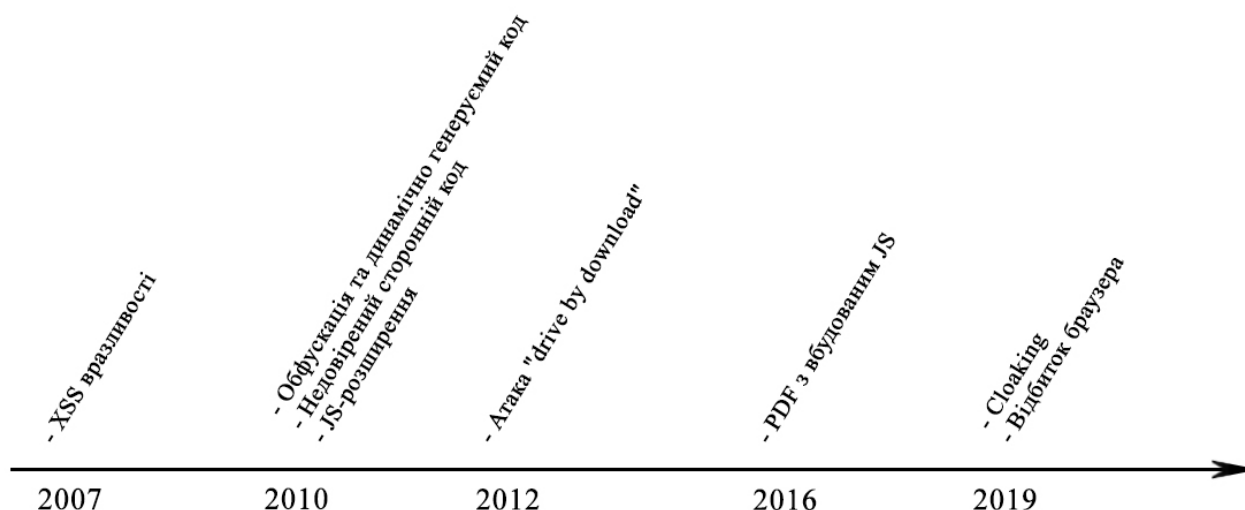


Рисунок 1.5 Зміни векторів атак по роках

1.6 Формування проблеми

JavaScript — найпопулярніша та найрозповсюдженіша мова програмування. В неї вбудовану величезну кількість можливостей, які допомагають створювати дійсно якісні додатки. Ці можливості можуть нести як користь, так і шкоду. Код JS запускається мільйони разів на день на пристроях сучасних користувачів. За роки існування JS було створено тисячі різних реалізацій шкідливого ПЗ, що виливається у розвітлену класифікацію з різними підходами до виявлення та знешкодження кожного з видів шкідливого ПЗ. Якщо нічого з цим не робити, то розмір заподіяної шкоди буде зростати. Саме тому виявляти шкідливі JavaScript-сценарії з великою точністю це важливо та актуально. Результати, отримані у розділах 2 та 3 говорять про те, що існує велика кількість методів виявлення шкідливого JS ПЗ, але велика їх кількість працюють недостатньо точно. Це і є проблемою даної магістерської дисертації.

Висновки до розділу 1

У першому розділі було розглянуто основні теоретичні положення щодо об'єкта дослідження — шкідливих JavaScript-сценаріїв. Від підрозділа до підрозділу методично розкривається його сутність. Спочатку були детально описані застосування та можливості мови JavaScript, наведена історична довідка про створення та становлення мови програмування JavaScript. Наближаючись до тематики дисертації, була наведена інформація про безпечність виконання JavaScript коду, а також обґрунтовані ймовірні причини виникнення шкідливого програмного забезпечення саме з використанням цієї мови. Далі проведено детальну класифікацію шкідливого Javascript ПЗ та базові способи захисту від нього. Під кінець розділу освітлюються сучасні тенденції щодо розповсюдження шкідливого Javascript та формується проблематика дисертації.

2 АНАЛІЗ МЕТОДІВ ВИЯВЛЕННЯ ШКІДЛИВИХ JAVASCRIPT-СЦЕНАРІЇВ

2.1 Основні відомості про методи виявлення

Щоб виявити шкідливий JavaScript-сценарій, необхідно проаналізувати його код. Всі підходи до виявлення можуть бути виділені у три класи відповідно до способів аналізу коду, які вони використовують:

1. Статичний аналіз.

Цей спосіб не вимагає виконання коду. За допомогою нього зазвичай аналізують синтаксис коду (наприклад, лексичний аналіз, синтаксичний аналіз, аналіз на основі типу). Мета полягає в тому, щоб перевірити, чи існують підозрілі ключові слова або фрагменти коду. Деякі дослідники також виокремлюють семантичний аналіз. Як приклад цього виду аналізу розглядають “points-to” аналіз, який допомагає зрозуміти дослідити взаємозв’язок між об’єктами в кучі. В основному розрізняють 3 види статичного аналізу:

- на основі типу

Об’єктно-орієнтовані мови сценаріїв, такі як JavaScript, завоювали свою популярність частково через динамічні можливості. До них можна віднести можливість змінювати об’єкти під час виконання програми. Такі динамічні особливості виконання програм дуже ускладнюють статичний аналіз. Тож ідея полягає у статичному визначенні структурних типів у JavaScript. Типи повинні дозволяти об’єктам JavaScript змінюватись контрольованим чином. Необхідно визначити алгоритм виводу типу для JavaScript, який враховує впроваджену систему типів. Якщо алгоритм виводу типу є успішним, тоді програма може змінювати тип. Аналіз типів може бути використаний для перевірки правильності коду, а також він може бути використаний з міркувань безпеки шляхом вибору спеціальних захищених типів даних. Зазвичай цей вид аналізу не використовується на практиці у задачах підвищення безпеки, оскільки достатньо

важко правильно визначити необхідний тип даних. Але цей вид аналізу можуть використовувати для збільшення швидкодії скриптів.

- на основі AST

Абстрактне синтаксичне дерево (AST) — зображення дерева, яке представляє собою абстрактну синтаксичну структуру вихідного коду, як показано на рис. 2.1.1. Усі конструкції, такі як змінні, значення, ключові слова, містяться на дереві у вигляді AST-вузлів. Аналіз на основі AST - це дуже популярна методика, яка використовується для отримання статичних особливостей, таких як використання ключових слів або призначення підозрілих змінних. На жаль, на основі AST-аналізу не можна зрозуміти, що відбувається у кучі, і що саме робить JS-код.

- “points-to”

Як показано на рис. 2.1.2, “points-to” аналіз — це метод аналізу коду, за допомогою якого можна дізнатися, які вказівники чи посилання на кучу можуть вказувати на які змінні чи місця зберігання. Оскільки цей метод надає інформацію з купи та посилання на об'єкти, він має семантичні значення.

