МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук
«_____»_____ 2020 р.
На правах рукопису
УДК

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Удосконалений алгоритм захисту від шкідливих Javascript сценаріїв

Виконавець: Т.В. Самойлова
Науковий керівник: к.т.н., доц. А.Б. Петренко
Нормоконтролер: к.т.н., доц. А.Б. Петренко

Київ 2020

ВСТУП

JavaScript - це динамічна мова сценаріїв на стороні клієнта, яка використовується для створення контенту для Інтернету разом з HTML і CSS. Вона використовується на більшості веб-сайтів та підтримується всіма сучасними веб-браузерами. Мова широко використовується для розробки інтерактивних веб-сторінок, однак в останні роки JavaScript стала найбільш поширеною і успішною мовою побудови атак. Шкідливий код JavaScript може бути вставлений на веб-сторінку і запускатися при завантаженні в будь-якому браузері, обходячи інструменти безпеки, такі як брандмауер і антивірусне програмне забезпечення. Кіберзлочинці регулярно маніпулюють кодом на незліченних веб-сайтах, щоб змусити його виконувати шкідливі функції. JavaScript - настільки динамічна мова програмування, що її неправильні реалізації можуть створити лазівки для зловмисників. Коли користувачі відвідують веб-сайт, файли JavaScript завантажуються автоматично.

Нові веб-атаки відбуваються щодня, що змушує компанії, спільноти і окремих осіб серйозно ставитися до питань безпеки.

Метою дипломної роботи є реалізація удосконаленого алгоритму захисту від шкідливих Javascript сценаріїв.

Об'єкт дослідження: процес захисту користувача від шкідливих Javascript сценаріїв.

Предмет дослідження: удосконалений алгоритм захисту від шкідливих Javascript сценаріїв.

Виходячи з мети, завданнями даної дипломної роботи є:

огляд основних вразливостей мови Javascript; аналіз існуючих методів виявлення шкідливих Javascript сценаріїв; розробка удосконаленого алгоритму захисту для підвищення рівня виявлення шкідливих Javascript сценаріїв; дослідження удосконаленого алгоритму.

Практична цінність роботи полягає в удосконаленні алгоритму Дерево рішень для виявлення шкідливих Javascript сценаріїв.

Наукова новизна. Удосконалено алгоритм Дерево рішень, за рахунок застосування власного набору функцій і реалізації рейтингу шкідливості коду, що дозволило проводити виявлення шкідливих Javascript сценаріїв з більшою точністю.

Самойлова Т.В. Виявлення вразливостей Javascript бібліотек / Самойлова Т.В., Петренко А.Б. // Materiały XVI Międzynarodowej naukowi-praktycznej konferencji ,«Nauka I inowacja - 2020» , Volume 8 Przemyśl: Nauka i studia – P. 71-73.

Самойлова Т.В. Засоби криптографічного захисту конфіденційної інформації за допомогою електронно-цифрового підпису / Самойлова Т.В. // Materiály XVI Mezinárodní vědecko - praktická konference «Aplikované vědecké novinky», Volume 2 : Praha. Publishing House «Education and Science» - P. 36-39.

Розділ 1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО JAVASCRIPT, ВРАЗЛИВОСТІ ТА ПРОБЛЕМИ БЕЗПЕКИ

1.1. Основні визначення та поняття

JavaScript (далі JS) – це інтерпретуєма прототипно-орієнтована клієнтська сценарна мова програмування, що функціонує на основі подій [1], одна з найпопулярніших мов веб-розробників на ряду з HTML та CSS. На самому початку своєї 25-річної історії ця мова отримала назву «LiveScript» [2], яка пояснювала основну мету її створення – зробити веб-сторінки «живими», а саме додати інтерактивності та привабливості для відвідувачів.

Сайт DOU.ua [3] представили результати щорічного опитування, щодо актуальних зараз мов програмування, зібрали обробили 9747 анкет, і як результат, можна побачити, що JS випередив Java на 3% та заняв лідируючу позицію в Україні. Результати опитування представлено на Рис. 1.

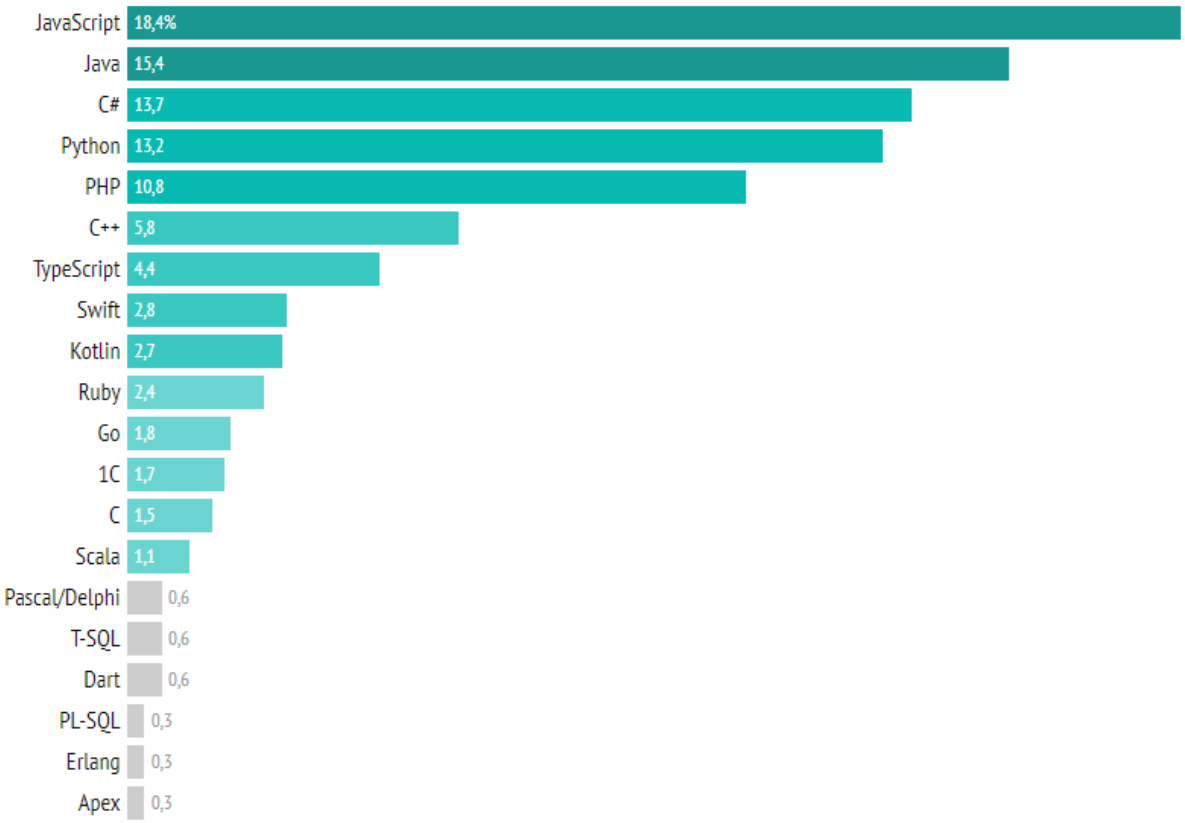

Рис. 1. Результати опитування DOU.ua

Indeed, американська система пошуку роботи по всьому світу, щорічно виставляє свій рейтинг [4], які мови програмування користуються попитом у роботодавців. На Рис.2 показані результати за останні 4 роки.

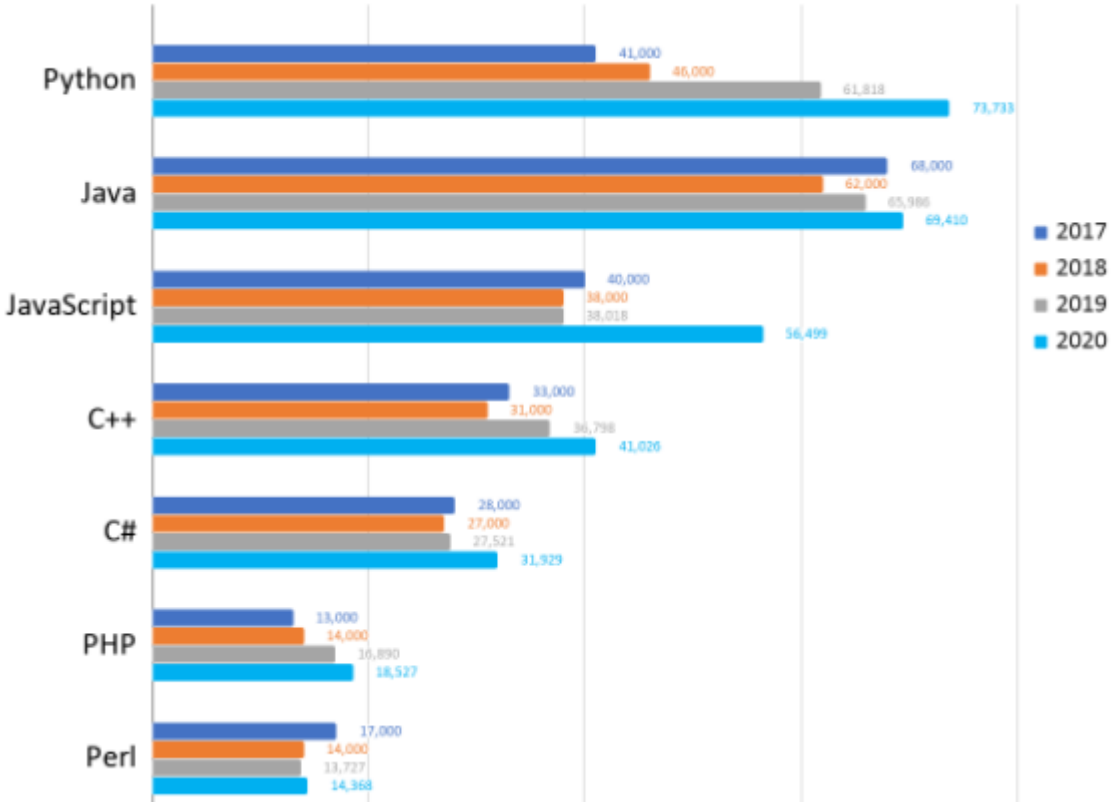

Рис. 2. Рейтинг мов програмування за версією Indeed

Використання JS відкриває такі можливості як створення спеціальних ефектів, які надають враження, що кнопка або виділена, або натиснута при наведенні курсора на неї; перевірка інформації, яку користувачі вводять в веб-форми; відкриття сторінки в нових вікнах і налаштування зовнішнього вигляду цих нових вікон; визначення можливостей браузера користувача і відповідна заміна змісту веб-сторінки; створення власної сторінки без необхідності використання серверної мови, такої як PHP та багато інших [5].

JS – це не Java, хоча обидві мови при написанні виглядають однаково. Java – це повнофункціональна і всеосяжна мова програмування, схожа на C або C ++, і хоча JS може взаємодіяти з веб-додатками Java, їх не слід плутати.

Щоб пояснити перше визначення JS в цьому розділі, звертається увага на наступні моменти:

1. Інтерпретація відноситься до того факту, що код JS виконується (обробляється), коли він завантажується в браузер. Це пришвидшує темп в порівнянні мовами, що компілюються, такими як Java, які ретельно перевіряють по рядкам програму перед запуском коду.

2. JS зазвичай використовується в веб-браузерах, а розширення його можливостей за допомогою реалізації об'єктів дозволяє організувати взаємодію з користувачем, управляти веб-браузером і змінювати вміст документа, що відображається в межах вікна веб-браузера. Ця вбудована версія JS запускає сценарії, впроваджені в HTML-код веб-сторінок. Як правило, ця версія називається клієнтською мовою JS, щоб підкреслити, що сценарій виконується на клієнтському комп'ютері, а не на веб-сервері [6].

3. Функціонування на основі подій означає здатність JS запускати певні фрагменти коду тільки при настанні певної події. Подією може бути завантаження сторінки, відправка форми, натискання посилання або зображення, на яке вказує курсор.

4. Об'єктно-орієнтованість вказує на те, що можливості JS для контролю над HTML-сторінкою засновані на маніпулюванні об'єктами на цій сторінці, JS емулює класи об'єктів за допомогою функції-конструктора [7].

Інтерпретатор JS – це програма, яка буде читати і виконувати код. Всі браузери, які підтримують JS, є інтерпретаторами JS.

JS-движок – це програма або інтерпретатор, здатний розуміти і виконувати JS-код [8]. JS-движки зазвичай використовуються в веб-браузерах, включаючи V8 в Google Chrome, SpiderMonkey в Mozilla Firefox і Chakra в Microsoft Edge. Кожен движок подібний мовному модулю, який дозволяє додатку підтримувати певну підмножину мови JS.

JS – сценарна мова загального призначення, відповідна специфікації ECMAScript (далі ES).

ES – описана в ECMA-262 специфікація створення сценарної мови загального призначення.

ECMA-262 – це назва і стандарту, і специфікації сценарної мови ES. ES містить правила, відомості та рекомендації, які повинні дотримуватися сценарною мовою, щоб він вважався сумісним з ES [9].

Говорячи про підтримку в браузерах, зазвичай згадується про сумісність з ES, а не про сумісність з JS.

Як вже було зазначено раніше, ES – це специфікація того, як може виглядати сценарна мова. Поява нової версії ES не означає, що у всіх движків JS з'являться нові функції. Все залежить від груп або організацій, які відповідають за оновлення JS-движків з урахуванням новітньої специфікації ES.

Тому розробники, як правило, дізнаються яку версію ES підтримує цей браузер або які функції ES підтримує цей браузер, щоб дізнатися чи вдалося Google, Mozilla і Microsoft наділити JavaScript-движки (відповідно V8, SpiderMonkey і Chakra) властивостями, описаними в останній версії ES. Якщо вийде нова версія ES, то в JS-движках не з'являться разом всі нововведення, вони будуть впроваджуватися поступово, реліз за релізом.

Кожні кілька років ES випускає нову версію. По суті, нова версія містить список нових функцій, які необхідно додати. Не всі оновлюють свій браузер, і не всі браузери підтримують одні й ті ж функції JS. Остання версія ES – ES 2020. Однак, не всі браузери підтримують цю нову версію ES (див. таблицю 1.1), оскільки їм, в свою чергу, необхідно реалізувати нові функції.

Крім того, є багато людей з застарілими версіями свого браузера, тому може пройти деякий час, перш ніж всі користувачі зможуть використовувати ці нові функції ES 2020.

