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

«До захисту допущено»
В.о. завідувача кафедри

_____ М.В.Грайворонський
(підпис)
“___”_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

на тему: Аналіз методів захисту JavaScriptфреймворків, використовуваних під час розробки веб-застосунків

Виконав (-ла): студент (-ка) 4 курсу, групи ФБ-51
(шифр групи)

Цвіленко Олег
(прізвище, ім’я, по батькові) (підпис)

Керівник: доцент к.т.н. Литвинова Т.В..
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

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

Засвідчую, що у цій дипломній роботі 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 – перший (бакалаврський)

Напрям підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)
«___»_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Цвіленку Олегу_____
(прізвище, ім'я, по батькові)

1. Тема роботи Аналіз методів захисту JavaScriptфреймворків, використовуваних під час розробки веб-застосунків_____ ,
науковий керівник роботи доцент к.т.н. Литвинова Т.В. _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

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

2. Термін подання студентом роботи 10 червня 2019 р.

3. Вихідні дані до роботи _____
_____
_____

4. Зміст роботи_____
_____
_____
_____
_____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) _____
_____
_____
_____

6. Дата видачі завдання _____

РЕФЕРАТ

Робота обсягом 66 сторінок містить 18 ілюстрацій, 5 таблиць та 26 літературних посилань.

Метою даної кваліфікаційної роботи є оцінка захисту JavaScript фреймворків, які використовуються під час розробки веб-застосунків, побудова свого методу захисту для одного із фреймворків, та порівняльний аналіз написаного методу з наявними.

Об’єктом дослідження є JavaScript фреймворки.

Предметом дослідження є методи захисту сучасних JavaScript фреймворків.

Результати роботи викладені у вигляді таблиці, що демонструє, захищеність базових методів в обраних для аналізу JavaScript фреймворків і їх порівняння з розробленим власним методом захисту одного із фреймворків.

Результати роботи можуть бути використані при розробці методів захисту односторінкових веб-застосунків.

JAVASCRIPT, ФРЕЙМВОРКИ, XSS, DOMSANITIZER, BYPASSSECURITYTRUSTX

ABSTRACT

The work of 66 pages contains 18 illustrations, 5 tables and 26 literary references.

The purpose of this qualification work is to evaluate the protection of JavaScript frameworks used in developing web applications, build its own method of protection for one of the frameworks, and a comparative analysis of the written method with the available ones.

The subject of the study is the JavaScript framework.

The subject of the research is the methods of protecting modern JavaScript frameworks.

The results of the work are presented in the form of a table demonstrating the security of the basic methods in selected for analysis of JavaScript frameworks and their comparison with the developed own method of protection of one of the frameworks.

The results of the work can be used in developing methods for protecting one-page web applications.

JAVASCRIPT, FRAMEWORKS, XSS, DOMSANITIZER, BYPASSSECURITYTRUSTX

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 7
Вступ 8
1 JavaScript фреймворки, які використовуються під час розробки веб-застосунків та їх базовий захист 10
1.1 JavaScript фреймворки 10
1.2 Базовий захист JavaScript фреймворків 16
Висновки до розділу 1 20
2 Аналіз методів захисту JavaScript фреймворків, які використовуються під час розробки веб-застосунків від найпоширеніших атак 21
2.1 Основні вразливості JavaScript фреймворків 21
2.2 Досліджувані фреймворки і атаки на них 30
2.3 Аналіз запобігання крос-сайт сценаріїв (XSS) у обраних JavaScript фреймворках 31
2.4 Аналіз запобігання включення міжсайтового сценарію (XSSI) у обраних JavaScript фреймворках 36
Висновки до розділу 2 41
3 Розробка методу захисту одного із JavaScript фреймворків та його порівняльний аналіз із базовими методами захисту 43
3.1 Розробка методу захисту JavaScript фреймворка 43
3.2 Порівняльний аналіз написаного та наявних методів захисту JavaScript фреймворків 50
Висновки до розділу 3 61
Висновки 62
Перелік джерел посилань 64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JS – JavaScript

XSS – cross site scripting

CSS – cascading style sheets

CDN – content delivery network

SPA – single page aplications

API – application programming interface

CSP – content security policy

CSRF – cross site request forgery

ВСТУП

Ми живемо в той час, коли безперервний розвиток інтернету та інформаційних технологій в цілому зробили так, що навіть маленька компанія, не кажучи вже про великі корпорації, державні установи, має свій веб-застосунок.

Фреймворк - це каркас веб - застосунку. Він зобов'язує програміста будувати архітектуру програми відповідно до певної логіки. Зазвичай середовище надає можливість для подій, сховищ і з'єднань даних. Як і машина, рама забезпечує готовий каркас, кузов і двигун. Можна додавати, знімати або замінювати певні деталі за умови, що автомобіль залишається в робочому стані.

Каркаси знаходяться на більш високому рівні абстракції в порівнянні з бібліотекою і дозволять непотрібні зусилля з розробки приблизно 80% додатків. Але слід пам'ятайти:

• Останні 20% можуть викликати значні труднощі через обмеження, що накладаються рамками.

• Оновлення структури може бути зумовлене складністю, а іноді й неможливою.

• Основний код і поняття рідко відповідають вимогам творців у їх первісному вигляді. Завжди треба знаходити "кращій" спосіб зробити щось.

Більшість фреймворків пишуться за допомогою JavaScript. Сьогодні більше 90 відсотків веб-застосунків створенні на базі саме JavaScript фреймворків.

Незважаючи на гарний базовий захист у всіх сучасних фреймворках, все одно є лазівки, за допомогою яких хакери отримують доступ до важливих даних та інформації про користувачів, банківські рахунки, тощо. Саме через це ця робота є актуальною на сьогодні, бо кожен розробник під час написання веб-застосунку за допомогою JavaScript фреймворку може покращити базовий

захист, який йому надає фреймворк. Результатом цієї роботи і є такий метод, та його порівняння з базовими методами захисту інших фреймворків.

Мета роботи - оцінка захисту JavaScript фреймворків, які використовуються під час розробки веб-застосунків, побудова свого методу захисту для одного із фреймворків, та порівняльний аналіз написаного методу з наявними.

Завданням роботи є розробка методу захисту одного із JavaScript фреймворків та його порівняльний аналіз із базовими методами.

Об’єктом дослідження є сучасні JavaScript фреймворки.

Предметом дослідження є методи захисту сучасних JavaScript фреймворків.

1 JAVASCRIPT ФРЕЙМВОРКИ, ЯКІ ВИКОРИСТОВУЮТЬСЯ ПІД ЧАС РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ ТА ЇХ БАЗОВИЙ ЗАХИСТ

1.1 JavaScript фреймворки

Інструменти спрощують процес програмування. Наприклад, використовуйте препроцесор Sass замість чистого CSS, оскільки він надає можливість використання циклів, функцій, локальних змінних і багато іншого. Браузери не розуміють синтаксис Sass / SCSS, тому код перекладається на CSS.

Структура CSS-Bootstrap

Bootstrap є найпопулярнішим фреймворком з готовими стилями і скриптами, для яких потрібно перезаписати необхідні класи і атрибути елементів HTML. Завдяки цій структурі легко створювати мобільні проекти за допомогою Bootstrap, що дозволяє легко налаштовувати будь-яку сторінку та відображати її на будь-якому пристрої.

CSS framework - це набір або декілька файлів з готовим кодом, який підключається до сторінки в головному розділі, після чого може використовуватися структура цієї структури.

Bootstrap значно спрощує компонування. Перш за все, рамки використовують різні браузери і варіанти адаптації, які є основними питаннями, яким повинен звернутися програміст. Bootstrap робить його дуже легко реалізувати.

По-друге, структура ідеально підходить для колективної роботи. Розміщення початкового завантаження з правильними навичками та розумінням займає в 3-5 разів швидше, а однорідність коду дозволяє кожному внести зміни. За відсутності рамки кожен розробник може мати свій власний стиль, а інша людина буде витрачати час на вивчення його коду.

Перевагою цієї структури є наявність великої громади і наявність хорошої документації. Через цю перевагу Bootstrap з'являвся шаблони, де вже витончений дизайн всіх основних елементів. На основі цих шаблонів можна внести лише невеликі зміни на веб-сайті.

Помилки завантаження

1. Код у бібліотеці написаний для кожного випадку, у випадку дизайну більше, ніж потрібно.

2. Дизайн шаблону. Цю проблему можна легко вирішити, оскільки вона буде мати місце лише тоді, коли ви використовуєте тільки готові компоненти скелета і нічого не робите.

Будівельні компоненти

Bootstrap є всеосяжною платформою, має багато встановлених компонентів, все, що може знадобитися для створення типових сайтів, таких як розкривні меню, кнопки, закладки, індикатори стану, трикутник, букви, назви тощо. додається до документів HTML.

Рамка містить знаковий шрифт, що полегшує роботу з іконками, тому вам не потрібно завантажувати ці піктограми у вікні перегляду зображень. За замовчуванням Bootstrap має близько 200 значків;

Початок роботи з кадрами

Ви, звичайно, повинні почати відвідування офіційного сайту getbootstrap.com. Сторінка "Перші кроки" або "Перші кроки" дозволяє завантажувати кадр одним із способів.

Крім того, ви можете приєднати необхідні файли через CDN, необхідно зареєструватися на головній сторінці і зв'язати ці файли з архівом cdn.

Щоб налаштувати структуру, використовуйте лише деякі з його компонентів, перейдіть до http://getbootstrap.com/customize. Він повинен буде вибрати компоненти, які він буде містити. За допомогою такого підходу ви можете позбутися непотрібного коду і зберегти тільки те, що вам потрібно.

Сторінка налаштування дуже довга, і ви можете встановити багато параметрів, наприклад кольори за замовчуванням для різних компонентів тощо.

JavaScript framework

JavaScript фреймворки - цеінструменти для побудови динамічних веб/мобільних/настільних додатків на Javascript. Розробникивикористовують js-фреймворки там, денеможливо/складно/довго виконуватизавдання звичайними засобами. У переважній більшостівипадків, фреймворки використовуютьсядля написання, так званих, Single Page Applications.

Односторінковий веб-застосунок - цевеб-застосунок чи веб-сайт, якийвміщується на одній сторінці. Водносторінковому веб-застосунку весь необхідний код - HTML, JavaScript, та CSS - завантажується разом ізсторінкою або динамічно довантажується запотребою, зазвичай, у відповідь на діїкористувача. Сторінка неоновлюється і не переправляє користувача на іншу сторінку. Взаємодія з застосунком часто здійснюється черездинамічний зв'язок з веб-сервером.

За допомогою JavaScript можуть бути розроблені як повні веб-сайти, так і функціональні модулі (онлайн-інструменти). Зазвичай, повні кадри більше підходять для першого завдання, а для інших рекомендується використовувати більш світлі кадри або бібліотеки.

Рамки забезпечують прозору структуру програми та реалізуються за допомогою програми Найчастіше використовуються MVC (Model-View-Controller), Model-View-Presenter (MVP) і Model-View-ViewModel (MVVM).