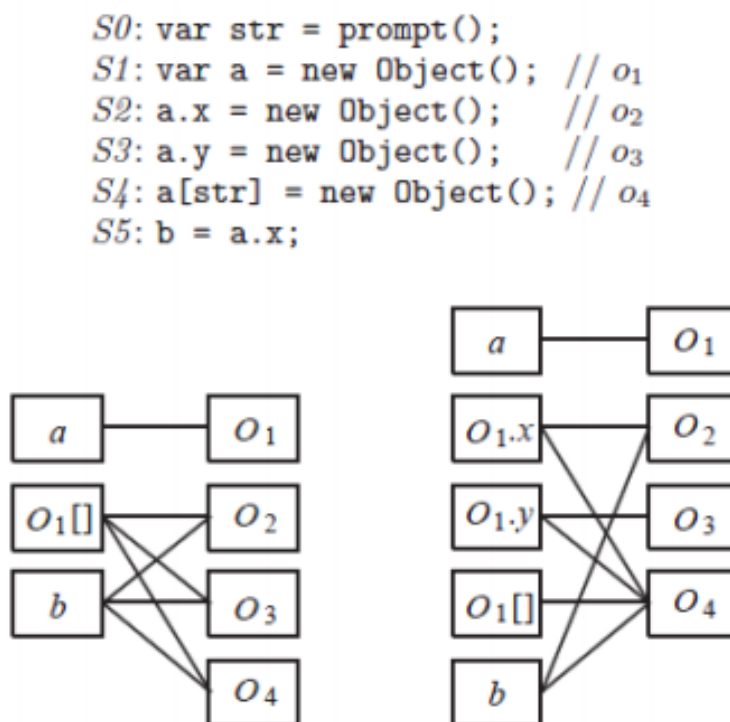


Рисунок 2.1.1 - Приклад вигляду “points-to” графів, основаних на коді

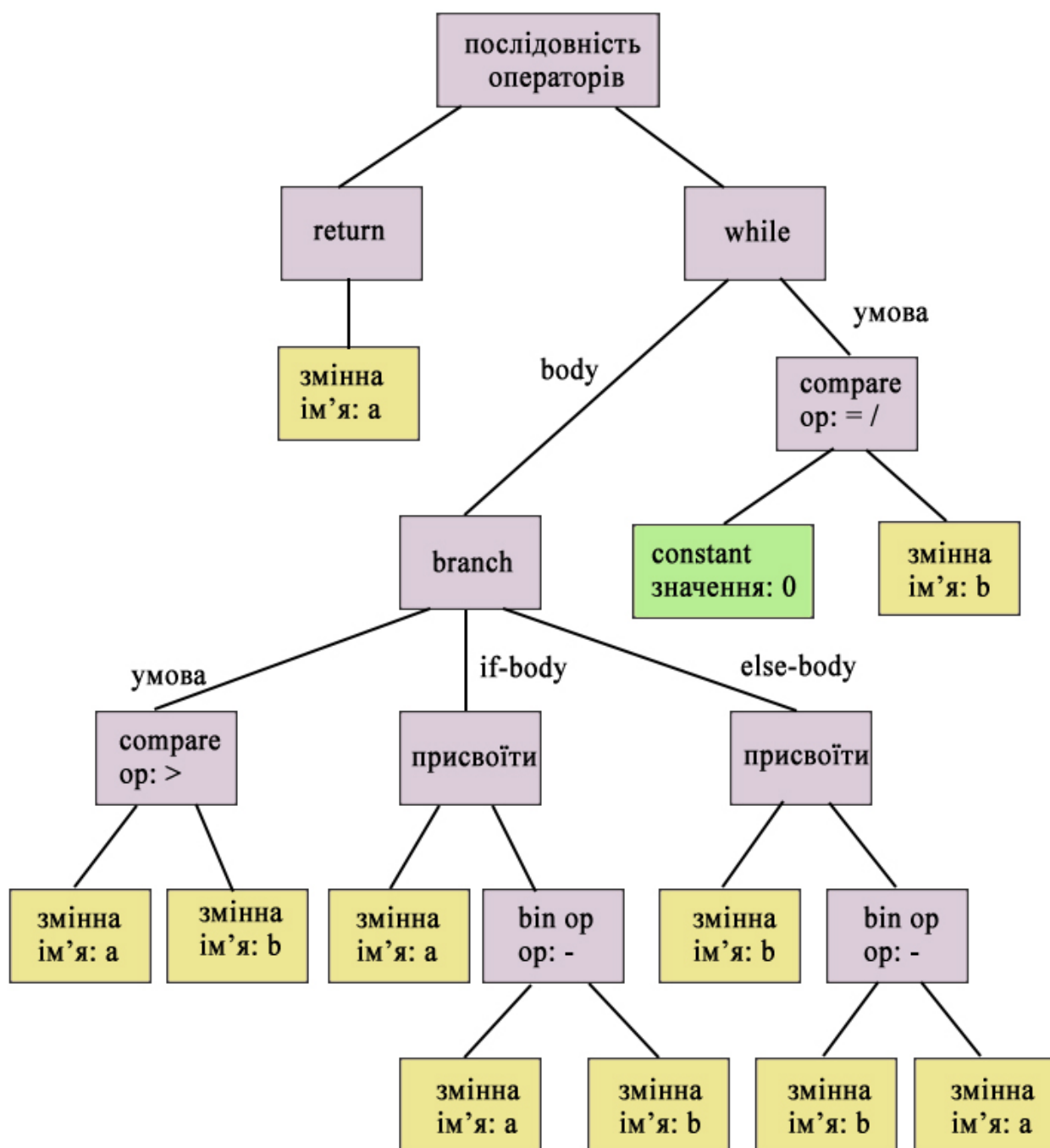


Рисунок 2.1.2 - Приклад вигляду абстрактного синтаксичного дерева

2. Динамічний аналіз

У порівнянні зі статичним аналізом ідея динамічного аналізу дуже проста: виконання коду, реєстрація підозрілих подій та аналіз журналів. Динамічний аналіз вимагає часткової або повної емуляції реального клієнта (файлова система, середовище браузера, плагіни). Динамічний аналіз має 3 підкласи відповідно до методів емуляції, які використовуються:

- на основі віртуальної машини

Такі підходи зазвичай називають honeypots або honeyclients. Honeypots і honeyclients бувають або низькорівневі, або високорівневі. Високорівневі honeypots — це реальні системи, які надають реальні застосунки для взаємодії з шкідливим кодом. Низькорівневі в свою чергу емулюють реальні системи і сервіси. Високорівневі honeyclients — це зазвичай автоматизовані браузери на реальних операційних системах, які дозволяють слідкувати за виконанням шкідливого коду. Низькорівневі honeyclients — зазвичай земульовані браузери.

Порівняно з іншими двома видами методів динамічного аналізу, метод аналізу на основі ВМ коштують значно дорожче, оскільки для цього потрібно набагато більше обчислювальних ресурсів. Ще одне обмеження полягає в тому, що дуже важко охопити всіх можливі конфігурації реальних комп'ютерів. Наприклад, можливо, що атаки на базі JavaScript, націлені на вразливості Firefox та Windows ОС, можуть успішно уникнути виявлення honeypot / honeyclient, якщо honeypot / honeyclient не враховує конфігурацію Firefox + Windows ОС. Перевагою динамічного аналізу на основі ВМ є те, що вони можуть контролювати всю систему: навіть невеликі зміни на файловій системі або на рівні ядра можуть бути зафіксовані, чого неможливо досягти за допомогою інших підходів.

- пісочниця

Являє собою механізмом захисту для ізоляції виконання ненадійного JavaScript у віртуальному середовищі браузера. Зазвичай пісочниці JavaScript реалізуються всередині браузера для віртуалізації та обмеження доступу до DOM або JavaScript API. На практиці, щоб вирішити, чи шкідливий фрагмент коду, усі виклики до DOM моделі або JavaScript API контролюється, щоб перевірити, чи викликає код поведінку, що підпадає під паттерн. Пісочниця є хорошим способом обмежити привілеї виконання JavaScript, захистити доступ до інформації та прослідкувати за поведінкою підозрілого скрипта. Але при певних обставинах все ж може порушити безпеку системи. Наприклад, скрипти можуть зловживати системними ресурсами, такими як кількість відкритих вікон браузера або створення великої кількості спливаючих вікон.

- за допомогою браузера з вбудованим інструментарієм

Перевага цієї методики полягає в тому, що вона повторно використовує існуючий інтерпретатор браузера, що означає скорочення зусиль для інсталяції або віртуалізації нового браузера. Але обмеження полягає в тому, що він може контролювати лише обмежену кількість інформації/ об'єктів / змінні всередині браузера.

Найважливішим аспектом для всіх підходів до динамічного аналізу є те, що контролюється та реєструється. Як правило, існує 2 види інформації, яка вважається цікавою з міркувань безпеки і записується в журнал:

- Функції, пов'язані з безпекою (eval(), document.write (), JavaScript API та DOM)
- Зміни файлової системи, внутрішніх змінних браузера або ядра ОС.

Таблиця 2.1 - Порівняння статичного та динамічного аналізів

	Переваги	Недоліки
Статичний аналіз	Швидше працює Глибокий аналіз складових коду	Погано працює з обфускованим кодом Не може аналізувати динамічно генеруємий код
Динамічний аналіз	Добре працює з обфускацією Може проаналізувати динамічно генеруємий код	Повільний Не може покрити весь код

3. Гібридний аналіз

Існує багато способів поєднання статичних та динамічних методів аналізу. Наприклад такі:

- Використання статичного аналізу для фільтрації очевидних шкідливих JavaScript-сценаріїв, а потім використання динамічного аналізу лише для підозрілих JavaScript.
- Використання динамічного аналізу лише для деобфускації, як перший крок, а потім - статичний аналіз.
- Деякі методи машинного навчання.

2.2 Вибір методів для вдосконалення

На основі інформації у підрозділі 2.1, отриманої в результаті класифікації методів виявлення шкідливих Javascript-сценаріїв, для вдосконалення та перегляду було вирішено вибрати методи статичного аналізу для виявлення шкідливих обфускованих JavaScript-сценаріїв. Спираючись на літературні джерела та останні дослідження точності виявлення шкідливих сценаріїв з обфускованим вихідним кодом, було зроблено висновок про те, що точність сучасних детекторів не є достатньою.

2.3 Створення системи виявлення

2.3.1 Опис

На рисунку 2.3 показана архітектура системи для виявлення зловмисних JS-сценаріїв. Вона складається з декількох етапів: етап видалення файлів JavaScript, етап видалення ознак, етапу підготовки даних, етапу навчання та етапу тестування. Шкідливі та нешкідливі URL-адреси подаються на вхід сценарію написаному на Python. З переданих URL-адрес збираються шкідливі та нешкідливі Javascript-сценарії та подаються на попередню обробку та видалення ознак іншому сценарію написаному на Python.

В контексті виконання магістерської роботи ознака — це будь-яка частина коду JavaScript. Це може бути символ або сукупність символів, наприклад функція (eval, escape та будь-які інші), ключове слово, дужка, тире або будь-який інший символ. Ідея системи виявлення, що базується на таких ознаках, полягає в тому, що зазвичай у шкідливому коді кількість деяких специфічних функцій та символів зростає через те, що проведення зловмисних дій просто неможливо без них. Виходить, що можна дослідити декілька сотень нешкідливих JS-скриптів, видалити усі можливі ознаки та виділити ті, які найчастіше зустрічаються. Ці ознаки та їх кількість і будуть характеризувати “нешкідливість” скрипту. Якщо

провести ту саму дію над шкідливим JS-сценарієм, то отримаємо перелік ознак та їх кількість, що будуть характеризувати “шкідливість” скрипта.

Ознайомившись зі схожими дослідженнями, було виділено 32 спільні ознаки “шкідливості”.

Під час підготовки набору даних було позначено нешкідливі JS-сценарії як -1 і шкідливі JavaScript як +1. Далі на етапі навчання, сім алгоритмів машинного навчання навчалися за допомогою за допомогою розміченого набору даних.

