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

_____ Аполонова Юлія Андріївна _____
(прізвище, ім'я, по батькові)

1. Тема роботи Порівняльний аналіз механізмів захисту JavaScript фреймворків і бібліотек та формування оцінки їх захищеності _____
_____ ,
науковий керівник роботи _____
_____,
(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 _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник роботи _____ _____
(підпис) (ініціали, прізвище)

ТИТУЛКА

ЗАВДАННЯ

КАЛЕНДАРНИЙ ПЛАН

РЕФЕРАТ

Робота обсягом 89 сторінок містить 5 ілюстрацій, 12 таблиць, 19 літературних посилань та 4 додатки.

Метою даної кваліфікаційної роботи є дослідження захищеності JavaScript фреймворків і бібліотек та формування оцінки їх захищеності.

Об’єктом дослідження є JavaScript бібліотеки і фреймворки.

Предметом дослідження є захищеність JavaScript бібліотек і фреймворків.

Результати роботи представлені у вигляді таблиць, що характеризують різні складові захищеності бібліотек і фреймворків, а також оцінок, обчислених для кожного обраного для розгляду фреймворку чи бібліотеки згідно з розробленими критеріями та експертними оцінками їх важливості.

Отримані результати мають практичне застосування при виборі технології для розробки веб-застосунків з метою підвищення захищеності системи, що розробляється, а також у більш детальних дослідженнях та оцінці захищеності систем, що використовують розглянуті технології.

Javascript, безпека веб-застосунків, бібілотека, фреймворк, вразливість, експертне оцінювання, метод аналітичної ієрархії

ABSTRACT

The work includes 89 pages, 5 figures, 12 tables 19 literary references and 4 additions.

The aim of this qualification is research of security of JavaScript frameworks and libraries as well as evaluation of their security.

The object of researches is JavaScript libraries and frameworks.

The subject of research is a security of JavaScript libraries and frameworks.

The results of work are presented in the form of tables describing various components of libraries and frameworks security, as well as assessments calculated for each selected for consideration framework and library, according to the developed criteria and expert evaluation of theit importance.

The obtained results have practical application in the process of choosing technology for developing web applications for the purpose of enhancement the secyrity of system being developed, as well as in more detailed studies and security evaluation odf systems using the technologies considered.

Javascript, web-application security, library, framework, vulnerability, expert evaluation, method of analytical hierarchy

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

MVC (Model-view-controller) — Модель-представлення-контролер, архітектурний шаблон проектування, що використовується при розробці програмного забезпечення.

MVVM (Model-View-ViewModel) — Модель-представлення-модель_представлення, архітектурний шаблон проектування, що використовується при розробці програмного забезпечення

MVP (Model-View-Presenter) — Модель-представлення-пред’явник, архітектурний шаблон, що використовується при розробці програмного забезпечення

OWASP (Open Web Application Security Project) — відкритий проект з безпеки веб-застосунків, організація, що пропонує статті, технології та рекомендації для по покращення безпеки веб-застосунків.

ASVS (Application Security Verification Standard) — стандарт перевірки відповідності безпеки застосунків проекту OWASP, основною метою якого є надання критеріїв, за якими відбувається встановлення рівня довіри в безпеці веб-застосунків.

CVE (Common Vulnerabilities and Exposures) — база даних загальновідомих вразливостей у сфері інформаційної безпеки, що підтримується організацією MITRE.

CVSS (Common Vulnerability Scoring System) — система оцінювання вразливостей та ранжування ризиків інформаційної безпеки.

DOM (Document Object Model) — об’єктна модель документа, програмний інтерфейс, що дозволяє отримати доступ до структури HTML-розмітки та можливість її змінювання.

HTML (HyperText Markup Language) — мова розмітки документів, стандартизована і призначена для формування структури веб-сторінок.

JSON (JavaScript Object Notation) — формат файлів, в якому для передачі даних, що складаються з пари атрибут-значення, використовується текст.

CSP (Content Security Policy) — політика безпеки контенту, призначена для зменшення наслідків атак та їх вчасного виявлення.

JSX – розширення синтаксису мови JavaScript, що використовується бібліотекою React.

W3C (World Wide Web Consortium) — організаці, основною метою якої є впровадження технологічних стандартів для всесвітньої павутини. Наприклад, XML, HTML, CSS.

JavaScript — мова програмування, що надає можливість взаємодії з користувачем на стороні клієнта, обміну даними з сервером та зміни структури веб-сторінки.

Cookie (англ. Cookie — тістечка, печиво) — інформація, що відправляється сервером і зберігається у вигляді текстових даних на стороні клієнта.

ВСТУП

Сучасний світ інформаційних технологій вже неможливо уявити без веб-ресурсів, сфера використання яких поширюється з року в рік. Але з їх розповсюдженням зростає також кількість вразливостей та небезпек, що дозволяють зловмисникам виконувати широкий спектр дій: отримувати важливу інформацію, проникати у мережу, а також порушувати режим функціонування веб-застосунку. Цей негативний вплив є суттєвим і для діяльності підприємств, які збільшують свої доходи шляхом використання електронної комерції, і для банківської сфери, пов’язаною із численними операціями з фінансами та банківськими рахунками, для користувачів, що довіряють веб-застосункам обробку важливих конфіденційних даних.

Більша частина потенційних вразливостей може бути врахована ще на етапі написання коду веб-застосунку, і дуже велике значення має тут попередження можливих видів атак на стороні клієнта. Все більше розробників вдаються до використання сторонніх бібліотек та фреймворків, що робить архітектуру веб-застосунку більш структурованою, а написання коду – простішим та ефективнішим. Ці технології певною мірою забезпечують якість та захищеність застосунку, але їх застосування може бути пов’язане і з додатковими загрозами та вразливостями. Крім того, дуже часто основні зусилля розробників направлені на забезпечення функціоналу, а безпеці приділяється недостатньо уваги, внаслідок чого спостерігається порушення конфіденціальності, цілістності та доступності інформації. Мало хто з потенційних клієнтів та користувачів захоче мати справу із небезпечними та сумнівними ресурсами, тому так важливо знати про те, яким вразливостям вони можуть бути підвладні, чи мають допоміжні засоби розробки механізмів захисту та чи є їх використання безпечним.

Метою даної роботи є дослідження захищеності сучасних JavaScript фреймворків і бібліотек. Завданням – формування оцінки їх захищеності згідно з вибраними критеріями перевірки.

Предметом дослідження є захищеність JavaScript фреймворків і бібліотек.

Науковий внесок та новизна роботи полягає у детальному розгляді існуючих механізмів захисту бібліотек і фреймворків та використанні експертних оцінок при визначенні важливості кожного з критеріїв, що впливає на точність та коректність визначення загальної оцінки захищеності.

Робота має практичне застосування, тому що результати, отримані в ході її виконання, можна використовувати в якості критерію на етапі обрання технологій розробки з метою підвищення захищеності системи та у дослідженнях комлексних систем, що використовують розглянуті технології.

1 ОСНОВНІ ЗАГРОЗИ БЕЗПЕКИ ДЛЯ ВЕБ-ЗАСТОСУНКІВ

1.1 Архітектура веб-застосунку

Веб-застосунок – це програмний продукт, що застосовує технологію «клієнт-сервер». У ній користувач взаємодіє з сервером, де зберігаються дані, через браузер.

Веб-застосунки складаються з двох частин: клієнтської і серверної. На стороні клієнта реалізується інтерфейс користувача, відображаються дані, що пересилаються з сервера та формуються запити до нього. Серверна частина відповідає за обробку запитів, що надходять від клієнта, формування відповіді та здійснення операцій, необхідних для реалізації бізнес-логіки. Схему клієнт-серверної взаємодії наведено на Рисунку 1.1.

Рисунок 1.1 – Клієнт-серверна взаємодія

Для кожної з частин існують свої технології та мови програмування. На стороні клієнта використовується JavaScript, а також різні архітектурні шаблони, які можуть реалізовуватися засобами фреймворку чи бібліотеки. До них належать MV*-шаблони (MVC, MVP, MVVM) та архітектура веб-компонентів, яка застосовується у деяких з обраних для дослідження фреймворках [1].

1.1.1 Архітектура MVC

MVC-архітектура скаладається з трьох ключових компонентів. Це модель, представлення і контролер (Model-View-Controller).

Модель використовується у веб-застосунку для управління бізнес-даними. Вона створює для даних унікальну зручну форму, не стосуючись при цьому представлення та користувацького інтерфейсу. Коли модель змінюється внаслідок оновлення формату даних, вона повідомляє про це представлення для створення можливості коректного реагування на зміни, що відбулися.

Представлення є візуальними відображеннями моделей і залежать від їх поточного стану. Користувачі мають можливість взаємодії з представленнями, що надають для цього зручний інтерфейс, в контексті здійснення операцій читання та редагування.

Окремо слід зазначити, що представлення тісно пов’язане із поняттям шаблонування у JavaScript. Це рішення часто використовується для того, щоб запобігти створенню в пам’яті великих блоків HTML-розмітки шляхом конкатенації рядків. Воно засноване на використанні у розмітці змінних, відмежованих за допомогою спеціального синтаксису. Важливо, що шаблони не є представленнями, але можуть бути способом повного або часткового відображення об’єкту представлення.

Посередником між моделлю та представленням є контролер. Він відповідає за оновлення моделі внаслідок дій користувача та управління логікою у веб-застосунку.

Використання моделі, представлення і контролеру в шаблоні MVC формує принцип модулярності у розробці, що дозволяє уникнути багаторазового дублювання коду та відділити основну логіку, що виконується на клієнті, від інтерфейсу користувача.

Схема архітектурного шаблону MVC представлена на Рисунку 1.2.

Рисунок 1.2 – Схема архітектурного шаблону MVC

1.1.2 Архітектура MVP

Архітектура MVP (Model-View-Presenter) була сформована на основі вже розглянутої архітектури MVC. Основною ідеєю для неї стало перетворення та покращення логіки представлення [2].

Від MVC дана архітектура відрізняється наявністю такого компонента, як пред’явник, замість контролера. Він також пов’язує між собою представлення та модель, а ще містить бізнес-логіку користувацького інтерфейсу, яка необхідна для представлення. Пред’явник спостерігає за моделлю та оновлює представлення, коли вона зазнає змін, такчим чином маніпулюючи його станом.

Найбільш поширеною реалізацією MVP є пасивне представлення, що передбачає мінімальне використання логіки. Пред’явники визначають, що повинне відображатися в представленні, на основі обробки подій та отриманих

від користувача даних. При цьому доступ між представленням і моделлю є закритим.

Прив’язка даних у MVP можлива у різновиді, який передбачає використання наглядового конролеру. Ця варіація архітектури є ближчою до шаблонів MVC та MVVM.

MVP-архітектура забезпечує більш чітке розмежування між моделлю та представленням і найчастіше використовується у програмах корпоративного рівня, де є важливою взаємодія з користувачем та використання якомога більшого обсягу логіки представлення, побудованої на обробці користувацьких дій.

Схема архітектурного шаблону MVP представлена на Рисунку 1.3.

Рисунок 1.3 – Схема архітектурного шаблону MVP

1.1.3 Архітектура MVVM

MVVM (Model-View-ViewModel) є архітектурним шаблоном, що грунтується на MVC та MVP та намагається чітко визначати і відокремлювати бізнес-логіку (або логіку поведінки програми) та логіку користувацьких інтерфейсів. Вона було створена для розділення праці дизайнера та розробника.

Модель у MVVM – це ніщо інше, як структура даних, необхідних у роботі програми. Вона на впливає на те, як інформація відображується для користувача. За форматування даних відповідає представлення, а поведінка визначається на іншому рівні, який взаємодіє з моделлю – ViewModel.

Представлення є інтерактивним інтерфейсом, який є відображенням стану ViewModel. Воно може бути пасивним, виконуючі функцію демонстрації на екрані, або активним, приймаючи віхдні дані від користувача.

Модель вигляду (ViewModel) може розглядатися в якості контролеру, що перетворює дані, які до нього надходять. Вона пропонує методи оновлення моделі на основі дій користувача, обробляє дані та спілкується з представленням за допомогою подій та прив’язки даних.

Архітектура MVVM полегшує паралельну розробку інтерфейсу та логіки веб-застосунку.