Переваги побудови програми в рамках JS:

- Ефективність - Проекти, розроблені протягом багатьох місяців з сотнями рядків коду, тепер можуть бути реалізовані набагато швидше з добре підготовленими готовими шаблонами та функціями. Код стає значно меншим і чистішим, що позитивно впливає на швидкість розвитку, а також на підтримку і усунення помилок програмного коду.

- Безпека - найкращі структури JavaScript мають фірмову систему безпеки і підтримуються великою спільнотою, члени та користувачі якої виступають лише тестерами.
- Витрати - більшість кадрів відкриті і безкоштовні. Оскільки вони допомагають програмістам швидше розвивати власні рішення, остаточна ціна веб-програми буде нижчою. Ви можете легко реалізувати SPA (Single Page Application). Наявність структури означає модульність програми, що полегшує роботу багатьох програмістів одночасно.
- Можливість швидко створити мобільний або комп'ютерний мультиплатформовий додаток з веб-версії за допомогою PhoneGap або Apache Cordova.

Існує багато програм на js-фреймворках, і цей сегмент швидко зростає.

Таблиця 1.1 - Популярні JS-фреймворки

	AngularJS	Angular 2	ReactJS	Vue.js	Ember.js	Meteor.js
Definition	MVW framework	MVC framework	JavaScript library	MVC framework	MVC framework	JavaScript app platform
1st Release	2009	2016	2013	2014	2011	2012
Homepage	angularjs.org	angular.io	reactjs.net	vuejs.org	emberjs.com	www.meteor.com
# Contributors on GitHub	1,562	392	912	62	636	328
GitHub Star Rating	54,402	19,832	57,878	39,933	17,420	36,496

Angular

Angular.js часто називають MVW (Model-View-Whatever) фреймворком, а основні переваги для нових і середніх компаній включають: швидке кодування, швидке тестування всіх частин програми та двосторонні канали передачі даних

(зміни, які миттєво відображаються у користувачеві) Інтерфейси), В даний час найпопулярнішою односторонньої прикладною рамкою для односторінкових додатків (додатків з одним SPA-сайтом) і є найбільша спільнота програмістів.

Angular2 має великий список функцій, які дозволяють розробляти все, починаючи від Інтернету, до настільних комп'ютерів і мобільних додатків. Рамка побудована на Microsoft TypeScript. Angular2 включає в себе архітектуру на основі компонентів, вдосконалену DI (ін'єкцію залежностей), ефективні служби журналів, взаємодії компонентів і багато іншого.

Обидві версії Angular є найкращим вибором для високоякісних корпоративних додатків або середовищ програмування, перш ніж читати код.

ReactJS

Це бібліотека, а не фреймворк і доводить свою ефективність в динамічних додатках з високим трафіком (пропускною здатністю) як приклад Facebook і Instagram. Це найшвидше зростаюча платформа JS, сьогодні існує близько 1000 авторів Github.

У MVC (Model-View-Controller) React.js працює як "V" і може бути легко інтегрована в будь-яку архітектуру. Використання віртуального дерева DOM забезпечує більш високу продуктивність у порівнянні з кутом 1.x. Респонденти можуть бути створені та повторно використані в інших програмах або навіть відправлені для загального користування.

Відповідь важко вивчити, але це полегшує та спрощує створення додатків. Ідеально підходить для складних, високопродуктивних програмних рішень.

Vue.js

Vue 2.0 був введений в 2016 році, ввів найкраще в Ember, React і Angular. Vue.js пропонує двонаправлене підключення до даних (як у AngularJS), візуалізацію на стороні сервера (як у Angular2 та ReactJS), Vue-cli (інструмент лісів) та додаткову підтримку JSX. Автори стверджують, що Vue2 є одним з найшвидших вільних мислителів.

Vue.js буде найкращим вибором для швидкої розробки багатоплатформних додатків. Це може бути міцною основою для ефективних односторінкових додатків (SPA) та реальних рішень, де продуктивність важливіша, ніж хороша організація коду або структура програми.

Ember.Js

У 2015 році Ember була визнана найкращою структурою JS, залишивши React та AngularJS. В даний час у нього є велика спільнота розробників, регулярні оновлення та широко використовувані кращі практики в Java Script. Ember має двостороннє підключення до даних, як у AngularJS, все ще синхронізуючи вигляд і модель. Використання Fastboot.js забезпечує швидку візуалізацію (перерахунок) дерева DOM на стороні сервера, покращуючи представлення складних інтерфейсів користувача.

Emberjs широко використовується в складних багатофункціональних веб-додатках і веб-сайтах. Провідні користувачі включають Chipotle, Blue Apron, Nordstrom, Kickstarter, LinkedIn, Netflix та багато іншого. Більш того, найпростіше навчитися і має багато навчальних посібників та онлайн-каталогів.

Meteor.js

Meteor є одним з найпопулярніших JavaScript фреймів, який має багато функцій для створення бекенда і переднього коду відтворення, управління базами даних і бізнес-логіки. З моменту його випуску в 2012 році його екосистема різко зросла. Ця повнофункціональна платформа дозволяє швидко створювати веб- та мобільні додатки. З точки зору продуктивності, всі зміни в базі даних негайно передаються користувальницькому інтерфейсу і назад без значних втрат часу через відмінності в мові або у відповідь на серверний час.

Meteor.js охоплює всі етапи циклу розробки програмного забезпечення і використовується для створення веб-додатків реального часу, таких як Mazda, IKEA, Honeywell і багато інших.

1.2 Базовий захист JavaScript фреймворків

Інтернет-додатки постійно піддаються різним загрозам. Більше того, найсерйознішою загрозою є те, що вони є хакерами та вірусами. Перший може отримати доступ до конфіденційної інформації, розміщення серверів, хакерських сайтів і заміни їх контенту, а також порушити трафік сервера через розподілені атаки (DDoS атаки). Однак віруси, які заражають веб-сервери, перетворять їх на інфекції розсади. Крім того, вони значно сповільнюють його роботу, а також займають інтернет-канал. На перший погляд, ці загрози, здається, в основному різні. Насправді це не зовсім так. Виявляється, багато вірусів, особливо інтернет-хробаків, використовуються для поширення прогалин у програмному забезпеченні. Хакери також віддають перевагу атакам, націленим на відомі отвори в програмному забезпеченні. І в цьому немає нічого особливого.

Використовуючи слабкі місця безпеки, обидва вони отримують відносно легкий доступ до віддаленого комп'ютера, навіть якщо він добре захищений.

Практично у кожній програмі є прогалини. І оскільки його вихідний код більший, то більше він може знайти "отвори". Розрив дуже легко пояснити. Людина не машина, він може помилятися. Існує навіть спеціальне правило програмування, яке визначає, скільки помилок експерт може допустити при написанні декількох рядків коду. Крім того, пам'ятайте, що велике програмне забезпечення написано не однією людиною, а цілою групою. Часто виникають помилки при створенні модулів, створених різними програмістами. Крім того, наявність уразливості не завжди залежить від якості написаного програмного забезпечення.

З точки зору уразливості в інтернет-додатках, переважна більшість людей відразу згадує "дірки" у своєму програмному забезпеченні. Це стосується самих серверних програм, таких як Apache, Microsoft Internet Information Server тощо. У цьому немає нічого дивного. Проте, програмне забезпечення досить обширне і складне, тому "дірка" в ній завжди доступна. Крім того, ми не можемо забувати,

що сучасний веб-сервер не можна уявити без багатьох інших функцій, таких як мови програмування, такі як Perl, PHP тощо. Все це можливо за допомогою встановлення додаткового програмного забезпечення в веб-додатку. Він також може містити отвори безпеки.

В даний час сайти інформаційної безпеки постійно повідомляють про нові помилки в програмному забезпеченні веб-додатків. Цей процес включає захист даних і хакерів. А інша, яка виявляє нову "дірку", може заспокоїти її і спробувати використати її для своїх власних цілей. Але вірно протилежне. Хакер намагається говорити про нові вразливості для всіх, включаючи розробників програмного забезпечення.

Головною особливістю виробничих прогалин є їх залучення до конкретних версій програмного забезпечення. Справа в тому, що "дірки" часто не зустрічаються в ряді веб-додатків, а лише в деяких їх виданнях. Однак слід зазначити, що чим популярнішим є програмне забезпечення, тим більше шансів знайти нові уразливості.

І це не залежить від якості написаного програмного забезпечення. Це просто розглядається рядом фахівців, тому ймовірність виявлення «дірок» є відносно високою.

З метою захисту від типу програмного забезпечення, що оцінюється, уразливості можуть бути лише одним способом - своєчасне встановлення всіх виробників розробило оновлення та виправлення. Справа в тому, що програмісти регулярно дізнаються про офіційні оновлення своїх продуктів. Коли виявлено критичне отвір безпеки, патч швидко звільняється. Якщо прогалини, виявлені знову, є більш теоретичними, ніж реальна загроза, масові оновлення звільняються, коли вони накопичуються.

Проблеми з безпекою можуть виникати через неправильні налаштування програмного забезпечення веб-застосунку.

Можливо, не секрет, що безпека будь-якої речі залежить від того, як ви її використовуєте. Те ж саме можна сказати і про веб-додаток. Багато що залежить

від конфігурації програмного забезпечення. Загалом, переважна більшість веб-додатків мають досить великий набір параметрів, які охоплюють практично всі аспекти їхнього бізнесу. Безпека, отже, значною мірою залежить від адміністраторів, що беруть участь у їхніх послугах. Але ми не можемо забувати, що адміністратори - це люди. А це означає, що вони можуть помилятися через свою неуважність, відсутність кваліфікації або з будь-якої іншої причини. І ці помилки можуть відкрити шлях до хакерів або вірусних веб-серверів.

Правильні налаштування не допоможуть вам встановити виправлення. Насправді, при оновленні програмного забезпечення його конфігурація не змінюється. Це означає, що існує ймовірність помилки в системі захисту після встановлення патча. Головною загрозою для типу "дірок" вважається тому складність їх знаходження. Єдиний спосіб надійно захистити від таких розривів - використання спеціальних сканерів безпеки. Ці програми, використовуючи спеціальні методи, досліджують захист веб-серверів і знаходять всі потенційно небезпечні сайти.

Веб-скрипти також можуть містити дірки в захисті.

Сучасні інтернет-програми та супутнє програмне забезпечення часто створюють специфічну основу для реалізації програм, написаних користувачем. Звичайно, це сценарії, які працюють на більшості сучасних веб-сайтів. Справа в тому, що більшість мов веб-програмування є серверними. Це означає, що сценарії, написані на них, запускаються безпосередньо на сервері, і тільки результати їх роботи (в даному випадку відвідувача сайту) надсилаються на комп'ютер користувача.

Це дуже серйозна небезпека. Справа в тому, що сценарії веб-сайту не завжди розробляються дійсно хорошими фахівцями. Багато інтернет-проектів використовують безкоштовне або самостійне програмне забезпечення. Звичайно, він може також містити пробіли. Деякі з них можуть бути дуже серйозними, дозволяючи зловмисникам отримати несанкціонований доступ до самого сервера. Крім того, слід враховувати, що деякі сценарії виконуються з

підвищеними привілеями. Тому прогалини в них можуть бути гарним інструментом для хакерів.