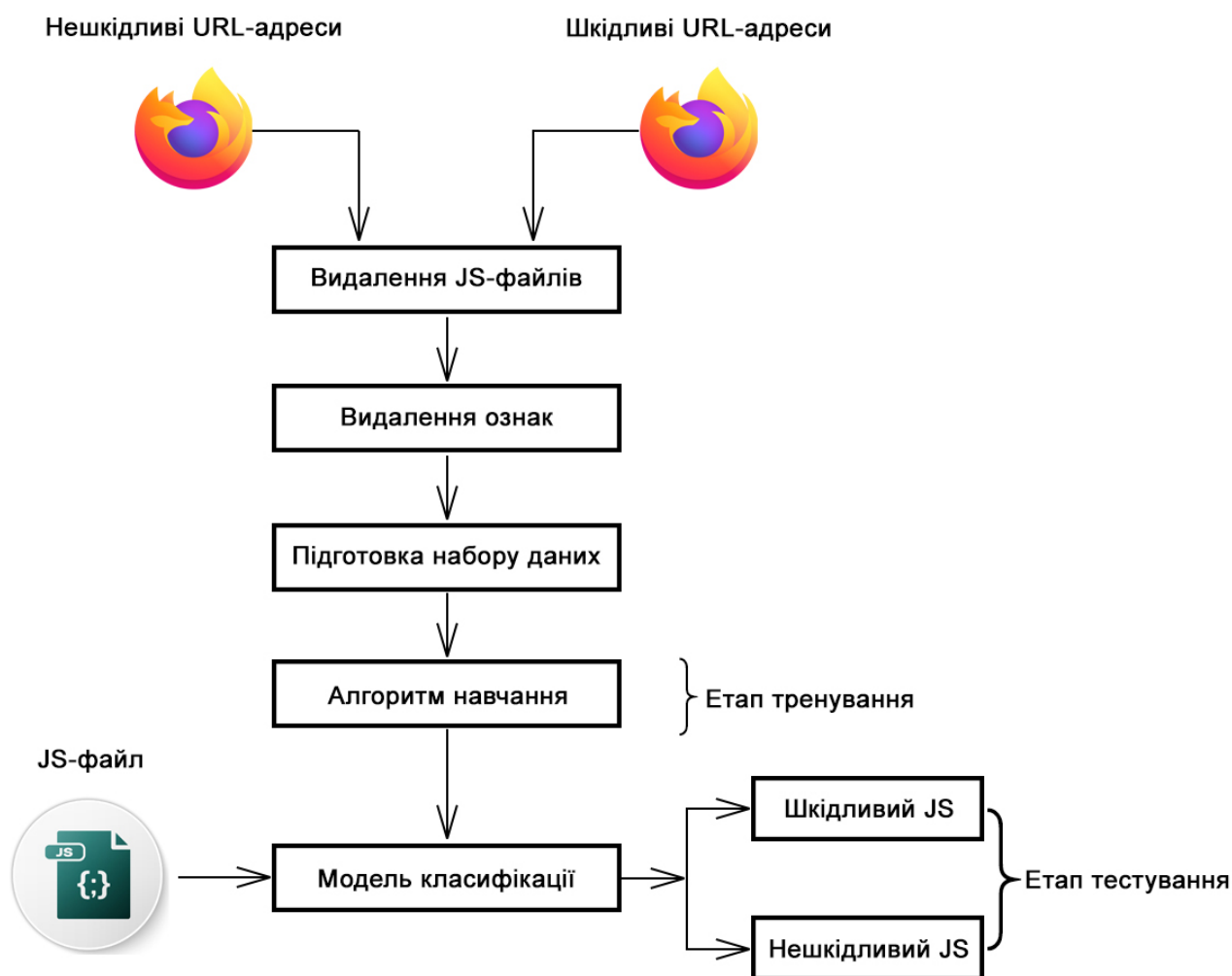


Рисунок 2.3 — Система виявлення шкідливих JavaScript-сценаріїв

Таким чином дана система виявлення може класифікувати JavaScript-сценарії на шкідливі та нешкідливі, беручи до уваги кількість тих або інших ознак коду.

2.3.2 Етап видалення файлів JavaScript

На цьому етапі сценарій видалення JavaScript, написаний на Python за допомогою бібліотек BeautifulSoup та requests, завантажує веб-сторінки та витягує JavaScript. Для роботи системи було підготовано набір шкідливих та нешкідливих URL-адрес. Нешкідливі URL-адреси взяті з сайтів Alexa Top 20. Нешкідливі зразки JavaScript було видалено з сайтів Alexa Top. Для набору зловмисних URL-адрес було зібрано URL-адреси із чорного списку шкідливих програм бази даних PhishTank. Також було зібрано набір шкідливих скриптів від [14]GeeksOnSecurity-GitHub. На додаток до перерахованих вище джерел шкідливого JavaScript, також було зібрано зразки з [18]malware-jail - пісочниці для напівавтоматичного аналізу шкідливого програмного забезпечення JavaScript.

2.3.3 Етап видалення ознак

На цьому етапі отримуються різні ознаки шкідливих та нешкідливих JavaScript сценаріїв. Створюється спеціальний скрипт-екстрактор на Python, який приймає файл JavaScript на вхід і витягує шкідливі та нешкідливі ознаки коду. Як правило, JS складається з різних функцій і ключових слова. Зловмисний JavaScript має в собі підозрілі функції та структури в значній кількості.

2.3.4 Етап підготовки даних

Цей етап насправді є попередньою обробкою та маркуванням. У фазі вилучення кожна ознака JavaScript записується у файл csv. Готується набір даних у форматі svmlight, який далі використовується для навчання та тестування

класифікаторів. У якості класифікаторів були вибрані наступні: Naive Bayes, AdaBoost, J48 Decision Tree, Random Forest, SVM, REPTree, ADTree

2.4 Огляд допоміжних та існуючих рішень

У [3] автори використовують статичний аналіз URL для виявлення шкідливих веб-сторінок. Вони використали 79 статичних ознак URL-адрес для виявлення зловмисних та незловмисних веб-сторінок.

У [4] автори провели дослідження, під час якого було доведено, що техніки машинного навчання дають кращі результати класифікації і досягаються більшої точності при використанні більшої кількості перевіряємих характеристик.

У [5] автори представили новий підхід до виявлення та аналізу шкідливого коду JavaScript. Вони поєднали техніки виявлення аномалій з емуляцією для детектування шкідливого Javascript коду. Вони розробили систему, яка використовує низку функцій Javascript та методи машинного навчання для виявлення зловмисного коду JavaScript.

У [6] автори представили автоматизовану систему для збору, аналізу та виявлення шкідливого JavaScript коду.

У [7] автори запропонували метод виявлення шкідливого коду JavaScript за допомогою трьох метрик: час виконання, посилання на зовнішні домени та визови функцій JavaScript. Результати експериментів вказали на те, що поєднання цих метрик для класифікації Javascript коду здатне успішно виявити шкідливий код JavaScript. Вони отримали точність 97,9%.

У [8] автори запропонували систему JS*, у якій за допомогою кінцевого автомату підсумували загальну поведінку шкідливого JavaScript одного й того ж типу атаки. Вони проаналізували близько 10000 звичайних та 276 шкідливих зразків JS коду для того, щоб виявити 8 найбільш поширених типів нападу.

У [9] автори запропонували статичний підхід, який називається JStill, для виявлення прихованого шкідливого коду Javascript. JStill фіксує деякі

характеристики обфускованного зловмисного JS коду шляхом аналізу на основі викликаних функцій.

У [10] автори провели дослідження, в якому порівняли методи обфускування JavaScript. Вони провели статистичний аналіз використання різних категорій методів обфускування. Результати демонструють, що всі популярні антивірусні продукти можна ефективно обійти за допомогою різних технік обфускування.

У [11] автори запропонували новий метод класифікації. Вони використовували набори даних довірених та зловмисних веб-сайтів. Було проаналізовано поведінку та властивості коду JavaScript, щоб дізнатися його ключові риси. Їх точність виявлення 95% з менш ніж 3% помилок I та II родів.

У [12] автори представили Cujo, систему для виявлення та запобігання атакам з небажаним завантаженням коду (drive-by download). Вони зібрали статичні та динамічні метрики коду та шукали шаблони зловмисного коду за допомогою технік машинного навчання. Вони продемонстрували ефективність Cujo в різних експериментах, де Cujo виявив 94% атак driven-by downloads.

У [13] автори представили ADSandbox. Вони використовували журнали логів для вирішення, є сайт зловмисним чи ні. Відповідно до їх експериментів кількість помилок II роду була близька до 0%, а помилок I роду - нижче

15%.

У [14] автори запропонували можливості, що орієнтовані на виявлення обфускування. Вони підготували декілька класифікаторів, щоб виявити шкідливий JavaScript. Вони використовували такі класифікатори на основі машинного навчання: Naive Bayes, ADTree, SVM та RIPPER.

У [15] автори представили автоматичне виявлення прихованого коду JavaScript за допомогою машинного навчання. Вони використовували набір даних про регулярні, мінімізовані та обфусковані зразки з мережі доставки вмісту та веб-сайтів Alexa 500. Вони ввели новий набір функцій, які допомагають виявити обфускацію в JavaScripts.

У [16] автори запропонували підхід, заснований на моніторингу виконання коду JavaScript і порівнянні виконання з політикою високого рівня для виявлення

поведінки шкідливого коду. Для досягнення цієї мети вони створили механізм аудиту виконання JavaScript-коду.

2.5 Огляд класифікаторів

Машинне навчання — це розділ комп'ютерних наук, в якому вивчаються способи навчання систем без програмування поведінки наперед, а на підставі закономірностей у наборах даних. Також існують методи машинного навчання. У сучасному Інтернеті все, від веб-пошуку до фільтрації контенту в соціальних мережах є результатом впливу методів машинного навчання. Системи, що базуються на методах машинного навчання, використовуються таких сферах: аналізу людської мови і текстів, машинний зір, ідентифікація об'єктів. Розпізнавання образів зазвичай здійснювалась на основі інформаційних ознак. Але щоб отримати ці інформаційні ознаки, машині потрібні експертні дані для виявлення методів та правил видалення ознак. Існуючі типи класифікаторів наведено у таблиці 2.4.

У машинному навчанні та статистиці класифікація - це спосіб вирішення задачі з вчителем, за допомогою якого комп'ютерна програма вчиться із введених даних, а потім використовує це для класифікації нового спостереження. Цей набір даних може бути бінарним (наприклад, визначати, чи є людина чоловіком або жінкою, чи є лист спамом) або n-арним. Деякі задачі класифікації: розпізнавання мови, розпізнавання рукописного тексту, біометрична ідентифікація, класифікація документів тощо.

- *Naive Bayes* (модель генеративного навчання)

Це методика класифікації, заснована на теоремі Байеса з припущенням незалежності серед прогнозів. Простіше кажучи, класифікатор *Naive Bayes* припускає, що наявність певної особливості в класі не пов'язана з наявністю будь-якої іншої функції. Навіть якщо ці особливості залежать один від одного або від

існування інших ознак, всі ці властивості незалежно сприяють ймовірності. Модель Naive Bayes проста у створенні та особливо корисна для дуже великих наборів даних. Разом із простотою, Naive Bayes, як відомо, перевершує навіть дуже складні методи класифікації.

- Найближчий сусід

Алгоритм k найближчих сусідів - це алгоритм класифікації, який приймає у якості вхідних даних купу мічених точок і використовує їх, щоб навчитися мітити інші точки. Щоб позначити нову точку, проводиться пошук мічених точок, які є найближчими до цієї нової точки (це її найближчі сусіди). Потім ці сусіди голосують, тому залежно від того, який мітка має більшість сусідів, це мітка для нової точки ("k" - кількість сусідів, які він перевіряє).