Таблиця 1.1

Список номерів версій JS, ES та сумісність з десктопними браузерами

Рік	Версія JS	Версія ECMA	Браузер
1996	1.0		Netscape 2
1997	1.1,1.2	ES 1	Internet Explorer 4
1998	1.3		Netscape 4

1999		ES 2	Internet Explorer 5
2000		ES 3	Internet Explorer 5.5
2000	1.5		Netscape 6, Firefox 1
2011		ES 5	Internet Explorer 9 (окрім функції «use strict»)
2011	1.8.5		Mozilla Firefox 4 (за виключенням початкових нулів в parseInt), Internet Explorer 10, Google Chrome 23, Safari 6, Mozilla Firefox 21, Opera 15 [10]
2015	2.0	ES 2015	Microsoft Edge 14, Mozilla Firefox 54, Google Chrome 58, Safari 10, Opera 55
2016		ES 2016	Google Chrome 68, Opera 47
2017		ES 2017	Mozilla Firefox 79, Google Chrome 85, Microsoft Edge 83, Safari 13.1 (частково), Opera 69
2018		ES 2018	Mozilla Firefox 79, Google Chrome 85, Microsoft Edge 83, Safari 13.1, Opera 69
2019		ES 2019	Mozilla Firefox 79, Google Chrome 85, Microsoft Edge 83, Safari 13.1 (частково), Opera 69
2020		ES 2020	Mozilla Firefox 79, Google Chrome 85, Microsoft Edge 83, Safari 13.1 (частково), Opera 69 [11]

У цьому середовищі виконується JS-код і інтерпретується JS-движком. Середовище виконання надає хост-об'єкти, на яких і з якими може працювати JS.

Середовище виконання JS – це "існуючий об'єкт або система", згадані у визначенні сценарної мови. Код проходить через JS-движок, в якому об'єкт або система аналізує код і розбирає його роботу, а потім виконує інтерпретовані дії.

JS-сценарії можуть звертатися до додатків, тому що ті надають "хост-об'єкти" в середовищі виконання. На стороні клієнта середовищем виконання JS буде веб-браузер, в якому стають доступними для маніпуляцій такі хост-об'єкти, як вікна і HTML-документи. Це веб-API, об'єкти, що надаються браузером, що діє як хост-середовище JS.

На серверній стороні середовище виконання JS – це Node.js, програмна платформа, заснована на движку V8, що перетворює JS з вузькоспеціалізованого мови в мову загального призначення. Node.js додає можливість JS взаємодіяти з пристроями введення-виведення через свій API, написаний на C ++, підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JS-коду. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js і десктопні віконні додатки, за допомогою NW .js, AppJS або Electron для Linux, Windows і Mac OS, і навіть програмувати мікроконтролери, наприклад, tessel і espruino.

Різні середовища виконання JS можуть використовувати один і той же JavaScript-движок. Наприклад, V8 – це движок, який використовується в двох абсолютно різних середовищах - в Google Chrome і Node.js.

Бібліотека JS – це багаторазово використовуваний фрагмент коду, який пропонує набір функцій, об'єктів і класів, які можна використовувати в додатку [12]. Бібліотека абстрагує різні рівні, тому не потрібно турбуватися про їх реалізацію. JS надає можливість викликати бібліотечну функцію і передати їй деякі параметри, а бібліотека виконає її і поверне керування.

У цьому дослідженні [13] показано використання бібліотек JS у мережі. Окремо скануються веб-сайт Alexa з найвищим рейтингом 75 тисяч і веб-сайти, розташовані в домені .com верхнього рівня. Результати представлені в таблиці 1.2.

Таблиця 1.2

15 JavaScript бібліотек, що найчастіше використовуються сайтами

Бібліотека	Використання на сайтах, %	
	Alexa	.com
jQuery	84.5	62.6
jQuery-UI	24.7	8.6
Modernizr	22.1	9.3
Bootstrap	13.7	4.8
Yepnope	13.6	5.3
jQuery-Migrate	11.3	10.7
SWFObject	10.7	5.2
Underscore	5.8	2.4
jQuery-Tools	4	1.2
Flexslider	3.6	1.6
Moment	3.5	1.4
RequireJS	3.4	2.3
jQuery-Form	2.7	3.4
Backbone	2.7	1.6
Angular	2.5	1.6

Згідно з отриманими результатами, 46,5% сайтів домену .com використовують принаймні одну добре відому бібліотеку JS, причому jQuery є найпопулярнішою для переважної більшості.

Як і всі мови програмування, JS має певні переваги і недоліки, які слід враховувати. Багато з них пов'язані з тим, що JS часто виконується безпосередньо в браузері клієнта. Але тепер є й інші способи використання JS, які дозволяють їй мати переваги серверних мов. Далі було розглянуто наступні переваги мови:

1. Швидкість. JS, як правило, є дуже швидкою, тому що вона часто запускається відразу в браузері клієнта. Поки він не вимагає зовнішніх ресурсів, JS не вповільнюється через виклики внутрішнього сервера. Крім того, всі основні

браузери підтримують Just-In-Time-компіляцію для JS, а це означає, що немає необхідності компілювати код перед його запуском.

2. Простота. Синтаксис JS був натхненний Java, і його відносно легко вивчити порівняно з іншими популярними мовами, такими як C ++.

3. Популярність. JS застосовується всюди в мережі, а з появою Node.js він все частіше використовується в серверній частині. Є незліченна безліч ресурсів для вивчення JS. І StackOverflow, і GitHub демонструють зростання кількості проектів, що використовують JS, і очікується, що популярність, отримана в останні роки, тільки зросте.

4. Взаємодія. На відміну від PHP або інших мов сценаріїв, JS можна вставити на будь-яку веб-сторінку. JS може використовуватися в багатьох різних додатках завдяки підтримці інших мов, таких як Pearl і PHP.

5. Навантаження на сервер. JS є клієнтською мовою, тому вона знижує потребу в серверах в цілому, а простим додаткам сервер може взагалі не знадобитися.

6. Розширена функціональність. Розробники можуть розширити функціональність веб-сторінок, написавши фрагменти коду JS для сторонніх надбудов.

7. Універсальність. Є багато способів використовувати JS через сервери Node.js. Якщо потрібно було завантажити Node.js за допомогою Express, використовувати базу даних документів, таку як MongoDB, і використовувати JS в зовнішньому інтерфейсі для клієнтів, можна було б розробити ціле додаток JS від початку до кінця, використовуючи тільки JS [14].

8. Оновлення. З моменту появи ES 5, ECMA International щорічно оновлює JS.

На противагу до цих численних переваг стоять два недоліки:

1. Підтримка браузера. В той час, коли серверні сценарії завжди виробляють однаковий результат, браузери іноді інтерпретують код JS по-різному, в наш час відмінності мінімальні. Також не всі браузери підтримують нові версії JS, що було розглянуто раніше в таблиці 1.1.

2. Безпека на стороні клієнта. Оскільки код JS виконується на стороні клієнта, помилки та упущення іноді можуть використовуватися в зловмисних цілях. Через це деякі люди вважають за краще повністю відключити JS. Це є найвагомішим недоліком мови, який потребує невідкладного вирішення.

1.2. Вразливості JavaScript

Бібліотеки JS можуть піддаватися злочинним втручанням, і це порушення призводить до сценаріїв, які дозволяють відстежувати активність в Інтернеті і порушувати конфіденційність користувачів. Веб-сайт може бути зламаний, навіть якщо він досить безпечний, через уразливість в бібліотеці JS, яку веб-розробник використовуєте на своєму веб-сайті. Далі розглянуті три типи зловмисного використання JS:

1. Відстеження слів, що вводяться.

Зловмисники можуть відстежувати текстові поля, в яких користувачі оновлюють статус, пишуть коментарі ті інше, використовуючи невеликий вбудований код JS. У липні 2012 року пара дослідників [15][16] зібрала дані від 5 мільйонів користувачів Facebook в США і Великобританії. Дослідники ясно дали зрозуміти, що вони використовують тільки записаний контент, який являє собою «наявність або відсутність введеного тексту», а не «натискання клавіш або контент». Проте можна було відстежувати натискання клавіш і контент, вони просто вирішили не робити цього спираючись на чинне законодавство. Цілком очевидно, що зловмисники не будуть вести себе так само.

Невеликий обсяг вбудованого JS - це все, що потрібно для запису будь-яких дій на веб-сторінці, навіть якщо користувач насправді нічого не відправляє. Прокрутка веб-сторінок, рух миші, натискання клавіш: все це можна відстежувати і записувати проти волі або без відома відвідувача.

2. Відстеження звичок перегляду

Речі, які можна відстежувати за допомогою JS, не обмежуються тільки вмістом натискання клавіш, трохи вбудованого коду JS може відстежувати файли cookie браузера. Завдяки магії файлів cookie браузера компанії можуть зберігати всіляку інформацію, специфічну для користувача: тип браузера, переваги, місце

розташування та інше. Багато веб-сайтів вже відстежують файли cookie браузера [17], щоб забезпечити кращий користувальницький досвід. Грубо кажучи, можна подумати, що вони постійно спостерігають за користувачами, коли вони займаються серфінгом в Інтернеті.

3. Впровадження шкідливого коду

Одним з найбільш хитрих способів використання JS є міжсайтовий скриптинг (далі – МСС), він та інші проблеми безпеки будуть більш детально розглянутий далі в цьому розділі. Простіше кажучи, МСС – це вразливість, яка дозволяє хакерам впроваджувати шкідливий код JS в легальний веб-сайт, який в кінцевому підсумку запускається в браузері користувача, який відвідує цей веб-сайт.

Якщо це відбувається на веб-сайті, який обробляє конфіденційну інформацію про користувачів, наприклад фінансові дані, шкідливий код потенційно може перехопити і вкрасти цю інформацію. Зробивши ще один крок,МСС можна використовувати для відтворення вірусів і шкідливих програм, що і сталося, коли Twitter був заражений черв'яком StalkDaily [18].

Хоча в базовій мові JS і базовій клієнтській об'єктній моделі відсутні файлова система і мережеві функції, які потрібні для більшості шкідливих програм. У багатьох веб-браузерах JS використовується як «механізм сценаріїв» для інших програмних компонентів, таких як елементи керування ActiveX в Internet Explorer і Plug-in в Netscape. Ці компоненти можуть мати файлову систему і мережеві можливості, і той факт, що програми JS можуть управляти ними, затуманює картину і викликає проблеми безпеки. Це особливо вірно по відношенню до елементів управління ActiveX, і Microsoft час від часу доводилося випускати виправлення безпеки, щоб запобігти використання кода JS можливостей об'єктів ActiveX з можливістю створення сценаріїв.

Хоча це навмисна відсутність функцій в клієнтському JS забезпечує базовий рівень захисту від найбільш обурливих атак, інші проблеми безпеки залишаються. В першу чергу це проблеми конфіденційності - програмам на JS можна дозволяти експортувати інформацію про користувача браузера, якщо передбачається, що ця інформація є приватною.

Коли користувачі переглядають Інтернет, одна з частин інформації про себе, яку відвідувач за замовчуванням погоджуєтеся розголошувати, - це використовуваний їм веб-браузер. Як стандартна частина протоколу HTTP, рядок, що ідентифікує браузер, його версію і постачальника, відправляється з кожним запитом веб-сторінки. Ця інформація є загальнодоступною, як, наприклад, IP-адреса Інтернет-з'єднання. Інша інформація, яка, однак, не повинна бути загальнодоступною включає адресу електронної пошти, яка не повинна бути розголошеною, якщо користувач не вирішує зробити це, надіславши електронного повідомлення або дозволивши автоматичне електронне повідомлення для відправки від його імені.

Також історія переглядів (запис про те, які сайти вже відвідувались) і вміст списку закладок повинні залишатися конфіденційними. Історія переглядів і закладки багато говорять про інтереси користувача; це інформація, за яку маркетологи та інші особи платять великі гроші, щоб вони могли більш ефективно націлювати продажі. Якщо веб-браузер або JS дозволять вкрасти цю цінну особисту інформацію, деякі люди будуть красти її кожен раз, коли відвідувач переходить на їх сайти, і вона з'явиться на ринку всього через кілька секунд [19].

У кожної мови програмування є свої недоліки та вразливості. Однією з причин є популярність використання мови, тому що вона стає об'єктом підвищеного інтересу для хакерів, шахраїв та інших небажаних проявів сторонніх спеціалістів, які намагаються знайти уразливості і слабкі місця в безпеці.

Далі в цій роботі розглянуто деякі поширені проблеми безпеки, про які повинен пам'ятати кожен веб-розробник:

1. MCC. Це широко використовуваний метод, який дозволяє запускати зовнішній JS в контексті атакованого веб-сайту. MCC дозволяє отримати доступ до повного веб-API. Найпростіший приклад MCC-атаки реалізовується таким чином, що хакер знаходить вразливе поле введення на сторінці і створює посилання, яке вводить фрагмент коду на іншу сторінку. Після того, як посилання буде відкрито користувачем, хакер вирішить, що буде далі.

МСС – це вразливість системи безпеки з високим рейтингом, оскільки зловмисник може отримати доступ до LocalStorage, SessionStorage або файлів cookie. Ось чому не рекомендується зберігати в цих сховищах конфіденційні дані.

2. Міжсайтова підробка запиту (далі – МСПЗ). МСПЗ – це атака, яка використовує механізм відправки HTTP-запитів з браузера. Якщо на комп'ютері користувача зберігаються файли cookie з певного веб-сайту, ці файли cookie будуть відправлені разом із запитом, і не має значення, хто запускає даний запит. Таким чином хакер може вкрасти облікові записи ваших користувачів.

Простий приклад МСПЗ-атаки реалізується таким чином, що хакер знаходить незахищений елемент <form> на веб-сторінці. Потім може створити свою URL-адресу, яка викликає дію даної форми. Наприклад, він може оновити адресу електронної пошти користувача, запросити нагадування пароля і заволодіти обліковим записом.

3. Ін’єкція серверного JS. Одна з найбільш поширених вразливостей веб-додатків в Інтернеті в даний час. Досить часто розробник випадково вводить схильність до цього в веб-додаток шляхом простої неправильної настройки. Наприклад, функція eval може бути досить відкрита до атак і легко використана.

Якщо використовувати конкатенацію рядків несанкціонованого динамічного введення користувачем, треба бути готовим до високого ризику вразливостей. Крім того, якщо для серверної команди потрібне введення даних користувачем, то воно повинно бути правильно перевірено.

4. Вибір між автентифікацією на основі JSON веб-токенів (далі – JWS) і автентифікацією на основі сеансу. JWS – це метод безпечного представлення даних для передачі між двома або більше сторонами у вигляді JSON-об'єкта [20]. Він включає деякі дані на стороні користувача, наприклад, ім'я та адресу електронної пошти, разом з двома іншими значеннями: iat (видається в) і exp (закінчується в). Ці маркери підписані секретним ключем, але не можна використовувати будь-які конфіденційні дані в JWS, оскільки корисні дані не зашифровані. Кожен

токен складається з трьох частин в кодуванні base64, які описують використовуваний алгоритм, містять фактичне корисне навантаження і підпис. Цей метод зазвичай використовується для автентифікації в SPA і може спростити автентифікацію з використанням різних API, розміщених на різних піддоменів.

У більшості випадків JWT зберігаються в локальному сховищі або сховище сеансів. Такий підхід є потенційно вразливим для MCC-атак. Тому, якщо говорити про безпеку і автентифікацію, слід віддати перевагу автентифікації на основі сеансів. У такому випадку всі файли cookie будуть захищені від доступу JS за допомогою прапора HttpOnly. Крім того, файли cookie набагато менше, що може бути корисно в разі мобільних додатків без доступу до широкосмугового Інтернету. JWT можна використовувати в разі, якщо токени використовуються протягом короткого часу. Наприклад, для автентифікації сайту на субдомені або для завантаження файлів.