Ви можете знайти "скрипти" у скриптах, використовуючи сканери безпеки. Тому кожен власник веб-сервера дійсно піклується про безпеку свого сайту, він повинен регулярно перевіряти його. І слово "звичайний" у попередньому реченні не було випадковим. Справа в тому, що хакери постійно придумують нові способи атаки на віддалену атаку. Крім того, в оригінальному програмному забезпеченні постійно виявляються нові "діри", які, у поєднанні зі скриптами, які раніше вважалися безпечними, становлять реальну загрозу.

Покращення інформаційної безпеки при використанні новітніх інформаційних технологій є однією з стратегічних цілей розвитку інформаційного суспільства в Україні.

Суть інформаційної безпеки полягає в захисті інформаційного середовища, особистості, суспільства і держави від навмисних і ненавмисних загроз і впливів. Тому явище спаму можна вважати одним з джерел його загроз.

Зокрема, загрози для технологій спаму інформаційної безпеки полягають у наступному: складність інформаційних систем і ресурсів через непотрібний пошук інформації, витрачання багато часу на аналіз спаму і психічний дискомфорт, пов'язаний з подальшим видаленням багатьох повідомлень, ризик видалення необхідного спаму, що може призвести до різні види неприємних наслідків, висока ймовірність реалізації різних підроблених програм, які можуть бути жертвами як приватного, так і юридичного CSSR, одержувач оплачує спам або роботу провайдера, який отримує небажані листи з поштового сервера. [3]

Небажана пошта існує тому, що існуючі умови для її існування є законними, зокрема відсутність обов'язкового закону, що забороняє його розповсюдження та заборону економічної діяльності. Включення спаму як суб'єкта обміну інформацією також дає можливість розрізняти його суб'єктів: клієнта, творців (дистриб'юторів) і споживачів [4].

Правовими (правовими) методами боротьби зі спамом можуть бути: прийняття положення про заборону спаму, створення спеціального підрозділу для виявлення та переслідування спамерів на основі служби захисту інформації, що надає постачальникам поштових послуг певні обов'язки та дозволи на фільтрування пошти [5].

Існують подібні системи:

- CleanTalk: захист від завантаження та спаму в коментарях, анти-спам-онлайн форум без captcha.

- Анти-спам ACP: інструмент пошуку IP, список спаму, перевірка профілю, підозрілі виправлення користувачів.

Висновки до розділу 1

Ми живемо в той час, коли безперервний розвиток інтернету та інформаційних технологій в цілому зробили так, що навіть маленька компанія, не кажучи вже про великі корпорації та державні установи, має свій веб-застосунок.

Стандартом сьогодні є односторінкові веб-застосунки. Вони створюються за допомогою фреймворків.

Захищеність веб-застосунків – це відкрите питання, для дослідження якого потрібно більше детально розглянути основні вразливості та обрати декілька з них для аналізу використовуваних ними лазівок.

2 АНАЛІЗ МЕТОДІВ ЗАХИСТУ JAVASCRIPT ФРЕЙМВОРКІВ, ЯКІ ВИКОРИСТОВУЮТЬСЯ ПІД ЧАС РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ ВІД НАЙПОШИРЕНІШИХ АТАК

2.1 Основні вразливості JavaScript фреймворків

На рисунку 1.2 зображено статистику веб-додатків за високою та середньою ступінню ризику вразливості, що була проведена міжнародною компанією, що спеціалізується на розробці програмного забезпечення у сфері інформаційної безпеки Positive Technologies [11]. Високий ступінь позначений червоним кольором, середній ризик позначений жовтим кольором.

Рисунок 2.1 - Статистика веб-додатків за ступенями ризику вразливості

Як видно в рисунку 2.1, платформа має найменшу частку веб-додатків за високим ризиком вразливості.

Якщо розглянути тенденцію щодо частки сайтів, а саме – їх схильності до різних атак та врахувати мову програмування, на якій цей застосунок створено – побачимо наступне (див табл.2.1). [28]

Таблиця 2.1 - Показники схильності сайтів до атак за критерієм «мова програмування»

PHP	Частка сайтів, %	Java	Частка сайтів, %	ASP.NET	Частка сайтів, %
Cross Site Scripting	90	Cross Site Scripting	80	Cross Site Scripting	73
Credential/Session Prediction	86	Fingerprinting	60	Brute Force	73
Brute Force	81	Brute Force	45	Fingerprinting	55
Information Leakage	67	Credential/Session Prediction	45	Cross Site Request Forgery	55
SQL Injection	62	Server Misconfiguration	35	Credential/Session Prediction	45
Fingerprinting	43	Information Leakage	30	Information Leakage	45

Загальномовне середовище виконання CLR і платформа .NET Framework надають багато корисних класів і служб, які дозволяють розробникам легко створювати безпечний код, а адміністраторам – налаштовувати доступ до захищених ресурсів. Крім того, середовище виконання і платформа.NET надають корисні класи і служби, що дозволяють використовувати шифрування і безпеку на основі ролей, що і буде реалізовано в захищеному веб-додатку. Систему безпеки в .NET можна представити у вигляді декількох функціональних блоків (Рисунок 2.2).

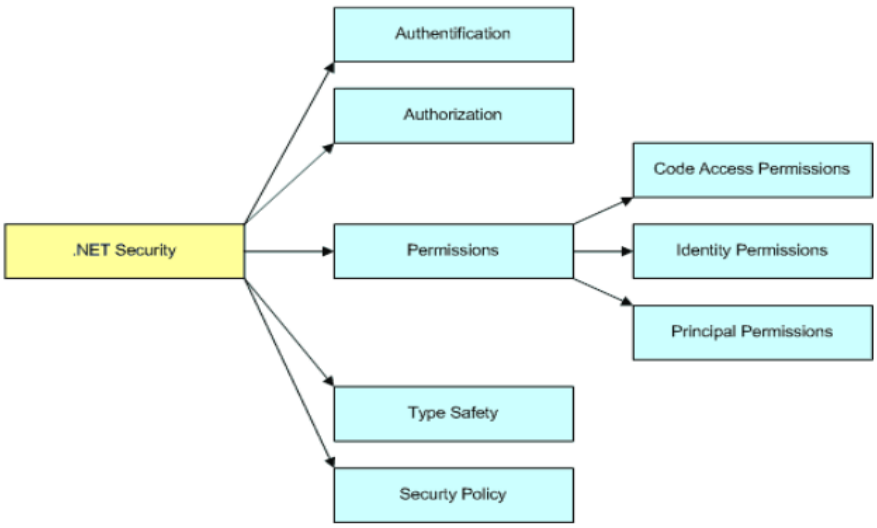

Рисунок 2.2 - Система безпеки

Робота і управління цими службами доступні як адміністраторам, так і прикладним програмістам. В .NET Framework визначено чотири групи політик безпеки (Policy Level): Enterprise, Machine, User і Application Domain. Кожен з рівнів містить свій набір правил, на основі яких визначається які права можна надати коду.[29]

Рівень Enterprise поширюється на кожен комп'ютер в домен і зазвичай управляється адміністраторами домену. За замовчуванням на цьому рівні всім надаються повні права.

Рівень Machine поширює свою дію на весь комп'ютер і може адмініструватися локальним адміністратором або адміністратором домену. За замовчуванням саме на цьому рівні визначається більшість налаштувань безпеки.

Рівень Personal використовується для управління настройками безпеки поточного користувача. За замовчуванням, як і на рівні Enterprise, всім надаються повні права.

Рівень AppDomain –спеціальний рівень. Особливість даного рівня в тому, що будь-які налаштування можна змінити лише програмно.

Захист веб-додатків ASP.NET полягає в тому, що платформа спільно з інформаційними службами Інтернету (IIS) може виконувати аутентифікацію облікових даних користувача, наприклад, імена та паролі, використовуючи один з наступних методів аутентифікації.

1) Windows: стандартна, зашифрована або вбудована автентифікація Windows (NTLM або Kerberos).

2) Аутентифікація у формах, в яких розробник створює сторінку входу та керує перевіркою додатків.

3) Аутентифікація сертифіката клієнта.

ASP.NET контролює доступ до інформації на веб-сайті, порівнюючи перевірені повноваження або їхню відповідність до дозволів файлової системи

NTFS або XML-файл, який містить уповноважених користувачів, уповноважені ролі (групи) або авторизовані команди HTTP.

Інтернет-додатки є однією з найбільш вразливих систем. Чим більш критичним і чутливим є магазин магазинів програмного забезпечення, тим важливішим є контроль їх безпеки.

Забезпечення високої доступності додатків, високоякісних серверних рішень і захисту конфіденційних даних забезпечує надійне та безпечне програмне забезпечення. Використання методів захисту Інтернет-додатків від потенційних атак полягає в запобіганні потенційним помилкам безпеки. Особливу увагу слід звернути на запобігання атакам XSS, атакам CSRF та ін'єкціям SQL для вашого власного веб-додатка. [8] Оскільки ці типи атак найбільш небезпечні, якщо критерієм прийняття домену є інтернет-домен (інформаційні технології).

Останніми роками, завдяки діяльності членів спільноти WWW, було створено величезну кількість CID. Конкурент проаналізував визначення CID на існуючому веб-сайті та визнав їх високоспеціалізованими та суперечливими. Запропоновано нове концептуальне визначення: KIN to IN, який створив користувача веб-сайту, який не є представником організації власника і розміщений на веб-сайті як частина творчого, комунікативного, рекламного чи іншого процесу без попереднього професійного редагування.

Аутентифікація - це перевірка ідентифікатора користувача. Механізм реєстрації і реєстрації використовує модуль passport.js. Забезпечує параметри аутентифікації для проміжного програмного забезпечення Node.js. Надзвичайно гнучкий і модульний паспорт може бути легко доданий і налаштований до будь-якого веб-додатка для експрес-додатків. Комплексний набір аутентифікації з іменем користувача та паролем, Facebook, Twitter і багато іншого. Для цього необхідно створити необхідні контури в файлі конфігурації системи і необхідні об'єкти в контролерах.

Під час доступу до програми будь-який користувач, який не ввійшов до системи, буде перенаправлено на сторінку входу.