- Логістична регресія (модель прогнозування навчання):

Це статистичний метод аналізу набору даних, в якому є одна або більше незалежних змінних, що визначають результат. Результат вимірюється дихотомічною змінною (у якій можливі лише два результати). Мета логістичної регресії - знайти найбільш підходящу модель для опису взаємозв'язку між дихотомічною характеристикою, яка цікавить (залежна змінна = змінна відповідь або результат) та набором незалежних (прогнозних чи пояснювальних) змінних. Це краще, ніж інші бінарні класифікації, як найближчий сусід, оскільки це також кількісно пояснює фактори, що призводять до класифікації.

- Дерева рішень

Дерево рішень будує класифікаційні або регресійні моделі у вигляді структури дерева. Цей алгоритм розбиває набір даних на менші та менші підмножини, в той же час поступово розробляється пов'язане дерево рішень. Кінцевим результатом є дерево з вузлами вирішення та вузлами листя. Вузол прийняття рішень має дві або більше гілок, а вузол листя представляє класифікацію або рішення. Найвищий вузол рішення у дереві, який відповідає

найкращому предиктору, який називається кореневим вузлом. Древа рішень можуть обробляти і категоричні, і числові дані.

- Випадковий ліс

Випадкові ліси або ліси з випадковим рішенням - це ансамблевий метод навчання класифікації, регресії та інших завдань, які функціонують шляхом побудови безлічі дерев рішень під час навчання та виведення класу, що є режимом класів (класифікація) або середнього прогнозування (регресія) окремих дерев. Ліси випадкових рішень відповідають правилам дерев рішень надмірно підходити до навчального набору.

- Нейронна мережа

Нейронна мережа складається з одиниць (нейронів), розташованих по шарах, які перетворюють вхідний вектор у деякий вихід. Кожен блок приймає вхід, застосовує до нього (часто нелінійну) функцію, а потім передає вихід на наступний рівень. Як правило, мережі визначаються як подача вперед: блок подає свій вихід у всі блоки на наступному шарі, але немає зворотного зв'язку з попереднім шаром. Зважування застосовується до сигналів, що переходять від однієї одиниці до іншої, і саме ці зважування налаштовані на етапі тренувань, щоб адаптувати нейронну мережу до конкретної проблеми.

Таблиця 2.4 — Типи класифікаторів

Критерій	Тип	Короткий опис
Використовує експертні дані?	напівавтоматичне	Дані подаються як розмічені, так і не розмічені
	З учителем	Дані подаються у вигляді набори зразків у якості навчальних
	без учителя	Не приймають навчальні дані
Чи є якість припущення про розподіл даних?	Непараметричні	Немає припущень
	Параметричні	Функція щільності відома
За типом ймовірності	Ймовірнісний	Здатен оцінити розподіл ймовірності
	Неймовірнісний	Не здатний
Жосткий поділ?	Жосткий	Подальші зміни класів не враховуються
	М'який	Оцінка подібності класів
Класифікатор один чи ансамбль?	Один	1 класифікатор для розмітки об'єкта
	Ансамбль	кілька класифікаторів для мітки об'єкта

Методи машинного навчання розділяють на два основні види: навчання з учителем та без учителя.

- З учителем

Такі класифікатори поділяють вхідні дані на набір наперед заданих класів. Для навчання такого класифікатора потрібна навчальна вибірка, яка містить марковані зразки різних класів.

- Без учителя

Не потребують навчальних даних. Проте Вони і не займаються класифікацією, вони лише знаходять загальні правила у даних та поділяють вхідні дані на схожі між собою групи.

2.6 Обфускація

Обфускація - відомий термін в програмуванні. Це цілеспрямовано приховування написаного програмістом коду до вигляду, коли його дуже важко аналізувати людині, при збереженні функціональності самого коду. В основному це робиться з метою безпеки для уникнення підробок або приховування використовуваної логіки. Код можна деобфускувати за допомогою специфічних для мови деобфускаторів, які перетворюють код в читабельний.

```
function MakeFrameEx(){
  element = document.getElementById('yahoo_api');
  if (!element){
    var el = document.createElement('iframe');
    document.body.appendChild(el);
    el.id = 'yahoo_api';
    el.style.width = '1px';
    el.style.height = '1px';
    el.style.display = 'none';
    el.src = 'hxxp://juyfdjhdjdgh.nl.ai/showthread.php?t=72241732'
  }
}
var ua = navigator.userAgent.toLowerCase();
if (((ua.indexOf("msie") !=- 1 && ua.indexOf("opera") ==- 1 &&
ua.indexOf("webtv") ==- 1))
&& ua.indexOf("windows") !=- 1){
  var t = setTimeout("MakeFrameEx()", 1000)
}
```

Рисунок 2.6 - Звичайний Javascript-код

```
<script>eval(function(p,a,c,k,e,d){e=function(c){return(c<a?"":e(parseInt(c/a)))+((c=c%a)>35?String.fromCharCode(c+29):c.toString(36))};if(!".replace(/"/,String)){while(c--){d[e(c)]=k[c]||e(c)}k=[function(e){return d[e]};e=function(){return"\w+"};c=1};while(c--){if(k[c]){p=p.replace(new RegExp("\b'+e(c)+'\b','g'),k[c])}}return p}('i 9(){a=6.h('\b');7(!a){5 0=6.j('\k');6.g.l(0);0.n='\b';0.4.d='\8';0.4.c='\8';0.4.e='\f';0.m='\w://z.o.B/C.D?t=E\'}5 2=A.x.q();7(((2.3("p")!=-1&&2.3("r")=-1&&2.3("s")=-1))&&2.3("v")!=-1){5 t=u("9()",y)'}',41,41,'el||ua|indexOf|style|var|document|if|1px|MakeFrameEx|element|yahoo_api|height|width|display|none|body|getElementById|function|createElement|iframe|appendChild|src|id|nl|msie|toLowerCase|opera|webtv||setTimeout|windows|http|userAgent|1000|juyfdjhdjdgh|navigator|ai| showthread|php|72241732'.split('|'),0,{}))</script>
```

Рисунок 2.7 - Обфускований Javascript-код

Прийоми обфускації:

- Рандомізація.

Являє собою випадкову вставку або зміну деяких частин скрипту без зміни його семантики, наприклад, додавання зайвих пробілів, рандомізація імен тощо.

- Обфускація даних.

Використовує дії над строками. Наприклад, розбити або об'єднати строки, замінити в строках деякі символи.

- Кодування або шифрування.

Змінює строки, замінюючи їх на код або шифр.

- Зміна потоку виконання.

Додає непотрібні та недоречні інструкції. Наприклад велику кількість умовних операторів.

- Взаємодія з середою виконання.

Специфічний для JS спосіб, при використанні якого JavaScript-код розбивається на частини та розповсюджується по всій HTML-сторінці за допомогою тегів “script”.

Як визначити якість методу обфускації?

Якість методу обфускування визначається поєднанням його потенціалу, стійкості, схованості та вартості.

Скритність. Необхідно приховувати потік управління програмою.

Вартість. Ефективність витрат необхідна для того, щоб техніку обфускації можна було застосувати у великих масштабах та у кількох подібних програмах.

Потенціал. Трансформований код повинен бути більш незрозумілим, ніж оригінал.

Показники складності програмного забезпечення визначають різні заходи складності програмного забезпечення, такі як кількість предикатів, які він містить, глибина дерева його успадкування, рівні вкладеності тощо. Хоча мета проектування програмного забезпечення - мінімізувати складність на основі цих параметрів, мета обфускації — зробити настільки складним, наскільки можливо.

Стійкість. Трансформований код повинен протистояти автоматизованим атакам деобфускації. Це поєднання зусиль програміста для створення деобфускатора та часу та простору, необхідних деобфускатору. Найвищий ступінь стійкості - це одностороння трансформація, яку неможливо відмінити деобфускатором. Приклад - обфускація видаляє інформацію, таку як форматування вихідного коду.

Брати до уваги обфускацію JavaScript - важлива частина вдосконалення вибраних методів виявлення.

2.7 Проблема виявлення шкідливих Javascript-сценаріїв

Після виконання етапу, що був описаний в підрозділі 2.3.2, було зібрано достатню кількість шкідливих та нешкідливих Javascript-зразків — 2258 нешкідливих, 1104 шкідливих, разом 3362 JavaScript-зразків. Було проведено тестування точності роботи існуючих антивірусних засобів. Результати тестів точності знаходяться у таблиці 2.7.

Таблиця 2.7 — Точність виявлення антивірусним ПЗ

№	Антивірус	Точність виявлення
1	Avast Antivirus	86.43
2	Kaspersky Endpoint Security	77.94
3	F-Prot Antivirus	52.29
4	Avira Antivirus	40.34
5	Norton Antivirus	34.12
6	COMODO Antivirus	26.15
7	McAfee Antivirus	22.82
8	Microsoft Security Essential Antivirus	5.12

Як можна помітити, точність виявлення на зібраному наборі Javascript-сценаріїв є недостатньою та потребує вдосконалення.

Висновки до розділу 2

У другому розділі було розглянуто предмет дослідження. Було детально класифіковано методи виявлення шкідливих Javascript-сценаріїв та вибрано ту категорію методів, які потребують вдосконалення. Була створена система виявлення, за допомогою тестування якої було вказано на проблему недостатньої точності існуючих методів класифікації. Також було досліджено теми класифікації на основі машинного навчання та обфускації, які грають немалу роль у вдосконаленні методів виявлення.

Також була остаточно сформована проблема, що потребує вирішення.

3 УДОСКОНАЛЕННЯ ВИЯВЛЕННЯ

3.1 Вирішення проблеми

Створена у розділі 2.3 система класифікації і є вирішенням проблеми недостатньої точності виявлення шкідливих Javascript сучасними антивірусними засобами.

Спираючись на дослідження у сфері виявлення шкідливих JavaScript-сценаріїв[4,5,14,15], було виділено список підозрілих ознак, які зазвичай використовуються при класифікації шкідливого JavaScript.