Схема архітектури MVVM наведена на Рисунку 1.4.

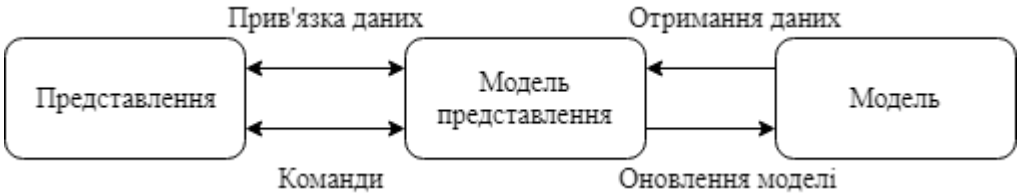

Рисунок 1.4 – Схема архітектури MVVM

1.1.4 Архітектура веб-компонентів

Веб-компоненти як різновид архітектури дуже поширилися у сфері веб-розробки останніми роками та суттєво змінили процес створення веб-застосунків. Вони є специфікацією консорціума W3C (World Wide Web Consortium) і основою компонентно-орієнтованого програмування, дозволяючи створення у веб-застосунках інкапсульованих повторно використовних віджетів [3].

Веб-компоненти пов’язані з такими технічними стандартами, як шаблони, користувацькі елементи, Shadow DOM (тіньова модель документу) та HTML imports (методи імпортування HTML документів в інші документи). Ці поняття є основними блоками, з яких складається веб-компонент [4].

З метою збільшення рівня абстракції навколо веб-компонентів будуються бібліотеки, їх концепція надихнула велику кількість сучасних фреймворків, що отримують власну реалізацію, використовуючи ідею створення веб-компонентів у різних способах.

1.1.5 Фреймворки та бібліотеки у побудові веб-застосунків

В даній роботі об’єктами дослідження є сучасні JavaScript фреймворки та бібліотеки.

Фреймворком у програмуванні називають певний шаблон для середовища виконання, який визначає деякі вимоги до архітектури програми, надаючи для цього необхідні методи та компоненти для полегшення процесу розробки.

Бібліотека є сукупністю програмних модулів та об’єктів, які використовуються для вирішення певних задач в ході розробки програмного забезпечення.

Одною з найважливіших грунтовних властивостей фреймворків, що дозволяє відрізняти їх від бібліотек, є інверсія управління, коли не не тільки розробник викликає зручні йому методи, а й сам фреймворк передбачає використання тих чи інших конструкцій для збереження загальної структури [5].

Чим більш розвиненою є бібліотека, тим більш вона стає схожою за структурою на фреймворк, і тому теж вимагає дотримання певних правил організації коду та побудови архітектури веб-застосунку, створюючи та пропонуючи відповідні шаблони. Цей факт дозволяє здійснювати порівняння між фреймворками та деякими бібліотеками, що розрізняються між собою за компонентним складом, доступною функціональністю та впливом на архітектуру веб-застосунку.

1.2 Основні вразливості веб-застосунків

Безпека веб-застосунків сьогодні є важливішим полем битви серед тих, хто покращує захищеність комп’ютерних ресурсів і даних, та зловмисниками.

Для того, щоб вміти оцінити рівень небезпеки у системі, аналізувати її та попереджувати, треба ознайомитися з переліком її можливих вразливостей.

Вразливістю називається недолік системи, який призводить до порушення її цілістності та подальшої некоректної роботи.

Деякі з вразливостей належать до теоретично існуючих, а деякі вже активно використовуються зловмисниками для виконання атак – неправомірних

дій, направлених на отримання контролю над системою, оволодіння конфіденційною інформацією чи на дестабілізацію системи.

Найбільш часто на поточний час зустрічаються вразливості, розвинуті з переліку вже відомих. Можливість їх експлуатація зросла і з появою нових технологій .

Згідно з класифікацією OWASP (Open Web Application Security Project) станом на 2017 рік існує десять найбільш поширених вразливостей у веб-застосунках, що можуть стати ризиком для безпеки [6]:

- ін’єкція (Injection);
- некоректна автентифікація (Broken Authentication);
- розголошення конфеденційних даних (Sensitive Data Exposure);
- зовнішні сутності (External Entities);
- недоліки контролю доступу (Broken Access Control);
- некоректне налаштування параметрів безпеки (Security Misconfiguration);
- міжсайтове виконання сценаріїв (Cross-Site Scripting);
- небезпечна десереалізація (Insecure Deserialization);
- використання компонентів із відомими вразливостями (Components with known vulnerabilities);
- недоліки журналювання та моніторингу (Insufficient Logging & Monitoring);

Ін’єкція – вразливість, що виникає внаслідок некоректної валідації вхідних даних і дозволяє зловмиснику виконати команди, що не передбачені поведінкою веб-застосунку, або отримати доступ до конфіденційної інформації,

що є недоступною без проходження авторизації. До цього типу належать SQL, NoSQL OS та LDAP Ін’єкції.

Некоректна автентифікація може бути проексплуатована внаслідок некоректної реалізації функцій, пов’язаних з управлінням сесіями, компрометацією паролів, токенами авторизації та іншими можливостями короткочасного або систематичного перехоплення даних облікових записів користувачів.

Розголошення конфіденційних даних відбувається через недостатнє приділення уваги методам шифрування інформації, що зберігається та обробляється у веб-застосунках. Це дозволяє здійснення несанкціонованих операцій з персональними, фінансовими або медичними даними користувача, що може призвести до отримання збитків, фізичної та моральної шкоди.

Зовнішні сутності, посилання на які піддаються обробці застарілими або погано налаштованими XML-процесорами, можуть бути використані для отримання доступу до внутрішніх файлів системи через віддалене виконання коду, спільні каталоги, відмову в обслуговуванні та сканування портів.

Недоліки контролю доступу складаються з відсутності контролю доступу на функціональному рівні та наявності небезпечних прямих посилань на об’єкти, що може призвести до зміни прав доступу та користувацьких даних.

Подібні наслідки можуть з’язвитися і у випадку некоректного налаштування параметрів безпеки (HTTP-заголовків, докладних повідомлень про помилки). Їм можна запобігти шляхом правильного налаштування операційних систем і фреймворків та їх систематичного оновлення.

Небезпечна десереалізація призводить до віддаленого виконання коду або сприяє атакам з повторним відтворенням та перевищенням привілеїв.

Бібліотеки, фреймворки та програмні модулі запускаються з правами веб-застосунку. Якщо вони мають у складі вразливості, їх експлуатація може призвести до втрати даних, перехоплення контролю над сервером та порушення захисту.

Недоліки журналювання і моніторингу пов’язані з відсутністю системи реагування на інциденти та дозволяють зловмиснику приховати свою присутність у системі, змінити дані або зруйнувати їх.

Кожна з перелічених вразливостей може стати ризиком та передумовою для порушення безпеки системи, але тільки деякі з них можна попередити за допомогою одних лише методів, застосованих на клієнтській стороні, або засобів фреймворку чи бібліотеки. Цей факт необхідно враховувати при проектуванні застосунку та реалізації його функціоналу, передбачати необхідні способи захисту даних за допомогою аутентифікації та авторизації, використання коректної системи машрутизації, валідації форм на веб-сторінках та інших механізмів.

1.3 Безпека веб-застосунків на стороні клієнта

JavaScript є потужним інструментом, що використовується у розробці веб-застосунків на стороні клієнта, але з його появою з’явилися і деякі проблеми безпеки, що отримали широку увагу.

Спосіб, за допомогою якого JavaScript взаємодіє з об’єктною моделлю документу (DOM), дозволяє зловмисникам впроваджувати скрипти через інтерфейс веб-застосунку і запускати їх на комп’ютері клієнта. Існують два методи, застосовуючи які можна цьому запобігти. Перший з них зумовлює створення пісочниць (sandboxing) або відокремлене виконання скриптів, завдяки чому отримується доступ тільки до визначених ресурсів і можливе

виконання лише пених завдань. Другий метод полягає у впровадженні політики обмеження домену, яка запобігає скриптам одного сайту мати доступ до даних, що використовуються і обробляються скриптами інших сайтів.

Багато вразливостей, знайдених у безпеці JavaScript, є результатом недостатньої продуманості веб-браузерів та відсутності у них засобів протидії ризикам безпеки, що пов’язані з об’єктною моделлю документу.

Дуже часто вразливості клієнтської частини веб-застосунку є результатом відсутності необхідної валідації вхідних/вихідних даних та експлуатуються для маніпуляцій з вихідним кодом програми або отримання неавторизованого доступу.

Також, з появою додаткових вразливостей пов’язане використання у розробці сучасних бібліотек та фреймворків, що, окрім зручності структури, створює додаткові шляхи для маніпуляцій із веб-застосунком та інформацією, що у ньому обробляється. Прикладом можуть слугувати атаки, що експлуатують недоліки шаблонів та прив’язки даних, що надають фреймворки, а також відомі вразливості у складових компонентах цих продуктів: залежних бібліотеках та модулях.

Розглянемо вразливості та атаки, які часто зустрічаються на стороні клієнта і мають повну або часткову можливість їх попередження.

1.3.1 Міжсайтове виконання сценаріїв

Міжсайтове виконання сценаріїв, або cross-site scripting (XSS) - це вразливість, що може бути знайдена у веб-застосунку у випадку обробки веб-застосунком ненадійних даних, що не назнали належної перевірки та форматування, або виконання на стороні клієнта ненадійого скрипту, який

змінює дерево DOM. Це одна з найпоширеніших атак для веб-застосунків, націлених на користувачів, щоб отримати доступ до облікових записів, активувати Трояни або змінити вміст веб-сторінки. Вона призводить до того, що при відкритті ненадійної сторінки на компютері користувача виконується небезпечний код, що взаємодіє з сервером зловмисника. Наслідками виконання такого небезпечного коду можуть бути:

- викрадення деталей сессії та кукі користувача;
- отримання доступу до історії браузера та вмісту буфера обміну;
- отримання доступу до віддаленого керування браузером;
- змінювання дерева DOM та порушення структури веб-сайту;
- перенаправлення користувачів на шкідливий веб-сайт.

Атака міжсайтового виконання сценаріїв може бути активною і пасивною.

У пасивному різновиді атаки для її спрацьовування потрібна дія користувача: прехід за посиланням, вставка певного коду у поле для вводу даних форми. Скрипт не виконується в браузері жертви автоматично.

При активній атаці скрипт зловмисника зберігається на сервері і автоматично спрацьовує при відкритті сайту жертвою.

1.3.2 Підробка міжсайтових запитів

Підробка міжсайтових запитів (Cross-Site Request Forgery, CSRF або XSRF, також відома як one-click attack or session riding) – це атака, що була у списку найбільших загроз безпеці веб-застосунків згідно з класифікацією OWASP у 2013 році.

Здійснюючи цю атаку, зловмисник отримує можливість виконувати дії під ім’ям авторизованого користувача, якому веб-сайт довіряє.

Атакуючий може вкрасти дані сесії користувача та його cookie, які він використовує для входу на довірений веб-сайт (наприклад, веб-сайт банку) та виконати якісь дії від його імені (наприклад, зробити грошовий переказ). Цей запит може бути відправлений за допомогою різних способів, таких як:

- відкриття сайту зловмисника у своєму веб-браузері;
- відкриття посилання на сайт зловмисника, що міститься в отриманому на електронну пошту листі;
- отримання доступу до зображення, яке може привести до перехоплювання інформації і виконування небажаних дій.

1.3.3 Включення міжсайтового сценарію

Включення міжсайтового сценарію (XSSI , Cross-Site Script Inclusion), також відома як вразливість спеціального формату даних JSON, знаходиться серед найбільш суттєвих та легко реалізовуваних атак. Вона дозволяє веб-сайту атакуючого прочитати дані з JSON API. Атака спрацьовує у старих браузерах через перевизначення конструкторів об’єктів у JavaScript та подальше включення API URL із використанням тегу <script>. Ця атака реалізується успішно тільки у тому випадку, якщо повернений JSON є виконуваним як JavaScript.

Висновки до розділу 1

В першому розділі були розглянуті основні загрози безпеки для веб-застосунків. Значною частиною вони залежать від коректної реалізації захисту від них як на стороні сервера, так і на стороні клієнта.