Функція входу в систему така:

```
/**
 * `UserController.login()`
 */
login: function (req, res) {
  return res.login({
    successRedirect: '/'
  });
},
```

Рисунок 2.3 - Функція входу в систему

Також, якщо користувач вже залогінився, тобто увійшов в систему, він може розлогінитись, тобто вийди із системи:

```
/**
 * `UserController.logout()`
 */
logout: function (req, res) {
  req.logout();
  return res.ok('Logged out successfully.');
},
```

Рисунок 2.4 - Функція виходу

В разі якщо користувач не зареєстрований, то йому буде запропоновано пройти реєстрацію, для цього використовується відповідна функція:

```
/**
 * `UserController.signup()`
 */
signup: function (req, res) {
  User.create(req.params.all()).exec(function (err, user) {
    if (err) return res.negotiate(err);
    req.login(user, function (err){
      if (err) return res.negotiate(err);
      return res.redirect('/welcome');
    });
  });
}
```

Рисунок 2.5 - Функція реєстрації

Всі маршрути налаштовуються в файлі «config/routes.js», приклад зображено на рисунку 1.7:

```
module.exports.routes = {
  '/': { view: 'homepage' },
  'get /login': { view: 'user/login' },
  'get /signup': { view: 'user/signup' },
  '/welcome': { view: 'user/welcome' },
  'post /login': 'UserController.login',
  'post /signup': 'UserController.signup',
  '/logout': 'UserController.logout'
}
```

Рисунок 2.6 - Маршрути для системи автентифікації

Крос-сайт сценарії (XSS)

Крос-сайт сценірій - це часто використовувана техніка, яка дозволяє запускати зовнішній JavaScript у контексті атакованого веб-сайту. XSS дозволяє отримувати доступ до повного веб-API. Найпростіший приклад атаки XSS передбачає, що хакер знаходить уразливе поле введення на сторінці і створює посилання, яке він вставляє на іншу сторінку.

XSS є високо оціненою вразливістю, оскільки зловмисник може отримати доступ до LocalStorage, SessionStorage або cookies. Тому рекомендується не зберігати будь-які конфіденційні дані в цих сховищах.

Є також вразливість XSS яка виникає, коли користувальницький вхід з URL або даних POST відображається на сторінці, не зберігаючись, що дозволяє зловмиснику вводити шкідливий вміст. Це означає, що зловмисник має надіслати пошкоджену URL-адресу або форму для повідомлення жертві, щоб вставити корисну навантаження, і жертва повинна натиснути посилання. Цей тип корисного навантаження також зазвичай вловлюється вбудованими фільтрами XSS у браузерах користувачів, наприклад, Chrome, Internet Explorer або Edge.

ContentSecurityPolicy

Політика захисту конфіденційності або вмісту - це інший рівень безпеки, який допомагає виявляти та пом'якшувати певні типи атак, включаючи міжсайтові скрипти (XSS) та ін'єкції даних. Ці атаки використовуються для усього, від крадіжки даних до пошкодження сайту та розповсюдження шкідливих програм.

Правила можуть бути вказані за допомогою HTTP-відповідей або тегів <meta> HTML.

Таблиця 2.2 - Заголовки HTTP

Заголовок	Веб-браузер
Content-SecurityPolicy	(W3C Standard header) Chrome version >= 25, Firefox version >= 23, Opera version >= 19
X-Content-SecurityPolicy	Firefoxversion< 23, IE version 10

Директиви, які підтримуються наведені в таблиці 2.3:

Таблиця 2.3 - Директиви CSP

default-src	Завантаження політики для всіх типів ресурсів
script-src	Визначає, які скрипти може виконати захищений ресурс
object-src	Визначає, звідки захищений ресурс може завантажувати плагіни
style-src	Визначає, якістилі (CSS) користувач використовує до захищеного ресурсу
img-src	Визначає, звідки захищений ресурс може завантажувати зображення
media-src	Визначає, звідки захищений ресурс може завантажувати відео і аудіо
frame-src	Визначає, звідки захищений ресурс може вставляти кадри
font-src	Визначає, звідки захищений ресурс можез авантажувати шрифти
form-action	Визначає, які ідентифікатори URI можуть бути використані в якості дії елементів форми HTML

Cross-OriginResourceSharing

Cross-OriginResourceSharing (CORS) - це специфікація технології веб-переглядача, яка описує, як мережеві служби можуть надавати сценарії з домену, який не відповідає політиці одного джерела. CORS - це сучасна альтернатива для JSONP. JSONP підтримує лише GET, CORS та інші запити. Використання CORS дозволяє веб-розробникам використовувати простий XMLHttpRequest, що дозволяє краще обробляти помилки, ніж JSONP. З іншого боку, JSONP працює на старих браузерах, які не мають підтримки CORS. CORS підтримує більшість сучасних браузерів.

Щоб увімкнути перехресні запити з білого списку довірених доменів з будь-яким контуром додатку, потрібно зареєструвати всі маршрути та вказати налаштування в розділі "config / cors.js."

Щоб увімкнути запити з будь-якого каналу на будь-якому маршруті, введіть параметри, як показано на рисунку 2.7:

```
allRoutes: true,
origin: '*',
credentials: false
```

Рисунок 2.7 - Файл конфігурації CORS

Крім глобальної конфігурації CORS, ще можна налаштувати всі ці параметри базуючись на параметрі кожного маршруту в «config / routes.js».

```
'POST /signup': {
   controller: 'UserController',
   action: 'signup',
   cors: false
}
```

Рисунок 2.8 - Налаштування для окремого маршруту

Cross-siterequestforgery

CSRF - це "фальсифікація сторінок", також відома як XSRF, яка є своєрідною атакою користувачів сайту, які використовують помилки HTTP. Якщо жертва потрапляє на веб-сайт, створений зловмисником, особа таємно відправляється на інший сервер (наприклад, сервер платіжної системи), який виконує шкідливу операцію (наприклад, обліковий запис атакуючого). Для виконання цієї атаки потерпілий повинен бути аутентифікований на сервері, на який було надіслано запит, і запит не повинен вимагати будь-якого підтвердження користувача, який не може ігнорувати або фальсифікувати атакуючий сценарій.

Розроблена система підтримує захист від такого виду атаки для того, щоб увімкнути її. Необхідно налаштувати файл "config / csrf.js" таким чином:

```
csrf: true
```

Рисунок 2.9 - Активований захист від CSRF атак

Потім необхідно створити маркер CSRF і включити його як параметр або заголовок у ці необхідні запити.

Цей веб-сервіс сумісний із середнім програмним забезпеченням для захисту від таких атак. Це проміжне програмне забезпечення реалізує Token Pattern. Якщо увімкнено захист CSRF, будь-які запити, які не передані на сервер Sails, повинні бути доповнені спеціальними тегами, які були визначені заголовками або параметрами в рядку запиту або вмісті HTTP.

2.2 Досліджувані фреймворки і атаки на них

Для подальшої роботи потрібно обрати три JavaScript фреймворки та найпоширеніші атаки, які використовують найбільше лазівок у захисті фреймворків, для того, щоб провести детальний аналіз цих лазівок та написати атаки базовані на них.

Фреймворки дозволяють створювати веб-сайти, які діють як настільні програми, але виконуються в браузері. Таким чином, з часом такі сайти стали називатись веб-застосунками.

Сам фреймворк має гарний базовий захист, але все ж в цьому захисті є свої недоліки та дірки. Тому обрання фреймворків та атак проводилося за такими критеріями:

1. Актуальність – фреймворки повинні використовуватися для розробки сучасних односторінкових веб-застосунків а атаки повинні використовуватися для нападів на ці веб-застосунки;

2. Розповсюдження – фреймворк і атака повинні використовуватися в багатьох проектах і повинні бути поширені.

Виходячи з цих критеріїв з декількох актуальних, для наступного аналізу було обрано такі фреймворки і атаки:

Обрані фреймворки:

1. Angular.js
2. React.js
3. Vue.js

Обрані атаки:

1. XSS – найпоширеніша атака серед усіх вразливостей
2. XSSI – модернізована форма атаки XSS

2.3 Аналіз запобігання крос-сайт сценаріїв (XSS) у обраних JavaScript фреймворках

Перехресний сценарій (XSS) дозволяє зловмисникам вводити шкідливий код на веб-сторінки. Потім такий код може, наприклад, викрасти дані користувача (зокрема, дані для входу) або виконати дії для виокремлення користувача. Це одна з найпоширеніших атак в Інтернеті.

Щоб заблокувати XSS-атаки, необхідно запобігти потраплянню шкідливого коду до DOM (Document Object Model). Наприклад, якщо зловмисники можуть підвести вас до вставки тега <script> у DOM, вони можуть виконувати довільний код на вашому веб-сайті. Атака не обмежується тегами <script> - багато елементів і властивостей DOM дозволяють виконувати код, наприклад, <img onerror = "..."> і <a href="javascript:...">. Якщо контрольовані зловмисниками дані надходять у DOM, очікують уразливості безпеки.

Angular

Щоб заблокувати XSS-атаки в Angular, необхідно запобігти введенню зловмисного коду до DOM. Коли значення вставляється в DOM з шаблону, через властивість, атрибут, стиль, прив'язку класу або інтерполяцію, Angular дезактивує і видаляє недостовірні значення.

Дезинфекція - це перевірка ненадійного значення, перетворюючи його на значення, яке можна безпечно вставити в DOM

Метою DomSanitizer є очищення ненадійних частин значень. Скелет класу виглядає так:

```
export enum SecurityContext { NONE, HTML, STYLE, SCRIPT, URL, RESOURCE_URL }

export abstract class DomSanitizer implements Sanitizer {
  abstract sanitize(context: SecurityContext, value: SafeValue|string|null): string|null;
  abstract bypassSecurityTrustHtml(value: string): SafeHtml;
  abstract bypassSecurityTrustStyle(value: string): SafeStyle;
  abstract bypassSecurityTrustScript(value: string): SafeScript;
  abstract bypassSecurityTrustUrl(value: string): SafeUrl;
  abstract bypassSecurityTrustResourceUrl(value: string): SafeResourceUrl;
}
```

Рисунок 2.10 - DOMSanitizer.

Як ви можете бачити, існують два типи шаблонів методів. Перший - це метод bypassSecurityTrustX, який отримує ненадійне значення відповідно до використання значення і повертає довірений об'єкт (про це ми поговоримо пізніше). Другий - метод sanitize, який отримує контекст безпеки і ненадійне значення і повертає довірене значення. Контекст безпеки - це значення.

Якщо для контексту довіряється значення, цей метод очищення розгортає вміст, що міститься, та використовує його безпосередньо. В іншому випадку значення буде дезінфіковане, щоб бути безпечним відповідно до контексту безпеки.

У конкретних ситуаціях може бути необхідно вимкнути санітарну обробку, наприклад, якщо програма дійсно потребує створення стилю `javascript:` з динамічним значенням. Користувачі можуть обійти безпеку, побудувавши значення з одним з методів bypassSecurityTrustX, а потім прив'язавши до цього

значення з шаблону. І саме тоді, коли DomSanitizer вимкнений Angular відкритий для XSS атак через змінні HTML. Приклад атакуючого скрипта:

```
interface SafeValue {}
interface SafeHtml extends SafeValue {}

abstract class SafeValueImpl implements SafeValue {
  abstract getTypeName(): string;
  ...
}

class SafeHtmlImpl extends SafeValueImpl implements SafeHtml {
  getTypeName() { return 'HTML'; }
}

class DomSanitizerImpl extends DomSanitizer {
  bypassSecurityTrustHtml(value: string): SafeHtml {
    return new SafeHtmlImpl(value);
  }
}
```

Рисунок 2.11 - XSS атака на Angular за допомогою HTML.