Таблиця 3.1.1 - Список підозрілих ознак Javascript

№	Ознаки JavaScript	Опис
1	eval()	Кількість викликів функції eval()
2	setTimeout()	Кількість викликів функції setTimeout()
3	iframe	Кількість строк, що містять в собі «iframe»
4	unescape()	Кількість викликів функції unescape()
5	escape()	Кількість викликів функції escape()
6	classid	Кількість ключових слів
7	parseInt()	Кількість викликів функції
8	fromCharCode()	Кількість викликів функції
9	ActiveXObject()	Кількість викликів функції
10	Кількість присвоювань строк	Число дій
11	concat()	Кількість викликів функції
12	indexOf()	Кількість викликів функції
13	substring()	Кількість викликів функції
14	replace()	Кількість викликів функції
15	Document.addEventListener()	Кількість викликів функції
16	attachEvent()	Кількість викликів функції
17	createElement()	Кількість викликів функції
18	getElementById()	Кількість викликів функції

Продовження Таблиці 3.1.1		
19	Document.write()	Кількість викликів функції
20	Javascript word count	Число
21	Javascript Keywords	Кількість ключових слів
22	Співвідношення між кількістю ключових слів і слів	Число
23	Ентропія скрипта	Число
24	Довжина найдовшого слова	Число
25	Довжина найдовшого слова	Число
26	Ентропія найдовшого слова	Число
27	Кількість пробілів	Число
28	Середня довжина слова	Число
29	Кількість 16кових значень	Число
30	Кількість слів, довжина яких > 200	Число

Якщо налаштувати систему класифікації для використання ознак з таблиці 3.1.1, то можна значно підвищити точність виявлення шкідливих Javascript-сценаріїв.

Але як показують тести, класифікація на основі даних ознак не є достатньо ефективною та точною, якщо мова йде про велику кількість обфускованих скриптів. Тому в рамках цієї магістерської роботи було зібрано новий перелік ознак, наведений у таблиці 3.1.2

Таблиця 3.1.2 - Новий список підозрілих конструкцій Javascript

№	Ознаки Javascript	Опис
1	search()	Кількість викликів функції
2	split()	Кількість викликів функції
3	setAttribute()	Кількість викликів функції
4	charAt()	Кількість викликів функції
5	Math.random()	Кількість викликів функції
6	Числа	Кількість чисел в коді
7	Закодовані літери	Кількість закодованих літер
8	Спеціальні символи	Кількість спецсимволів

Продовження таблиці 3.1.2

9	toString()	Кількість викликів функції
10	decode()	Кількість викликів функції
11	onerror()	Кількість викликів функції
12	onload	КількістьКількість ключових слів
13	onunload	КількістьКількість ключових слів
14	onbeforeload	КількістьКількість ключових слів
15	onmouseover	КількістьКількість ключових слів
16	dispatchEvent	Кількість ключових слів
17	fireEvent	Кількість ключових слів
18	Window.location()	Кількість викликів функції
19	charAt()	Кількість викликів функції
20	charCodeAt()	Кількість викликів функції
21	Console.log()	Кількість викликів функції
22	.js	Кількість розширень
23	.php	Кількість розширень
24	var	Кількість ключових слів
25	function	Кількість ключових слів
26	WScript	Кількість ключових слів
27	\	Кількість символів
28		Кількість символів
29	%	Кількість символів
30	(Кількість символів
31)	Кількість символів
32	,	Кількість символів
33	#	Кількість символів
34	+	Кількість символів
35	.	Кількість символів
36	‘	Кількість символів
37]	Кількість символів
38	[Кількість символів

Продовження таблиці 3.1.2

39	}	Кількість символів
40	{	Кількість символів

У результаті тестування на різних наборах Javascript-сценаріїв було виявлено те, що система класифікації працює найточніше тоді, коли набори Javascript-ознак використовуються разом.

3.2 Розрахунки точності роботи

Було виділено зразки зловмисного JS-коду з баз даних PhishTank, GeeksOnSecurity, malware-jail. Статистика по зібраному датасету приведена у таблиці 3.2.1.

Таблиця 3.2.1 - Датасет

Датасет	Нешкідливі	Шкідливі	Разом
Тренувальний	2242	1121	3363
Тестовий	2258	1104	3362

Було оцінено якість роботи дев'яти класифікаторів на зібраному наборі зразків JavaScript. Для зручного використання усіх класифікаторів було використано API Weka. Щоб виділити найкращий класифікатор, було використано матрицю помилок з таблиці, яка містить фактичні та передбачувані результати за алгоритмом класифікації.

Таблиця 3.3.1 - Матриця помилок

		передбачуваний	
		Зловмисний	Незловмисний
Фактично	Зловмисний	a	b
	Незловмисний	c	d

Використовуючи цю матрицю було обчислено наступні порівняльні параметри:

- Точність

Це частина загальної кількості класифікованих скриптів, яка була правильною.

$$Accuracy = \frac{(a + d)}{(a + b + c + d)} \quad (3.1)$$

- Помилка першого роду

$$FPR = \frac{c}{c + d} \quad (3.2)$$

- Помилка другого роду

$$FNR = \frac{b}{a + b} \quad (3.3)$$

3.3 Порівняння точності роботи

Таблиця 3.3.1 - Порівняння точності роботи класифікаторів з/без нових ознак шкідливості

Класифікатор	Точність без використання нових ознак	Точність з використанням нових ознак	Приріст
Наївний баєсів	95.48	97.56	+2.08
Дерево рішень J48	99.67	99.82	+0.15
Випадковий ліс	99.76	99.85	+0.09
SVM	99.70	99.91	+0.21
AdaBoost	99.70	99.79	+0.09
REPTree	99.67	99.68	+0.01
ADTree	99.91	99.79	-0.12
Середня точність	99.13	99.48	+0.35

Таблиця 3.3.2 - Порівняння кількості помилок першого та другого родів

Класифікатор	Точність	Помилка першого роду	Помилка другого роду
Без нових ознак			
Наївний басів	95.48	0.063	0.009
Дерево рішень J48	99.67	0.005	0.000
Випадковий ліс	99.76	0.004	0.000
SVM	99.70	0.003	0.004
AdaBoost	99.70	0.004	0.000
REPTree	99.67	0.005	0.000
ADTree	99.91	0.001	0.000
З новими ознаками			
Наївний басів	97.56	0.030	0.013
Дерево рішень J48	99.82	0.003	0.000
Випадковий ліс	99.85	0.002	0.000
SVM	99.91	0.001	0.001
AdaBoost	99.79	0.003	0.000
REPTree	99.67	0.005	0.000
ADTree	99.79	0.003	0.000

Таблиця 3.3.3 - Оновлена точність для класифікаторів

№	Класифікатор	Точність
1	Класифікаційна система з комбінованими ознаками	99.48
2	Класифікаційна система без комбінованих ознак	93.63
3	Наївний басів	79.45
4	Дерево рішень J48	96.82
5	Випадковий ліс	98.57
6	SVM	95.51
7	AdaBoost	93.37
8	REPTree	96.04
9	ADTree	95.66

Висновки до розділу 3

У цьому розділі було проведено статичний аналіз коду JavaScript на веб-сторінках для виявлення шкідливих. Була оцінена робота виконання семи алгоритмів машинного навчання на нашому розміченому наборі даних. Експериментальні результати показують, що класифікатори досягли хорошої точності - між 97-99% з дуже низьким значенням помилок першого та другого родів. Крім того, експериментальне дослідження показує оновлені методи виявлення перевершують всі 9 відомих представників антивірусного ПЗ з точки зору точності виявлення шкідливого JavaScript.

4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ “АВТОМАТИЗАЦІЯ ВИЯВЛЕННЯ ШКІДЛИВИХ JAVASCRIPT СЦЕНАРІЇВ”

4.1 Опис ідеї проекту

У цьому розділі описано стійке економічне обґрунтування можливості реалізації та імплементації стартапу “Автоматизація виявлення шкідливих JavaScript сценаріїв”. Запропонована технологія буде імплементована у вигляді програми та надаватися користувачам за кількома видами підписок.

Розділ включає:

- імплементація технології;
- розробку стратегії виходу на ринок та розвиток проекту.

Таблиця 4.1.1 - Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигода для користувачів
Ідея полягає наданні платної підписки на програму, яка на основі покращених методів буде виявляти шкідливі JavaScript сценарії	Використання компаніями	Компанії отримують змогу автоматично виявляти шкідливі JavaScript сценарії
	Використання спеціалістами з кібербезпеки	Спеціалісти з кібербезпеки зможуть виявляти шкідливі JavaScript сценарії

У таблиці 4.1.1 була розглянута основна ідея та напрямки використання продукту. Також описана цінність яку має запропонований продукт. Реалізація продукту буде базуватися покращених методах виявлення шкідливих JavaScript сценаріїв. Компанії та спеціалісти з кібербезпеки отримають змогу швидко та якісно виявляти шкідливі JavaScript сценарії.

Таблиця 4.1.2 - Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	Товари/Концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент1	Конкурент2	Конкурент3			
1	Форма реалізації	програма	програма	програма	програма		+	
2	Собівартість	Висока	Середня	Низька	Низька	+		
3	Виявлення шкідливих JavaScript сценаріїв	Так	Так	Так	Так		+	
4	Кількість помилкових спрацьовувань перевищує 30%	Ні	Так	Так	Так			+
5	Крос-платформеність	Так	Ні	Так	Ні			+

В таблиці 4.1.2 були розглянуті слабкі та сильні сторони майбутнього продукту, це беззаперечно є основною складовою його конкурентноздатності. Сильною стороною є використання покращених методів виявлення шкідливих JavaScript сценаріїв, які знижують кількість помилкових спрацьовувань. Слабкою сторонами є висока собівартість.

4.2 Технологічний аудит ідеї проекту

У даному підрозділі описуються технології за якими створена програма для виявлення шкідливих JavaScript сценаріїв

Таблиця 4.2.1 - Технологічна здійсненність ідеї проекту

№ п /п	Ідея проекту	Технології реалізації	Наявність технології	Доступність технології
1	Створення програми	C	Наявна	Безкоштовна
		JavaScript	Наявна	Безкоштовна
		Python	Наявна	Безкоштовна
Для розробки програми буде використана мова програмування Python				

Розробку програми вирішено виконувати на мові програмування Python через те що команда стартап-проекту знайома з нею більше.