Клієнтська частина веб-застосунків базується на використанні такого інструменту, як JavaScript, і дуже часто виникаючі у ній вразливості є наслідком відсутності належної перевірки вхідних та вихідних даних. Деякі з них, повністю або частково, можна попередити на стороні клієнта.

Окремо слід зазначити існування вразливостей, що виникають у зв’язку з наявністю залежностей, або особливостями реалізованих у програмі архітектурних шаблонів, які надають сучасні фреймворки та бібліотеки. Використання цих продуктів певною мірою збільшує ризик порушення безпеки у веб-застосунках.

З плином часу виникають нові загрози безпеки, але значною кількістю вони засновані на використанні вже існуючих механізмів здійснення атак, тому їх розуміння є важливим для оцінки рівню небезпеки у системі, її аналізу та попередження.

2 ОГЛЯД СУЧАСНИХ ФРЕЙМВОРКІВ І БІБЛІОТЕК ТА ІСНУЮЧИХ МЕХАНІЗМІВ ЇХ ЗАХИСТУ

2.1 Вибір фреймворків і бібліотек для аналізу їх захищеності

Для здійснення аналізу рівня захищеності були прийняті до розгляду фреймворки та бібліотеки, що отримали найбільше розповсюдження у сфері веб-розробки. Згідно зі статистикою similartech.com [8] та опитуванням stateofjs [7], в якому брали участь понад 20000 розробників із різних країн з різною кількістю років досвіду роботи, проведеним щодо збору статистики використання бібліотек і фреймворків станом на 2018 рік, ними виявилися React, Vue.js, Angular, Ember.js та Polymer.js.

Це фреймворки, які розробники застосовують найчастіше, цікавляться ними та їх вивченням. До уваги приймалося співвідношення між відсотковими показниками серед людей, які використовують та збираються використовувати фреймворк у подальшій роботі, використовували, але не планують працювати з фреймворком, чули і хотіли б його вивчати, не зацікавлені і роботі з фреймворком чи бібліотекою або ніколи про нього не чули.

При аналізі та порівнянні характеристик фреймворків будуть розглянуті їх останні версії. Це Angular 7.2.15, React 16.8.6, Polymer 3.2.0, Vue.js 2.6.10 та Ember.js 3.10.0.

На Рисунку 2.1 відтворена діаграма статистичних даних щодо вказаного дослідження. Відсоткове співвідношення між категоріями опитаних показане у таблиці А.1 додатку А.

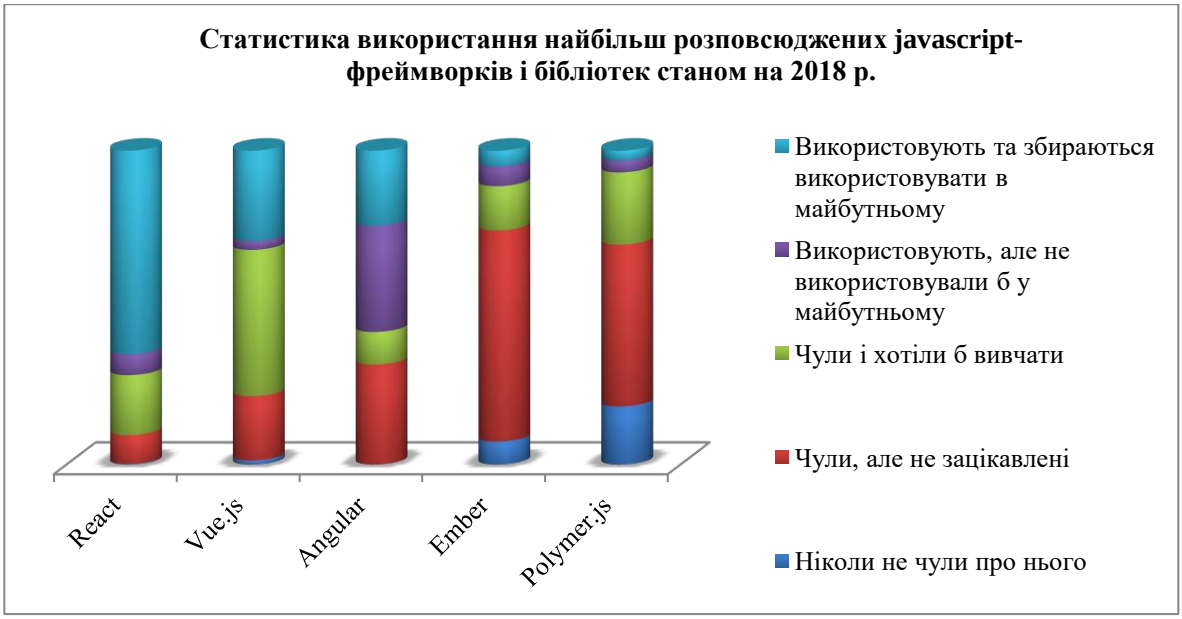

Рисунок 2.1 – Статистика використання найбільш розповсюджених JavaScript-фреймворків і бібліотек станом на 2018 р.

2.1.1 React

React або React JS– це JavaScript-бібліотека для побудови користувацьких інтерфейсів з компілятором JSX-синтаксису. Вона розроблюється та підтримується Facebook Inc. і зосереджена на полегшенні створення інтерфейсів за рахунок розбиття кожної сторінки на фрагменти, що зазвичай називають веб-компонентами. Змінюючи дані на сторінці без її перевантаження, може використовуватись для розробки мобільних застосунків та односторінкових веб-застосунків, в основі яких знаходиться один єдиний документ HTML.

Існує думка, що React належить до MVC-фреймворків, та це не зовсім так. Він є бібліотекою з відкритим вихідним кодом, що дозволяє рендеринг представлень, і має при цьому власну архітектуру.

Деякі з фреймворків передбачають чітке оформлення архітектури веб-застосунку, та багато з них дозволяють використання одного з декількох шаблонів або їх комбінацію, залишаючи вибір розробникам системи. Цей підхід реалізується і у React, роблячи можливим для нього застосування компонентної концепції.

Слід також пам’ятати, що веб-компоненти та React були розроблені з різною метою для вирішення різних завдань. Якщо головною ідеєю веб-компонентів є забезпечення інкапсуляції, то React створений для синхронізації об’єктної моделі документа DOM з даними веб-застосунку [9]. Обидва завдання є сумісними, і бібліотека дозволяє її використання у веб-компонентах та створення веб-компонентів, тим самим формуючи зручний інструмент розробки веб-застосунків.

2.1.2 Vue.js

Vue.js в останні роки стає все більш популярним у спільноті веб-розробників. Може застосовуватись як фреймворк, або як бібліотека функцій в залежності від потреб веб-застосунку.

Швидкий та продуктивний, він підходить для створення інтерфейсів користувача та односторінкових застосунків. Складається з основної бібліотеки, що в більшому зосереджена на рівні представлення, і сукупності допоміжних бібліотек, що розширюють його можливості.

Фреймворк також використовує систему компонентів, і в цьому має певну схожість з React. Архітектура Vue.js заснована на використанні шаблону NVVM, але реалізує його не повністю, залишаючись зосередженою на моделі вигляду, ViewModel частині цього патерну.

Розробники обирають Vue.js через його гнучкість та легкість у вивченні.

2.1.3 Angular

Angular створювався як вдосконалена версія AngularJS, розробленого командою Google. Оскільки його архітектура була переписана з нуля, згодом став позиціонуватися як окремий фреймворк, набувши значного поширення. При цьому переході була виправлена велика кількість помилок, враховуючі численні вразливості безпеки, та значно змінена структура.

Angular написаний на TypeScript і передбачає використання цієї мови у процесі написання коду з використанням продукту. На відміну від своєї попередньої версії, AngularJS, він не є повністю MVC-фреймворком, тому що заснований на компонентному підході. Кожен створений у процесі розробки компонент, визначений за всіма необхідними параметрами, можна представити як окрему реалізацію MVC-шаблону, оскільки він містить залежності, деталі представлення та оголошення класу з відповідною логікою, що може розглядатися як контролер. Представлення та логіка застосунку відокремлені, що спрощує розмітку і загальну структуру.

Фреймворк містить у собі велику кількість інструментів для розробки веб-застосунку, але це становить додаткову складність при його вивчанні. Він підтримує розробку для декількох платформ, включаючи десктопні та мобільні застосунки.

Маючи велику спільноту, Angular продовжує постійно розвиватися та вдосконалюватися.

2.1.4 Ember

Ember.js з’явився раніше за інші фреймворки, що розглядаються, і є усталеним у своїй концепції створення розширюваних односторінкових веб-застосунків. Він побудований на основі двох бібліотек: jQuery, створеній для полегшення роботи з DOM, та Handlebars, яка пропонує систему HTML шаблонів.

Хоча розповсюдженою думкою є те, що фреймворк реалізує MVC-архітектуру, слід зазначити, що контролер у ньому фактично виконує роль моделі вигляду. З цього випливає, що Ember.js заснований на MVVM-шаблоні, що дозволяє використовувати всі його переваги.

Ember.js має велику підтримку своєї спільноти, а також може позмагатися в універсальності та продуктивності з іншими сучасними фреймворками.

2.1.5 Polymer.js

Polymer — це JavaScript бібліотека для розробки веб-застосунків із використанням веб-компонентів. Вона була створена розробниками з Google та контриб'юторами на GitHub і являє собою набір поліфілів (надбудов для браузерів, що не підтримують ту чи іншу технологію) та синтаксичного цукру над стандартом веб-компонентів, запропонованим W3C [10].

Polymer дозволяє створювати власні елементи, подібні до елементів HTML. Розробка програм на основі елементів збільшує можливості для повторного використання коду у різних веб-застосунках.

Наявність поліфілів зменшує продуктивність фреймворку, тому, щоб її компенсувати, команда розробників Polymer зробила прив’язку даних та наявність зручних методів для обчислення та обробки даних дуже обмеженими.

Бібліотека підтримує створення односторінкових веб-застосунків, а також пропонує готові для використання компоненти.

2.2 Огляд механізмів захисту

Оскільки розробники кожного фреймворку прагнуть розширити аудиторію, що буде зацікавлена у його застосуванні, та підтримувати високий рівень якості програмного продукту, окрім вдосконалення функціональності, вони впроваджують і додаткові засоби безпеки. Треба пам’ятати, що використання їх на стороні клієнта не робить веб-застосунок повністю безпечним, тому що належний стан захищеності може бути досягнутий тільки при комплексному застосуванні методів як зі сторони клієнта, так і зі сторони сервера, а також при коректній автентифікації, авторизації та реалізації інших захисних механізмів на програмному рівні.

Існуючі механізми направлені на унеможливлення проведення певної атаки та автоматичний захист від неї, або часткову підтримку техніки захисту зі сторони клієнта. Кожній атаці відповідають різні способи протидії, деякі з котрих можна забезпечити засобами фреймворків, що використовуються у розробці.

Попередження деяких атак та вразливостей відбувається у фреймворках автоматично за замовчуванням (атаки міжсайтового виконання сценаріїв), для інших існують спеціальні методи, застосування яких розробниками сприяє підвищенню рівню захищеності веб-застосунку. Опис цих методів зазвичай

знаходиться у загальній документації продукту, або у документації з приводу безпеки, якщо така існує.

Частину існуючих технік захисту від потенційних загроз можна реалізувати засобами та методами JavaScript (наприклад, встановлення флагів в HTTP-cookie), вони не стосуються свідомого вибору того чи іншого фреймворку або бібліотеки. Використання цих методів у контексті вибраного програмного каркасу буде мати свої особливості та свій синтаксис, але не характеризуватиме програмний продукт с точки зори вбудованих наявних механізмів захисту, передбачаючи обізнаність розробника у необхідності їх використання, тому подібні техніки протидії у роботі не розглядалися.

Певні засоби захисту пов’язані із налаштуванням самих фреймворків. До них належить налаштування політики безпеки контенту, попередження несанкціонованого перенаправлення з веб-сайту, а також система оброблення помилок.

Існують також захисні механізми, що дозволяють попередження одразу декількох видів атак схожого типу, оскільки вони направлені на експлуатацію одних і тих самих ресурсів, прикладами яких можуть слугувати вхідні та вихідні дані програми, а також інформація, введена користувачем.