React

Проект Open Web Application Security, на щастя, має великий вибір ресурсів про запобігання XSS . Щоб уникнути цієї уразливості, потрібні деякі заходи безпеки в програмах:

1. Всі користувальницькі вхідні дані повинні мати виділені HTML-об'єкти. React запобігає більшості вразливостей XSS через DOM-вузли і текстовий вміст.

2. При серіалізації стану на сервері, який буде відправлено клієнту, необхідно його серіалізувати таким чином, щоб уникнути HTML-сутностей. Це пояснюється тим, що для створення рядку не використовувався React, а значить, що рядок не буде автоматично виведено. Однак і ця вразливість може бути знівельована використанням модуля Serialize JavaScript, який скинути до будь-якої програми.

Спочатку необхідно встановити модуль за допомогою: npm install --save serialize-javascript

Тоді ми можемо змінити фрагмент з попередніх, щоб виглядати наступним чином.

Замість JSON.stringify, призначаючи значення __PRELOADED_STATE__

```
var serialize = require("serialize-javascript")

function renderFullPage(html, preloadedState) {
  return `
    <!doctype html>
    <html>
      <head>
        <title>Redux Universal Example</title>
      </head>
      <body>
        <div id="root">${html}</div>
        <script>
          window.__PRELOADED_STATE__ = ${serialize(preloadedState, { isJSON: true })}
        </script>
        <script src="/static/bundle.js"></script>
      </body>
    </html>
    `
}
```

Рисунок 2.12 - Виправлений фрагмент, який видаляє уразливість XSS за допомогою модуля JavaScript Serialize JavaScript.

Що стосується отримання даних на стороні клієнта, то він працює точно так само, як і раніше, за винятком того, що тепер можна використати будь-які HTML-об'єкти в рядку (вони будуть виглядати так: \\u003C\\u002Fscript\\u003E як </script>)

Vue

Спробуємо виконати основну атаку XSS на директиву v-html :

```
 <div id = "app">
Ласкаво просимо:
 <span v-html = "attack">
 </span>
</ div>
 new Vue ({
 el: '#app',
```

```
дані: {
атака: "john doe",
}
});
```

Спробуємо ввести різні рядки в директиву v-html і перевірити, чи можемо витягти файли cookie.

```
new Vue ({
el: '#app',
дані: {
attack: '<script> alert (document.cookie) </ script>',
}
});
```

Нічого не відбувається. Браузер перешкоджає виконанню введених тегів скриптів після початкового завантаження сторінки.

Спробуємо ввести в події HTML-теги.

```
new Vue ({
el: '#app',
дані: {
атака: '<a onmouseover=alert(document.cookie)> натисніть </a>',
}
});
```

Використовуємо подію onmouseover для отримання файлів cookie.

Ось перша можлива атака XSS з Vue.

```
new Vue ({
el: '#app',
дані: {
attack: '<img src = "validUrl" onload = alert (document.cookie)>',
}
});
```

```
        new Vue ({
       el: '#app',
         дані: {
         attack:  '<img  src  =  "notValidUrl"  onerror  =  alert
(document.cookie)>',
         }
        });
```

Вони також працюють. Тоді просто потрібно уникнути v-html.

XSS в Github Vue.js 23, тепер ми можемо спробувати шукати можливі атаки XSS у Vue.js. Фактично, ми можемо виконувати деякі ін'єкції, не використовуючи безпосередньо v-html. Тепер спробуємо атакувати його. В основному, хочемо мати можливість виконувати шкідливу XSS при використанні компонентів інтерфейсу.

Використання зовнішньої бібліотеки користувальницького інтерфейсу без перевірки використовуваних методів візуалізації ризикує стати XSS.

Щоб уникнути атак XSS необхідно скористатися максимально можливим інтерпольованим виразом {{}} , вони підлягають вирівнюванню і не можуть бути виконані браузером.

2.4 Аналіз запобігання включення міжсайтового сценарію (XSSI) у обраних JavaScript фреймворках

XSSI - це форма XSS, яка використовує той факт, що браузери не заважають веб-сторінкам включати ресурси, такі як зображення та скрипти, які розміщуються на інших доменах і серверах. Сценарії, наприклад, можуть надавати функціональні можливості, які розробник повинен створити для певної сторінки - багато сайтів, включаючи цей, включають бібліотеку jQuery JavaScript, розміщену на https://developers.google.com/speed/libraries/#jquery . Ця умова, однак, може бути використана для читання даних користувача з одного домену, коли цей користувач отримує доступ до іншого домену. Наприклад,

якщо сайт Bank ABC має скрипт, який читає інформацію про особистий обліковий запис користувача, хакер може включити цей скрипт у свій зловмисний сайт (www.fraudulentbank.com), щоб витягнути інформацію з серверів банку ABC, коли клієнт банку ABC відвідує сайт хакера.

Існують різні заходи, які розробники повинні реалізувати для захисту від атак XSSI. Перший - це передавати користувачеві унікальний, непередбачуваний маркер авторизації і вимагати, щоб він був відправлений назад як додатковий параметр HTTP, перш ніж сервер відповість на будь-які запити. Сценарії повинні відповідати лише запитам POST. Це зупиняє виявлення маркера автентифікації як параметра URL у запиті GET, а також запобігає завантаженню скрипта через тег сценарію. Браузери можуть повторно видати GET-запити, що може призвести до виконання дії більше одного разу, тоді як переоформлені POST-запити вимагають згоди користувача.

При обробці відповідей JSON префікс відповіді з деяким не виконуваним префіксом, наприклад "n", щоб переконатися, що скрипт не є виконуваним. Сценарій, що виконується в цьому ж домені, може читати вміст відповіді і видаляти префікс, але сценарії, які виконуються в інших доменах, не можуть. Також не використовуйте JSONP (JSON з додаванням) для завантаження конфіденційних даних з іншого домену, оскільки це відкриває двері для фішингових сайтів, які збирають дані. Надсилання заголовка відповіді "X-Content-Type-Options: nosniff" також допоможе захистити користувачів Internet Explorer і Google Chrome від атак XSSI.

Для боротьби з XSS-атаками загалом, слід вказати CHARSET в заголовку відповіді HTTP-змісту або в http-equiv атрибуті в мета-тезі HTML-коду, щоб браузери не інтерпретували спеціальні кодування символів з інших наборів символів. Для тих, що розробляють сайти в ASP.NET, бібліотека сценаріїв антисайтників Microsoft може допомогти захистити веб-додатки від помилок між сценаріями.

Існує безліч інструментів для сканування уразливостей з відкритим кодом, які розробники можуть використовувати для перевірки того, чи їх код не відкритий для XSS-атак, таких як Vega, Wapiti, Zed Attack Proxy та Skipfish. Сайти повинні перевірятися на регулярній основі і, звичайно, щоразу, коли впроваджуються зміни базового коду, або функціональність, що спирається на сторонні бібліотеки, інтегрується в різні сторінки.

Angular

Включення міжскриптових сценаріїв, також відоме як уразливість JSON, дозволяє веб-сайту зловмисника читати дані з JSON API. Атака працює на старих браузерах, перекриваючи власні конструктори об'єктів JavaScript, а потім включаючи URL-адресу API, використовуючи тег <script>.

Ця атака є успішною, лише якщо повернений JSON виконується як JavaScript. Сервери можуть запобігти атаці шляхом префіксації всіх відповідей JSON, щоб зробити їх невиконаними, за згодою, використовуючи відомий рядок ")]} ', \ t

Бібліотека HttpClient Angular визнає цю конвенцію і автоматично видаляє рядок ")]} "," n "з усіх відповідей до подальшого аналізу.

Щоб отримати додаткові відомості, перегляньте розділ XSSI цього блогу Google для веб-безпеки.

Аудит кутових додатків

Кутові програми повинні слідувати тим же принципам безпеки, що й звичайні веб-додатки, і повинні бути перевірені як такі. Спеціальні API, які повинні бути перевірені в огляді безпеки, такі як методи bypassSecurityTrust, позначені в документації як чутливі до безпеки.

Захист REST API Spring Security

Інтерфейс REST API можна захистити за допомогою конфігурації Spring Security Java. В даному випадку цілком можна використовувати форму з автентифікацією HTTP Basic в якості резервного варіанта, а також підключити

захист від CSRF і можливість жорсткого визначення, що всі методи машинного інтерфейсу можуть бути доступні тільки через HTTPS.

Таким чином, машина запропонувала використовувати форму для входу, а після успішного втручання в нього були створені певні браузерні клієнти, але при цьому буде працювати і з іншими клієнтами, підтримуючими звичайний HTTP в авторизації.

У відповідності до рекомендацій OWASP REST-сервіси можна програмувати з мінімальним збереженням стану (вся інформація про стан сервера обмежується тим самим конфігом, що використовувався для автентифікації). Це робиться, щоб не перенести початкові дані за мережами при кожному запиті.

Приклад конфігурації безпеки REST API:

```
http
      .authorizeRequests()
      .antMatchers("/resources/public/**").permitAll()
      .anyRequest().authenticated()
      .and()
      .formLogin()
      .defaultSuccessUrl("/resources/calories-tracker.html")
      .loginProcessingUrl("/authenticate")
      .loginPage("/resources/public/login.html")
      .and()
      .httpBasic()
      .and()
      .logout()
      .logoutUrl("/logout");

 if ("true".equals(System.getProperty("httpsOnly"))) {
       LOGGER.info("launching the application in HTTPS-only
mode");
     http.requiresChannel().anyRequest().requiresSecure();
 }
```

Рисунок 2.13 - Прикад конфігурацііі безпеки REST API

Така конфігурація враховує автентифікацію лише в контексті безпеки, при цьому стратегія авторизації вибирається в залежності від вимог безпеки, попереднього API. Якщо вам потрібно буде перевіряти авторизацію, ознайомтеся з переліком контролів доступу до списку Spring Security ACL і перевірте, чи підключені вони до рішень стоянки перед вами.

Тепер порівнялємо такий спосіб створення веб-застосунку з іншими поширеними підходами.

Порівняння стека Spring MVC / Angular з іншими поширеними варіантами

Для того, щоб скористатися JavaScript в кліентській частині та Java - в роботі з базовими даними буде робочий процес і підвищить продуктивність.

Коли машинний інтерфейс вже працює, не потрібно ніяких спеціальних інструментів або плагінів, щоб розігнати гаряче розгону в кліентській частині на повну потужність: просто опублікуйте ресурси на сервері при допомозі IDE (наприклад, натисніть Ctrl + F10 в IntelliJ) і оновіть сторінку в браузері.

Класи машинного інтерфейсу можна перезавантажити за допомогою JRebel, але в клієнтській частині нічого не можна робити не потрібно. В принципі, можна витягти всю клієнтську частину, зімітувавши машинний інтерфейс при допомозі, скажем, json-server. У такому випадку різні фахівці можуть паралельно розлучати клієнтську частину і машинний інтерфейс, якщо це потрібно.

Потенціальний недолік цього підходу: ці розробки повинні знати і HTML, і CSS, і JavaScript, але в останній раз таку компетенцію зустрічається все частіше.