4.3 Аналіз ринкових можливостей стартап-проекту

Аналіз ринкових можливостей необхідний для того щоб визначити в якома є його динаміка, чи існують якісь перешкоди та чи є конкурентні продукти. Ми також можемо змодельовавши запланувати подальший розвиток продукту.

Таблиця 4.3.1 - Попередня характеристика потенційного ринку стартап-проекту

№ п /п	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	60000
3	Динаміка ринку	Зростає
4	Наявність обмежень для виходу	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає

Продовження таблиці 4.3.1

6	Середня норма рентабельності в галузі, %	$R = (3000000 * 100) / (1100000 * 12) = 27\%$
---	--	---

Рентабельність галузі за потенційними показниками є більшою за прибуток від вкладання грошей до банку, тому стартап-проект є досить привабливим для інвесторів. Необхідно зазначити, що на даний момент на ринку немає обмежень для виходу стартап-проекту на ринок та якихось перешкод таких як проходження специфічної сертифікації.

В таблиці 5.3.2 обрані можливі групи потенційних клієнтів та переваг, які надає продукт.

Таблиця 4.3.2 - Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія	Відмінність у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
	Програма для виявлення шкідливих JavaScript сценаріїв	Компанії, фахівці з кібербезпеки	Цільова група намагається виявити шкідливі JavaScript сценарії але є досить велика кількість помилкових спрацювань	Програма повинна вирішувати проблему помилкових спрацювань та бути більш надійною

Основною характеристикою потенційних клієнтів є те, що вони зацікавлені у програмі, яка з більшою точністю зможе виявляти шкідливі JavaScript сценарії. Основними частинами ринку є: великі, середні, малі компанії які вже намагаються виявити шкідливі JavaScript сценарії.

Далі нам необхідно визначити фактори загрози, які наведені у таблиці 4.3.3.

Таблиця 4.3.3 - Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція	Поява на ринку великої компанії	-Вихід стартапу; -Поглинання великою компанією стартапу -Вихід на IPO та залучення інвестицій
2	Зміна потреб потенційних користувачів	Користувачі жадають програму яка виконує ще інші функції	Предметне вивчення ринку і настроїв користувачів для прогнозування необхідних функцій користувачів
3	Збільшення витрат на розробку продукту	Невдале реагування на потреби користувачів	Кропітке планування необхідних коштів на розробку і підтримку продукту
4	Винайдення інноваційного методу до виявлення шкідливих JavaScript сценаріїв	Нова інноваційна технологія, що може витіснити вже існуючу	Адаптація продукту до нової технології

Були розглянуті загрози які становлять загрози і можуть виникнути під час реалізації стартап-проекту. Найбільшою загрозою є досить висока конкуренція яка може з'явитися через появу великої компанії. Стартапу необхідно провести дослідження на предмет використання користувачами програми. Це дозволить збільшити клієнтську базу та покращити розуміння користувачів. Реакція на вихід великої компанії на ринок може бути: вихід стартапу, поглинання великою компанією стартапу, вихід на IPO та залучення інвестицій

Таблиця 4.3.4 - Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Збільшення кількості покупців	Зростання кількості користувачів потенційно може збільшити прибутки	Розробка додаткових функцій за які користувачі будуть готові платити більше
2	Падіння довіри до конкурентних компаній	Клієнти втрачають довіру до програм конкурентів через їх компрометацію	Підкреслення переваг і надійності програми
3	Зменшення витрат	Збільшення продуктивності	Збільшення заробітної платні
4	Зростання зацікавленості до автоматичного виявлення шкідливих JavaScript сценаріїв	Все більше компаній думає над використання автоматизованих рішень задля виявлення шкідливих JavaScript сценаріїв	Необхідно зв'язатися з цими компаніями і запропонувати їм за знижкою використовувати розроблену програму

Були описані основні позитивні фактори: збільшення кількості покупців та недовіри до конкурентів, а також ріст зацікавленості до автоматичного виявлення шкідливих JavaScript сценаріїв. Реакція компанії полягає у використанні можливостей які вадні спричинити позитивну динаміку сприйняття продукту.

Таблиця 4.3.5 - Ступеневий аналіз конкуренції ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
Тип конкуренції: -олігополія	Є декілька компаній які розробляють продукт який виконує такі функції але все вони використовують застарілі підходи до виявлення	Вплив є позитивним тому що стартап пропонує програму на основі покращених методів виявлення і це дає змогу знизити кількість помилкових спрацювань майже у 3 рази.

За рівнем конкурентної боротьби: -міжнародний	Усі три компанії конкурента працюють на міжнародному ринку	Програму слід розробляти англійською для користувачів з усього світу
За галузевою ознакою: -внутрішньогалузева	Всі конкуренти мають схожий продукт та працюють у тій самій галузі	Програма повинна мати змогу змінити обране напрямлення за необхідністю
Конкуренція за видами товарів: -товарно-видова	Товари схожі проте вони дають більшу кількість помилкових спрацювань	Програма зменшує помилкові спрацювання майже в три рази
За характером конкурентних переваг: -націлена	Програма повинна добре вирішувати проблему виявлення шкідливих JavaScript сценаріїв	Програма яка зменшує помилкові спрацювання тому є ліпшою за конкурентів
За інтенсивністю: -Не марочна	Немає відомих брендів	-

Був проведений аналіз конкуренції який існує на ринку та тип конкуренції: олігополія; за ознакою конкурентної боротьби: міжнародна; за галузевою ознакою: внутрішньогалузева. Було розглянуто яка є конкуренція за видами товарів: товарно-видова; інтенсивність: не марочна. Також були описані дії стартапу необхідні для забезпечення конкурентоспроможності. Дії стартапу на початку; фокусування на вирішенні специфічної проблеми та зниження кількості помилкових спрацювань майже у три рази.

Далі описаний перелік факторів конкурентоспроможності на потенційному ринку який був зроблений за допомогою аналізу конкуренції в галузі за методом М.Портера.

Таблиця 4.3.6 - Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Перелік конкурентів	Бар'єри входження в ринок	Фактори сили постачальників	фактори сили споживачів	Фактори загроз з боку замінників
Висновки	В даний час існує три конкуренти програми яких мають досить високий рівень помилкових спрацювань	Немає бар'єрів для входження на ринок	Необхідних постачальників немає	Для користувача є важливим достовірність отриманих результатів	Конкуренти можуть проаналізувати використаний покращений алгоритм та запровадити його у свої програми в іншій формі

Конкурентна ситуація на ринку є дуже привабливою через те що конкурентні програми мають досить велику кількість помилкових спрацювань. Розроблюємо програма знижує кількість помилкових спрацювань майже у три рази. Основними сильними сторонами продукту покращений алгоритм виявлення, надійність та швидкість.

Таблиця 4.3.7 - Обґрунтування факторів конкурентоспроможності

№ п/п	Фактори конкурентоспроможності	Обґрунтування
1	Використання покращеного методу виявлення шкідливих JavaScript сценаріїв	Використання покращених методів виявлення дає змогу зменшити кількість помилкових спрацювань

Продовження таблиці 4.3.7		
2	Швидкість	Продукт спроектований таким чином щоб використовувати переваги покращених методів виявлення, одним з яких є більша швидкість роботи
3	Надійність	Використовуючи покращені методи виявлення програма буде більше надійною

Були виділені фактори конкурентоспроможності на які стартап планує спиратись: використання покращеного методу виявлення шкідливих JavaScript сценаріїв , швидкість та надійність.

На основі цих факторів визначаються сильні та слабкі сторони стартапу.

Таблиця 4.3.8 - Порівняльний аналіз сильних та слабких сторін страп-проекту

№ п /п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів конкурентів						
			-3	-2	-1	0	1	2	3
1	Використання покращеного методу виявлення шкідливих JavaScript сценаріїв	20	+						
2	Швидкість, надійність	15		+					

Були визначені сильні та слабкі сторони продукту в порівнянні з рішеннями конкурентів. Сильною стороною є використання покращеного методу виявлення шкідливих JavaScript сценаріїв .

Нижче наведений SWOT-аналіз стартапу

Таблиця 4.3.9 - SWOT аналіз стартап-проекту

Сильні сторони: використання покращеного методу виявлення шкідливих JavaScript сценаріїв, швидкість, надійність	Слабкі сторони: конкуренти в лиці великих компаній можуть знизити ціну та працювати деякий час у мінус
Можливості: враховуючи те що програми конкурентів дають досить велику кількість помилкових спрацювань є гарна можливість вийти на ринок	Загрози: висока конкуренція, зміна потреб користувачів

У стартапу є наступні сильні та слабкі сторони. Використання використання покращеного методу виявлення шкідливих JavaScript сценаріїв, краща швидкість та надійність. Слабкими сторонами є - потрібен час для прийняття програми. Ринкові можливості стартапу мають наступні можливості і загрози: можливості - вийти на ринок та запропонувати зменшити кількість помилкових спрямувань; загрози - висока конкуренція та зміна потенційних потреб користувачів.

На базі вже проведеного SWOT-аналізу було розроблені альтернативні можливості ринкового впровадження які наведені нижче у таблиці 4.3.1.1.

Таблиця 4.3.1.1 - Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива	Ймовірність отримання ресурсів	Сроки реалізації
1	Створення програми за допомогою мови програмування Python	90%	5 місяців
2	Створення програми за допомогою мови програмування JavaScript	70%	6 місяців
3	Створення програми за допомогою мови програмування C	40%	9 місяців

Альтернативою ринкового впровадження є розробка програми мовою програмування Python через те що команда краще з нею знайома та її вдасться швидше реалізувати.

4.4 Розроблення ринкової стратегії проекту

Далі сформувані ринкову стратегію охоплення ринку а точніше груп потенційних кінцевих користувачів.

Таблиця 4.4.1 - Вибір цільових груп потенційних споживачів

№/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприяти продукту	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Компанії	Отримання відгуків про користування продуктом	Середній	Існує 3 конкуренти програми яких	Висока
2	Спеціалісти з кібербезпеки	Отримання відгуків про користування продуктом	Високий	дають досить велику кількість помилкових спрацювань	Середня
Було обрано: Спеціалісти з кібербезпеки і компанії.					