2.2.1 Наявність механізму автоматичної перевірки параметрів шаблонів та вилучення потенційно небезпечних конструкцій

Некоректна перевірка вхідних даних є найбільш поширеною слабкістю безпеки веб-застосунків. Для того, щоб захистити веб-застосунок від атак і їх наслідків, всі дані, що відправляються користувачами (через API або відправку форми), повинні бути перевірені та конвертовані в надійні значення, тому саме з

цією метою у багатьох JavaScript фреймворках проводиться санітизація значень – інспектування коду, видалення його ненадійних складових та форматування у валідне значення, яке є безпечним для його подальшого поміщення у DOM.

Зазвичай валідація та трансформування вхідних та вихідних даних слугують для попередження багатьох вразливостей і атак, таких як різного типу ін’єкції, атаки файлової системи або міжсайтове виконання сценаріїв, і є автоматичними, але є фреймворки, у яких перевірка не виконується за замовчуванням, і отримання методів санітизації передбачає підключення окремих модулів або бібліотек.

У веб-затосунках, побудованих з використанням бібліотек або фреймворків, важливою є фільтрація та автоматичне вилучення небезпечних символів та значень у шаблонах.

2.2.2 Захист від підробки міжсайтових запитів (CSRF або XSRF)

Для того, щоб попередити цю атаку, треба впевнитися у тому, що запит користувача йде від справжнього веб-застосунку, а не з іншого ресурсу. З метою запобігання наслідкам клієнт і сервер повинні діяти разом.

У загальній анти-XSRF техніці сервер застосунку надсилає випадково згенерований токен (ключ) автентифікації у файлі cookie. Код клієнта читає cookie і додає спеціальний заголовок до запиту з токеном 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, тому що всі браузери реалізують політику того ж походження. Лише код з веб-сайту, на якому встановлюються файли cookie,

може читати файли cookie з цього сайту і встановлювати користувацькі заголовки на запити до цього сайту. Це означає, що тільки конкретна програма може прочитати цей токен cookie і встановити спеціальний заголовок. Зловмисний код з будь-якого іншого сайту не має такої можливості.

Фреймворк або бібліотека можуть пропонувати часткову релізацію цієї техніки на стороні клієнта.

2.2.3 Захист від включення міжсайтового сценарію (XSSI)

Атаку можна попередити зі сторони сервера шляхом префіксації всіх відповідей JSON, щоб зробити їх невиконуваними, за конвенцією, використовуючи відомий рядок ")]} ', \ t.

Засобами бібліотеки або фреймворку можуть бути впроваджені методи роботи з HTTP-запитами, що передбачають розпізнавання цього перевірочного рядку та видалення його перед подальшим зчитуванням файлу та його обробкою.

Ця техніка, як і вбудовані методи захисту від підробки міжсайтових запитів, забезпечує лише підтримку обраних засобів захисту, застосування яких відбувається на сервері.

2.2.4 Налаштування політики безпеки контенту

Політика безпеки контенту – це комп’ютерний стандарт, що сприяє запобіганню XSS-атакам, клікджекінгу та ін’єкціям коду. Він визначає правила використання вбудованих стилів, скриптів, а також динамічної оцінки JavaScript і джерел завантаження на сторінку даних.

Налаштування політики безпеки контенту передбачає розміщення на веб-сторінці Content-Security-Policy заголовку та надання йому значення, що містить певні правила управління ресурсами, на які можуть відправлятися запити та які можуть бути завантажені на сторінці користувача.

Згідно з критеріями оцінки безпечності використання шаблонів на стороні клієнта, запропонованими проектом mustache-security [11], фронт-енд фреймворк може дозволяти або сприяти використанню безпечних правил політики безпеки контенту, підтримуючи її налаштування за замовчуванням.

2.2.5 Попередження несанкціонованого перенаправлення з веб-сайту

Якщо в результаті дій користувача перенаправлення на інший, потенційно небезпечний сайт, вже відбувається, фреймворки та бібліотеки можуть застосовувати засоби інформування користувача про можливу загрозу від ненадійного посилання, або запобігати перенаправленню, вдаючись до попередження поведінки посилань за замовчуванням.

2.2.6 Система автоматичної обробки помилок

Зловмисники часто використовують той факт, що при виникненні помилки у веб-застосунках поведінка програми іноді може стати непередбачуваною, надавши можливість для перегляду конфіденційної інформації, або будь-яких даних, за допомогою яких цю інформацію можна отримати.

Розробникам програмного забезпечення важко передбачити всі сценарії, які можуть визвати нестандартну реакцію системи та спричинити сбій у її поведінці, але сучасні засоби розробки, зокрема фронт-енд фреймворки, можуть мати вбудовану систему автоматичної обробки помилок, що зменшує ймовірність ризику, пов’язаного із їх виникненням.

Висновки до розділу 2

У другому розділі був здійснений огляд розповсюджених JavaScript фреймворків і бібліотек, а також інсуючих вбудованих механізмів захисту.

Для дослідження були обрані саме ті технології, які мають тенденцію до поширення у використанні. Вони розрізняються за архітектурою і були розроблені для виконання різних завдань, але всі пропонують зручний набір інструментів для створення веб-застосунків.

Часто фреймворки та бібліотеки, що використовуються для розробки клієнтської частини веб-застосунків, мають вбудовані механізми захисту від потенційно можливих загроз, або підтримку певних засобів захисту, що реалізуються на сервері.

Існуючі способи протидії грунтуються на виконанні певних вимог до обробки та передачі даних у веб-застосунках, а також застосуванні та підтримці спеціального функціоналу чи певних властивостей.

Наявність механізмів захисту дозволяє зменшити ймовірність ризиків для веб-застосунку та підвищити рівень його захищеності.

3 ФОРМУВАННЯ ОЦІНКИ ЗАХИЩЕНОСТІ JAVASCRIPT-ФРЕЙМВОРКІВ І БІБЛІОТЕК

3.1 Вибір критеріїв захищеності та методів їх оцінювання

В ході дослідження обраних фреймворків і бібліотек з метою формування оцінки їх захищеності були створені спеціальні критерії, кожний з яких характеризується певним показником і впливає на процес визначення загальної оцінки безпечності фреймворку.

При формуванні цих критеріїв враховувалась необхідність наявності захисту від переіліку найбільш розповсюджених загроз для веб-застосунків OWASP Top 10, використовувалась документація фреймворків, що досліджуються, а також стандарт перевірки відповідності безпеки застосунків ASVS 4.0 (Application Security Verification Standard) проекту OWASP, основною метою якого є надання критеріїв, за якими відбувається встановлення рівня довіри в безпеці веб-застосунків [12]. Сформовані критерії грунтуються на тих вимогах, які мають можливість реалізації вибраними в ході дослідження фреймворками та бібліотеками. До критеріїв належать наступні:

1. Серйозність знайдених вразливостей.
2. Наявність вбудованих механізмів захисту від розповсюджених атак.
3. Серйозність вразливостей у залежностях.
4. Наявність політики безпеки.
5. Наявність контактів для звернення з приводу питань безпеки.
6. Наявність документації з приводу безпеки.
7. Наявність Bug Bounty програм.

Кожному критерію призначається оцінка, значення її для кожного критерію нормується і виражається в шкалі від 0 до 1 в залежності від ступеню відповідності критерію фреймворку або бібліотеки.

3.1.1 Серйозність знайдених вразливостей

Вразливості, знайдені у фреймворках та бібліотеках, становлять загрозу для веб-застосунку, що використовує їх в якості інструменту розробки, тому при визначенні захищеності треба враховувати їх вплив на безпеку програми та наявність у фреймворку тенденції до виникнення помилок безпеки.

У контексті критерію розглядаються вразливості, знайдені починаючи з моменту виходу першої версії фремворку, тому що кількість вже допущених помилок, пов’язаних з безпекою, може свідчити про можливість і частоту появи інших помилок такого типу в майбутньому та загальну схильність до них команди розробників фреймворку.

Існує декілька баз даних, що зберігають списки вразливостей програмних продуктів там систем, наприклад, SecurityFocus BID, NVD, OSVDB, Secunia, IBM ISS X-Force. Декі з них припинили свою роботу або публікацію та надання відкритого доступу до баз даних.

Для більш повного аналізу вразливостей, знайдених у програмних продуктах, у роботі використовувались співставлені дані з декількох баз даних вразливостей: CVE, Snyk та Vulners.

База даних загальновідомих вразливостей CVE (Common Vulnerabilities and Exposures) ведеться з 1999 року, являє собою достатньо повний список вразливостей та містить велику кількість посилань на інші БД і виробників програмних та апаратних засобів. Основною перевагою цієї бази даних є те, що

вона є найбільш повною та систематизованою, тому її часто використовують як основу у вказівках відповідностей записів і інших базах [13].

База даних Snyk.io є досить популярним інструментом та дозволяє знайти вразливості у залежностях, якими можуть вважатися бібліотеки та фреймворки, встановлені у форматі додаткових пакетів.

Дуже великою базою з неперервним поповненням даних є Vulners [14]. Вона містить опис вразливостей, представлених у багатьох базах даних та досліджувальних центрах (наприклад, Vulnerability Lab, CERT, ICS, Positive Technologies, ERPScan), надає можливість пошуку експлоітів з Exploit-DB і Metasploit, помилок безпеки, знайдених за допомогою Bug Bounty програм, та інших публікацій, пропонуючи різноманітність ресурсів.

Кількість вразливостей не завжди може дати точну характеристику стану захищеності системи, тому в якості критерію було обрано серйозність знайдених вразливостей. Цей параметр визначає їх загальний вплив на безпеку і заснований на використанні CVSS (Common Vulnerability Scoring System) – системи ранжирування вразливостей [15]. Він обчислюється за формулою:

$Vulnerability(p_k) = \sum_{i=1}^{n} BaseScore(v_i)$, де

$BaseScore(v_i)$ – базова оцінка CVSS для вразливості v_i.

i – порядковий номер вразливості у фреймворку, $i = [1, n]$, де n – кількість вразливостей,

p_k – k-й продукт, що розглядається.

Для отримання значення порівняно з іншими фреймворками обчислюємо показник і нормуємо його відносно найбільшого показника з отриманих:

$$N\_Vulnerability(p_k) = 1 - \sum_{i=1}^{n} \frac{Vulnerability(p_k)}{\max\limits_{k} Vulnerability(p_k)}$$

3.1.2 Наявність вбудованих механізмів захисту від розповсюджених атак

Наявність вбудованих механізмів захисту у фреймворках буде визначатися за наступним списком:

1. Наявність механізму автоматичної перевірки параметрів шаблонів та вилучення потенційно небезпечних конструкцій.
2. Підтримка техніки захисту від CSRF зі сторони клієнта.
3. Автоматичний захист від включення міжсайтового сценарію (XSSI).
4. Налаштування політики безпеки контенту.
5. Попередження несанкціонованого перенаправлення з веб-сайту.
6. Система автоматичної обробки помилок.

Як зазначається у стандарті ASVS, найбільш поширеним недоліком безпеки веб-застосунків є відсутність належної перевірки вхідних даних, які надаються клієнтом або середою, перед їх безпосереднім використанням у процесі обробки та передачі. Саме ця слабкість використовується для багатьох атак, які завдають значної шкоди веб-застосунку, наприклад, різних типів ін’єкцій та міжсайтового виконання сценаріїв. У певних випадках важко забезпечити надійну перевірку даних, тому критично важливим для безпеки застосунку є використання більш безпечного API, наприклад, наявність автоматичного вилучення небезпечних значень з параметрів шаблонів у фреймворках.

Згідно з рекомендаціями стандарту до контролю доступу на операційному рівні, слід перевірити, що застосунок або фреймворк має механізм захисту від

атаки CSRF для попередження несанкціонованого доступу до функціональності та даних, для перегляду яких потрібна автентифікація.

Також вимоги стандарту щодо валідації та санінтизації у веб-застосунках кажуть про необхідність наявності захисту від ін'єкцій JavaScript та JSON, серед яких атаки, побудовані на використанні функціїї eval, та CSP bypass. Це передбачає необхідність існування у застосунку або фреймворку захисту від атаки включення міжсайтового сценарію та налаштувань політики безпеки контенту.