React

Атаки, які впливають на уразливості XSS, існують з 1990-х років, і більшість великих веб-сайтів, таких як Google, Yahoo і Facebook, були вразливі вразливостями XSS в певний момент. Атаки, засновані на XSS, відрізняються від більшості атак на прикладному рівні (наприклад, ін'єкції SQL), коли вони атакують користувачів програми, а не програми або сервера. Ці атаки працюють шляхом введення коду - зазвичай скрипту на стороні клієнта, такого як JavaScript

- у вихід веб-програми. Більшість сайтів мають численні точки ін'єкції, такі як поля пошуку, файли cookie та форми. Хоча ці шкідливі сценарії не можуть безпосередньо впливати на інформацію на стороні сервера, вони все одно можуть порушити безпеку сайту. Використовуючи маніпуляцію об'єктної моделі документа, щоб змінити значення форм, змінити вигляд сторінки або змінити дію форми, щоб розмістити подані дані на сайті зловмисника, атаки можуть вкрасти дані, взяти під контроль сеанс користувача, запустити шкідливий код або використовувати як частина фішинг-афери .

Висновки до розділу 2

Незважаючи на гарну захищеність сучасних JavaScript фреймворків існує багато вразливостей, якими кожного для користуються зловмисники. Для більше детального аналізу було обрано найпоширенішу та найбільш загрозливу вразливість серед всіх описаних вище – XSS атака та її модернізацію XSSI.

Для того, щоб дослідити яким чином діє XSS атака та якими лазівками в захисті користується, було обрано три найпопулярніші і найпоширеніші фреймворки:

1. React
2. Angular
3. Vue.js

Було проаналізовано захист кожного з обраних фреймворків від XSS та XSSI. З цього аналізу стало зрозуміло, що незважаючи на гарний захист, в кожному із фреймворків є лазівки для проведення успішних атак:

1. Асинхронна операція з відправкою запиту на API
2. HTML створений на стороні сервера
3. Підробка запиту (to-do компонент)

Базуючись на цих лазівках в наступному розділі будуть написані три типові операції з шкідливим кодом (XSS атакою) та власний захист від них для

одного з фреймворків щоб провести порівняльний аналіз з базовим захистом інших двох фреймворків.

3 РОЗРОБКА МЕТОДУ ЗАХИСТУ ОДНОГО ІЗ JAVASCRIPT ФРЕЙМВОРКІВ ТА ЙОГО ПОРІВНЯЛЬНИЙ АНАЛІЗ ІЗ БАЗОВИМИ МЕТОДАМИ ЗАХИСТУ.

3.1 Розробка методу захисту JavaScript фреймворка

Метод захисту розроблений для моделі безпеки Angular, що містить крос-сайти.

Щоб систематично блокувати помилки XSS, Angular трактує всі значення як недостовірні за замовчуванням. Коли значення вставляється в DOM з шаблону, через властивість, атрибут, стиль, прив'язку класу або інтерполяцію, Angular дезактивує і видаляє недостовірні значення.

Кутові шаблони збігаються з виконуваним кодом: HTML, атрибути і вираз прив'язки (але не значення, пов'язані) в шаблонах можна вважати безпечними. Це означає, що програми повинні запобігати значенням, які зловмисник може контролювати, коли-небудь роблячи його в вихідний код шаблону. Ніколи не можна генерувати вихідний код шаблону шляхом об'єднання вводу користувача та шаблонів. Щоб запобігти цим вразливостям, потрібно використовувати автономний компілятор шаблонів, також відомий як ін'єкція шаблонів.

Дезінфекція та контексти безпеки

Дезінфекція - це перевірка ненадійного значення, перетворюючи його в безпечне для вставки значення в DOM. У багатьох випадках санітарія взагалі не змінює значення. Дезінфекція залежить від контексту: значення, яке є нешкідливим у CSS, є потенційно небезпечним у URL-адресі.

Angular визначає наступні контексти безпеки:

HTML використовується при інтерпретації значення як HTML, наприклад, при прив'язуванні до internalHtml.

Стиль використовується при прив'язуванні CSS до властивості стилю.

URL-адреса використовується для властивостей URL, наприклад <a href>.

URL-адреса ресурсу - це URL-адреса, яка буде завантажена і виконана у вигляді коду, наприклад, у <script src>.

Angular дезактивує недостовірні значення для HTML, стилів і URL, дезактивація URL-адрес ресурсу неможлива, оскільки вони містять довільний код.

Приклад обробки

Наступний шаблон пов'язує значення htmlSnippet, один раз шляхом інтерполяції його в вміст елемента, і один раз, прив'язавши його до властивості innerHTML елемента:

```
src/app/inner-html-binding.component.html
<h3>Binding innerHTML</h3>
<p>Bound value:</p>
<p class="e2e-inner-html-interpolated">{{htmlSnippet}}</p>
<p>Result of binding to innerHTML:</p>
<p class="e2e-inner-html-bound" [innerHTML]="htmlSnippet"></p>
```

Рисунок 3.1 - Шаблон поєднання htmlSnippet

Інтерпольований вміст завжди відхиляється - HTML не інтерпретується, а браузер відображає кутові дужки у текстовому вмісті елемента.

Щоб інтерпретувати HTML, потрібно прив'язати його до властивості HTML, такій як innerHTML. Але прив'язка значення, яке зловмисник може контролювати в innerHTML, зазвичай викликає уразливість XSS. Наприклад, виконується код, що міститься в тезі <script>:

```
src/app/inner-html-binding.component.ts (class)
export class InnerHtmlBindingComponent {
  // For example, a user/attacker-controlled value from a URL.
  htmlSnippet = 'Template
<script>alert("0wned")</script><b>Syntax</b>';
}
```

Рисунок 3.2 – Вразливий скрипт з XSS

Angular розпізнає значення як небезпечне і автоматично деактивує його, що видаляє тег <script>, але зберігає безпечний вміст, такий як елемент <b>.

Безпосереднє використання API

Вбудовані API-інтерфейси DOM браузера не захищають автоматично від вразливостей безпеки. Наприклад, документ, вузол, доступний через ElementRef, і багатосторонні API містять небезпечні методи. Таким же чином, якщо ви взаємодієте з іншими бібліотеками, які маніпулюють DOM. Ймовірно ми не матимемо такої ж автоматичної санітарії, як з інтерполями Angular. Потрібно уникати безпосереднього взаємодії з DOM і використовувати кутові шаблони там, де це можливо.

У випадках, коли це неминуче, потрібно використовувати вбудовані функції кутової санітарної обробки. Дезінфікувати недостовірні значення методом DomSanitizer.sanitize і відповідним текстом SecurityContext. Ця функція також приймає значення, які були позначені як довірені за допомогою функцій bypassSecurityTrust, і не буде їх дезінфікувати.

Політика безпеки контенту

Політика безпеки контенту (CSP) є технікою захисту в глибині для запобігання XSS. Щоб увімкнути CSP, потрібно налаштувати веб-сервер для повернення відповідного HTTP-заголовка Content-Security-Policy.

Автономний компілятор шаблонів

Автономний компілятор шаблонів запобігає всьому класу вразливостей, які називаються ін'єкцією шаблонів, і значно покращує продуктивність додатків. Використовувати автономний компілятор шаблонів потрібно у виробничих розгортаннях, не створювати шаблони динамічно. Кутовий шаблон довіряє шаблону, тому генерування шаблонів, зокрема шаблонів, що містять дані користувача, обходить вбудований захист Angular.

Захист Angular використовує цей компілятор, як базовий, проте через те, що він створенний динамічно – це не є ефективним.

Захист XSS на стороні сервера

Створений на сервері HTML є вразливим до атак ін'єкцій. Введення коду шаблону в додаток Angular є таким же, як ін'єкція виконуваного коду в застосунок: він надає зловмиснику повний контроль над застосунком. Щоб запобігти цьому, потрібно скористатися мовою шаблонів, яка автоматично вилучає значення для запобігання уразливості XSS на сервері. Не створювати шаблони Angular на стороні сервера, використовуючи мову шаблонів - це призводить до високого ризику впровадження вразливостей для шаблонів ін'єкцій.

Іноді програми дійсно повинні включати виконуваний код, відображати <iframe> з певного URL або створювати потенційно небезпечні URL-адреси. Щоб запобігти автоматичній санітарній обробці в будь-якій з цих ситуацій, можна сказати Angular, що ми перевірили значення, перевірили, як вона була створена, і переконалися, що вона завжди буде безпечною. Але потрібно бути обережними. Якщо ми довіряємо значенню, яке може бути зловмисним, ми вводимо у наш застосунок вразливість системи безпеки.

Щоб позначити значення як довірене, потрібно ввести DomSanitizer і викликати один з наступних методів:

bypassSecurityTrustHtml

bypassSecurityTrustScript

bypassSecurityTrustStyle

bypassSecurityTrustUrl

bypassSecurityTrustResourceUrl

Значення безпечне залежить від контексту, тому потрібно вибирати правильний контекст для призначеного нами значення. Уявімо, що наступний шаблон повинен пов'язувати URL-адресу зі скриптом javascript: alert (...):

```
src/app/bypass-security.component.html (URL)
<h4>An untrusted URL:</h4>
<p><a  class="e2e-dangerous-url"  [href]="dangerousUrl">Click
me</a></p>
<h4>A trusted URL:</h4>
<p><a    class="e2e-trusted-url"    [href]="trustedUrl">Click
me</a></p>
```

Рисунок 3.3 – Шаблон з’єднання URL-адреси зі скриптом

Як правило, Angular автоматично деактивує URL, вимикає небезпечний код і в режимі розробки реєструє цю дію на консолі. Щоб запобігти цьому, позначимо значення URL як довірену URL-адресу за допомогою виклику bypassSecurityTrustUrl:

```
src/app/bypass-security.component.ts (trust-url)
constructor(private sanitizer: DomSanitizer) {
  // javascript: URLs are dangerous if attacker controlled.
  // Angular sanitizes them in data binding, but you can
  // explicitly tell Angular to trust this value:
  this.dangerousUrl = 'javascript:alert("Hi there")';
  this.trustedUrl
sanitizer.bypassSecurityTrustUrl(this.dangerousUrl);
```

Рисунок 3.4 – Код виклику методу bypassSecurityTrustUrl

Якщо потрібно конвертувати вхід користувача в довірене значення, потрібно використовувати метод контролера. Наступний шаблон дозволяє користувачам вводити ідентифікатор відео YouTube і завантажувати відповідне відео в <iframe>. Атрибут <iframe src> - це контекст безпеки URL ресурсу, оскільки ненадійний джерело може, наприклад, переправляти завантаження файлів, які могли б виконувати не підозрювані користувачі. Тому необхідно викликати метод на контролері, щоб побудувати URL-адресу надійного відео,

що призводить до того, що Angular дозволить прив'язку до <iframe src>:

```
src/app/bypass-security.component.html (iframe)
<h4>Resource URL:</h4>
<p>Showing: {{dangerousVideoUrl}}</p>
<p>Trusted:</p>
<iframe       class="e2e-iframe-trusted-src"       width="640"
height="390" [src]="videoUrl"></iframe>
<p>Untrusted:</p>
<iframe      class="e2e-iframe-untrusted-src"      width="640"
height="390" [src]="dangerousVideoUrl"></iframe>
src/app/bypass-security.component.ts (trust-video-url)
updateVideoUrl(id: string) {
  this.dangerousVideoUrl = 'https://www.youtube.com/embed/' +
id;
  this.videoUrl =

this.sanitizer.bypassSecurityTrustResourceUrl(this.dangerousV
ideoUrl);
}
```

Рисунок 3.5 – Шаблон для безпечного вводу ідентифікатора відео YouTube

Вразливості на рівні HTTP

Angular має вбудовану підтримку для запобігання двох поширених вразливостей HTTP, підробки мультисайтових запитів (CSRF або XSRF) та включення міжкристових скриптів (XSSI). Обидва вони повинні бути пом'якшені насамперед на стороні сервера, але Angular надає помічникам можливість полегшити інтеграцію на стороні клієнта.