Було обрано Спеціалісти з кібербезпеки через найлегшу можливість входу та високий попит серед цільової групи.

Таблиця 4.4.2 - Визначення базової стратегії розвитку

№/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
-----	--------------------------------------	---------------------------	--	---------------------------

Продовження таблиці 4.4.2				
1	Створення програми мовою програмування Python яка базуватиметься на покращених методах виявлення шкідливих JavaScript сценаріїв	Ринкове позиціювання	Надійність, Швидкість,	Диференціація

Стратегія розвитку продукту є диференціація, яка має на меті позиціювання. Через те що основними перевагами продукту є надійність та швидкість.

Таблиця 4.4.3 - Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект першопрохідцем на ринку	Чи буде компанія шукати нових споживачів або забирати існуючих у конкурентів	Чи буде компанія копіювати основні характеристики	Стратегія конкурентної поведінки
1	Ні	Так	Стартап буде копіювати вигляд інтерфейсів але пропонувати більш надійну та швидку програму	Стартап буде виставити себе на ринку досить агресивно фокусуючись на перевагах продукту

Було обрано стратегію конкурентної поведінки, яка полягає у агресивному маркетингу. Також було визначено, що компанія буде копіювати вже добре відомі інтерфейси та більш надійний та швидкий продукт.

Таблиця 4.4.4 - Визначення стратегії позиціонування

№/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможності і позиції стартап-проекту	Вибір асоціацій які мають сформувати комплексну позицію власного проекту
1	Програма повинна чітко виконувати основну функцію “Виявлення шкідливих Javascript-сценаріїв”	Диференціація	Виявлення шкідливих JavaScript сценаріїв базуючись на покращених методах виявлення	Передова технологія
2	Швидкість	Диференціація	Програма використовує покращені методи тому є швидшою за конкурентів	Швидкодія
3	Надійність	Диференціація	Надійність продукту полягає у зменшенні кількості помилкових спрацювань	Надійність

Були розглянуті основні вимоги цільової аудиторії до товару: зниження кількості помилкових спрацювань та краща швидкодія. Також була обрана стратегія розвитку, ключові позиції конкурентоспроможності та набір асоціацій, які повинні сформуватися у користувачів.

4.5 Розроблення маркетингової програми стартап-проекту

Для розробки маркетингової стратегії стартапу, необхідно сформулювати маркетингову концепцію продукту.

Таблиця 4.5.1- Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода яку пропонує товар	Ключові переваги перед конкурентами
1	Автоматизація виявлення шкідливих JavaScript сценаріїв	Автоматизоване рішення проблеми виявлення шкідливих JavaScript сценаріїв	Програма автоматизує процес виявлення шкідливих JavaScript сценаріїв базуючись на покращених методах. Це зменшує кількість помилкових спрацювань
2	Швидкість	Швидкість роботи програми	Програма працює швидше за конкурентів тому що використані покращені методи виявлення
3	Надійність	Надійність роботи програми	Програма знижує кількість помилкових спрацювань тому є більш надійною

Перейдемо до створення трирівневої маркетингової моделі товару яка представлена у таблиці 4.5.2.

Таблиця 4.5.2 - Опис трьох рівнів моделі товару

Рівні товару	Сутність та складова		
Товар за задумом	Товар автоматизує процес виявлення шкідливих Javascript-сценаріїв		
Товар у реальному виконанні	Властивості	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Автоматизація виявлення шкідливих JavaScript сценаріїв	Нм	Тх

	2. Швидкість		
	3. Надійність		
	Якість: Програма буде розроблена за покращеними методами виявлення і тому буде зменшувати кількість помилкових спрацювань		
	Пакування відсутнє		
	MaliciousJSDetection		
Товар із підкріпленням	3-х недільне безкоштовне користування програмою		
	Підтримка користувачів		
За рахунок чого потенційний товар буде захищено від копіювання: патент			

У таблиці вище були описані три рівні моделі проекту. Стартап-проект буде виконаний у вигляді програми та допомагатиме автоматично виявляти шкідливі JavaScript . Основними характеристиками є: швидкість та надійність. Програму буде захищено від копіювання за патентом.

Далі, була визначена цінова межа, вона визначалася за експертним методом.

Таблиця 4.5.3 - Визначення між встановлення ціни

№/п	Рівень цін на товари-замінники, грн/рік	Рівень цін на товари-аналоги, грн/рік	Рівень доходів цільової групи споживачів, грн/рік	Верхня та нижня межі встановлення ціни на товар/послугу, грн/рік
1	70000	60000	500000	50000, 40000

Були визначені межі можливих цін за підписку на використання програмою; рівень цін товарів-замінників та доходів споживачів. Верхньою межею є 40000, нижньою є 50000 грн/рік відповідно.

Оптимальна система збуту у межах якої можуть прийматися потенційні рішення наведена у таблиці 5.5.4.

Таблиця 4.5.4 - Формування системи збуту

№/п	Специфіка закупівельної поведінки клієнтів	Функції збуту, яка має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Клієнти купують місячну або річну підписку за користування програмою	Продаж	За власними каналами	За власними каналами

Було розроблено систему збуту, яка має форму підписки на місяць або рік.

Збут буде виконуватися за власними каналами.

Далі необхідно розробити концепцію маркетингових комунікацій стартапу, які наведені у таблиці 4.5.5.

Таблиця 4.5.5 - Концепція маркетингових комунікацій

№/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Завантаження програми з офіційного сайту стартапу	Інтернет	Надійність, Швидкість	Показати що продукт зменшую помилкові спрацювання майже у три рази	Ролик з демонстрацією роботи та зниження помилкових спрацювань у процесі виявлення

Були створена концепція потенційних маркетингових комунікацій. Завантаження для кожної з платформ буде можливе з офіційного сайту, а каналом комунікації є інтернет. Рекламне повідомлення повинно мати форму

демонстраційного відео та показувати зниження помилкових спрацювань у процесі виявлення.

Висновки до розділу 4

У розділі була розглянута ідея та потенційна концепція стартапу. Програма націлена на компанії та спеціалістів з кібербезпеки які намагаються виявити шкідливі JavaScript сценарії та прагнуть зменшити кількість помилкових спрацювань.

У рамках розділу були визначені сильні та слабкі сторони стартапу через необхідність формування його конкурентоспроможності на ринку. Продукт буде створений у вигляді програми, розроблених за допомогою мови програмування Python, яка є безкоштовною. Також було проведено та представлено: ступеневий аналіз конкуренції, SWOT аналіз, менеджмент ризиків.

Через те що розробляються програма базується на покращених методах виявлення, які дозволяють зменшити помилкові спрацювання майже в три рази та комерційна привабливість більша ніж від банківського депозиту можна зробити висновок, що стартап-проект є доцільним.

ВИСНОВКИ

У дипломній роботі запропоновано актуальне для сфери виявлення шкідливих програмних засобів, що стосується значного покращення статичних методів детектування шкідливих . В ході вирішення поставленого завдання були отримані наступні аналітичні та практичні результати:

- 1) Була проведена повна, детальна класифікація існуючих шкідливих JS-сценаріїв, а також методів їх виявлення.
- 2) Був розроблений допоміжний програмний засіб для витягу ознак шкідливості.
- 3) На основі нових ознак шкідливості Javascript, була підвищена точність детектування методів статичного аналізу.
- 4) Доведено економічну доцільність розробки стартапу і впровадження удосконалених методів.

Для остаточного висновку про ефективність внесених удосконалень необхідно проведення більш масового produciton-тестування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Zaharia A. JavaScript Malware - A Growing Trend Explained for Everyday Users. 2017.
2. Wang WH, Yin-Jun LV, Chen HB, Fang ZL. A static malicious JavaScript detection using svm. Proceedings of the International Conference on Computer Science and Electronics Engineering; 2013; 40:21-30.
3. Seshagiri P, Vazhayil A, Sriram P. AMA: Static code analysis of web page for the detection of malicious scripts. Procedia Computer Science. 2016 Dec 31; 93:768-73.
4. Wang J, Xue Y, Liu Y, Tan TH. JSDC: A hybrid approach for JavaScript malware detection and classification. Proceedings of the 10 th ACM Symposium on Information, Computer and Communications Security; 2015. p. 109-20.
5. Canfora G, Mercaldo F, Visaggio CA. Malicious JavaScript detection by features extraction. e-Informatica Software Engineering Journal. 2014; 8(1).
6. Xue Y, Wang J, Liu Y, Xiao H, Sun J, Chandramohan M. Detection and classification of malicious JavaScript via attackbehavior modeling. Proceedings of the International Symposium on Software Testing and Analysis; 2015. p. 48-59.
7. Gu B, Zhang W, Bai X, Champion AC, Qin F, Xuan D. JS-Guard: Shellcode detection in JavaScript. Proceedings of the International Conference on Security and Privacy in Communication Systems; 2012. p. 112-30.
8. Xu W, Zhang F, Zhu S. JStill: Mostly static detection of obfuscated malicious JavaScript code. Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy; 2013. p. 117-28.
9. Likarish P, Jung E, Jo I. Obfuscated malicious JavaScript dytection using classification techniques. Proceedings of the 4 th International Conference on Malicious and Unwanted Software (MALWARE); 2009. p. 47-54.
10. Fraiwan M, Al-Salman R, Khasawneh N, Conrad S. Analysis and identification of malicious JavaScript code. Information Security Journal: A Global Perspective. 2012; 21(1):1-1.