5. Безпека паролів

Паролі ніколи не повинні зберігатися у вигляді звичайного тексту. Крім того, треба використовувати сіль (модифікатор), рядок даних, який передається геш-функції разом з паролем. Без неї хакер може використовувати так звані райдужні таблиці, список всіх можливих перестановок відкритого тексту зашифрованих паролів, специфічних для даного алгоритму гешування, для зміни паролів. Такі функції, як Bcrypt, дозволяють додавати сіль у паролі.

Якщо говорити про обробку секретів, таких як паролі баз даних, ніколи не можна записувати їх в систему контролю версій у вигляді звичайного тексту. Щоб зберігати таку цінну інформацію в VCS, можа зашифрувати їх за допомогою таких додатків, як git-crypt або git-secret. Ще один зручний інструмент – Vault, він дозволяє зберігати і контролювати доступ до токенів, паролів та інших секретних даних.

Розробники JS, спираючись на існуючі проблеми, створили свої обмеження для потенційних авторів шкідливого коли. Крім того, розробники браузерів вносять додаткові обмеження у відповідь на що мають місце зловживання. У наступному пункті автор більш детально розкриває це питання.

1.3. JavaScript і безпека

Через відкритий характер Інтернету безпека є важливим питанням. Це особливо актуально з появою таких мов, як Java і JS, оскільки вони дозволяють вбудовувати виконуваний контент в статичні веб-сторінки. Оскільки завантаження веб-сторінки може призвести до виконання довільного коду на кінцевому комп'ютері, необхідні суворі заходи безпеки, щоб не допустити, щоб шкідливий код завдав якоїсь шкоди даними або конфіденційності користувача.

1.3.1. Перша та друга лінії захисту JavaScript

Перша лінія захисту JS від шкідливого коду полягає в тому, що мова просто не підтримує певні можливості. Наприклад, клієнтський JS не дозволяє записувати, видаляти файли або каталоги на клієнтському комп'ютері. Без об'єкта файлу та без функцій доступу до файлу програми JS не може видалити дані користувача або встановити віруси в системі користувача. Точно так же клієнтський JS не має мережевих примітивів будь-якого типу. Програма на JS може завантажувати URL-адреси і відправляти дані HTML-форм на веб-сервери, сценарії CGI і адреси електронної пошти, але не може встановити пряме з'єднання з будь-якими іншими хостами в мережі. Це означає, наприклад, що програма JS не може використовувати клієнтський комп'ютер в якості платформи атаки, з якої можна спробувати зламати паролі на іншій машині. Це було б особливо небезпечно, якби програма JS була завантажена з Інтернету через брандмауер і потім могла б спробувати проникнути у внутрішню мережу, захищену брандмауером.

Друга лінія захисту полягає в тому, що JS накладає обмеження на певні функції, які він підтримує. Наприклад, клієнтський JS підтримує метод close () для об'єкта Window, але більшість реалізацій веб-браузера обмежують цей метод, так що сценарій може закрити тільки вікно, відкрите сценарієм з того ж веб-сервера. Зокрема, сценарій не може закрити вікно, відкрите користувачем; якщо він спробує це зробити, користувачеві буде показано вікно підтвердження з питанням, чи дійсно він хоче закрити вікно. Нижче наводиться список інших обмежень

безпеки, що зустрічаються в більшості реалізацій клієнтського JS. Це не остаточний список. У кожного браузера може бути свій набір обмежень, і власні функції кожного браузера цілком можуть мати власні обмеження безпеки.

1. Об'єкт History спочатку був розроблений як масив URL-адрес, які представляють повну історію переглядів браузера. Однак, як тільки наслідки цього для конфіденційності стали очевидні, весь доступ до фактичних URL-адрес був обмежений, а об'єкту History залишилися тільки методи back (), forward () і go () для переміщення браузера по масиву історії без розкриття вмісту масиву.

2. Неможливо встановити властивість value об'єкта FileUpload. Якби це властивість могло бути встановлено, сценарій міг би встановити для нього будь-яке бажане ім'я файлу і змусити форму завантажувати вміст будь-якого зазначеного файлу (наприклад, файлу паролів) на сервер.

3. Сценарій не може відправити форму, наприклад, за допомогою методу submit () об'єкта Form, на адресу mailto: або news: без явного схвалення користувача в діалоговому вікні підтвердження. Така відправка форми буде містити адресу електронної пошти користувача, який не повинен публікуватися без отримання дозволу користувача.

4. Програма на JS не може закрити вікно браузера без підтвердження користувача, якщо вона сама не відкриє вікно. Це перешкоджає тому, щоб шкідливі сценарії викликали self.close (), щоб закрити вікно перегляду користувача, що призведе до завершення програми.

5. Сценарій не може відкрити вікно із стороною менш як 100 пікселів або змінити розмір вікна до менше 100 пікселів. Точно так же такий сценарій не може перемістити вікно за межі екрану або створити вікно, що перевищує розмір екрану. Це запобігає відкриттю сценаріями вікон, які користувач не бачить або може легко пропустити; такі вікна можуть містити сценарії, які продовжують виконуватись після того, як користувач думає, що вони зупинилися. Крім того, сценарій не може створювати вікно браузера без заголовка, тому що таке вікно може бути створено для підробки діалогового вікна операційної системи і, наприклад, обманом змусити користувача ввести конфіденційний пароль.

6. Сценарій не може змусити вікно або фрейм відображати URL-адресу about :, наприклад about: cache, тому що ці URL-адреси можуть розкривати системну інформацію, таку як вміст кеша браузера.

7. Сценарій не може встановлювати будь-які властивості об'єкта Event. Це запобігає підробку подій сценаріями. Сценарій не може реєструвати прослуховувачі подій або захоплювати події для документів, завантажених з джерел, відмінних від сценарію. Це запобігає відстеження сценаріями даних, що вводяться користувачем, таких як натискання клавіш, складові введення пароля, на інші сторінки.

1.3.2. Правило обмеженого домену (Політика того ж походження)

В JS є одне важливе обмеження безпеки, яке відомо як «Політика того ж походження» (далі – ПТП): сценарій може читати тільки властивості вікон і документів, що мають одне і те ж походження [21] (які були завантажені з того ж хоста, через той же порт і по тому же протоколу), що і сам скрипт. ПТП фактично не застосовується до всіх властивостей всіх об'єктів у вікні з іншого джерела, але може бути застосована до багатьох з них, зокрема, практично до всіх властивостей об'єкта Document. Призначені для користувача властивості об'єктів з різним походженням можуть бути обмежені, хоча це може варіюватися від реалізації до реалізації. ПТП – досить серйозне обмеження, але воно необхідне для запобігання крадіжки конфіденційної інформації скриптами. Без цього обмеження ненадійний сценарій (можливо, сценарій, завантажений через брандмауер в браузер в захищеній корпоративної інтрамережі) в одному вікні міг використовувати методи DOM для читання вмісту документів в інших вікнах браузера, які можуть містити особисту інформацію.

Проте, існують обставини, при яких ПТП є занадто суворою. Це створює особливі проблеми для великих веб-сайтів, які використовують більше одного сервера. Наприклад, скрипт з home.netscape.com може законно захотіти прочитати властивості документа, завантаженого з developer.netscape.com, або скриптам з orders.acme.com може знадобитися прочитати властивості з документів на catalog.acme.com. Для підтримки великих веб-сайтів такого типу в JS 1.1 було

введено властивість domain об'єкта Document. За замовчуванням властивість домену містить ім'я хоста сервера, з якого був завантажений документ. Можна встановити це властивість, але тільки для рядка, яка сама по собі є допустимим суфіксом домену. Таким чином, якщо домен спочатку є рядком «home.netscape.com», можна встановити для нього рядок «netscape.com», але не «home.netscape» або «cape.com», і вже тим більше не «microsoft. .com ". У значенні домену повинна бути хоча б одна точка, і не можна встановити для нього значення «com» або будь-який інший домен верхнього рівня.

Якщо два вікна (або фрейма) містять сценарії, які встановлюють для домену одне і те ж значення, то ПТП ослаблена для цих двох вікон, і кожне з вікон може зчитувати властивості іншого. Наприклад, спільні скрипти в документах, завантажених з orders.acme.com і catalog.acme.com, можуть встановити для своїх властивостей document.domain значення «acme.com», тим самим створюючи враження, що документи мають одне і те ж походження, і дозволяючи кожному документу читати властивості іншого.

1.3.3. Зони безпеки та підписані сценарії

Універсальна політика безпеки ніколи не буває повністю задовільною. Якщо політика занадто сувора, довірені скрипти не зможуть робити цікаві та корисні речі, які веб-розробники б хотіли від них. З іншого боку, якщо політика занадто дозвільної, ненадійні сценарії можуть викликати хаос. Ідеальне рішення – дозволити налаштувати політику безпеки таким чином, щоб довірені сценарії піддавалися меншій кількості обмежень безпеки, ніж ненадійні сценарії. Два основних постачальника браузерів, Microsoft і Netscape, використовували різні підходи до забезпечення безпеки; їх підходи коротко описані далі.

Microsoft Edge визначає «зони безпеки», в яких можна перерахувати веб-сайти, до сценаріям яких є довіра, і веб-сайти, до сценаріям яких немає. Потім можна налаштувати політику безпеки для цих двох зон окремо, надавши більше привілеїв і наклавши менше обмежень на довірені сайти. Також можна окремо налаштувати привілеї сайтів Інтернету і локальної мережі, які явно не вказані ні в одній з двох інших зон.

На жаль, це не повне або деталізоване рішення для безпеки JS, оскільки більшість параметрів безпеки, які дозволяє налаштовувати Microsoft Edge, не мають прямого відношення до JS. У бета-версії Microsoft Edge 6, наприклад, можна було вказати, чи дозволено сценаріям управляти об'єктами ActiveX і Java-аппле-тами і чи можуть вони виконувати операції вставки (як при вирізанні та вставлення). Не надається можливість, наприклад, відключити ПТП для надійного сайту або дозволити сценаріями з довірених сайтів відправляти повідомлення електронної пошти без підтвердження користувача.

Netscape 4 і Netscape 6 реалізують налаштування безпеку за допомогою підходу, відомого як «підписані сценарії». Підписані сценарії забезпечують повне детальне налаштування політик безпеки і роблять це криптографічно безпечним і теоретично дуже переконливим чином. На жаль, оскільки у Microsoft немає сумісної технології, процес створення підписаних сценаріїв є обтяжливим для авторів сценаріїв, а використання підписаних сценаріїв може збивати з пантелику кінцевих користувачів, використання цієї багатообіцяючої технології так і не прижилося.

Якщо коротко, підписаний сценарій має непідробний цифровий підпис, в якому вказується особа або організація, які написали сценарій або іншим чином прийняли на себе відповідальність за нього. Коли підписаним сценарієм необхідно обійти одне з описаних раніше обмежень безпеки, він спочатку запитує спеціальні «привілеї», які дозволяють йому це зробити. Коли сценарій запитує привілей, браузер підпорядковується користувачеві. Користувачеві повідомляють, хто підписав сценарій, і питають, чи хоче він надати запитаний привілей сценарію, написаним цією людиною або організацією. Як тільки користувач прийме рішення, він зможе запам'ятати його в браузері, щоб йому не задавали те ж питання в майбутньому. Фактично, ця процедура дозволяє користувачеві детально налаштовувати політику безпеки на льоту, коли виникає необхідність.

Процес створення підписаних сценаріїв кілька громіздкий. Крім того, деталі того, як це робиться, змінилися між Netscape 4 і Netscape 6. [22]

Створених обмежень не завжди буває достатньо, розвиток інформаційних технологій йде вгору, а хакери завжди шукають нові способи для своїх зловмисних дій. Так само важливу роль відіграє людський фактор, іноді сам автор сценарію може залишити слабке місце в своєму коді, яке потім зловмисники можуть використовувати в своїх цілях. Тому актуальним постає питання аналізу методів виявлення шкідливого JS коду на веб-сторінках.

1.4. Висновки до розділу

JS – це інтерпретуєма прототипно-орієнтована клієнтська сценарна мова програмування, що функціонує на основі подій; найпопулярніша мова програмування в Україні у 2020 році за результатами опитування сайта DOU; використовується, щоб додати інтерактивності та динамічності веб-сторінкам.

ES – специфікація створення сценарної мови загального призначення, яка кожен рік випускає оновлення та додає нові функції, які, в свою чергу реалізує JS. Не всі браузери одразу мають сумісність з новими версіями ES, тому що їм в свою чергу потрібно додати ці оновлення в свій функціонал.

До переваг JS відносять: швидкість, простоту, популярність, взаємодію, зниження навантаження на сервер, розширена функціональність, універсальність, оновлення. До недоліків – сумісність з браузером та безпека на стороні клієнта, тому що помилки та упущення в коді може буде використано в зловмисних цілях, таких як відстеження слів, що вводяться, відстеження звичок перегляду та впровадження шкідливого коду.

До поширених проблем безпеки входять MCC, МСПЗ, ін’єкція серверного JS, безпека паролів. Розробники JS, спираючись на існуючі проблеми, створили свої обмеження для потенційних авторів шкідливого коду. Крім того, розробники браузерів вносять додаткові обмеження у відповідь на місця зловживання. До них відносять першу лінію захисту (не підтримку певних можливостей), другу лінію захисту (накладання обмежень на існуючі можливості), ПТП (сценарій може читати тільки властивості вікон і документів, що мають одне і те ж походження) з боку JS; зони безпеки (налаштування безпеки для двох зон: сайтів,

до яких є довіра та сайтів, до яких нема) та підписані сценарії (накладання електронного підпису) – з боку браузера.

Але не завжди методи захисту JS та браузера є ефективними. Коли зловмиснику вдалось знайти слабке місце в коді та використати його для своєї мети, єдиним виходом, щоб застерегти користувача від пагубних наслідків є методи виявлення шкідливих JS-сценаріїв.

Розділ 2. ДОСЛІДЖЕННЯ АЛГОРИТМІВ ЗАХИСТУ ВІД ШКІДЛИВИХ JAVASCRIPT СЦЕНАРІЇВ

2.1. Загальні відомості про алгоритми класифікації виявлення

За способом виконання шкідливого сценарію методи виявлення діляться на методи статичного і динамічного аналізу. Метод статичного аналізу використовує статичні характеристики, тобто структуру сценаріїв, для виявлення шкідливих сценаріїв. Метод динамічного аналізу виявляє шкідливі сценарії спостерігаючи.

Сучасні підходи до виявлення шкідливих сторінок, як і інших типів шкідливих програм, засновані на пошуку функцій відомих загроз. Так, наприклад, антивірусні сканери, як правило, регулярно оновлюють свої бази, поповнюючи їх все новими функціями, приклад таких функцій показано в таблиці 2.1.