Неперевірені перенаправлення з веб-сайту та переадресації вважалися одними з найбільш поширених атак за OWASP Top 10 2013 року [16], тому захист від несанкціонованого перенаправлення, що може бути реалізований фреймворком або бібліотекою, був доданий до списку необхідних механізмів захисту.

Відповідно до вимог з обробки помилок ASVS, наявність системи обробки помилок у фреймворку також враховувалась у обчисленні оцінки відповідності критерію.

Кожен фреймворк або бібліотека отримує по цьому критерію бал від 0 до 6, після чого значення переводиться у шкалу від 0 до 1. Тобто:

$$Mechanisms\_Score\ (p_k) = \frac{Existing\_Score(p_k)}{General\_Score}$$

Де $Existing\_Score(p_k)$ – бал, отриманий програмним продуктом в залежності від кількості наявних вбудованих механіхмів захисту.

$General\_Score$ – загальна кількість існуючих механізмів захисту. Значення дорівнює 6.

3.1.3 Серйозність вразливостей у залежностях

Використання компонентів із відомими вразливостями є однією з десяти найбільш впливових загроз згідно з проектом OWASP Top 10 2017, отже наявність у фреймворках залежностей від інших бібліотек або потенційно вразливих модулів може стати джерелом додаткової загрози, тому слід враховувати цей показник при підрахунку оцінки захищеності.

Аналогічно критерію знайдених вразливостей, будемо оцінювати не їх загальну кількість, а серйозність впливу на безпеку веб-застосунку. Оскільки немає сенсу враховувати вразливості залежностей за увесь час інсування через їх велику кількість та відсутність чіткого зв'язку між версією залежного компоненту та версією досліджуваного фреймворку, будемо розглядати вразливості, що присутні в останній стабільній версії фреймворку або бібліотеки.

В якості інструменту знаходження вразливостей у складових компонентах було обрано Retire.js – сканер, що запускається за допомогою командного рядку і допомагає ідентифікувати наявність вразливих модулів. Інструмент по черзі було застосовано до кожного з веб-застосунків, створених стандартним способом з використанням інструменту командного рядку того чи іншого фреймворку або бібліотеки без будь-яких модулів, що потребують додаткового встановлення.

Показник серйозністі вразливостей у залежностях $N\_Dep\_Vulnerability(p_k)$ обчислюється подібно до показнику серйозності знайдених вразливостей.

3.1.4 Наявність політики безпеки

Згідно з вимогами щодо безпечного циклу розробки програмного забезпечення стандарту ASVS, треба впевнитися у наявності в системі чіткої документації або рекомендацій щодо безпеки та політики безпеки.

Політика безпеки належить до організаційних засобів захисту інформації і є набором рекомендацій, спрямованих на досягнення інформаційної безпеки і мінімізацію ризиків для системи.

Показник відповідності цьому критерію обчисляються наступним чином:

$$Security\_Policy\_Score = \begin{cases} 1, & \text{якщо політика безпеки існує} \\ 0, & \text{інакше} \end{cases}$$

3.1.5 Наявність контактів для звернення з питань безпеки

Критерій наявності контактів з приводу питань безпеки є дуже важливим, оскільки велика кількість вразливостей у програмному продукті може бути знайдена самими користувачами. Якщо вони не матимуть можливості повідомити про це розробників, недоліки безпеки так і залишаться невиправленими, і їх зможе використати зловмисник.

Запитів на прийняття змін на Github (pull request) та звичайних контактів офіційного сайту, призначених для обговорення всіх питань, що стосуються підтримки фреймворку або бібліотеки, може виявитися недостатньо, оскільки повідомлення про існуючу загрозу може просто загубитися серед безлічі інших.

Окремі контакти для питань безпеки передбачають акцентованість на знайденні та виправленні вразливостей та свідчать про зацікавленість розробників у покращенні стану захищеності продукту.

Оцінка критерію:

$$Security\_Contacts\_Score = \begin{cases} 1, & \text{якщо контакти безпеки існують} \\ 0, & \text{інакше} \end{cases}$$

3.1.6 Наявність документації з приводу безпеки

Один з критеріїв ASVS зазначає, що первірка визначеності всіх компонентів програми у документації з точки зору функцій безпеки є обов'язковою вимогою до архітектури бізнес-логіки.

Документація щодо безпеки містить опис найкращих практик вживання тих чи інших конструкцій або методів програмного продукту, рекомендацій для підтримання належного рівню безпеки та існуючих засобів захисту, що може пропонувати бібліотека або фреймворк. Окремо у ній зазначені дії, які можуть призвести до збільшення ризиків безпеки.

Наявність документації дозволяє зменшити ймовірність виникнення загроз для веб-застосунку шляхом попередження некоректного використання методів та висвітлювання важливих аспекти безпеки.

Оцінка відповідності критерію:

$$Security\_Documentation\_Score = \begin{cases} 1, & \text{якщо документація існує} \\ 0, & \text{інакше} \end{cases}$$

3.1.7 Наявність Bug Bounty програм

Bug Bounty програми пропонуються багатьма компаніями та розробниками, які турбуються про якість свого програмного продукту. Вони передбачають наявність певного механізму, за допомогою якого люди можуть отримати винагороду за знаходження вразливостей і помилок безпеки та повідомлення про них. Такі програми проводяться для того, щоб уникнути знаходженя програмних багів широкою аудиторією, виправивши вразливість до того, як вона буде проексплуатована.

Наявність подібних програм у власників сучасних фронт-енд бібліотек та фреймворків була додана як окремий критерій, тому що свідчить про схильність до вчасного попередження загроз та усування вразливостей.

Фреймворк отримує 1 бал, якщо має подібні програми, і 0 – якщо ні.

$$Bug\_Bounty\_Score = \begin{cases} 1, & \text{якщо програми існують} \\ 0, & \text{інакше} \end{cases}$$

3.2 Огляд методу оцінки захищеності

Оцінка захищеності фреймворку або бібліотеки формується з урахуванням кожної з оцінок критеріїв захищеності та коефіцієнтів важливості, визначених для кожного критерія за допомогою експертного оцінювання.

Визначимо вектор $G = (g_1,\ g_{12}, \dots ,\ g_n)$ як вектор оцінок критеріїв, де n – кількість критеріїв.

Тоді оцінка захищеності фреймворку I буде вираховуватися за формулою:

$$I = \sum_{j=1}^{n} w_j g_j,$$

Де w_j – експертна оцінка важливості критерію і $\sum_{j=1}^{n} w_j = 1$.

Для використання формули оцінки всіх критеріїв повинні бути нормалізовані в інтервалі $[0, 1]$, де 0 – повне невиконання вимог критерію, а 1 – повна відповідність вимогам критерію.

Максимальній захищеності фреймворку відповідає оцінка 1. Мінімальній захищеності – оцінка 0.

3.3 Експертне оцінювання важливості критеріїв

Для визначення важливості критеріїв захищеності сучасних JavaScript фреймворків і бібліотек було проведене експертне оцінювання – процедура отримання оцінки на основі думки фахівців з метою подальшого прийняття рішення.

Залучення експертів сприяє формуванню більш комплексного підходу у вирішенні проблеми. У цій роботі такою проблемою була нерівнозначність обраних критеріїв захищеності, необхідність їх порівняння та впорядкування.

При проведенні оцінювання використовувався підхід індивидуальних оцінок, отриманих в результаті анкетування, заснований на використанні висновків окремих фахівців незалежно один від одного.

3.3.1 Визначення чисельності та складу експертної групи

При виборі чисельності експертної групи слід керуватися такими факторами як достовірність інформації, що буде отримана після узгодження оцінок, та наявністю необхідних ресурсів для проведення експертизи.

Оптимальна чисельність експертів знаходиться в межах від п’яти до п’ятнадцяти людей. При меншій кількості статистичний аналіз не завжди є можливим, оскільки цих результати можуть виявитися занадто різними. Зі збільшенням числа експертів зростає значення середньої групової похибки [17]. Графік залежності чисельності експертів від значення групової похибки показано на Рисунку 3.1.

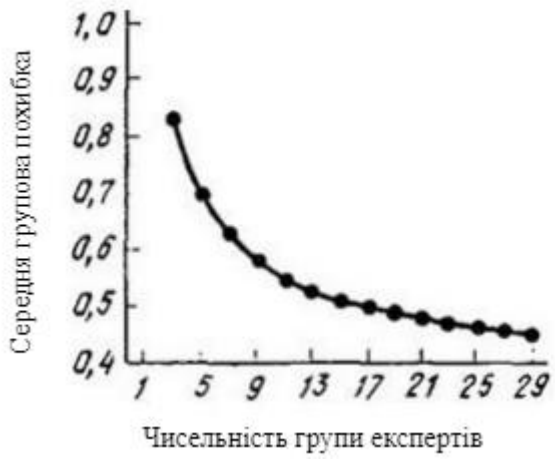

Рисунок 3.1 – Графік залежності середньої групової похибки від чисельності групи експертів

З урахуванням цих показників було встановлено чисельність групи у сім експертів.

До складу групи увійшли фахівці, що мають досвід у сфері інформаційних технологій більше п’яти років, а також сертифіковані лідерами галузі

інформаційної безпеки: Check Point, Cisco Systems, ESET, IBM, Oracle, Splunc, Symantec. Експерти володіють сертифікатами ISO 27001 ISMS Auditor/Lead Auditor та Certified Ethical Hacker.

3.3.2 Вибір методу експертного оцінювання

Для оцінки важливості критеріїв може використовуватись кількісна або порядкова шкала. У роботі було вирішено застосувати саме порядкову шкалу, оскільки згідно з численними випробуваннями, людина більш правильно (і з меншими труднощами) відповідає на питання якісного, наприклад, порівняльного характеру, ніж кількісного [18].

Для визначення важливості будемо використовувати метод бінарних порівнянь критеріїв за значимістю із застосуванням шкали Сааті, яку він визначив для методу аналітичної ієрархії.

1– рівна важливість критеріїв;

3 – помірна перевага одного критерію над іншим;

5 – суттєва перевага одного критерю над іншим;

7 – значна перевага одного критерію над іншим;

9 – дуже сильна перевага одного критерію над іншим.

Значення 2, 4, 6 і 8 використовуються як проміжні між двома сусідніми компонентами.

Експертам було запропоновано заповнити таблицю, де на перетині рядків та стовбців через дріб записуються значення, що є відображенням переваги одного критерію над іншим на думку експерта.

Для кожного j-го критерію, $j = 1,\ 2, \ldots, 7,$ оціненого i-м експертом визначалась загальна сума по рядку:

$$S_j^i = \sum_{i=1}^{n} O_{ij}\,, i = [1, n]$$

Де n - кількість критеріїв і O_{ij} - значення порівняння одного критерію з іншим.

В якості показника важливості, встановленого в результаті обробки даних опитування, для кожного з n критеріїв вираховувалася вага:

$$w_j^i = \frac{S_{ij}}{\sum_{j=1}^{n} S_{ij}}$$

Отримані показники на думку кожного з експертів використовувались при розрахунку загальної експертної оцінки.

Приклад заповненої анкети експерта наведено у Таблиці 3.2.

Таблиця 3.2 – Приклад заповненої анкети експерта

Номер критерію	1	2	3	4	5	6	7	Сума по рядку	Вага
1	1/1	1/7	1/3	1/5	3/1	1/5	7/1	11.88	0.098
2	7/1	1/1	7/1	5/1	7/1	1/5	9/1	36.20	0.298
3	3/1	1/7	1/1	3/1	3/1	1/3	7/1	16.33	0.134
4	5/1	1/5	1/3	1/1	3/1	3/1	7/1	19.53	0.161
5	1/3	1/7	1/3	1/3	1/1	1/5	7/1	9.34	0.077
6	5/1	5/1	3/1	1/3	5/1	1/1	7/1	26.33	0.217
7	1/7	1/9	1/7	1/7	1/7	1/7	1/1	1.83	0.015

3.3.3 Узгодження експертних оцінок

Формування узагальненої оцінки можливе при об'єднанні індивідуальних експертних суджень. Згідно з дослідженнями Т.Сааті, для виконання цього існує спосіб, що полягає у перемноженні відповідних чисельних значень суджень та добуванні кореня степеню k, де k – число учасників, а саме знаходження геометричного середнього [19].