11. Rieck K, Krueger T, Dewald A. Cujo: efficient detection and prevention of drive-by-download attacks. Proceedings of the 26 th Annual Computer Security Applications Conference; 2010. p. 31-9.
12. Dewald A, Holz T, Freiling FC. ADSandbox: Sandboxing JavaScript to fight malicious websites. Proceedings of the ACM Symposium on Applied Computing; 2010. p. 1859-64.
13. P. Vogt, F. Nentwich, N. Jovanovic, and E. Kirda. Cross-site scripting prevention with dynamic data tainting and static analysis. In Proceeding of the Network and Distributed System Security Symposium, Jan 2007.
- 14 I. Witten and E. Frank. Data Mining: Practical machine learning tools and techniques, 2nd ed. Morgan Kaufman, San Francisco, 2005.
15. Hallaraker O, Vigna G. Detecting malicious JavaScript code in Mozilla. Proceedings of the 10 th IEEE International Conference on In Engineering of Complex Computer Systems(ICECCS); 2005. p. 85-94.
16. Alexa: Alexa top 500 global websites. 2016. Crossref
17. PhishTank: Phishtank developer information. 2016. Crossref
18. HynekPetrak: Sandbox for semi-automatic JavaScript malware analysis, deobfuscation and payload extraction. 2017.
19. Dharmaraj R. Patil * and J. B. Patil. Detection of Malicious JavaScript Code in Web Pages. 2017.
- . Cova M, Kruegel C, Vigna G. Detection and analysis of drive-by-download attacks and malicious JavaScript code. Proceedings of the 19 th International Conference on World Wide Web; 2010. p. 281-90.
20. Schwenk G, Bikadorov A, Krueger T, Rieck K. Autonomous learning for detection of JavaScript attacks: Vision or reality? Proceedings of the 5 th ACM Workshop on Security and Artificial Intelligence; 2012. p. 93-104.

ДОДАТОК А

Наївний байесівський класифікатор

```
import math
import statistics
import csv.reader

train_set=[]
tset=[]

def load_csv(path):
    with open(path, "r") as f:
        ls = csv.reader(f);
        data = list(ls)
        for i in range(len(data)):
            for j in data[i]:
                data[i] = float(j)
    return dataset

def get_probability(a, m, s):
    e = math.exp(-(math.pow(a-m,2)/(2*math.pow(s,2))))
    return 1 / (math.sqrt(2*math.pi) * s*s) * e

def main():
    dataset=load_csv('js_feature.csv')
    size=0.5
```

```

for i in range(len(dataset)*size):
    tset.append(dataset[i])
for i in range(len(dataset)*size+1,len(dataset)):
    train_set.append(dataset[i])
print('Довжина тестового сету',len(tset))
print('Довжина тренувального сету -- ',len(train_set))

class=[]
for i in dataset:
    if i[-1] not in class:
        class.append(i[-1])

class={}
class1={}
prob={}
for i in class:
    class[i]=[]
    class1[i]=[]
    prob[i]=1

for i in class:
    for row in train_set:
        if row[-1]==i:
            class[i].append(row[:-1])

for val,datt in class.items():
    for col in zip(*datt):

class1[val].append((st.m(col),st.s(col)))

count=0

```

```

for row in tset:
    for i in cls:
        prob[i]=1
    for val,datt in cls1.items():
        for i in range(len(row[:-1])):
            m,std=datt[i]
            t=row[i]
            prob[val]*=calprob(x,t,std)
    print(prob," для рядку ",row)
    mini=0
    cl=0
    for c,d in prob.items():
        if d>mini:
            mini=d
            cl=c

    if row[-1]==cl:
        count+=1

if __name__ == "__main__":
    acc=count/len(tset)
    print("Точність відповіді --- ", acc)

```

Випадковий ліс

```
import numpy as np
from scipy.stats import mode
from decisiontree import DecisionTreeClassifier
from utilities import shuffle_in_unison

class RandomForestClassifier:
    def __init__(self, mdepth, mfeatures=np.sqrt,
                 mnsample_split, b=0.9, n=32):

        self.mdepth = max_depth
        self.mnsamples_split = mnsample_split
        self.bootstrap = b
        self.forest = []
        self.n = n_estimators
        self.mfeatures = max_features

    def fit(self, X, y):
        self.forest = []
        n_samples = len(y)
        n_sub_samples = round(n_samples*self.bootstrap)

        for i in xrange(self.n_estimators):
            shuffle_in_unison(X, y)
            X_subset = X[:n_sub_samples]
            y_subset = y[:n_sub_samples]
```



```

        tree =
DecisionTreeClassifier(self.max_features, self.max_depth,
self.min_samples_split)
        tree.fit(X_subset, y_subset)
        self.forest.append(tree)

```

```

def score(self, X, y):
    y_predict = self.predict(X)
    n_samples = len(y)
    correct = 0
    for i in xrange(n_samples):
        if y_predict[i] == y[i]:
            correct = correct + 1
    accuracy = correct/n_samples
    return accuracy

```

```

def predict(self, X):
    n_samples = X.shape[0]
    n_trees = len(self.forest)
    predictions = np.empty([n_trees, n_samples])
    for i in xrange(n_trees):
        predictions[i] = self.forest[i].predict(X)

    return mode(predictions)[0][0]

```

Видалення потенційно шкідливих

```
import os
import time
import base64
import logging
import os.path
from datetime import datetime

import log
import utils

from googleapiclient.discovery import build
from httplib2 import Http
from oauth2client import file, client, tools

SCOPES = "https://www.googleapis.com/auth/gmail.readonly"

bd = os.path.dirname(os.path.abspath(__file__))

LOG_FILE = f"{bd}/prices.log"
TOKEN_FILE = f"{bd}/creds/token.json"
CONFIG_FILE = f"{bd}/config.json"
CREDENTIALS_FILE = f"{bd}/creds/credentials.json"

BASE_DIR = "{} /files/".format(os.path.dirname(bd))
ARCHIVE = "{} /files/archives".format(os.path.dirname(bd))
```

```

logger = logging.getLogger()
logging.getLogger("googleapiclient.discovery").setLevel(log
ging.ERROR)

```

```

def auth_gmail():
    store = file.Storage(TOKEN_FILE)
    creds = store.get()
    if not creds or creds.invalid:
        flow =
client.flow_from_clientsecrets('credentials.json', SCOPES)
        creds = tools.run_flow(flow, store)
    return creds.authorize(Http())

def get_email_subject(msg_payload):
    subject = "<Тема отсылки>"
    for p in msg_payload:
        if p.get("name", "") == "Subject":
            subject = p["value"] if p["value"] else subject
            break
    return subject

def get_new_emails_from_sender(email, after_time, service):
    query = f"from:{email} after:{after_time}"
    results = service.users().messages().list(userId='me',
labelIds=['INBOX'], q=query).execute()
    fetch_time = time.time() - 1
    return results.get('messages', []), fetch_time

```

```

def save_attachments_from_emails(messages, base_dir,
service):
    saved = 0

    for message in messages:
        msg = service.users().messages().get(userId='me',
id=message['id']).execute()

        attachments_number = len([p for p in msg["payload"]
["parts"] if p['filename']])
        subject = get_email_subject(msg['payload']
['headers'])
        email_time =
datetime.fromtimestamp(int(msg['internalDate'])/1000).strft
ime("(%d.%m.%y)")

        if not attachments_number:
            logger.info(f'Лист не має закріплень')
            continue

        logger.info(f'Письмо під темою "{subject}" містить
вкладень: {attachments_number}')

        for part in msg["payload"]["parts"]:
            if part['filename']:
                attach_id = part["body"]["attachmentId"]
                logger.info(f'Загружаю вложение
{part["filename"]}')
```

```

        attachment =
service.users().messages().attachments().get(id=attach_id,
messageId=msg['id'], userId='me').execute()

        file_data =
base64.urlsafe_b64decode(attachment["data"].encode('UTF-
8'))

        path = ''.join([base_dir, f"{email_time}
{part['filename']}"])
        with open(path, "wb") as f:
            f.write(file_data)

        file_ext = os.path.splitext(path)[1]

        if file_ext == ".zip":
            utils.create_dir_if_not_exists(ARCHIVE)
            logger.info("Вкладення формату '.zip'.
Підозріле. Деархівірую...")
            utils.unzip(path)
        elif file_ext == ".rar":
            utils.create_dir_if_not_exists(ARCHIVE)
            logger.info("Вкладення формату '.rar'")
            utils.unrar(path)

        saved += 1
        logger.info('Вложение успешно сохранено.')

    return saved

```

```

def run_fetcher(last_fetch, sender, timeback):
    fetch_timestr = utils.prettify_time(last_fetch)
    logger.info(f"Поизвожу поиск писем от {sender},
полученных после {fetch_timestr}...")

    logger.debug("Соединяюсь...")
    service = build('gmail', 'v1', http=auth_gmail(),
cache_discovery=False)
    logger.debug("Аутентификация прошла успешно. Соединение
с Gmail установлено.")

    logger.debug("Ищу новые письма...")
    messages, fetch_time =
get_new_emails_from_sender(sender, last_fetch, service)
    logger.info(f"Новых писем -- {len(messages)}")

    fetch_timestr = utils.prettify_time(fetch_time)
    logger.info(f"Новое время последней проверки на наличие
новых писем -- {fetch_timestr}")
    logger.debug(f"Зміню час в конфігураційному файлі")
    try:
        utils.write_config(CONFIG_FILE, fetch_time, sender)
    except Exception as exc:
        logger.error(f"Помилка -- {exc}")

    if not messages:
        logger.info(f"Завершення роботи.\n\n")
        utils.notify_me("Нових листів немає. Fetcher...")
        exit()

    logger.debug("Шукаю вкладення и перевіряю...")

```

```

        saved_number = save_attachments_from_emails(messages,
BASE_DIR, service)
        logger.info(f"Підозрілих вкладень: {saved_number}")

def main():
    utils.notify_me("Javascript..")

    args = utils.parse_args()
    log.setup_logging(LOG_FILE, args.verbose)

    logger.debug("Завантажую конфігураційний файл...")
    try:
        cfg = utils.load_config(CONFIG_FILE)
    except Exception as exc:
        msg = f"Помилка-- {exc}"
        logger.error(msg)
        utils.notify_me(msg)

    last_fetch = int(cfg["last_fetch"])
    if args.timeback:
        logger.info("Ввімкнено режим таймбек")
        last_fetch -= args.timeback

    logger.info("Програма працює! Запускаю Fetcher...")
    try:
        run_fetcher(last_fetch, cfg["sender"],
args.timeback)
    except Exception as exc:
        msg = f"Помилка -- {exc}"
        logger.error(msg)

```

```
        utils.notify_me(msg)

    logger.info(f"Завершення роботи.\n\n")
    utils.notify_me("Закінчую Fetcher...")

if __name__ == '__main__':
    main()
```