Таблиця 2.1

Список функцій відомих загроз

JS функція	Опис
eval()	Кількість функцій eval()
setTimeout()	Кількість функцій setTimeout()
iframe	Кількість строк, які містять iframe
unescape()	Кількість функцій unescape()
escape()	Кількість функцій escape()
classid	Кількість класів
parseInt()	Кількість функцій parseInt()
fromCharCode()	Кількість функцій fromCharCode()
ActiveXObject()	Кількість функцій ActiveXObject()
No. of string direct assignments	Кількість прямих назначених строк
concat()	Кількість функцій concat()
indexOf()	Кількість функцій indexOf()

substring()	Кількість функцій substring()
replace()	Кількість функцій replace()
document.addEventListener()	Кількість функцій document.addEventListener()
attachEvent()	Кількість функцій attachEvent()
createElement()	Кількість функцій createElement()
getElementById()	Кількість функцій getElementById()
document.write()	Кількість функцій document.write()
JavaScript word count	Кількість слів
JavaScript Keywords	Кількість ключових слів
No. of characters in JavaScript	Кількість символів
The ratio between keywords and words	Співвідношення між словами і словами
Entropy of JavaScript	Ентропія
Length of Longest JavaScript Word	Довжина найдовшого слова
The No. of Long Strings >200	Кількість довгих строк більше 200
Length of shortest JavaScript Word	Довжина найкоротшого слова
Entropy of the Longest JavaScript Word	Ентропія найдовшого слова
No. of Blank Spaces	Кількість пробілів
Average Length of Words	Середня довжина слів
No. Hex Values	Кількість шістнадцятковий значень
Share of space characters	Частка пробілів

Даний підхід має істотні недоліки:

1. Простота обходу функції;
2. Необхідність в регулярному і досить частому оновленні;
3. Мала ефективність виявлення нових типів атак і шкідливих програм.

На додаток до функцій в сучасних антивірусних сканерах представлені евристичні алгоритми. Вони здатні виявляти підозрілі елементи в коді і, таким чином, виявляти деякі нові типи атак, проте дають велику кількість помилкових спрацьовувань.

Останнім часом набирає популярність підхід програм на основі динамічного аналізу - використання так званих пісочниць. Стосовно до проблеми виявлення driveby-download атак, пісочниця може бути реалізована у вигляді віртуальної машини, зі встановленою на ній повноцінною операційною системою (наприклад, Windows XP). У віртуальному середовищі запускається браузер, який відвідує сторінку, що перевіряється. Через деякий час відбувається перевірка системи на наявність нових процесів, файлів і ключів реєстру. Якщо такі виявляються, то це розцінюється як факт атаки зі сторінки. Такий підхід є досить ефективним, однак і він має ряд недоліків:

1. Перевірка мільйонів сторінок таким способом є витратною по ресурсам і за часом;
2. Існують так звані нерезидентні шкідливі програми, що не створюють нових файлів і процесів, а лише виконують ряд викликів в контексті браузера і самознищується;
3. Існують атаки, орієнтовані на певний браузер, тому при такому підході процес аналізу однієї сторінки вимагає її відвідування з декількох браузерів.

У зв'язку з вищевикладеними проблемами виникає необхідність дослідження потенціалу алгоритмів машинного навчання в рішенні задач комп'ютерної безпеки.

2.1.1. Наївний баєсів класифікатор

Наївний баєсів класифікатор заснований на теоремі Баєса (див. Формулу 2.1). Наївну баєсову модель побудувати легко і швидко. Це особливо корисно для дуже великих наборів даних. Модель може бути змінена з використанням нових навчальних даних без необхідності перебудовувати модель.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \quad (2.1)$$

де B - доказ,

A - гіпотеза.

Тут зроблено припущення, що предиктори / ознаки незалежні. Використовуючи теорему Баєса, ми можемо знайти ймовірність того, що A відбудеться, враховуючи, що B відбулося. [23] Тобто наявність однієї конкретної ознаки не впливає на іншу. Звідси його називають наївним.

Типи наївного баєсова класифікатора:

1. Багаточленний наївний баєсів класифікатор

В основному використовується для проблеми класифікації документів, тобто, чи належить документ до категорії спорту, політики, технологій тощо. Особливостями / предикторами, що використовуються класифікатором, є частота слів, присутніх у документі.

2. Наївний баєсів класифікатор Бернуллі

Схожий на багаточленний, але предиктори - це логічні змінні. Параметри, які використовуються для прогнозування змінної класу, приймають лише значення так чи ні, наприклад, якщо слово зустрічається в тексті чи ні.

3. Гауссів наївний баєсів класифікатор

Коли предиктори приймають безперервне значення і не є дискретними, припускається, що ці значення відбираються з гауссового розподілу (див. Рис. 2.1).

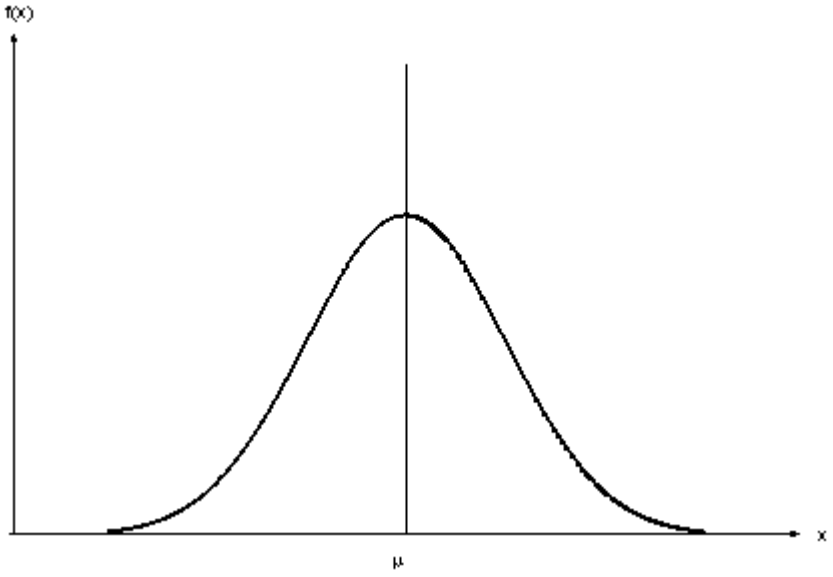

Рис 2.1. Крива гауссового розподілу

Оскільки спосіб значень у наборі даних змінюється, формула умовної ймовірності (Формула 2.1) змінюється на Формулу 2.2.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\pi\sigma_y^2}\right), \quad (2.2)$$

де y – змінна класу,

x – змінна параметрів/ознак,

σ - середньоквадратичне відхилення,

μ - математичне очікування (середнє значення), медіана і мода розподілу.

Наївні алгоритми Байєса в основному використовуються в фільтрації спаму, системах рекомендацій тощо. Вони швидкі та прості у впровадженні, але найбільшим їх недоліком є те, що предиктори повинні бути незалежними. У більшості випадків реального життя предиктори залежні, це заважає роботі класифікатора.

2.1.2. J48 Дерево рішень

Класифікатор J48 - це дерево рішень C4.5 для класифікації. Він створює двійкове дерево (див. Рис. 2.2). Підхід до дерева рішень є найбільш корисним у проблемі класифікації. За допомогою цієї техніки будується дерево для моделювання процесу класифікації. Після побудови дерева воно застосовується до кожного кортежу в базі даних і призводить до класифікації для цього кортежу. [24]

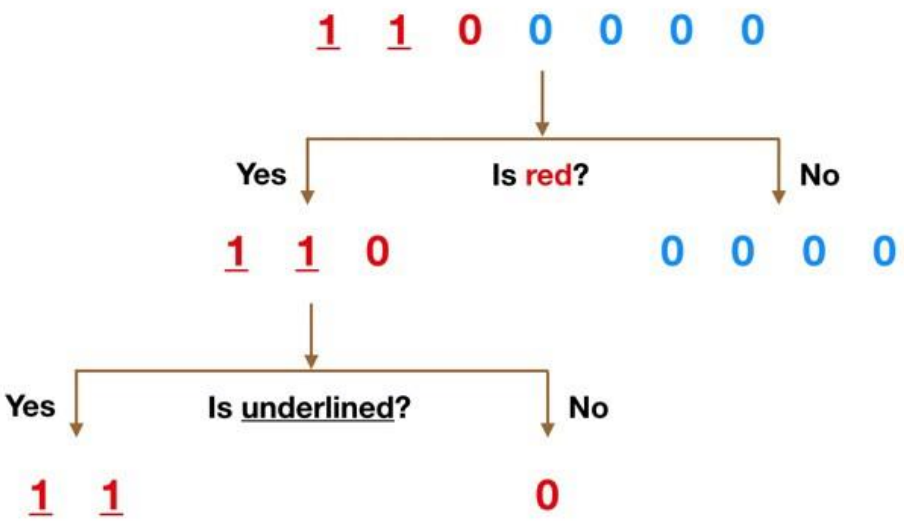

Рис. 2.2. Приклад простого дерева рішень

У навчальній вибірці кількість прикладів має бути значно більше кількості класів, до того ж кожен приклад повинен бути заздалегідь асоційований зі своїм класом. З цієї причини C4.5 є варіантом машинного навчання з учителем.

Побудова дерева відбувається наступним чином: нехай є *T* - навчальна вибірка прикладів, а *C* - безліч класів, що складається з *k* елементів. Для кожного прикладу з *T* відома його приналежність до якої-небудь з класів $C_1 \dots C_k$.

На першому кроці є корінь і асоційоване з ним безліч *T*, яке необхідно розбити на підмножини. Для цього необхідно вибрати один з атрибутів в якості перевірки. Обраний атрибут A має *n* значень, що дає розбиття на *n* підмножин. Далі створюються *n* нащадків кореня, кожному з яких поставлено у відповідність своя підмножина, отримана при розбитті *T*. Процедура вибору атрибута і розбиття по ньому рекурсивно застосовується до всіх *n* нащадкам і зупиняється в двох випадках:

1. Після чергового розгалуження в вершині виявляються приклади з одного класу (тоді вона стає листом, а клас, якому належать її приклади, буде рішенням листа).

2. Вершина виявилася асоційованої з порожнію множиною (тоді вона стає листом, а в якості рішення вибирається клас, який найчастіше зустрічається у предка цієї вершини).

У дереві рішень основною проблемою є ідентифікація атрибута для кореневого вузла на кожному рівні. Цей процес відомий як вибір атрибута, є два популярні заходи вибору атрибутів:

1. Отримання інформації
2. Індекс Джині

Коли використовується вузол в дереві рішень для поділу навчальних примірників на більш дрібні підмножини, ентропія змінюється. Приріст інформації є мірою цієї зміни ентропії.

$$Gain(S, A) = Entropy(S) - \sum v \in Values(A) \frac{|S_v|}{|S|} \times Entropy(S_v), \quad (2.3)$$

де S - набір екземплярів,

A - атрибут,

S_v - підмножина S з A = v,

Values (A) набір всіх можливих значень A.

Ентропія - це міра невизначеності випадкової величини, вона характеризує нечистоту довільного набору прикладів. Чим вище ентропія, тим більше інформативність.

Індекс Джині - це показник для вимірювання того, як часто випадково обраний елемент буде неправильно ідентифікований. Це означає, що перевагу слід віддавати атрибуту з більш низьким індексом Джині.

$$GiniIndex = 1 - \sum_j p_j^2 \quad (2.4)$$

2.1.3. Випадковий ліс

Випадковий ліс, як випливає з назви, складається з великої кількості окремих дерев рішень, які діють як ансамбль. Кожне окреме дерево в випадковому лісі дає передбачення класу, і клас з найбільшою кількістю голосів стає прогнозом моделі (див. Рис. 2.3).

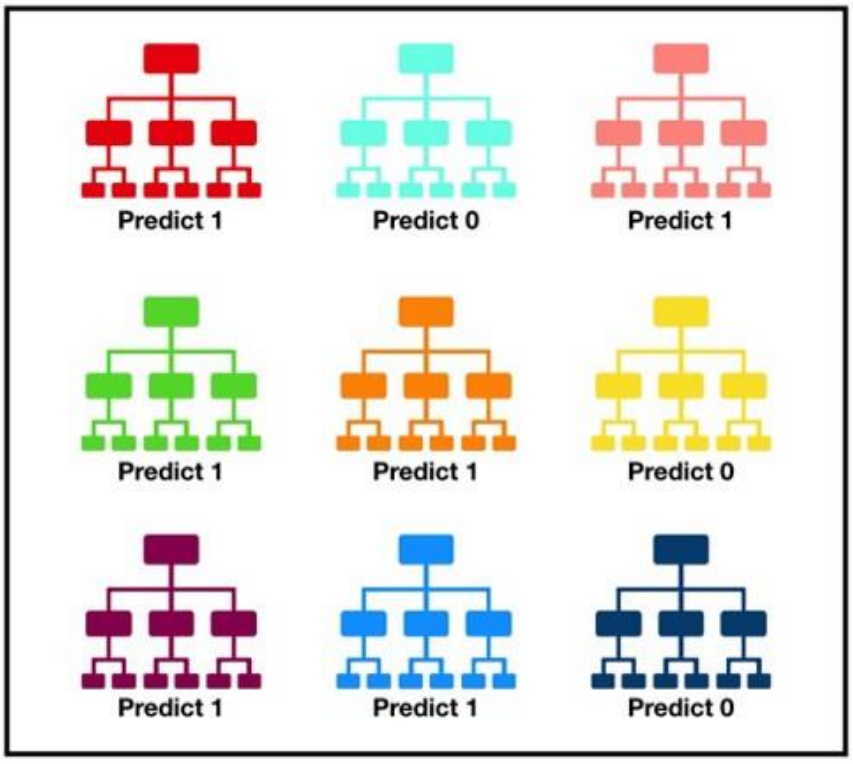

Рис. 2.3. Візуалізація моделі випадкового лісу для прогнозування

Основним поняттям, що стоїть за випадковим лісом, є проста, але потужна концепція - мудрість натовпу. У науці даних говорять, причина того, що випадкова лісова модель працює так добре, полягає в тому, що велика кількість відносно некорельованих моделей (дерев), що діють як комітет, перевершить будь-яку з окремих складових моделей. [25]

Низька кореляція між моделями є ключовим фактором. Подібно до того, як інвестиції з низькою кореляцією (наприклад, акції та облігації) об'єднуються, щоб сформувати портфель, який перевищує суму їх частин, некорельовані моделі можуть створювати ансамблеві прогнози, які є більш точними, ніж будь-які окремі прогнози. Причиною цього чудового ефекту є те, що дерева захищають одне одного від своїх індивідуальних помилок (якщо вони постійно не всі помиляються в одному напрямку). Хоча деякі дерева можуть помилятися, багато інших дерева матимуть рацію, тому як група дерев можуть рухатись у правильному напрямку. Отже, передумовами того, щоб довільний ліс добре працював, є:

1. У функціях повинен бути якийсь фактичний сигнал, щоб моделі, побудовані з використанням цих функцій, спрацьовували краще, ніж випадкові вгадування.

2. Прогнози (і, отже, помилки), зроблені окремими деревами, повинні мати низьку кореляцію між собою.

2.1.4. Метод опорних векторів

Метод опорних векторів (далі – МОВ) – це модель машинного навчання з вчителем, яка використовує алгоритми класифікації для класифікаційних задач двох груп. Після представлення моделі МОВ наборів навчальних даних для кожної категорії вони можуть класифікувати новий текст.

Завданням МОВ є пошук гіперплощини в N-мірному просторі (N - кількість ознак), яка чітко класифікує точки даних.