Для обробки експертних даних будемо використовувати наступні позначення:

m – кількість учасників експертизи;

$i = 1, 2, \ldots, m$ – порядковий номер експерта;

n – кількість критеріїв, для яких здійснюється оцінка важливості.

$j = 1, 2, \ldots, n$ – порядковий номер критерію;

m_j – кількість експертів, що оцінили j-й критерій;

m'_j – максимальна кількість оцінок, що отримує j-й критерій;

C_{ij} – оцінка важливості i-м експертом j-го критерію.

Тоді узагальнена оцінка важливості:

$$w_j = \sqrt[k]{\prod_{i=1}^{m} C_{ij}}$$

Чим більшим є значення w_j, тим вище важливість j-го критерію.

Спираючись на дані результатів експертного опитування, отримуємо:

$$w_1 = \sqrt[7]{\prod_{i=1}^{7} C_{i1}} = \sqrt[7]{0.37 * 0.07 * 0.27 * 0.1 * 0.25 * 0.24 * 0.26} = 0.1955 = 0.2$$

$$w_2 = \sqrt[7]{\prod_{i=1}^{7} C_{i2}} = \sqrt[7]{0.16 * 0.11 * 0.2 * 0.3 * 0.29 * 0.13 * 0.21} = 0.1882 = 0.19$$

$$w_3 = \sqrt[7]{\prod_{i=1}^{7} C_{i3}} = \sqrt[7]{0.26 * 0.25 * 0.17 * 0.13 * 0.16 * 0.2 * 0.21} = 0.1921 = 0.19$$

$$w_4 = \sqrt[7]{\prod_{i=1}^{7} C_{i4}} = \sqrt[7]{0.04 * 0.12 * 0.12 * 0.16 * 0.14 * 0.1 * 0.14} = 0.1088 = 0.1$$

$$w_5 = \sqrt[7]{\prod_{i=1}^{7} C_{i5}} = \sqrt[7]{0.04 * 0.12 * 0.1 * 0.08 * 0.02 * 0.15 * 0.1} = 0.0734 = 0.07$$

$$w_6 = \sqrt[7]{\prod_{i=1}^{7} C_{i6}} = \sqrt[7]{0.07 * 0.12 * 0.12 * 0.22 * 0.11 * 0.1 * 0.05} = 0.1029 = 0.1$$

$$w_7 = \sqrt[7]{\prod_{i=1}^{7} C_{i7}} = \sqrt[7]{0.05 * 0.2 * 0.02 * 0.02 * 0.02 * 0.06 * 0.02} = 0.0371 = 0.04$$

Результати узгодження експертних оцінок в якості отриманих значень показників важливості наведені у Таблиці 3.3.

Таблиця 3.3 – Значення показників важливості критеріїв, отримані в результаті експертного оцінювання

Критерій	Оцінка важливості
Серйозність знайдених вразливостей	0.2
Наявність вбудованих механізмів захисту від розповсюджених атак	0.19
Серйозність вразливостей у залежностях	0.19
Наявність політики безпеки	0.1
Наявність контактів для звернення з приводу питань безпеки	0.07
Наявність документації з приводу безпеки	0.1
Наявність Bug Bounty програм	0.04

3.4 Результати дослідження фреймворків

Для кожного фреймворку та бібліотеки, що розглядаютсья в роботі, приведені дані, отримані в результаті їх дослідження за кожним з розроблених критеріїв. До них належать опис та основні характеристики вразливостей програмного продукту із вказанням тяжкості та оцінки за шкалою CVSS, вразливості, наявні у залежних модулях та бібліотеках, та список існуючих засобів захисту від загроз.

Для зручності результати оформлені у вигляді таблиць.

3.4.1 Результати React

У React за час існування було знайдено три вразливості середнього та високого рівню тяжкості. Опис вразливостей та відповідні їм оцінки по шкалі CVSS наведені у Таблиці 3.4.

Таблиця 3.4 – Вразливості React за час існування

Вразливість	Опис вразливості	Тяжкість вразливості (severity)	CVSS оцінка
CVE-2018-6341	React applications which rendered to HTML using the ReactDOMServer API were not escaping user-supplied attribute names at render-time. That lack of escaping could lead to a cross-site scripting vulnerability. This issue affected minor releases 16.0.x, 16.1.x, 16.2.x, 16.3.x, and 16.4.x. It was fixed in 16.0.1, 16.1.2, 16.2.1, 16.3.3, and 16.4.2.	Medium	6.1

Продовження таблиці 3.4

CVE-2013-7035	Affected versions >=0.5.0 <0.5.2 || >=0.4.0 <0.4.2 of the package contain Cross-site Scripting (XSS) vulnerability due to unescaped text before inserted into the DOM.	Medium	6.5
npm:react:20150318	Affected versions <0.14.0 are vulnerable to Cross-site Scripting (XSS) due to the createElement method not validating the object, allowing a malicious user to pass a specially crafted JSON object and renders them as an element.	High	7.1

Синтаксис, що використовується при написанні шаблонів React - JSX – автоматично перетворює всі інтерпольовані значення у безпечні та надійні та конвертує в рядок перед відображенням їх у DOM, таким чином попереджуючи виконання скриптів, що містяться у вхідних даних користувача. Метод innerHTML, який дозволяє вставку HTML-фрагменту у сторінку, у React також має обгортку dangerouslySetInnerHTML, що валідує дані, які до нього надходять, тим самим не дозволяючи використовувати метод для атак.

Бібліотека також автоматично перевіряє код CSS перед тим як застосувати його до властивостей стилів елементів HTML.

React не має в API жодного методу для роботи з CSRF-токенами. Веб-застосунки, що використовують цю бібліотеку, повинні мати власний впроваджений захист від цієї атаки.

React наразі не є сумісним із сайтами із підключеною політикою безпеки контенту та не містить жодних методів API для валідації JSON на стороні клієнта для захисту від XSSI, хоча реалізація його засобами бібліотеки могла б виявитись не дуже складною.

Бібліотека не має системи автоматичної обробки помилок, але попереджує несанкціоноване перенаправлення зі сторінки.

Вбудовані механізми захисту для бібліотеки ілюструє Таблиця 3.5.

Таблиця 3.5 – Вбудовані механізми захисту у React

Механізм захисту	Наявність
Автоматична перевірка параметрів шаблонів та вилучення потенційно небезпечних конструкцій	Так
Підтримка техніки захисту від CSRF зі сторони клієнта	Ні
Автоматичний захист від XSSI	Ні
Підтримка політики безпеки контенту	Ні
Попередження несанкціонованого перенаправлення з веб-сайту	Так
Система автоматичної обробки помилок	Ні

Вразливості складових компонентів React, кожна з яких має рівень medium, відображені у Таблиці 3.6.

Таблиця 3.6 – Вразливості у залежностях бібліотеки React

Назва модулю	Вразливий компонент	Назва вразливості	Тяжкість (Severity)	Опис вразливості	Оцінка CVSS
webpack-dev-server	jquery 1.8.1	CVE-2012-6708	medium	Selector interpreted as HTML	6.1
		CVE-2015-9251	medium	3rd party CORS request may execute	6.1
		CVE-2015-9251	medium	parseHTML() executes scripts in event handlers	6.1
		CVE-2019-11358	medium	jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution	6.1

Політика безпеки, контакти з приводу питань безпеки та будь-яка документація щодо безпеки на офіційному сайті відсутня.

Компанія Facebook, що займається розробкою бібліотеки, має спеціальну bug bounty програму для забезпечення своєчасного розкриття помилок безпеки. Це зазначено на офіційному сайті React.

3.4.2 Результати Vue.js

Vue.js має дві вразливості, знайдених під час існування, серед яких нема жодної, що мала б високий рівень. Опис вразливостей наведено у Таблиці 3.7.

Таблиця 3.7 – Вразливості фреймворку Vue.js

Вразливість	Опис вразливості	Тяжкість вразливості (severity)	CVSS оцінка
npm:vue:20180222	Affected versions <=2.5.14 of this package are vulnerable to Regular Expression Denial of Service (ReDoS) attacks.	Low	3.7
npm:vue:20170401	Affected versions <2.5.17 of the package are vulnerable to XSS	Medium	6.5

Vue.js не має вбудованого санітайзеру значень. Засновник фреймворку пояснює це тим, подібний вбудований модуль додасть ваги фреймворку і разом із тим буде рідко застосовуватись, оскільки у більшості випадків директива v-html, призначена для додавання на сторінку фрагментів html коду використовується для роботи з надійними значеннями.

З усіх засобів захисту, що пропонуються до розгляду, фреймворк має тільки попередження несанкціонованого перенаправлення з веб-сайту, демонструючи при цьому надзвичайно низький показник.

Наявність вбудованих механізмів захисту показана у Таблиці 3.8.

Таблиця 3.8 – Вбудовані механізми захисту Vue.js

Механізм захисту	Наявність
Автоматична перевірка параметрів шаблонів та вилучення потенційно небезпечних конструкцій	Ні
Підтримка техніки захисту від CSRF зі сторони клієнта	Ні
Автоматичний захист від XSSI	Ні
Підтримка політики безпеки контенту	Ні
Попередження несанкціонованого перенаправлення з веб-сайту	Так
Система автоматичної обробки помилок	Ні

Вразливості складових компонентів фреймворку теж не відрізняються великою кількістю. Їх список наведено у Таблиці 3.9. До нього входить лише одна вразливість середнього рівня тяжкості.

Таблиця 3.9 – Вразливості залежностей Vue.js

Назва модулю	Вразливий компонент	Назва вразливості	Рівень	Опис вразливості	CVSS оцінка
webpack-dev-server	jquery 3.3.1	CVE-2019-11358	Medium	jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution	6.1

Будь-яка офіційна документація з приводу безпеки побудови веб-застосунків із Vue.js, контакти та політика безпеки – відсутні.

3.4.3 Результати Angular

Angular має близько двадцяти знайдених вразливостей із урахуванням вразливостей ранніх версій фреймворку – AngularJS. Це пов’язано із

популярністю фреймворку та, відповідно, великою кількістю людей, зацікавлених в цьому продукті та його якості.

Деякі вразливості мають високий рівень тяжкості, що говорить про велику ймовірність ризиків, з якими було пов’язане використання Angular.

Вразливості фреймворку з їх описом та вказаним рівнем тяжкості наведені у таблиці Б.1 додатку Б.

Для захисту від ін’єкцій всі дані, що передаються та обробляються в Angular, розглядаються як ненадійні. Усі фрагменти та рядки, які потенційно можуть стати частиною DOM за допомогою використання шаблонів (через інтерполяцію значень, прив’язку даних, властисвості та атрибути елементів) підлягають автоматичному форматуванню. Фреймворк конвертує значення у безпечні,видаляючи з них ненадійні компоненти, а у режимі розробки ще й виводячи повідомлення про це у консоль.

Методи, що застосовуються для перетворення значень, залежать від контексту, в якому використовуються. Серед контекстів Angular визначає HTML, STYLE, URL та Resource URL.

Фреймворк має модуль HttpClient, у якому існують методи для підтримки загального механізму захисту від CSRF зі сторони клієнта. Цей модуль також визнає конвенцію, згідно з якою шляхом префіксації відповідей JSON попереджається атака XSSI. Він автоматично видаляє рядок ")]} ', \ t з усіх відповідей перед подальшою обробкою інформації, реалізуючі захист зі сторони клієнта.

Angular підтримує політику безпеки контенту та заохочує до її використання. Він також має систему автоматичної обробки помилок та засоби попередження несанкціонованого перенаправлення з веб-сайту.

Кількість вбудованих механзмів захисту, що пропонує цей фреймворк, є найбільшою серед усіх програмних продуктів, що розглядаються в контексті роботи.

Таблиця 3.10 ілюструє вбудовані механізми захисту від потенційних загроз, що пропонує Angular.

Таблиця 3.10 – Вбудовані механізми захисту Angular

Механізм захисту	Наявність
Автоматична перевірка параметрів шаблонів та вилучення потенційно небезпечних конструкцій	Так
Підтримка техніки захисту від CSRF зі сторони клієнта	Так
Автоматичний захист від XSSI	Так
Підтримка політики безпеки контенту	Так
Попередження несанкціонованого перенаправлення з веб-сайту	Так
Система автоматичної обробки помилок	Так

У таблиці В.1 додатку В показані вразливості складових компонентів фреймворку.