Підробку запитів між сайтами

У підробці запитів між сайтами (CSRF або XSRF) зловмисник примушує користувача відвідувати іншу веб-сторінку (наприклад, evil.com) із злоякісним кодом, який таємно надсилає шкідливий запит до веб-сервера програми (наприклад, приклад- bank.com).

Припустимо, що користувач увійшов до програми на example-bank.com. Користувач відкриває електронний лист і натискає посилання на evil.com, яке відкривається в новій вкладці.

Сторінка evil.com негайно надсилає зловмисний запит до example-bank.com. Можливо, це запит на переказ коштів з облікового запису користувача на рахунок атакуючого. Браузер автоматично надсилає файли cookie в прикладі-bank.com (включаючи кукі-автентифікації) з цим запитом.

Якщо сервер-приклад-bank.com не має захисту XSRF, він не може визначити різницю між законним запитом від програми та підробленим запитом від evil.com.

Щоб запобігти цьому, програма повинна переконатися, що запит користувача походить від реального застосунку, а не з іншого сайту. Сервер і клієнт повинні співпрацювати, щоб перешкодити цій атаці.

У загальній техніці anti-XSRF сервер застосунків надсилає випадково згенерований маркер автентифікації у файлі cookie. Код клієнта читає cookie і додає спеціальний заголовок запиту з маркером у всіх наступних запитах. Сервер порівнює отримане значення cookie із значенням заголовка запиту і відхиляє запит, якщо значення відсутні або не збігаються.

Цей метод є ефективним, оскільки всі браузери реалізують таку ж політику щодо походження. Лише код з веб-сайту, на якому встановлюються файли cookie, може зчитувати файли cookie з цього сайту і встановлювати користувальницькі заголовки запитів на цей сайт. Це означає, що лише наша програма може прочитати цей маркер cookie і встановити спеціальний заголовок. Зловмисний код на сайті evil.com не може.

HttpClient Angular має вбудовану підтримку на стороні клієнта цієї техніки.

3.2 Порівняльний аналіз написаного та наявних методів захисту JavaScript фреймворків

Порівняльний аналіз методів захисту фреймворків проведемо в три етапи. На кожному з етапів реалізуємо ту чи іншу типову задачу та проаналізуємо кількість успішних атак на всіх трьох етапах.

Етап 1. Реалізація програми-лічильника, ймовірно, є одним з найбільш часто використовуваних прикладів в реактивному програмуванні. Він гранично простий і зрозумілий:

- Потрібен механізм для виведення значення, що зберігається в count та подання його користувачеві.
- Потрібні дві кнопки, прив'язані до відповідних методів, що дозволяють користувачу впливати на змінну count.

Ось реалізація цього прикладу з використанням розглянутих фреймворків.

React

```
import React from "react";
import ReactDOM from "react-dom";

Class Counter extends React.Component {
    constructor (props) {
        super (props);
        this .state = {count: 0};
    }

    down (value) {
      this .setState (state => ({count: state.count - value}));
    }
    up (value) {
      this .setState (state => ({count: state.count + value}));
    }

    render () {
```

```
            return (
                <div>
                    <h1> {this.state.count} </ h1>
                     <button onClick = {() => this.down (1)}> - </
    button>
                      <button onClick = {() => this.up (1)}> + </
    button>
                </ div>
            );
        }
    }

    ReactDOM.render (<Counter />, document.querySelector ( "#
    app"));
```

Vue

```
import Vue from "vue";
new Vue ({
    data: {count: 0},

    methods: {
        down: function (value) {
            this .count - = value;
        },
        up: function (value) {
            this .count + = value;
      }
    },
    render: function (h) {
        return (
            <div>
                <h1> {this.count} </ h1>
                      <button onClick = {() => this.down (1)}>
     - </ button>
                  <button onClick = {() => this.up (1)}> + </
     button>
```

```
                </ div> ); },
        el: "#app"
       });
```

Hyper

```
import {h, app} from "hyperapp";
const state = {
    count: 0
};

const actions = {
    down: value => state => ({count: state.count - value}),
    up: value => state => ({count: state.count + value})
};

const view = (state, actions) => (
    <div>
        <h1> {state.count} </ h1>
       <button onclick = {() => actions.down (1)}> - </ button>
        <button onclick = {() => actions.up (1)}> + </ button>
    </ div>
);
```

Аналіз

При використанні всіх трьох фреймворків на початку коду програми є команди import.

У React використовується об'єктно-орієнтована парадигма. Тут створюється клас для компонента Counter. Vue йде схожим шляхом. Тут створюється новий екземпляр класу Vue, йому передається інформація. І, нарешті, в Hyper застосовується функціональна парадигма, тут використовуються самостійні сутності view, state і actions.

Якщо говорити про змінну count, то в React вона ініціалізується в конструкторі компонента, а в Vue і Hyper вона являє собою властивість, відповідно, об'єктів data і state.

Якщо просунутися в дослідженні цих застосунків далі, то можна помітити, що в React і Vue використовуються дуже схожі методи для взаємодії зі змінною count, що є наймовірнішим об’єктом атак. У React, для зміни стану програми, застосовується метод setState, успадкований від React.Component. В Vue значення this.count змінюється безпосередньо. Методи в Hyper написані з використанням синтаксису стрілочних функцій ES6. Серед розглянутих фреймворків він єдиний використовує подібне, так як React і Vue змушені використовувати всередині своїх методів ключове слово this. Методи Hyper, з іншого боку, вимагають, щоб їм, як аргумент, передавали б об'єкт зі станом застосунка. Це означає, що їх, найімовірніше, можна буде повторно використовувати в різних контекстах.

Та частина програми, яка відповідає за виведення даних на сторінку, у всіх трьох прикладах виглядає практично однаково. Особливість Vue полягає в тому, що при використанні цього фреймворка в підсистему рендеринга треба передати функцію h. У Hyper, замість onClick, використовується onclick, а так само тут звернення до змінної count здійснюється не так як в React і Vue, що обумовлено особливостями того, як в кожному з фреймворків реалізовано зберігання стану програми.

І, нарешті, всі три фреймворки використовують прив'язку до елементу #app. У кожному з них ця операція виконується по-різному. Треба відзначити, що в Vue ця операція виглядає найпростішою і зрозумілою і дає розробнику більш гнучку конструкцію, працюючи з селектором елемента, а не з самим елементом.

Етап 2. Одна з найпоширеніших асинхронних операцій є відправка запиту якомусь API. Для цілей цього прикладу застосовується API JSONPlaceholder , що містить умовні дані і видає список публікацій. Ось що ми збираємося тут зробити:

- Збережемо масив для розміщення в ньому публікацій (posts) в стані додатки.

- Викличемо, скориставшись відповідним методом, fetch (), вказавши потрібний нам URL, почекаємо надходження даних, розпарсити отриманий JSON-код, який представляє собою масив об'єктів, і, нарешті, оновимо змінну posts, записавши в неї отримані дані.
- Виведемо на сторінку кнопку, яка викликає метод, що завантажує список публікацій.
- Виведемо список публікацій з posts з використанням ключів.

Розглянемо код, який реалізує вищеописану схему дій.

React

```
import React from "react";
import ReactDOM from "react-dom";

class PostViewer extends React. Component {
    constructor (props) {
        super (props);
        this .state = {posts: []};
    }
     getData () {
           fetch ( `https: // jsonplaceholder.typicode.com /
posts`)
        .then (response => response.json ())
        .then (json => {
            this .setState (state => ({posts: json}));
        });
    }

    render () {
        return (
            <Div>
                <button onClick = {() => this.getData ()}> Get
posts </ button>
```

```
{this.state.posts.map (post => (
                    <div key = {post.id}>
                              <H2> <font color = "# 3AC1EF">
{post.title} </ font> </ h2>
                         <P> {post.body} </ p>
                    </ Div>
                ))}
            </ Div>
        );
    }
}

ReactDOM.render (<PostViewer />, document.querySelector ( "#
app"));
```

Vue

```
import Vue from "vue";

new Vue ({
    data: {posts: []},

    methods: {
        getData: function (value) {
             fetch ( `https: // jsonplaceholder.typicode.com /
posts`)
            .then (response => response.json ())
            .then (json => {
                this .posts = json;
            });
        }
    },

    render: function (h) {
        return (
            <Div>
```

```
                        <button onClick = {() => this.getData ()}> Get
  posts </ button>
                        {this.posts.map (post => (
                            <div key = {post.id}>
                                      <H2> <font color = "# 3AC1EF">
  {post.title} </ font> </ h2>
                                <P> {post.body} </ p>
                            </ Div>
                        ))}
                    </ Div>
                );
            },

            el: "#app"
        });
```

Hyper

```
import {h, app} from "hyper";

const state = {
    posts: []
};

const actions = {
    getData: () => (state, actions) => {
        fetch ( `https: // jsonplaceholder.typicode.com /
posts`)
        .then (response => response.json ())
        .then (json => {
            actions.getDataComplete (json);
        });
    },
    getDataComplete: data => state => ({posts: data})
};
```

```
const view = (state, actions) => (
    <Div>
        <button onclick = {() => actions.getData ()}> Get
posts </ button>
        {state.posts.map (post => (
            <div key = {post.id}>
                <H2> <font color = "# 3AC1EF"> {post.title}
</ font> </ h2>
                <P> {post.body} </ p>
            </ Div>
        ))}
    </ Div>
);

app (state, actions, view, document .querySelector (
"#app"));
```

Аналіз

Розберемо цей код і порівняємо три досліджувані фреймворки. Так само, як і в попередньому прикладі, зберігання стану програми, висновок даних і підключення до елементу сторінки, у всіх трьох фреймворках досить схожі. Тут спостерігаються ті ж відмінності, про які ми вже говорили вище.

Завантаження даних за допомогою функції fetch () - операція досить проста, вона працює так, як очікується, у всіх фреймворках. Основна відмінність тут, однак, полягає в тому, що Hyper підтримує виконання асинхронних операцій трохи не так, як інші фреймворки. Замість того щоб модифікувати стан безпосередньо, всередині асинхронного дії, ця дія викликає інше, синхронну дію, яка отримує дані і перетворює їх у відповідний формат. Це робить ядро програми більш функціональним і краще підходить для розбиття на невеликі частини, які, потенційно, підходять для повторного використання, однак це негативно впливає на безпеку. Такий підхід, крім того, допомагає уникати деяких проблем, властивих вкладеним коллбекам, які можуть виникнути в ситуаціях, подібних розглянутим.