Для розділення двох класів точок даних існує безліч можливих гіперплощин, які можна вибрати (див. Рис. 2.4). Мета - знайти площину, яка має максимальний запас, тобто максимальну відстань між точками даних обох класів. Максимізація відстані поля забезпечує певне підкріплення, щоб майбутні точки даних можна було класифікувати з більшою впевненістю.

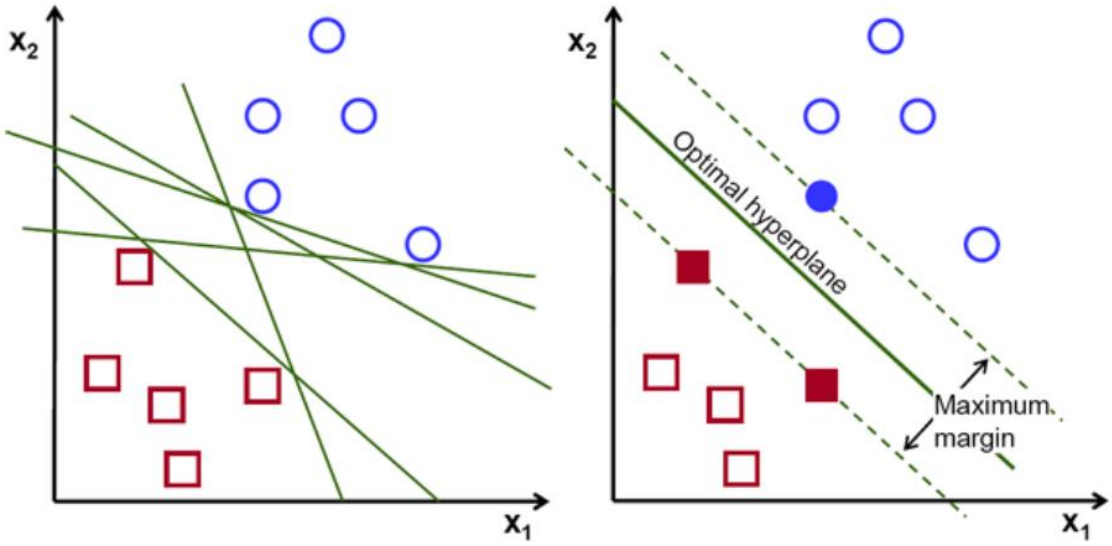

Рис. 2.4. Можливі гіперплощини

Гіперплощини - це межі прийняття рішень, які допомагають класифікувати точки даних (див. Формулу 2.5). Точки даних, що потрапляють по обидві сторони гіперплощини, можна віднести до різних класів. Крім того, розмір гіперплощини залежить від кількості ознак. Якщо кількість вхідних функцій дорівнює 2, тоді гіперплощина - це просто пряма. Якщо кількість вхідних ознак дорівнює 3, тоді гіперплощина стає двовимірною площиною.

$$w \times x - b = 0, \quad (2.5)$$

де w – перпендикуляр, що розділяє гіперплощини,

x – об’єкт даних.

Опорні вектори - це точки даних, які знаходяться ближче до гіперплощини і впливають на положення та орієнтацію гіперплощини. Використовуючи ці вектори, максимізується запас класифікатора. Видалення векторів опори змінить положення гіперплощини. Це пункти, які допомагають будувати МОВ.

В логістичній регресії береться вихід лінійної функції і стискається значення в межах [0,1] за допомогою сигмовидної функції. Якщо зведене значення

більше порогового значення (0,5), присвоюється йому мітка 1, інакше - мітку 0. У МОВ береться вихід лінійної функції, і якщо цей вихід більший за 1, то він ідентифікується це з одним класом, і якщо результат -1 - з іншим класом. Оскільки порогові значення змінені на 1 та -1 у МОВ, отримується цей діапазон значень [- 1,1], які діють як грані.

2.1.5. AdaBoost

AdaBoost, скорочення від "Adaptive Boosting", є першим практичним алгоритмом підвищення, запропонованим Фройндом і Шапері в 1996 році. Він фокусується на проблемах класифікації і має на меті перетворити набір слабких класифікаторів у сильний. Остаточне рівняння для класифікації можна представити у вигляді Формули 2.4.

$$F(x) = sign(\sum_{m=1}^{M} \theta_m f_m(x)), \text{ (2.6)}$$

де f_m – m-ий класифікатор,

θ_m – відповідна вага

Дано набір даних, що містить n точок, де $x_i \in R^d, y_i \in \{-1; 1\}$. Тут -1 позначає негативний клас, тоді як 1 представляє позитивний.

Ініціалізується вага для кожної точки даних як:

$$w(x_i, y_i) = \frac{1}{m}, m = 1 \dots M \text{ (2.7)}$$

Встановлюються слабкі класифікатори до набору даних і вибирається той, що має найменшу зважену помилку класифікації:

$$\epsilon_m = E_{w_m}[1_{y \neq f(x)}] \text{ (2.8)}$$

Обчислюється вага для m-го слабкого класифікатора:

$$\theta_m = \frac{1}{2} \ln \left(\frac{1-\epsilon_m}{\epsilon_m}\right) \text{ (2.9)}$$

Для будь-якого класифікатора з точністю вище 50% вага позитивна. Чим точніше класифікатор, тим більше вага. У той час як для класифікатора з точністю менше 50% вага негативна. Значить, комбінується його пророкування, перевертаючи знак. Наприклад, можна перетворити класифікатор з точністю 40% в точність 60%, змінивши знак прогнозу. Таким чином, навіть класифікатор працює гірше, ніж випадкове припущення, він все одно вносить свій внесок в

остаточний прогноз. Тільки не потрібен якийсь класифікатор з точністю 50%, що не додає ніякої інформації і, таким чином, не впливає на остаточний прогноз.

Оновлення вага для кожної точки даних виглядає як:

$$w_{m+1}(x_i, y_i) = \frac{w_m(x_i, y_i)\exp\left[-\theta_m y_i f_m(x_i)\right]}{Z_m}, \quad (2.10)$$

де Z_m - коефіцієнт нормалізації, який гарантує, що сума всіх ваг прикладів дорівнює 1.

Якщо неправильно класифікований випадок отриманий з класифікатора з позитивним зважуванням, член exp в чисельнику завжди буде більше 1 ($y \times f$ завжди дорівнює -1, θ_m позитивна). Таким чином, неправильно класифіковані випадки будуть оновлюватися з більш високими вагами після ітерації. Та ж логіка застосовна до класифікаторів з негативним вагою. Єдина відмінність полягає в тому, що вихідні правильні класифікації стануть помилковими після зміни знака.

Після M ітерації отримується остаточний прогноз, підсумовуючи зважений прогноз кожного класифікатора.

2.1.6. REPTree

REPTree (Reduced Error Pruning Tree - скорочення числа помилок) - ще один алгоритм, специфічний для Weka. Це засіб швидкого вивчення дерева рішень, оптимізоване для простоти і швидкості. Алгоритми використовують скорочення з зменшеним числом помилок з повторною підгонкою, щоб знайти найменше уявлення найбільш точного піддерева по відношенню до набору скорочення. [26]

Алгоритм REPTree будує бінарні дерева для задач класифікації і регресії, використовуючи відповідно ентропійний і статистичний критерії розгалуження.

Weka практично не надає будь-яких технічних подробиць REPTrees. Таким чином, досить складно провести точну межу між REPTree і J48 зі скороченням кількості помилок. Властивості, що надаються Weka, зазвичай дуже схожі на J48. Точніше, пропущені значення і числові атрибути обробляються так само, як в

C4.5. Крім того, можна вказати мінімальну кількість примірників на лист, максимальну глибину дерева і мінімальну дисперсію навчального набору для поділу (для числових класів).

2.1.7. ADTree

Модель ADTree (Alternating Decision Tree) була побудована шляхом поділу даних, в якому дані кожної ітерації були розділені відповідно до значень атрибутів. Таким чином, основна мета цього аналізу - розділити дані на підмножини, якщо тільки підмножина не містить однорідне цільове значення або передбачуваний атрибут. У кожному розбитті перевіряється вплив обраних змінних на передбачуваний атрибут. Якщо передбачуваний атрибут містить дискретні дані, результуюча деревоподібна модель називається деревом класифікації. Цей процес дерева рішень також називається індукцією дерева рішень. Вхідні дані навчального набору поділяються на вузол прогнозування з використанням спліт-тестів для отримання значень вузла прогнозування. [27]

Дерево рішень, що чередується, складається з вузлів рішень і вузлів прогнозування. Вузли прийняття рішень визначають умова предиката. Вузли прогнозу містять одне число. ADTrees завжди мають вузли прогнозування як кореневі, так і кінцеві. Примірник класифікується за допомогою ADTree, дотримуючись всіх шляхах, для яких все вузли рішення істинні, і підсумовуючи все пройдені вузли прогнозування. Це відрізняється від довічних дерев класифікації, таких як CART (дерево класифікації і регресії) або C4.5, в яких екземпляр слід тільки по одному шляху через дерево.

2.2. Проблеми алгоритмів виявлення

Навіть використовуючи алгоритми виявлення шкідливих сценаріїв неможливо бути повністю впевненим у безпеку веб-сторінки. Кожен алгоритм може давати як хибно позитивний результат, так і хибно негативний, виною цьому є слабкі місця класифікатора.

Хоча Наївний Баєсів класифікатор є дуже швидким і простим класифікатором, але є деякі недоліки, які можуть погіршити його роботу:

1. Передбачається, що атрибути незалежні. Це слабке місце можна вирішити, виконавши певний статистичний аналіз перед використанням наївного баєсова класифікатора для вимірювання ступеня кореляції між функціями, а потім вибравши найбільш некорельовані.

2. Він однаково ставиться до всіх атрибутів. Таким чином, деякі ваги можуть бути додані до важливих атрибутів, щоб збільшити їх вклад в остаточне рішення.

3. Проблема нульової ймовірності. Ця проблема може бути вирішена або додаванням значення один до частоти кожного атрибута, або використанням методу розподілу Гаусса.

4. Проблема з постійним значенням атрибута. Перетворення безперервного значення в дискретне є вирішенням такої проблеми.

Дерево рішень - один з найкорисніших алгоритмів машинного навчання, також має свої недоліки:

1. Для дерева рішень іноді обчислення можуть бути набагато складнішими у порівнянні з іншими алгоритмами. [28]

2. Дерево рішень часто вимагає більшого часу для навчання моделі.

3. Навчання дерева рішень є відносно дорогим, оскільки на нього йде більше складності і часу.

4. Алгоритм дерева рішень не підходить для застосування регресії і прогнозування безперервних значень.

Недоліки алгоритму Випадковий Ліс:

1. Хоча випадковий ліс може бути поліпшенням у порівнянні з одиночними деревами рішень, доступні більш складні методи. Точність прогнозування складних завдань зазвичай нижче, ніж у дерев з градієнтним посиленням.

2. Ліс важче інтерпретувати, ніж одне дерево рішень. Поодинокі дерева можна уявити як послідовність рішень.

3. Навченому лісу може знадобитися значний обсяг пам'яті для зберігання через необхідність зберігати інформацію з декількох сотень окремих дерев. [29]

Недоліки методу опорних векторів:

1. Алгоритм не підходить для великих наборів даних.

2. Не дуже добре працює, коли в наборі даних більше шуму, тобто цільові класи перекриваються.

3. У випадках, коли кількість функцій для кожної точки даних перевищує кількість вибірок навчальних даних, метод опорних методів буде працювати гірше.

4. Оскільки класифікатор опорних векторів працює, поміщаючи точки даних вище і нижче класифікуючої гіперплощини, імовірнісного пояснення класифікації немає. [30]

Кілька недоліків AdaBoost:

1. Техніка підвищення кваліфікації поступово навчається, тому важливо переконатися, що у алгоритм навчається на якісних даних.

2. AdaBoost також надзвичайно чутливий до гучних даних з шумом і, тому, якщо дійсно планується використовувати AdaBoost, настійно рекомендується їх усунути.

3. Також було доведено, що AdaBoost працює повільніше, ніж інші алгоритми. [31]

2.3. Порівняння ефективності алгоритмів

У експерименті було використано Weka API для всіх семі класифікаторів машинного навчання. Щоб вибрати найбільш ефективний класифікатор, застосовано матрицю неточностей (див. Таблиця 2.2), що містить фактичні та прогнозовані результати, виконані алгоритмом класифікацій.

Таблиця 2.2

Матриця неточностей

		Передбачений результат	
		Шкідливий	Доброякісний
Фактичний результат	Шкідливий	a	b
	Доброякісний	c	d

Щоб порівняти дію алгоритмів, їх ефективність у виявленні шкідливих сценаріїв було обрано 4 ознаки: точність, хибно позитивний результат, хибно негативний результат і крива помилок.

Точність - це частка від загального числа вірних прогнозів, визначається за допомогою рівняння:

$$\mathrm{A} = \frac{(a+d)}{(a+b+c+d)} \quad (2.11)$$

Хибно позитивний результат - це частка безпечних JS, які були неправильно класифіковані як шкідливі і розраховані з використанням рівняння:

$$FRP = \frac{c}{c+d} \quad (2.12)$$

Хибно негативний результат (FNR) - це частка шкідливих кодів JS, які були неправильно класифіковані як безпечні і розраховані з використанням рівняння:

$$FRP = \frac{b}{a+b} \quad (2.13)$$

Крива помилок - графічна характеристика якості бінарного класифікатора, залежність частки вірних позитивних класифікацій від частки помилкових позитивних класифікацій, визначається за формулою:

$$C = \frac{d}{b} \quad (2.14)$$

Докладний аналіз продуктивності класифікаторів машинного навчання по наборі даних JS з Таблиці 2.1 представлені нижче.

Таблиця 2.3

Аналіз продуктивності алгоритмів по набору з Таблиці 2.1

Класифікатор	Точність (%)	Хибно позитивний результат	Хибно негативний результат	Крива помилок
Наївний Баєсів	95.48	0.063	0.009	0.995
J48 Дерево рішень	99.67	0.005	0.000	0.998

Випадковий ліс	99.76	0.004	0.000	1.000
Метод опорних векторів	99.70	0.003	0.004	0.997
AdaBoost	99.70	0.004	0.000	1.000
REPTree	99.67	0.005	0.000	0.998
ADTree	99.91	0.001	0.000	1.000

По Таблиці 2.3 видно, що найкращий результат показує алгоритм ADTree з максимальним показником точності 99,91% та мінімальними хибно позитивними (0,001) та негативними (0,000) результатами. Найгірші результати у наївного баєсова класифікатора з точністю 95,48% та максимальними хибно позитивними (0,063) та негативними (0,009) випадками.

Щоб перевірити як змінюється точність статичного алгоритму при використанні іншого набору даних, береться новий набор функцій з Таблиці 2.4, а результати показані в Таблиці 2.5.