На сторінці, присвяченій безпеці, Angular має опис основних принципів, яких слід дотримуватись для підтримки відповідного рівня захищеності веб-застосунку, та посилання на Філософію Безпеки Google, яку можна трактувати і як загальну політику безпеки фреймворку. У тому ж розділі офіційного сайту

вказані контаки для звернення з приводу знайдених вразливостей або загроз та представлена офіційна документація, що висвітлює аспекти безпечної веб-розробки з використанням даного фреймворку та існуючі в ньому вбудовані механізми захисту.

3.4.4 Результати Ember

Кількість вразливостей за час існування фреймворку Ember дорівнює дев’яти. Їх перелік наведено у таблиці Г.1 додатку Г.

Ember запобігає багатьом формам атак. Всі отримані користувацькі значення та фрагменти, які відображуються за допомогою залежної бібліотеки Handlebars, автоматично валідуються та конвертуються в безпечні.

Фреймворк надає підтримку політики безпеки контенту з першого дня користуваня. Вона надає додатковий захист від певних атак, та в залежності від встановлених у ній правил, рівень захищеності буде відрізнятися. Інтерфейс командного рядка Ember оснащений доповненням ember-cli-content-security-policy для роботи з політикою безпеки контенту у режимі розробки.

У фреймворку представлена половина засобів захисту з існуючих, перелік яких знаходиться у Таблиці 3.11.

Таблиця 3.11 – Засоби захисту від потенційних загроз у Ember

Механізм захисту	Наявність
Автоматична перевірка параметрів шаблонів та вилучення потенційно небезпечних конструкцій	Так

Продовження Таблиці 3.11

Підтримка техніки захисту від CSRF зі сторони клієнта	Ні
Автоматичний захист від XSSI	Ні
Підтримка політики безпеки контенту	Так
Попередження несанкціонованого перенаправлення з веб-сайту	Так
Система автоматичної обробки помилок	Ні

Опис вразливостей складових компонентів фреймворку та їх оцінки можна побачити у Таблиці 3.12.

Таблиця 3.12 – Вразливості у залежностях Ember

Назва модулю	Вразливий компонент	Назва вразливості	Рівень	Опис вразливості	CVSS оцінка
ember-data	jquery 3.3.1	CVE-2019-11358	Medium	jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution	6.1

Продовження Таблиці 3.12

broccoli-sri-hash	moment.js 2.10.6		medium	reDOS - regular expression denial of service	5.9
sourcemap-validator	jquery 1.10.2	CVE-2015-9251	medium	3rd party CORS request may execute	6.1
		CVE-2015-9251	medium	parseHTML() executes scripts in event handlers	6.1
		CVE-2019-11358	low	jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution	6.1

Хоча Ember.js має не дуже розгорнутої документації щодо безпечної розробки, для нього наявні контакти для звернень та чітко визначена політика безпеки.

3.4.5 Результати Polymer.js

Polymer є бібліотекою, для якої на поточний стан не було знайдено жодної вразливості. Вона також не має у своєму складі залежних модулей та компонент, для яких можна визначити відомі загрози.

Для захисту від ін’єкцій та XSS-атак функція бібліотеки, що відповідає за відображення на сторінці фрагментів коду HTML, інтерполює тільки значення формату HTMLTemplateElement та htmlLiteral і не дозволяє атаки XSS через прив’язку даних.

Наявність вбудованих засобів захисту від загроз для бібліотеки представлена у Таблиці 3.13.

Таблиця 3.13 – Вбудовані механізми захисту Polymer.js

Механізм захисту	Наявність
Автоматична перевірка параметрів шаблонів та вилучення потенційно небезпечних конструкцій	Так
Підтримка техніки захисту від CSRF зі сторони клієнта	Ні
Автоматичний захист від XSSI	Ні
Підтримка політики безпеки контенту	Ні
Попередження несанкціонованого перенаправлення з веб-сайту	Ні
Система автоматичної обробки помилок	Так

На офіційному сайті бібліотеки присутні контакти для звернення з приводу питань безпеки та частково документація з описом існуючого механізму захисту і кращих практик побудови безпечного веб-застосунку із застосуванням веб-компонентів Polymer. Офіційної політики безпеки бібліотека не має.

3.5 Визначення оцінок для фреймворків та бібліотек

Для обчислення оцінок захищеності та оцінки серйозності знайдених вразливостей по кожному з досліджених бібліотек та фреймворків необхідно знайти максимальне значення $Vulnerability(p_k)$ та $Vulnerability\_Dep(p_k)$. Обчислимо ці показники для кожного з вибраних продуктів:

$p_1 - \text{React}, p_2 - \text{Vue.js}, p_3 - \text{Angular}, p_4 - \text{Ember.js}, p_5 - \text{Polymer.js}$

$Vulnerability(p_1) = 6.1 + 6.5 + 7.1 = 19.7$

$Vulnerability(p_2) = 3.7 + 6.5 = 10.2$

$Vulnerability(p_3)$

$$= 8.1 + 7.4 * 3 + 7.1 * 2 + 6.8 * 2 + 6.5 * 5 + 5.4 * 2 + 5.3 + 4.8 + 4.3 * 3 = 124.4$$

$Vulnerability(p_4) = 6.9 + 6.1 * 2 + 5.4 + 4.3 + 3.5 + 3.1 * 2 + 2.6 = 89.7$

$Vulnerability(p_5) = 0$

$\max_k Vulnerability(p_k) = 124.4$

Показники серйозності вразливостей у складових компонентах:

$Dep\_Vulnerability(p_1) = 6.1 + 6.1 + 6.1 = 18.3$

$Dep\_Vulnerability(p_2) = 6.1$

$$Dep\_Vulnerability(p_3) = 6.1 + 4.3 \ + 6.1 + 6.1 + 4.3 + 6.1 + 6.5 \ = 39.5$$

$$Dep\_Vulnerability(p_4) = 6.1 + 5.9 + 6.1 = 18.1$$

$$Dep\_Vulnerability(p_5) = 0$$

$$\max_k Dep\_Vulnerability(p_k) = 39.5$$

3.5.1 React

$$N\_Vulnerability(p_1) = 1 - \frac{19.7}{124.4} = 0.84$$

$$Mechanisms\_Score\ (p_1) = \frac{2}{6} = 0.33$$

$$N\_Dep\_Vulnerability(p_1) = 1 - \frac{18.3}{39.5} = 0.54$$

$$Security\_Policy\_Score \ = 0$$

$$Security\_Contacts\_Score \ = 0$$

$$Security\_Documentation\_Score = 0$$

$$Bug\_Bounty\_Score = 1$$

Оцінка захищеності: $0.2 * 0.84 + 0.19 * 0.33 + 0.19 * 0.54 + 0.1 * 0 + 0.07 * 0 + 0.1 * 0 + 0.04 * 1 = 0.373$

3.5.2 Vue.js

$$N\_Vulnerability(p_2) = 1 - \frac{10.2}{124.4} = 0.92$$

$$Mechanisms\_Score\ (p_1) = \frac{1}{6} = 0.17$$

$$N\_Dep\_Vulnerability(p_1) = 1 - \frac{6.1}{39.5} = 0.85$$

$$Security\_Policy\_Score\ = 0$$

$$Security\_Contacts\_Score\ = 0$$

$$Security\_Documentation\_Score = 0$$

$$Bug\_Bounty\_Score = 0$$

Оцінка захищеності: $0.2 * 0.92 + 0.19 * 0.17 + 0.19 * 0.85 + 0.1 * 0 + 0.07 * 0 + 0.1 * 0 + 0.04 * 0 = 0.378$

3.5.3 Angular

$$N\_Vulnerability(p_k) = 1 - \frac{124.4}{124.4} = 0$$

$$Mechanisms\_Score\ (p_1) = \frac{6}{6} = 1$$

$$N\_Dep\_Vulnerability(p_1) = 1 - \frac{39.5}{39.5} = 0$$

$$Security\_Policy\_Score\ = 1$$

$$Security\_Contacts\_Score\ = 1$$

$$Security\_Documentation\_Score = 1$$

$$Bug\_Bounty\_Score = 1$$

Оцінка захищеності: $0.2 * 0 + 0.19 * 1 + 0.19 * 0 + 0.1 * 1 + 0.07 * 1 + 0.1 * 1 + 0.04 * 1 = 0.5$

3.5.4 Ember

$$N\_Vulnerability(p_k) = 1 - \frac{89.7}{124.4} = 0.28$$

$$Mechanisms\_Score\ (p_1) = \frac{3}{6} = 0.5$$

$$N\_Dep\_Vulnerability(p_1) = 1 - \frac{18.1}{39.5} = 0.54$$

$$Security\_Policy\_Score\ = 1$$

$$Security\_Contacts\_Score\ = 1$$

$$Security\_Documentation\_Score = 0$$

$$Bug\_Bounty\_Score = 1$$

Оцінка захищеності: $0.2 * 0.28 + 0.19 * 0.5 + 0.19 * 0.54 + 0.1 * 1 + 0.07 * 1 + 0.1 * 0 + 0.04 * 1 = 0.464$

3.5.5 Polymer.js

$$N\_Vulnerability(p_k) = 1 - \frac{0}{124.4} = 1$$

$$Mechanisms\_Score\ (p_1) = \frac{2}{6} = 0.33$$

$$N\_Dep\_Vulnerability(p_1) = 1 - \frac{0}{39.5} = 1$$

$$Security\_Policy\_Score\ = 0$$

$$Security\_Contacts\_Score\ = 1$$

$Security\_Documentation\_Score = 0$

$Bug\_Bounty\_Score = 1$

Оцінка захищеності: $0.2 * 1 + 0.19 * 0.33 + 0.19 * 1 + 0.1 * 0 + 0.07 * 1 + 0.1 * 0 + 0.04 * 1 = 0.563$

Висновки до розділу 3

У третьому розділі були сформовані критерії захищеності JavaScript фреймворків та бібліотек та вирішена проблема їх нерівнозначності шляхом використання методу експертних оцінок важливості.

Згідно з результатами експертного оцінювання та відповідністю розглянутих фреймворків та бібліотек сформованим критеріям були представлені такі результати обчислення оцінки захищеності:

- React – 0.373;
- Vue.js – 0.378;
- Angular – 0.5;
- Ember – 0.464;
- Polymer.js – 0.563;

Проаналізувавши ці дані, слід зазначити, що найбільш захищеною технологією виявився Polymer.js, що більшою частиною пов’язано з відсутністю у ньому вразливих залежних компонентів та власних знайдених вразливостей.

Друге місце належить фреймворку Angular, який, незважаючи на найбільшу кількість знайдених вразливостей з усіх розглянутих продуктів, має всі механізми захисту від потенційних загроз з представлених, а також

розгорнуту документацію з приводу питань безпеки, контакти та політику безпеки.

Ember має середні показники як у наявності механізмів захисту, так і в серйозності вразливостей, але його перевагою є чітко визначена політика безпеки та контакти з приводу питань безпеки.

Низькі результати Vue.js та React зумовлені низькою кількістю вбудованих засобів захисту та відсутністю документації з приводу безпеки, політики безпеки та контактів з приводу її питань.

Отже, кожен фреймворк або бібліотека мають свої переваги та недоліки у вжитих засобах та заходах безпеки, але отримані показники захищеності дозволяють оцінити їх комплексно із урахуванням багатьох впливових факторів.

ВИСНОВКИ

В результаті виконання роботи були розглянуті основні загрози безпеки для веб-застосунків та методи захисту від потенційних загроз зі сторони клієнта, що пропонують сучасні JavaScript фреймворки і біліотеки.

Для виконання завдання оцінки захищеності фреймворків були створені спеціальні критерії, кожен з яких характеризує стан безпеки програмного продукту з різних боків. Для визначення важливості критеріїв використовувались показники, отримані після обробки даних проведеного експертного оцінювання.

За результатам аналізу п’яти обраних для розгляду фреймворків і бібліотек відповідно до розроблених критеріїв була визначена оцінка їх захищеності.

Згідно з обчисленими балами найбільш захищеним продуктом є Polymer.js з оцінкою 0.563 при максимальній оцінці 1, отже, його можна рекомендувати в якості інструменту для безпечної розробки програм. Наступними у порядку зменшення оцінки виявились Angular (0.5), Ember (0.464), Vue.js (0.378) та React (0.373).