Якщо говорити про розміри коду, то Hyper застосункам знову потрібно менше рядків коду для досягнення тієї ж мети, але код на Vue виглядає більш коротким, і, якщо порахувати кількість символів коду, він коротше інших варіантів.

Виконання асинхронних операцій виявилося однаково простим у всіх фреймворках. Hyper може схилити розробника до написання більш функціонального та модульного коду при роботі з асинхронними діями, але два інших фреймворка теж відмінно справляються з поставленим перед ними завданням, і, в цьому плані, дають розробнику можливість вибору.

Етап 3. Компонент елементу списку для To-Do-застосунка

Ймовірно, To-Do-застосунки - це найвідоміший приклад в області реактивного програмування. По всій видимості, щось подібне реалізовано з використанням майже кожного з існуючих фреймворків. Тут ми не будемо реалізовувати весь застосунок. Замість цього зупинимося на простому компоненті без стану для того, щоб вивчити можливості досліджуваних фреймворків зі створення невеликих будівельних блоків веб-застосунків, які підходять для повторного використання.

Розглянемо реалізацію компонента із застосуванням досліджуваних фреймворків. У цьому прикладі ми, однак, розширимо розглянуті варіанти коду за рахунок розгляду React-компонента, написаного в функціональному стилі.

React (функціональний стиль)

```
function TodoItem (props) {
    return (
        <li class = {props.done? "done": ""} onclick = {() =>
props.toggle (props.id)}>
            {props.value}
        </ li>
    );
}
```

React

```
class TodoItem extends React. Component {
    render () {
        return (
            <li class = {this .props.done? "done": ""} onclick
= {() => this .props.toggle (this .props.id)}>
                {This .props.value}
            </ li>
        );
    }
```

Vue

```
var TodoItem = Vue .component ( "todoitem", {
    props: [ "id", "value", "done", "toggle"],
    template:
        '<li v-bind: class = "{done: done}" v-on: click =
"toggle (id)"> {{value}} </ li>'
});
```

Hyper

Hyper так само використовує функціональний стиль.

```
const TodoItem = ({id, value, done, toggle}) = (
    <li class = {done? "done": ""} onclick = {() => toggle
(id)}>
        {value}
    </ li>
);
```

React, в тому, що стосується використання патернів кодування, є самим гнучким фреймворком. Він підтримує функціональні компоненти, а так само компоненти, оформлені у вигляді класів. Крім того, React, в його стандартному вигляді, підтримує і компоненти Hyper.

Hyper теж підтримує функціональні React-компоненти. Це означає, що при роботі з Hyper є великий простір для експериментів, а також для атак, які цим спровоковані.

Vue в цьому випробуванні займає останнє місце. У нього досить дивний синтаксис, який непросто відразу зрозуміти навіть тим, хто знайомий з React або Hyper.

Якщо говорити про довжину коду, то все приклади мають дуже схожі розміри. Єдине, що тут можна відзначити - це те, що код на React, в одному з варіантів, вийшов трохи об'ємніше, ніж в іншому.

Для отримання результатів оцінювання захищеності JavaScript фреймворків згідно обраних критеріїв кожен фреймворк був детально досліджений. Дослідження проводились використовуючи офіційні інформаційні ресурси та програмний код самих фреймворків. Результати аналізу приведені у Таблиці 3.1.

Таблиця 3.1 - Результати аналізу обраних фреймворків

Фреймворки	Кількість атак, які не було відхилено захистом фреймворка		
	Етап 1	Етап 2	Етап 3
React	0	0	2
Vue	3	1	1
Hyper (розроблений)	0	1	1

Отже, на першому етапі захист фреймворку React та розробленого нами впорався з атаками. Фреймворк Vue не зміг впоратися з жодною із трьох спроб цієї атаки.

На другому етапі захист фреймворку React впорався з атаками. Фреймворк Vue та розроблений нами не зміг впоратися з однією із трьох спроб цієї атаки. Це доводить, що метод захисту є ситуативний.

На третьому етапі захист фреймворку React не впорався з двома атаками. Фреймворк Vue та розроблений нами не зміг впоратися з однією із трьох спроб цієї атаки.

Це доводить, що методи захисту всіх фреймворків є недостатньо надійними і при написані власного веб-застосунку розробник повинен не тільки використовувати базові методи захисту фреймворків, а й писати власні дивлячись на те, для чого проектувався застосунок і які на нього атаки можуть бути проведені.

Висновки до розділу 3

В даному розділі був запропонований метод захисту сучасних JavaScript фреймворків, який базується на аналізі захисту фреймворків від XSS та XSSI атак.

Отриманий результат після проведення порівняльного аналізу методів захисту фреймворків в три етапи свідчить про те, що базові методи не є еталонними бо мають діри в захисті. Кожному розробнику під час написання веб-застосунку потрібно приділяти більше часу питанню захищеності його продукту та окрім використання базових методів захисту писати й свої, дивлячись на деталі майбутнього користування веб-застосунком.

ВИСНОВКИ

1.Результатом роботи є розроблений метод захисту для одного із фреймворків та його порівняльний аналіз із наявними базовими методами захисту інших фреймворків.

Для написання власного методу захисту було обрано три найпопулярніші та найпоширеніші фреймворки на сьогодні:

1. React
2. Angular
3. Vue.js

2.Проведений аналіз загроз для цих трьох фреймворків і було обрано найбільш загрозливу та найпоширенішу з них - XSS атаку та її модифікацію XSSI. Базуючись на лазівках які ці атаки використовують, був написаний власний метод захисту для Anglar. Суть методу в тому, що ми дезінфікуємо недостовірні значення, які використовуються під час XSS атак методом DomSanitizer.sanitize і відповідним текстом SecurityContext шляхом поєднання цих методів. Ця функція також приймає значення, які були позначені як довірені за допомогою функцій bypassSecurityTrust, і не буде їх дезінфінкувати.

3.Проведений в три етапи порівняльний аналіз методів захисту фреймворків. На кожному з етапів реалізували ту чи іншу типову задачу та проаналізували кількість успішних атак на всіх трьох етапах. Кожну типову задачу доповнили атакою XSS та аналізували як реагував захист кожного фреймворка на таку атаку. Всього на кожному етапі було проведено по три атаки на кожен фреймворк.

4.Результати свідчать про те, що методи захисту всіх фреймворків є недостатньо надійними і при написані власного веб-застосунку розробник повинен не тільки використовувати базові методи захисту фреймворків, а й писати власні дивлячись на те, для чого проектувався застосунок і які на нього атаки можуть бути проведені.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Digital, Social and Mobile in 2015 report indicates [Електронний ресурс] –Режим доступу до ресурсу: http://wearesocial.com/sg/special-reports/digital-social-mobile-2015

2. Веб-фреймворки и с чем их едят [Електронний ресурс] –Режим доступу до ресурсу: http://iwsm.ru/blog/show/veb-freymvorki-i-s-chem-ih-edyat.

3. Фреймворки в веб-разработке [Електронний ресурс] –Режим доступу до ресурсу: : https://web-creator.ru/articles/about_frameworks.

4. Результаты тестирования шести ведущих фреймворков на производительность [Електронний ресурс] –Режим доступу до ресурсу: http://www.alrond.com/ru/2007/jan/25/rezultaty-testirovanija-6-frameworks/.

5. Адаптивные CSS-фреймворки, сетки, классы видимости [Електронний ресурс] –Режим доступу до ресурсу: http://klondike-studio.ru/blog/responsive-css-framework/.

6. Обзор CSS-фреймворков [Електронний ресурс] –Режим доступу до ресурсу: http://iantonov.me/page/obzor-css-frejmvorkov.

7. Використання PHP фреймворків в розробці сайту [Електронний ресурс] –Режим доступу до ресурсу: http://ukrbukva.net/page,5,39718-Ispol-zovanie-PHP-freiymvorkov-v-razrabotke-saiyta.html.

8. Сравнение каркасов веб-приложений [Електронний ресурс] –Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Сравнение_каркасов_веб-приложений.

9. Обзоры Web-фреймворков [Електронний ресурс] –Режим доступу до ресурсу: https://praktikatech.wordpress.com/category/обзоры-web-фреймворков/

10. Полное руководство по Yii [Електронний ресурс] –Режим доступу до ресурсу: http://www.yiiframework.com/doc/guide/1.1/ru/index.

11. Каркас веб-приложений [Електронний ресурс] –Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Каркас_веб-приложений.

12. Что такое фреймворк? [Електронний ресурс] –Режим доступу до ресурсу: http://www.dbhelp.ru/what-is-framework/page/.

13. Десять причин избегать тяжеловесных фреймворков, а также лишних зависимостей в проекте [Електронний ресурс] –Режим доступу до ресурсу: http://eax.me/avoid-frameworks/.

14. 5 самых популярных фреймворков 2014года [Електронний ресурс] –Режим доступу до ресурсу: http://lpgenerator.ru/blog/2016/03/10/5-samyh-populyarnyh-frejmvorkov-2014-goda/.

15. What is a Web Framework? [Електронний ресурс] –Режим доступу до ресурсу: https://jeffknupp.com/blog/2014/03/03/what-is-a-web-framework/.

16. Популярные frontend и backend фреймворки [Електронний ресурс] –Режим доступу до ресурсу: http://web-diz.com.ua/poleznosti/populyarnye-frontend-i-backend-freymvorki/.

17. О фреймворках [Електронний ресурс] –Режим доступу до ресурсу: http://web-elive.com/stati/raznoe/o-frejmvorkax/.

18. Веллинг Л. Разработка веб-приложений с помощью PHP и MySQL / Л. Веллинг, Л. Томсон. –Москва: Вильямс, 2010. –848 с.

19. Шлосснейгл Д. Профессиональное программирование на PHP / Джордж Шлосснейгл.–Москва: Вильямс,2006. –624 с.

20. Кузнецов М. В. PHP 5 на примерах / М. В. Кузнецов, И. В. Симдянов, С.

21. 103В. Голышев.–Санкт-Петербург: БХВ-Петербург., 2005. –576 с.

22. Кухарчик А. С. PHP: обучение на примерах / А.С.Кухарчик.–Минск: Новое знание, 2004. –240 с

23. Аткинсон Л. PHP 5. Библиотека профессионала / Л. Аткинсон, З. Сураски.–Москва: Вильямс, 2006. –944 с.

24. Мазуркевич А. М. PHP: Настольная книга программиста / А. М. Мазуркевич, Д. С. Еловой.–Минск: Новое знание, 2004. –480 с.

25. Суэринг С. PHP и MySQL. Библия программиста / С. Суэринг, Д. Парк, Т. Конверс.–Москва: Вильямс, 2010. –912 с.

26. Дронов В. А. PHP 5/6, MySQL 5/6 и Dreamweaver CS4. Разработка интерактивных Web-сайтов / Владимир Александрович Дронов.–Санкт-Петербург: БХВ-Петербург., 2009. –544