Таблиця 2.4

Список нового набору функцій загроз

JS функція	Опис
search()	Кількість функцій search ()
split()	Кількість функцій split ()
onbeforeunload	Кількість подій onbeforeunload
onload	Кількість подій onload
onerror()	Кількість функцій onerror ()
onunload	Кількість подій onunload
onbeforeload	Кількість подій onbeforeload
onmouseover	Кількість подій onmouseover
dispatchEvent	Кількість подій dispatchEvent
fireEvent	Кількість подій fireEvent
setAttribute()	Кількість функцій setAttribute ()

window.location()	Кількість функцій window.location ()
charAt()	Кількість функцій charAt ()
console.log()	Кількість функцій console.log ()
.js	Кількість зовнішніх файлів JS
.php	Кількість файлів .php
var	Кількість ключових слів var, використовуваних в JS
function	Кількість ключових слів функцій, використовуваних в JS
Math.random()	Кількість функцій Math.random ()
charCodeAt()	Кількість функцій charCodeAt ()
WScript	Кількість WScript, використовуваних в JS
decode()	Кількість функцій decode ()
toString()	Кількість функцій toString ()
No. of Digits	Кількість цифр, що використовуються в JS
No. of Encoded Characters	Кількість закодованих символів, використовуваних в JS
No. of backslash Characters	Кількість символів /, використовуваних в JS
No. of Pipe Characters	Кількість символів вертикальної риси (|), які використовуються в JS
No. of % Characters	Кількість символів %, використовуваних в JS
No. of ‘(‘ Characters	Кількість символів '(', використовуваних в JS
No. of ‘)’ Characters	Кількість символів ')', які використовуються в JS
No. of ‘,’ Characters	Кількість символів ',', використовуваних в JS
No. of ‘#’ Characters	Кількість символів '#', використовуваних в JS
No. of ‘+’ Characters	Кількість символів '+', використовуваних в JS
No. of ‘.’ Characters	Кількість '.' символів, які використовуються в JS
No. of ‘ Characters	Кількість символів ', використовуваних в JS
No. of ‘[‘ Characters	Кількість символів '[', використовуваних в JS

No. of ‘]’ Characters	Кількість символів ']', використовуваних в JS
No. of ‘{‘ Characters	Кількість символів '{', використовуваних в JS
No. of ‘}’ Characters	Кількість '}' символи, використовувані в JS
Share of Encoded characters	Частка закодованих символів в JS
Share of Digits characters	Доля десяткових знаків
Share of Hex/Octal characters	Доля шістнадцятирічних / вісімкових знаків
Share of Backslash characters	Доля / знаків
Share of Pipe (|) characters	Доля | знаків
Share of % characters	Доля % знаків

Таблиця 2.5

Аналіз продуктивності алгоритмів по набору з Таблиці 2.3

Класифікатор	Точність (%)
Наївний Баєсів	79.45
J48 Дерево рішень	96.82
Випадковий ліс	95.51
Метод опорних векторів	93.37
AdaBoost	99.79
REPTree	96.04
ADTree	95.66

Як видно з результатів в Таблиці 2.4 лідируючу позицію займає AdaBoost з показником точності 99,79%. Найгірший результат має той ж наївний баєсів класифікатор з точністю 79,45%.

Для покращення результатів точності класифікаторів використано обидва набори даних (Табл. 2.5)

Таблиця 2.6

Аналіз продуктивності алгоритмів з наборами Таблиці 2.1 та Таблиці 2.4

Класифікатор	Точність	Хибнопозитивний результат	Хибнонегативний результат	Крива помилок
Наївний Баєсів	97.56	0.030	0.013	0.994
J48 Дерево рішень	99.82	0.003	0.000	0.998
Випадковий ліс	99.85	0.002	0.000	1.000
Метод опорних векторів	99.91	0.001	0.001	1.000
AdaBoost	99.79	0.003	0.000	1.000
REPTree	99.67	0.005	0.000	0.998
ADTree	99.79	0.003	0.000	1.000

Використовуючи нові 45 функцій, всі класифікатори забезпечують високу точність виявлення. Найгірший результат знов показує наївний баєсів класифікатор з найменшим показником точності 97,56% і найбільшою кількістю хибних результатів.

2.4. Висновки до розділу

Методи виявлення шкідливих JS сценаріїв поділяються на статистичні та динамічні. Статистичний аналіз працює в режимі реального часу; динамічний – спостерігаючи, що призводить до більшої витрати часу та складності в реалізації.

Сучасні підходи до виявлення шкідливих сторінок, як і інших типів шкідливих програм, засновані на пошуку функцій відомих загроз. Найпоширенішими алгоритмами для виявлення шкідливих сценаріїв є Наївний Баєсів, Дерево Рішень, Випадковий Ліс, Метод опорних векторів, AdaBoost, REPTree, ADTree. Основними недоліками більш продуктивних алгоритмів є використання великого обсягу пам’яті та швидкість роботи.

Щоб вибрати найбільш ефективний класифікатор, застосовано матрицю неточностей, що містить фактичні та прогнозовані результати, виконані алгоритмом класифікацій. За допомогою матриці було обрано 4 ознаки порівняння алгоритмів: точність, хибно позитивний результат, хибно негативний результат і крива помилок.

За результатами порівняння зроблено висновок, що всі алгоритми працюють з майже однаковою точністю, їх показники за ознаками варіюються в залежності від набора функцій шкідливих сценаріїв (експериментально було обрано 2 набори), якщо в першому випадку лідером був ADTree, то в другому – метод опорних векторів. Найгірші результати показав Наївний баєсів відстаючи на 2-3% точності в обох випадках.

Алгоритмом для удосконалення було обрано алгоритм дерева рішень. В обох випадках алгоритм показував одні з кращих результатів по ознакам. При його використанні немає необхідності нормалізувати та масштабувати дані. Етап попередньої обробки даних для дерев рішень вимагає менше коду, часу і аналізу. Концепція, що лежить в основі дерева рішень, більш відома програмістам і порівняно легше для розуміння, ніж інші подібні алгоритми.

Розділ 3. УДОСКОНАЛЕННЯ АЛГОРИТМУ ВИЯВЛЕННЯ JAVASCRIPT СЦЕНАРІЇВ

3.1. Опис середовища розробки

Мова C #, розроблена компанією Майкрософт, одна з найпопулярніших сучасних мов програмування. Вона затребувана на ринку розробки в різних країнах, C # застосовують при роботі з програмами для ПК, створення складних веб-сервісів або мобільних додатків. З'явилась як мова для власних потреб платформи Microsoft .NET, поступово ця мова стала дуже популярною. Отже, розробка мови почалася в 1998 році, а перша версія побачила світ в 2001. Групою розробників керував відомий в професійних колах фахівець Андерс Хейлсберг. Нові версії C # виходять порівняно часто, а поточні доопрацювання, виправлення багів і розширення бібліотек ведуться практично на постійній основі.

В результаті мова вийшла вкрай гнучкою, потужною і універсальною. На ній пишуть практично все, що завгодно, від невеликих веб-додатків до потужних програмних систем, які об'єднують в собі веб-структури, додатки для десктопів і мобільних пристроїв. Все це стало можливим завдяки зручному Сі-подібному синтаксису, суворому структурування, величезній кількості фреймворків і бібліотек (їх число досягає декількох сотень).

Довгий час платформа .NET поставлялася з закритим ядром, що створювало певні труднощі в розробці і знижувало популярність C # в професійному середовищі. Але в листопаді 2014 Майкрософт радикально змінила підхід і стала видавати безкоштовні ліцензії для Visual Studio вже з відкритим вихідним кодом для всіх наборів інструментів.

C# - дійсно цікавий інструмент, гідний уваги. Він впевнено займає високі позиції в рейтингах затребуваних мов програмування на ринку праці. Тому має сенс вивчити його можливості докладніше і зрозуміти, для чого і де варто застосовувати C #.

Компанія Microsoft приділяє значну увагу підтримці мови розробки, а тому регулярно з'являються оновлення та доповнення, виправляються виявлені баги в компіляторі, розширюються бібліотеки. Розробники зацікавлені в популяризації інструменту і докладають до цього масу зусиль.

Розробники надають докладну і розгорнуту документацію на своїх офіційних ресурсах. Крім того, відповіді практично на будь-які питання, пов'язані з роботою в C #, можна знайти в мережі. Популярність мови привела до появи безлічі професійних співтовариств, присвячених C#. Існує безліч підручників, курсів для новачків і мідл, відео добірок та інших навчальних матеріалів.

Інструментарій C # дозволяє вирішувати широке коло завдань, мова дійсно дуже потужна і універсальна. На ній розробляють:

1. Додатки для WEB;
2. Різні ігрові програми;
3. Додатки платформ Андроїд або iOS;
4. Програми для Windows.

Перелік можливостей розробки практично не має обмежень завдяки найширшому набору інструментів і засобів. Звичайно, все це можна реалізувати за допомогою інших мов, але деякі з них є вузькоспеціалізованими, в інших доведеться використовувати додаткові інструменти сторонніх розробників. У C # рішення широкого кола завдань можливі швидше, простіше і з меншими витратами часу і ресурсів.

Дозволяє в автоматичному режимі очистити пам'ять від об'єктів, які не використовуються, або знищених додатків.

За допомогою цього інструменту можна легко виявляти і обробляти помилки в коді. Спосіб є структурованим з широким набором функцій. При цьому важливо не зловживати можливостями роботи з винятками, так як при неправильному використанні з'являється ризик появи «багів».

У мові прийнята загальна система роботи з типами, починаючи від примітивів і закінчуючи складними, в тому числі, призначеними для користувача, наборами. Застосовується єдиний набір операцій для обробки і зберігання

значень типізації. Також можна використовувати посилальні типи користувача, що дозволить динамічно виділити пам'ять під об'єкту або зберігати спрощену структуру в мережі.

Мова програмування забороняє звернення до змінних, що не були ініційовані, що виключає можливість виконання безконтрольного приведення типів або виходу за межі певного масиву даних.

Керування версіями. Дуже цікава особливість мови програмування. Суть в тому, що багато мов не приділяють належної уваги цьому питанню, і програми нерідко перестають коректно працювати при переході на нову версію продукту. У C # це було виправлено.

Архітектура платформи. Для використання цієї функції на C # необхідно встановити і налаштувати платформу NET Framework. Вона поставляється повністю безкоштовно, застосовується вкрай широко, а тому проблем з користувацькими пристроями зазвичай не виникає. Платформа вбудована в інсталяційний пакет Windows, при необхідності її також можна скачати і «поставити» окремо. Існують версії для Лінукс і MAC.

В рамках платформи до обробки виконуваного коду підключається середа CLR - єдиний об'єднаний набір бібліотек і класів, який був розроблений Майкрософт і є реалізацією світового стандарту Common Language Infrastructure (CLI).

Після роботи компілятора текст програми перекладається в проміжну мову IL, яка «розуміє» CLI. Працює це так. IL і всі необхідні ресурси, включаючи рядки і малюнки формату BMP, зберігаються на жорсткий диск у вигляді виконуваного файлу dll або exe. З таких файлів з проміжним кодом формується збірка додатків, яка включає в себе опис з повною інформацією про всі важливі параметри роботи.

Безпосередньо при виконанні програми CLR звертається до збірки і виробляє дії в залежності від отриманих відомостей. Якщо код написаний правильно і проходить перевірку безпеки системи, проводиться компіляція з IL в інструкції в машинні команди. Середа CLR попутно виконує ще багато побічних функцій:

1. видалення «програмного» сміття;

2. робота з винятками;
3. розподіл ресурсів;
4. контроль типізації;
5. управління версіями;
6. типізація;
7. управління версіями.

В результаті код C # вважається керованим, тобто він компілюється в двійковий вид на призначеному для користувача пристрої з урахуванням особливостей встановленої системи.

Для сучасних розробників програмного забезпечення створюють усі необхідні умови для комфортної роботи. Однією з найбільш корисних розробок сміливо можна назвати інтегроване середовище розробки. Це програмне забезпечення, яке надає комплексні можливості програмістам для розробки програмного забезпечення. Як правило, воно складається з щонайменше редактора вихідних кодів, засобів автоматизації побудови та відладчика. Інтегровані середовища розробки створені для того, щоб максимізувати продуктивність програміста, надавши йому пов'язані інструменти розробки зі схожими інтерфейсами як одну програму, в якій відбуватиметься весь процес розробки й яка надає необхідні функції для модифікації, компілювання, розгортання та налагодження програмного забезпечення.

Переваги інтегрованого середовища розробки над звичайними редакторами значущі. Програміст 80% свого часу витрачає на розуміння написаного коду та переміщенню по ньому. Причому переміщенню саме по коду, а не по тексту, тому йому редактор не може допомогти абсолютно нічим. Список параметрів методу в підказці не покаже, перейти до визначення методу не дозволить, синтаксис не проконтролює і не підсвітить. А середовища, навіть найпростіші, з цим справляються просто і елегантно, завдяки наявності синтаксичного аналізатора мови програмування. Середовище, на відміну від редактора, «розуміє» що воно редагує код. А це і автодоповнення, і навігація, і підсвічування синтаксичних, а, іноді, і семантичних помилок. Здається, надмірність, приємна дрібниця,

баловство. Але воно, перетворюється в необхідність після того, як розмір проекту перевищує певну межу. А з урахуванням об'ємних сучасних проектів — ця межа настає практично відразу. Також саме поняття проекта існує тільки в інтегрованому середовищі розробки. До нього прив'язуються настройки, ресурси, можна здійснювати пошук файлів та словосполучень і т.п. У редакторах це в кращому випадку відкритий каталог файлової системи. Звичайно деякі редактори можна налаштувати чи встановити плагіни, але такі доповнення можуть бути складні в установці і ніколи не зможуть покрити весь функціонал якісного середовища розробки.

Для розробки на C++ існує безліч середовищ. Одні з найбільш популярних на даний час:

1. Dev-C++ — це безкоштовна інтегроване середовище розробки з відкритим вихідним кодом, написана на Delphi для Windows. Вона досить легка, для неї потрібно всього пару хвилин для установки. Це — найкраще середовище розробки для новачків, в ній можна встановити плагін для створення GUI — інтрефейса методом перетягування елементів.

2. Xcode — це не просто інтегроване середовище розробки, а повний набір інструментів для розробки програмного забезпечення, створена Apple для розробки програмного забезпечення для MacOS, iOS, WatchOS і tvOS.

3. Microsoft Visual Studio — це середовище розробки від Microsoft. Visual Studio в основному відома для створення програмного забезпечення, що включають в себе .NET. Це повний набір інструментів, що дозволяє зробити точне налагодження і налаштування програми.

4. Qt Creator — кросплатформене вільне середовище для розробки на C, C ++ і QML. Розроблено Trolltech(Digia) для роботи з фреймворком Qt. Включає в себе графічний інтерфейс відладчика і візуальні засоби розробки інтерфейсу. Підтримувані компілятори: GCC, Clang, MinGW, MSVC.

Як було зазначено в розділі 2.1, C++ — кросплатформна мова рівня компілювання. Це означає, що для того щоб написаний програмний продукт міг запускатися на усіх платформах, його необхідно окремо компілювати під кожну.