Визначені із урахуванням існуючих механізмів захисту, знайдених власних вразливостях фреймворків і бібліотек та їх складових компонентів, отримані показники надають можливість їх використання в процесі загальної оцінки та перевірки захищеності системи.

Врахування оцінки захищеності при виборі технології для розробки веб-застосунків з навчальною метою або метою використання їх у різних сферах може покращити стан безпеки та зменшити можливість виникнення загроз.

Отже, результати виконаної роботи можуть у застосовуватись декількома способами. Вони свідчать про те, що кожна розглянута технологія має свої слабкі та сильні сторони у захисті, тому для визначення її захищеності слід використовувати повну систематичну оцінку, яка формується в результаті багатостороннього аналізу і охоплює найбільшу кількість складових стану безпеки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1 Osmani A. Learning JavaScript Design Patterns / Addy Osmani., 2017.

2 Martin F. GUI Architectures. Model-View-Presenter (MVP) [Електронний ресурс] / Fowler Martin. – 2006. – Режим доступу до ресурсу: https://martinfowler.com/eaaDev/uiArchs.html#Model-view-presentermvp.

3 Sandeep Kumar Patel. What are web components? / Sandeep Kumar Patel // Learning Web Component Development / Sandeep Kumar Patel., 2015.

4 Справочник современных концепций JavaScript: часть 2 [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://medium.com/devschacht/%D1%81%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D1%87%D0%BD%D0%B8%D0%BA-%D1%81%D0%BE%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D1%85-%D0%BA%D0%BE%D0%BD%D1%86%D0%B5%D0%BF%D1%86%D0%B8%D0%B9-javascript-%D1%87%D0%B0%D1%81%D1%82%D1%8C-2-8ecf07f3f36a.

5 Fowler M. GUI Architectures. Forms and Controls [Електронний ресурс] / Martin Fowler. – 2006. – Режим доступу до ресурсу: https://martinfowler.com/eaaDev/uiArchs.html#FormsAndControls

6 The Ten Most Critical Web Application Security Risks [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf.

7 Front-end Frameworks - Overview [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://2018.stateofjs.com/front-end-frameworks/overview/.

8 Technologies [Електронний ресурс] – Режим доступу до ресурсу: https://www.similartech.com/technologies.

9 Web Components - React [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/docs/web-components.html#using-react-in-your-web-components.

10 Sandeep Kumar Patel. What is Polymer? / Sandeep Kumar Patel // Learning Web Component Development / Sandeep Kumar Patel., 2015.

11 Mustache-security [Електронний ресурс]. – 2013. – Режим доступу до ресурсу: https://code.google.com/archive/p/mustache-security/.

12 OWASP Application Security Verification Standard Project [Електронний ресурс] – Режим доступу до ресурсу: https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

13 Федорченко А. В. Исследование открытых баз уязвимостей и оценка возможности их применения в системах анализа защищенности компьютерных сетей / А. В. Федорченко, А. А. Чечулин, И. В. Котенко. // Информационно-управляющие системы. – 2014. – №5. – С. 73.

14 Complete Vulnerability DataBase & Security Scanner [Електронний ресурс] – Режим доступу до ресурсу: https://vulners.com/landing.

15 Common Vulnerability Scoring System [Електронний ресурс] – Режим доступу до ресурсу: https://www.first.org/cvss/.

16 The Ten Most Critical Web Application Security Risks [Електронний ресурс]. – 2013. – Режим доступу до ресурсу: https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf.

17 Бешелев С.Д. Математико-статистические методы экспертных оценок / Бешелев С.Д, Гурвич Ф.Г.., 1980. – 263 с.

18 Орлов А.И. Экспертные оценки / Орлов А.И. – Москва, 2002. – С. 16.

19 Саати Т. Метод анализа иерархий / Т. Саати, К. Кернс // Аналитическое планирование. Организация систем / Т. Саати, К. Кернс. – Москва: "Радио и связь", 1991. – С. 52

Додаток А

Статистика використання JavaScript фреймворків і бібліотек станом на 2018 рік

Таблиця А.1 – Статистика використання фреймворків

	React	Vue.js	Angular	Ember	Polymer.js
Ніколи не чули про нього	0.2%	1.3%	0.0%	7.3%	18.5%
Чули, але не зацікавлені	9.2%	20.5%	31.8%	67.3%	51.5%
Чули і хотіли б вивчати	19.1%	46.6%	10.4%	14.1%	23.0%
Використовують, але не використовували б у майбутньому	6.7%	2.8%	33.8%	6.3%	3.8%
Використовують та збираються використовувати в майбутньому	64.8%	28.8%	23.9%	5.0%	3.1%

Додаток Б

Вразливості Angular за час існування

Таблиця Б.1 – Вразливості Angular

Вразливість	Опис вразливості	Тяжкість вразливості (severity)	CVSS оцінка
npm:angular:20130625	Affected versions <1.1.5 of the package are vulnerable to Arbitrary Script Injection due to improper sanitization of the $event object passed to the native constructor function	High	8.1
npm:angular:20150310	Affected versions <1.5.0-beta.2 are vulnerable to Arbitrary Code Execution via unsafe svg animation tags.	High	7.4
npm:angular:20140909	Affected versions <1.2.24 >=1.2.19 of the package are vulnerable to Unsafe Object Deserialization.	High	7.4

Продовження таблиці Б.1

npm:angular:20131113	Affected versions <1.2.2 of the package are vulnerable to Protection Bypass via ng-attr-action and ng-attr-srcdoc allowing binding to Javascript	High	7.4
npm:angular:20150909	Affected versions <1.5.0-beta.2 of the package are vulnerable to Mutation Cross-site Scripting (mXSS). This error occurs	High	7.1
npm:angular:20150807	Affected versions <1.5.0-beta.0 >=1.0.0 of the package are vulnerable to Cross-site Scripting (XSS) due to no proper sanitization of xlink:href attributes.	High	7.1
npm:angular:20150807-1	Affected versions of the package are vulnerable to Clickjacking	Medium	6.8

Продовження таблиці Б.1

npm:angular:20130621	Affected versions of the package are vulnerable to Cross-site Scripting (XSS) attacks	Medium	6.8
npm:angular:20180202	Affected versions <1.6.9 of this package are vulnerable to Cross-site Scripting (XSS) through SVG files if enableSvg is set.	Medium	6.5
npm:angular:20171018	Affected versions <1.6.7 of the package are vulnerable to Cross-site Scripting (XSS) via ideographic space chararcters in URIs	Medium	6.5
npm:angular:20150315	Affected versions <1.6.1 of the package are vulnerable to JSONP Callbacks attacks.	Medium	6.5
npm:angular:20161101	Affected versions <1.5.9 >=1.5.0 of the package are vulnerable to CSP Bypass	Medium	6.5

Продовження таблиці Б.1

npm:angular:20141104	Affected versions <1.3.2 of the package are vulnerable to Arbitrary Command Injection due to the assignment functions accessing constructors functions, allowing attackers to execute their malicious code	Medium	6.5
npm:angular:20151130	Affected versions <1.4.10 of the package are vulnerable to Cross-site Scripting (XSS) attacks involving assignment on constructor properties.	Medium	5.4
npm:angular:20130622	Affected versions <1.2.0 >=1.0.0 of the package are vulnerable to Cross-site Scripting (XSS)	Medium	5.4

Продовження таблиці Б.1

npm:angular:20140908	Affected versions <1.3.0-rc.4 of the package are vulnerable to Cross-site Scripting (XSS) due to unsanitized URIs in ng-srcset	Medium	5.3
npm:angular:20160527	Affected versions <1.2.30 >=1.0.0 of the package are vulnerable to Arbitrary Script Injection	Medium	4.8
npm:angular:20160122	Affected versions <1.5.0-rc.2 >=1.3.0 of the package are vulnerable to Cross-site Scripting (XSS) due to the usemap attribute not being blacklisted	Medium	4.3
npm:angular:20151205	Affected versions <1.5.0-rc.0 of the package are vulnerable to Cross-site Scripting (XSS) via the SVG <use> element	Medium	4.3

Продовження таблиці Б.1

npm:angular:20151205	Affected versions <1.5.0-rc.0 of the package are vulnerable to Cross-site Scripting (XSS) via the SVG <use> element	Medium	4.3

Додаток В

Вразливості залежностей Angular

Таблиця В.1 – Вразливості залежностей фреймворку Angular

Назва модулю	Вразливий компонент	Назва вразливості	Рівень	Опис вразливості	CVSS Оцінка
webpack-dev-server	jquery 3.3.1	CVE-2019-11358	medium	jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution	6.1
selenium-webdriver	jquery 1.3.2	CVE-2011-4969	medium	XSS with location.hash	4.3
		CVE-2012-6708	medium	Selector interpreted as HTML	6.1

Продовження таблиці В.1

		CVE-2019-11358	medium	jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution	6.1
	jquery 1.4.4	CVE-2011-4969	medium	XSS with location.hash	4.3
		CVE-2012-6708	medium	Selector interpreted as HTML	6.1
		CVE-2015-9251	medium	3rd party CORS request may execute	6.1

Продовження таблиці В.1

		CVE-2019-11358	medium	jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution	6.1
	jquery-ui-dialog 1.8.10	CVE-2010-5312	medium	Title cross-site scripting vulnerability	4.3
		CVE-2016-7103	medium	summary: XSS Vulnerability on closeText option	6.1
	tinyMCE 4.0.26		medium	xss issues with media plugin not properly filtering out some script attributes.	6.5

Додаток Г

Вразливості за час існування Ember

Таблиця Г.1 – Вразливості за час інсування фреймворку Ember

Вразливість	Опис вразливості	Тяжкість вразливості (severity)	CVSS оцінка
CVE-2010-3355	Ember 0.5.7 places a zero-length directory name in the LD_LIBRARY_PATH, which allows local users to gain privileges via a Trojan horse shared library in the current working director	Medium	6.9
CVE-2015-7565	Cross-site scripting (XSS) vulnerability in Ember.js 1.8.x through 1.10.x, 1.11.x before 1.11.4, 1.12.x before 1.12.2, 1.13.x before 1.13.12, 2.0.x before 2.0.3, 2.1.x before 2.1.2, and 2.2.x before 2.2.1 allows remote attackers to inject arbitrary web script or HTML.	Medium	6.1
CVE-2015-1866	Cross-site scripting (XSS) vulnerability in Ember.js 1.10.x before 1.10.1 and 1.11.x before 1.11.2.	Medium	6.1

Продовження таблиці Г.1

CVE-2014-0013	Ember.js 1.0.x before 1.0.1, 1.1.x before 1.1.3, 1.2.x before 1.2.1, 1.3.x before 1.3.1, and 1.4.x before 1.4.0-beta.2 allows remote attackers to conduct cross-site scripting (XSS) attacks by leveraging an application that contains templates whose context is set to a user-supplied primitive value and also contain the `{{this}}` special Handlebars variable.	Medium	5.4
CVE-2013-4170	Potential Cross-site Scripting (XSS) >= 1.0.0-rc.6 <1.0.0-rc.6.1	Medium	4.3
CVE-2014-0014	Ember.js 1.0.x before 1.0.1, 1.1.x before 1.1.3, 1.2.x before 1.2.1, 1.3.x before 1.3.1, and 1.4.x before 1.4.0-beta.2 allows remote attackers to conduct cross-site scripting (XSS) attacks by leveraging an application using the "{{group}}" Helper and a crafted payload.	Low	3.5

Продовження таблиці Г.1

npm:ember:20140214	Affected versions >=1.2.0 <1.2.2 || >=1.3.0 <1.3.1 of this package are vulnerable to Cross-site Scripting (XSS).	Low	3.1
SNYK-DOTNET-EMBER-60147	Potential XSS Exploit With User-Supplied Data When Binding Primitive Values versions [1.2.0,1.2.2), [1.3.0,1.3.2)	Low	3.1
CVE-2014-0046	Cross-site scripting (XSS) vulnerability in the link-to helper in Ember.js 1.2.x before 1.2.2, 1.3.x before 1.3.2, and 1.4.x before 1.4.0-beta.6, when used in non-block form, allows remote attackers to inject arbitrary web script or HTML via the title attribute.	Low	2.6