Така особливість вимагає від розробника переписування коду згідно до вимог кожного компілятора. Це досить затратно по часу і до того ж ускладнює можливість модифікації та розширення програмного продукту. Щоб уникнути дану проблему було вирішено вести розробку під компілятор GCC для UNIX систем, та компілятор MinGW для Windows. MinGW — це набір вільного програмного забезпечення з відкритим кодом для розробки Windows додатків. Він включає в себе порт GNU Compiler Collection(GCC), GNU Binutils, набір вільно розповсюджуваних Windows файлів заголовків та статичних бібліотек які дозволяють використовувати Windows API, GNU debugger та інші утиліти. MinGW не використовує сторонніх C DLL файлів. Бібліотеки не розповсюджуються під ліцензією GNU's General Public License (GPL), тому не обов'язково надавати вихідний код разом з програмою. Таким чином, ми зможемо уникнути проблеми з адаптуванням коду під різні компілятори і єдиною умовою для цього є написання коду відповідно до вимог компілятору GCC, що не являється проблемою як такою.

Єдине інтегроване середовище розробки, яке задовільняє наведені вище умови це Qt Creator, так як воно являється кросплатформеним і підтримує компілятори GCC та MinGW. Також Qt Creator має безліч корисних особливостей для розробки, а саме:

1. розумний редактор коду;
2. майстер генерації проектів Qt5;
3. інтеграція довідки по коду;
4. інтеграція з Qt Designer;
5. зручний пошук по файлам словосполучень;
6. підтримка формату файлу проекту .pro;
7. наявність GDB.

3.2. Опис і тестування удосконаленого алгоритму

Дерево рішень - це спосіб подачі вирішальних правил в ієрархічній структурі, що складається з елементів двох типів - вузлів і листя. У вузлах знаходяться вирішальні правила і проводиться перевірка відповідності прикладів цього правила з якого-небудь атрибуту навчальної множини.

У найпростішому випадку, в результаті перевірки, безліч прикладів, які потрапили в вузол, розбивається на дві підмножини, в одне з яких потрапляють приклади, що задовольняють правилу, а в інше - що не задовольняють.

Потім до кожної підмножини знову застосовується правило і процедура повторюється поки не буде досягнута деяка умова зупинки алгоритму. В результаті в останньому вузлі перевірка і розбиття не проводиться і він оголошується листом. Лист визначає рішення для кожного потрапившего в нього прикладу. Для дерева класифікації - це клас, що асоціюється з вузлом.

Таким чином, на відміну від вузла, в листі міститься не правило, а підмножина об'єктів, які відповідають всім правилам гілки, яка закінчується даним листом.

Очевидно, щоб потрапити в лист, приклад повинен відповідати всім правилам, які лежать на шляху до цього листа. Оскільки шлях в дереві до кожного листу єдиний, то й кожен приклад може потрапити тільки в один лист, що забезпечує єдність розв'язки.

У випадку автора береться перелік функцій JS, які вважаються шкідливими (див. Рис. 3.1), що будуть вузлами дерева.

Document.cookie – властивість, яка використовується щоб отримати доступ до кукі безпосередньо з браузера. Кукі - це невеликі рядки даних, які зберігаються безпосередньо в браузері. Вони є частиною HTTP-протоколу, визначеного в специфікації RFC 6265. Один з найбільш частих випадків використання кукі - це автентифікація. Значення document.cookie складається з пар ключ = значення, розділених ;. Кожна пара являє собою окреме кукі. Щоб знайти певний кукі, досить розбити рядок з document.cookie по ;, і потім знайти потрібний ключ. Для цього ми можна використовувати як регулярні вирази, так і функції для обробки масивів.

```
class Program
{
    private static string[] cookieAccess = { "document.cookie", "window.localStorage", "document.URL", "getCookie", "setCookie" };
    private static string[] dynamicDOM = { "document.write", "document.writeln", ".innerHTML", ".outerHTML", ".insertAdjacentHTML" };
    private static string[] urlManipulations = { "document.URL", "location.assign", "location.replace" };
    private static string[] webBrowser = {"getAppName", "getUserAgent" };
    private static string[] dynamicCode = { "setInterval", "setTimeout", "eval" };
```

Рис. 3.1. Шкідливі функції JS

Властивість localStorage дозволяє отримати доступ до Storage об'єкту. localStorage аналогічно властивості sessionStorage. Різниця тільки в тому, що властивість sessionStorage зберігає дані протягом сеансу (до закриття браузера), на відміну від даних, які перебувають у властивості localStorage, які не мають обмежень за часом зберігання і можуть бути видалені тільки за допомогою JS. Слід зазначити, що дані, збережені як в localStorage, так і в sessionStorage, є специфічними для протоколу сторінки. Ключі та значення завжди рядки (так само, як і об'єкти, цілочисельні ключі автоматично будуть перетворені в рядки).

Document.URL повертає рядок URL документа HTML.

Найкоротший спосіб отримати доступ до кукі - це використовувати регулярні вирази. Функція getCookie (name) повертає куки з зазначеним name. setCookie (name, value, options) встановлює куки з ім'ям name і значенням value, з налаштуванням path = / за замовчуванням (можна змінити, щоб додати інші значення за замовчуванням).

Метод document.write працює тільки поки HTML-сторінка знаходиться в процесі завантаження. Він дописує текст в поточне місце HTML ще до того, як браузер побудує з нього DOM.

JS метод .writeln () об'єкта document записує в потік документа рядок тексту за якою слідує символ нового рядка. Щоб відкрити потік для запису документа необхідно скористатися методом open () об'єкта document. Для завершення запису документа (закрити потік відкритий за допомогою методу .open () об'єкта document) необхідно скористатися методом close () об'єкта document. Дані в потік можуть бути передані також за допомогою методу .writeln (). Основна відмінність методів write () і .writeln () полягає в тому, що метод .writeln () додатково додає символ нового рядка.

Властивість innerHTML є вміст елемента (елементи-нащадки, коментарі, текст і т.д.), яке зберігається в ньому у вигляді рядка. Властивість доступна для читання і запису, тому є можливість отримувати і змінювати вміст елемента.

Атрибут insertAdjacentHTML DOM-інтерфейсу element отримує серіалізовані HTML-фрагмент, що описує елемент, включаючи його нащадків. Можна встановити заміну елемента вузлами, отриманими з заданої рядки.

insertAdjacentHTML () розбирає вказаний текст як HTML або XML і вставляє отримані вузли в DOM дерево в зазначену позицію. Ця функція не переписує наявні елементи, що запобігає додаткову серіалізацію і тому працює швидше, ніж маніпуляції з innerHTML.

Метод Location.assign () запускає завантаження і відображення нового документа за вказаною URL.

Метод Location.replace () замінює поточний ресурс на новий по URL, вказаною в якості параметра. Відмінність від assign () в тому, що при використанні replace () поточна сторінка не буде збережена в History, і користувач не зможе використовувати кнопку назад, щоб повернутися до неї.

Властивість appName повертає ім'я браузера.

getUserAgent - нестатичний метод, що повертає запитаний заголовок юзерагента HTTP.

Є можливість викликати функцію не в даний момент, а пізніше, через заданий інтервал часу. Це називається «планування виклику». Для цього існують два методи: setTimeout дозволяє викликати функцію один раз через певний інтервал часу, setInterval дозволяє викликати функцію регулярно, повторюючи виклик через певний інтервал часу.

Вбудована функція eval дозволяє виконувати рядок коду.

Елемент <input> – найважливіший елемент управління веб-формою, визначає поле введення, в яке користувач може вводити дані, також може відображатися кількома способами, в залежності від атрибута type. Через величезну кількість можливих поєднань типів введення і атрибутів це один з найбільш потужних і складних елементів. Зловмисник може використовувати цей елемент для зміни веб-форми, що може призвести до витоку даних користувача. Елемент «isInput» - є кореневим вузлом (див. Рис.3.2), який розбиває дерево на перші дві

підмножини: якщо результат позитивний, то код випадає під загрозу Cookie Access, гілку «cookieAccessBranch», якщо негативний – перевіряється належність до типу <script>.

```
        var isButton = new DecisionQuery
        {
            Title = "Is button type element",
            Test = (elem) => string.Equals(elem.TagName, "button", StringComparison.OrdinalIgnoreCase),
            Positive = cookieAccessBranch,
            Negative = new DecisionResult { maliciousRating = 0 }
        };

        var isScript = new DecisionQuery
        {
            Title = "Is script type element",
            Test = (elem) => string.Equals(elem.TagName, "script", StringComparison.OrdinalIgnoreCase),
            Positive = cookieAccessBranch,
            Negative = isButton
        };

        var isInput = new DecisionQuery
        {
            Title = "Is input type element:",
            Test = (elem) => string.Equals(elem.TagName, "input", StringComparison.OrdinalIgnoreCase),
            Positive = cookieAccessBranch,
            Negative = isScript
        };

        return isInput;
    }
```

Рис. 3.2. Кореневий вузол дерева

Тег <script> призначений для опису скриптів, може містити посилання на програму або її частину тексту певною мовою. Скрипти можуть розташовуватися в зовнішньому файлі і зв'язуватися з будь-яким HTML-документом. Такий підхід дозволяє використовувати одні й ті ж загальні функції на багатьох веб-сторінках і прискорює їх завантаження, тому що зовнішній файл гешується при першому завантаженні, і скрипт викликається швидше при наступних викликах. Зловмисник може посилатися на шкідливу програму завдяки тегу <script>, тому цей елемент входить в гілку «cookieAccessBranch».

Елемент button відображається як прості кнопки, які можна запрограмувати для управління призначеними для користувача функціями в будь-якому місці веб-сторінки, наприклад, призначити функцію обробки події. Елемент не має поведінки за замовчуванням, щоб кнопки робили що-небудь, треба написати код JS для виконання роботи. Цей елемент також входить в гілку «cookieAccessBranch».

Кожному класу загрози («Possile cookie leak», «Dynamic Code Construction», «Url manipulation», «Web browser attributes leak», «Dynamic Code

Construction») призначається шкідливий рейтинг від 2 до 10 (див. Рис. 2.3), який в кінці роботи програми сумується.

```
var urlManipulationsBranch = new DecisionQuery
{
    Title = "\tUrl manipulation",
    Test = (elem) => urlManipulations.Any(c => elem.OuterHtml.Contains(c)),
    Positive = new DecisionResult { maliciousRating = 6 },
    Negative = webBrowserBranch
};
var dynamicDOMBranch = new DecisionQuery
{
    Title = "\tDynamic Code Construction",
    Test = (elem) => dynamicDOM.Any(c => elem.OuterHtml.Contains(c)),
    Positive = new DecisionResult { maliciousRating = 8 },
    Negative = urlManipulationsBranch
};
var cookieAccessBranch = new DecisionQuery
{
    Title = "\tPossile cookie leak",
    Test = (elem) => cookieAccess.Any(c => elem.OuterHtml.Contains(c)),
    Positive = new DecisionResult { maliciousRating = 10 },
    Negative = dynamicDOMBranch
};
```

Рис. 3.3. Шкідливий рейтинг

Останніми вузлами дерева (див. Рис. 3.4) є «Web browser attributes leak» та «Dynamic Code Construction», з шкідливим рейтингом 4 та 2 відповідно.

```
private static DecisionQuery MainDecisionTree()
{
    var dynamicCodeBranch = new DecisionQuery
    {
        Title = "\tDynamic Code Construction",
        Test = (elem) => dynamicCode.Any(c => elem.OuterHtml.Contains(c)),
        Positive = new DecisionResult { maliciousRating = 2 },
        Negative = new DecisionResult { maliciousRating = 0 }
    };
    var webBrowserBranch = new DecisionQuery
    {
        Title = "\tWeb browser attributes leak",
        Test = (elem) => webBrowser.Any(c => elem.OuterHtml.Contains(c)),
        Positive = new DecisionResult { maliciousRating = 4 },
        Negative = dynamicCodeBranch
    };
```

Рис. 3.4. Листя дерева рішень

На Рис. 3.5. показана побудова вузла дерева. Цей елемент коду проводить тест, виводить його результат і вирішує в яку гілку пуститися далі.

```
public class DecisionQuery : Decision
{
    public string Title { get; set; }
    public Decision Positive { get; set; }
    public Decision Negative { get; set; }
    public Func<HtmlElement, bool> Test { get; set; }

    public override Decision Evaluate(HtmlElement client)
    {
        bool result = this.Test(client);
        string resultAsString = result ? "yes" : "no";

        Warning += $"\n\t- {this.Title}? {resultAsString}";

        if (result)
            return this.Positive.Evaluate(client);
        else
            return this.Negative.Evaluate(client);
    }
}
```

Рис. 3.5. Побудова вузла дерева

Беручи до уваги рейтинг шкідливості, підсумований в кінці, якщо maliciousRating не дорівнює нулю (див. Рис. 3.6) – код містить шкідливий JS.

На Рис. 3.7 зображена частина коду, яка відповідає за процес виявлення, аналізу та обробки програмного коду веб-сторінки та перевіряє його на наявні загрози інформаційній безпеці.

```
public class DecisionResult : Decision
{
    public int maliciousRating { get; set; }
    public bool Result => maliciousRating != 0;
    public override Decision Evaluate(HtmlElement client)
    {
        Warning += $"\r\nMalicious : {(Result ? "YES" : "NO")}";
        return this;
    }
}
```

Рис. 3.6. Визначення шкідливості коду

```
private static void PageHtml_DocumentCompleted(object sender, WebBrowserDocumentCompletedEventArgs e)
{
    var browser = sender as WebBrowser;
    var tree = MainDecisionTree();
    int maliciousRating = 0;
    int maxRating = 0;
    foreach (HtmlElement element in browser.Document.All)
    {
        maxRating += string.Equals(element.TagName, "button", StringComparison.OrdinalIgnoreCase) ||
                     string.Equals(element.TagName, "script", StringComparison.OrdinalIgnoreCase) ||
                     string.Equals(element.TagName, "input", StringComparison.OrdinalIgnoreCase)
                     ? 10 : 0;
        if (tree.Evaluate(element) is DecisionResult result)
        {
            maliciousRating += result.maliciousRating;
            if (result.Result)
            {
                Console.WriteLine(element.OuterHtml);
                Console.WriteLine(Decision.Warning);
            }
            Decision.Warning = string.Empty;
        }
    }
```

Рис. 3.7. Частина коду, що витягує елементи

На Рис. 3.8 зображена головна частина програми, яка тестує аргументи на наявність загроз, запускає програму та виводить результати.

```
private static void Main(string[] args)
{
    Console.WriteLine($"Testing {args[0]} for XSS.");
    var t = new Thread(() =>
    {
        var pageHtml = new WebBrowser();
        pageHtml.DocumentCompleted += PageHtml_DocumentCompleted;
        pageHtml.ScriptErrorsSuppressed = true;
        pageHtml.Navigate(args[0]);
        Application.Run();
    });
    t.SetApartmentState(ApartmentState.STA);
    t.Start();
    t.Join();
}
```

Рис. 3.8. Головна частина програми

Для демонстрації роботи програми на сайті без шкідливих JS сценаріїв було обрано сайт Національного авіаційного університету (nau.edu.ua). Програма не знайшла жодного вразливого елементу коду, тому рейтинг шкідливості дорівнює 0 з 20 протестованих елементів.

```
C:\Users\Алексей>E:\XSSDecisionTree\XSSDecisionTree\obj\x64\Debug\XSSDecisionTree.exe nau.edu.ua
Testing nau.edu.ua for XSS.
Malicious Rating: 0/20
```

Рис. 3.9. Приклад сайту без шкідливого JS коду

Для демонстрації роботи програми на сайті з шкідливим JS сценарієм першим було обрано сайт seasonvar.ru. Програма знайшла дві вразливих частини

коду, в обох випадках функція document.write типу script припадає до класу Possible cookie leak, Dynamic Code Constrution. Програма вважає сайт шкідливим та ставить йому рейтинг 18 з 410 протестованих елементів (див. Рис. 3.10).

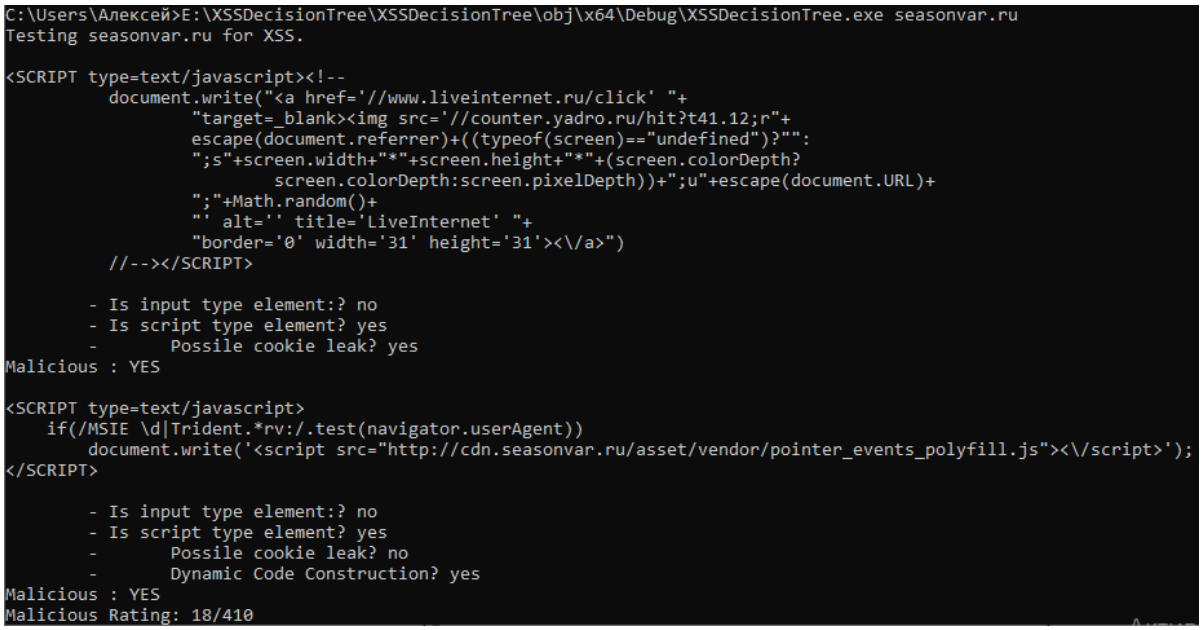

Рис. 3.10. Тестування сайту seasonvar.ru

Рис. 3.11. Тестування сайту securityweek.com

Для демонстрації роботи програми на сайті з шкідливим JS сценарієм другим було обрано сайт securityweek.com. Програма знайшла одну вразливу частину коду, в обох випадках атрибут script припадає до класу Dynamic Code Construction. Програма вважає сайт шкідливим та ставить йому рейтинг 2 з 240 протестованих елементів (див. Рис.3.11).

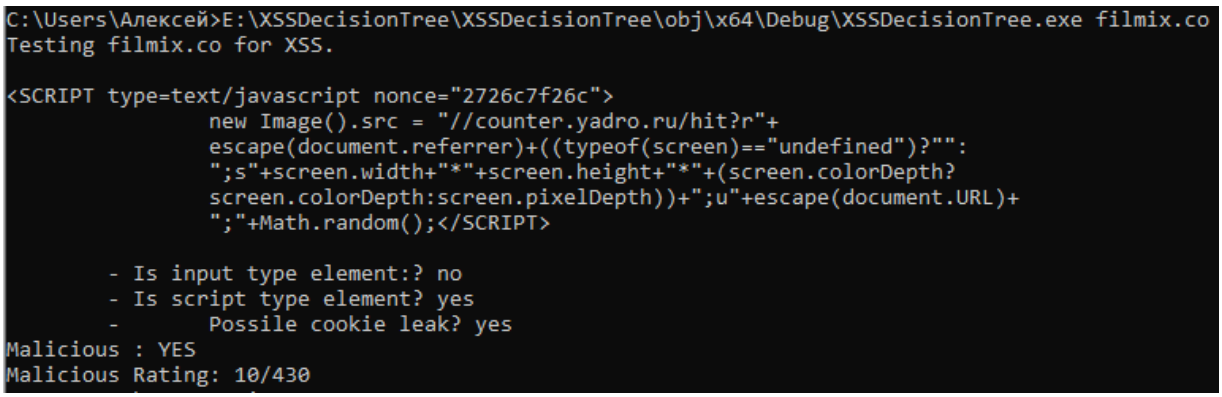

Рис. 3.12. Тестування сайту filmix.co

Для демонстрації роботи програми на сайті з шкідливим JS сценарієм другим було обрано сайт securityweek.com. Програма знайшла одну вразливу частину коду, в обох випадках атрибут script припадає до класу Possible cookie leak. Програма вважає сайт шкідливим та ставить йому рейтинг 10 з 430 протестованих елементів (див. Рис.3.12).

3.3. Розрахунок точності удосконаленого алгоритму та його переваги

Зібрано URL-адреси з еталонних джерел URL-адрес як для шкідливого, так і для безпечного JavaScript з процентним поділом 50% (по 100 шт.). Це означає, що набори мають однакову кількість зразків. Безпечні URL-адреси взяті з сайтів Alexa Top20. Для набору даних шкідливих URL-адрес зібрано URL-адреси з чорного списку шкідливих програм і фішингу в базі даних перевірених фішингових сторінок PhishTank. Також зібрано шкідливий набір даних JS з GeeksOnSecurity-GitHub22.

Веб-сайти було протестовано на власному наборі шкідливих функцій представлених в Таблиці 3.1.

Таблиця 3.1

Список власного набору функцій загроз

JS функція	Опис
document.cookie	Кількість функцій document.cookie ()
window.localStorage	Кількість функцій window.localStorage ()

document.URL	Кількість функцій document.URL
getCookie	Кількість функцій getCookie
setCookie	Кількість функцій setCookie
document.writeln	Кількість функцій document.writeln
.innerHTML	Кількість подій .innerHTML
.outerHTML	Кількість подій .outerHTML
.insertAdjacentHTML	Кількість подій .insertAdjacentHTML
document.URL	Кількість функцій document.URL
location.assign	Кількість функцій location.assign
location.replace	Кількість функцій location.replace
getAppName	Кількість функцій getAppName
getUserAgent	Кількість функцій getUserAgent
setInterval	Кількість функцій setInterval
setTimeout	Кількість функцій setTimeout

Таблиця 3.2

Аналіз продуктивності алгоритмів по власному набору

Класифікатор	Точність	Хибнопозитивний результат	Хибнонегативний результат	Крива помилок
J48 Дерево рішень	96.86	0.003	0.000	0.998

В Таблиці 3.2 показано результат тестування алгоритму на власному наборі функцій. Точність алгоритму складає 96,86%, що на 0,04% вище ніж в Таблиці 2.5.

Таблиця 3.3

Порівняння продуктивності алгоритмів з наборами Таблиці 2.1, Таблиці 2.4 та власним

Класифікатор	Точність (%)	Хибнопозитивний результат	Хибнонегативний результат	Крива помилок
J48 Дерево рішень (Набір 1)	99.67	0.005	0.000	0.998
J48 Дерево рішень (Набір 2+1)	99.82	0.003	0.000	0.998
J48 Дерево рішень (Набір 1+2+власний)	99.91	0.003	0.000	0.998

В Таблиці 3.3 представлене порівняння ефективності алгоритмів зі всіма наборами. В останньому випадку до Набору 1 і 2 додано 16 функцій з власного набору, що привело до збільшення точності на 0,09%. Хибнопозитивний, хибнонегативний результати та крива помилок залишилися без змін.

3.4. Висновки до розділу

Дерево рішень - математична модель, яка задає процес прийняття рішень так, що буде відображене кожне можливе рішення, попередні та наступні цьому рішенню події або інші рішення і наслідки кожного кінцевого рішення. Дерево рішень складається з наступних елементів: вузлів рішень, вузлів подій і кінцевих вузлів (листів).

Було реалізовано удосконалений алгоритм Дерева рішень, який протестовано на власному наборі з 16 функцій та який в результаті виводить рейтинг шкідливості JS-коду та вразливі фрагменти. Точність алгоритму складає 96,86%, що на 0,04% вище ніж при тестуванні на Наборі 2.

При додаванні функцій всіх наборів точність збільшується на 0,09%. Хибнопозитивний, хибнонегативний результати та крива помилок залишилися без змін.

ВИСНОВКИ

Створення JavaScript значно збагатило інтерактивні можливості клієнта. Однак зловмисники використовують динамічну функцію мови JavaScript для вбудовування шкідливого коду в веб-сторінки з метою завантаження, перенаправлення, відстеження та ін.

Проаналізувавши ефективність існуючих алгоритмів виявлення, таких як Наївний баєсів, Дерево рішень, Випадковий ліс, Метод опорних векторів, AdaBoost, REPTree, ADTree, можна зробити, що всі вони працюють майже з однаковою точністю. Їх показники за ознаками варіюються в залежності від набора функцій шкідливих сценаріїв (експериментально було обрано 2 набори). Найгірші результати показав Наївний баєсів відстаючи на 2-3% точності в обох випадках.

В обох випадках алгоритм Дерево рішень показував одні з кращих результатів по ознакам. При його використанні немає необхідності нормалізувати та масштабувати дані. Етап попередньої обробки даних для дерев рішень вимагає менше коду, часу і аналізу. Концепція, що лежить в основі дерева рішень, більш відома програмістам і порівняно легше для розуміння, ніж інші подібні алгоритми.

Результатом виконаної роботи є удосконалений алгоритм Дерево рішень, який протестовано на власному наборі з 16 функцій. В результаті алгоритм виводить рейтинг шкідливості JS-коду та вразливі фрагменти, а також його максимальна точність досягає 99,91%, що на 0,09% ніж у попередників. Хибнопозитивний, хибнонегативний результати та крива помилок залишилися без змін.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Что Такое JavaScript? Презентация JS Для Начинающих. URL: https://www.hostinger.com.ua/rukovodstva/chto-takoe-javascript/#JavaScript (дата звернення: 25.09.2020).
2. Кантор И. Язык Javascript. Москва : Издательство АСТ, 2020. 120 с.
3. Рейтинг мов програмування 2020: JavaScript випередив Java, а Dart увійшов у першу лігу. URL: https://dou.ua/lenta/articles/language-rating-jan-2020/ (дата звернення: 30.09.2020).
4. Top 7 Programming Languages Of 2020. URL: https://www.codingdojo.com/blog/top-7-programming-languages-of-2020 (дата звернення: 30.09.2020).
5. JavaScript for Begginers. Course notes. URL: https://winterstein.me.uk/javascript/JavaScript%20Course%20Notes.pdf (дата звернення: 25.09.2020).
6. Flanagan D. JavaScript: The Definitive Guide. 7th Edition. O'Reilly Media, 2019. 462 p.
7. Object-oriented JavaScript for beginners. URL: https://developer.mozilla.org/ru/docs/Learn/JavaScript/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D1%8B/Object-oriented_JS (дата звернення: 26.09.2020).
8. What is the Deal with Javascript Versions? URL: https://levelup.gitconnected.com/what-is-the-deal-with-javascript-versions-88334491906c (дата звернення: 26.09.2020).
9. Чем отличаются JavaScript и ECMAScript? URL: https://orkhanalyshov.com/blog/58 (дата звернення: 26.09.2020).
10. JavaScript Versions. URL: https://www.w3schools.com/js/js_versions.asp (дата звернення: 26.09.2020).

11. ECMAScript 2016+. URL: https://kangax.github.io/compat-table/es2016plus/ (дата звернення: 26.09.2020).

12. Essential JavaScript Libraries and Frameworks You Should Know About. URL: https://code.tutsplus.com/articles/essential-javascript-libraries-and-frameworks-you-should-know-about--cms-29540 (дата звернення: 27.09.2020).

13. NDSS 2017 JSlibs. URL: http://www.ccs.neu.edu/home/arshad/publications/ndss2017jslibs.pdf (дата звернення: 27.09.2020).

14. Advantages of Using JavaScript in Web Technologies. URL: https://dev.to/javablog/advantages-of-using-javascript-in-web-technologies-1c4g (дата звернення: 27.09.2020).

15. Rodriguez S. Start-up says 80% of its Facebook ad clicks came from bots. Los Angeles Times. July 31, 2012.

16. Sengupta S. Bots Raise Their Heads Again on Facebook. New York Times. July 31, 2012.

17. 3 Ways JavaScript Can Breach Your Privacy & Security. URL: https://www.makeuseof.com/tag/3-ways-javascript-can-used-breach-privacy-security/ (дата звернення: 27.09.2020).

18. A simple overview of the Twitter StalkDaily virus. URL: https://www.networkworld.com/article/2235261/a-simple-overview-of-the-twitter-stalkdaily-virus.html (дата звернення: 28.09.2020).

19. The Advantages and Disadvantages of JavaScript. URL: https://www.freecodecamp.org/news/the-advantages-and-disadvantages-of-javascript/ (дата звернення: 28.09.2020).

20. Internet Engineering Task Force. JSON Web Token (JWT). URL: https://tools.ietf.org/html/rfc7519 (дата звернення: 29.09.2020).

21. Same Origin Policy. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy (дата звернення: 29.09.2020).

22. Security Zones and Signed Scripts. URL: https://docstore.mik.ua/orelly/webprog/jscript/ch21_04.htm (дата звернення: 29.09.2020).

23. Naive Bayes Classifier. URL: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c (дата звернення: 09.10.2020).

24. J48 decision Tree. URL: http://data-mining.business-intelligence.uoc.edu/home/j48-decision-tree. (дата звернення: 15.10.2020).

25. Understanding Random Forest. URL: https://towardsdatascience.com/understanding-random-forest-58381e0602d2 (дата звернення: 17.10.2020).

26. The alternating decision tree learning algorithm. URL: https://cseweb.ucsd.edu/~yfreund/papers/atrees.pdf (дата звернення: 23.10.2020).

27. Naive Bayes Explained: Function, Advantages. URL: https://www.upgrad.com/blog/naive-bayes-explained/ (дата звернення: 30.10.2020).

28. Random Forest Algorithm Advantages and Disadvantages. URL: https://dhirajkumarblog.medium.com/random-forest-algorithm-advantages-and-disadvantages-1ed22650c84f (дата звернення: 10.11.2020).

29. SVM as a technique. URL: https://core.ac.uk/download/pdf/188978526.pdf (дата звернення: 20.11.2020).

30. A guide to AdaBoost: Boosting To Save The Day. URL: https://blog.paperspace.com/adaboost-optimizer/ (дата звернення: 20.11.2020).

31. Class REPTree. URL: https://weka.sourceforge.io/doc.dev/weka/classifiers/trees/REPTree.html (дата звернення: 20.11.2020).