УДК 519.711

Д.В. Савельєв, Київ

ЗАХИСТ ВЕБ-ДОДАТКІВ – КОМПЛЕКСНА ЗАДАЧА ВІД ПРОЕКТУВАННЯ ДО ЕКСПЛУАТАЦІЇ

Abstract. The article describes the stages of the lifecycle of developing web application and web services, provides guidance for design and development in terms of product security to avoid errors as potential risks of cybersecurity.

Вступ

На сьогоднішній день безпека веб-додатків знаходиться у першій десятці трендів та загроз інформаційної безпеки уже понад 10 років. Все більше аспектів сучасності залежить від використання веб-додатків – як у складних інфраструктурних системах, так і для IoT пристроїв.

Без сумніву, незалежно від сфери застосування усі використовувані ресурси повинні бути захищеними. Проте далеко не в усіх випадках доречним буде правило «чим більше захисту, тим краще» – додатки різних сфер потребують різних підходів.

Забезпечення захищеності веб-додатків та веб-систем є комплексною задачею, та потребує уваги як у процесі їх розробки так і у ході експлуатації.

У даній статті розглядаються етапи життєвого циклу розробки веб-додатків та веб-сервісів, наведені рекомендації при проектуванні та розробці з точки зору захищеності продукту що допоможуть уникнути помилки як потенційні вразливості інформаційної безпеки.

Основна частина

Захист систем на етапі розробки в першу чергу – це уникнення помилок у вихідному коді додатку та раціональне використання ресурсів. Навіть використавши усі необхідні інструменти для забезпечення безпеки додатку (шифрування повідомлень, політика встановлення паролів, тощо) розробник може допустити помилку, яка згодом буде використана як вразливість системи. Далі наведено деякі рекомендації, що допоможуть уникнути потенційні вразливості системи.

Некомерційна організація Open Web Application Security Project (OWASP) випустила рекомендації для розробників програмного забезпечення, які хочуть підтримувати актуальний рівень захисту своїх продуктів. Документ [1] містить 10 пунктів на які слід звернути увагу при створенні веб-додатків для уникнення потенційних вразливостей в коді:

1) дотримання актуальних вимог безпеки;
2) використання безпечних фреймворків та бібліотек;
3) забезпечення захищеного доступу до даних;
4) шифрування та безпека даних;

5) перевірка вхідних даних;
6) захист «цифрової особистості»;
7) управління доступом;
8) всебічний захист інформації;
9) моніторинг та ведення журналів безпеки;
10) коректне опрацювання помилок та виключень.

Дотримання актуальних вимог безпеки. Галузеві стандарти, законодавчі акти та накопичений досвід професійної спільноти формують набір уявлень про необхідні заходи безпеки. Розробникам варто проектувати свої продукти на існуючих практиках змість того, щоб винаходити власні підходи.

Використання безпечних фреймворків та бібліотек. Програмні інструменти з гарантованою відсутністю вразливостей допомагають створювати захищені додатки, інакше – розробка займе більше часу та може обійтись дорожче за рахунок додаткових перевірок та виправлень помилок.

До даного розділу також доречно віднести оновлення програмного забезпечення. Зловмисники регулярно знаходять та одразу ж застосовують нові вразливості операційних систем та іншого програмного забезпечення – HTTP-серверів чи систем управління контентом. Більшість розробників користуються менеджерами пакетів для встановлення залежних компонентів для додатків. У цих пакетах також знаходять вразливості, тому варто слідкувати за їх оновленнями також.

Забезпечення захищеного доступу до даних. Цей розділ передбачує забезпечення безпечності запитів, механізмів конфігурації, автентифікації та обміну даними з додатками для уникнення вразливостей, таких як SQL-ін'єкції.

SQL-ін'єкція представляє собою виконання довільного запиту до бази даних додатку за допомогою поля форми або параметра URL, в результаті чого можуть отримуватись, змінюватись або видалятись дані таблиць. Для уникнення такої ситуації рекомендується використовувати параметризовані запити, які підтримуються більшістю мов програмування.

Шифрування та безпека даних. До даного розділу можна віднести такий тип атаки, як міжсайтовий скриптинг (XSS) – впровадження в сторінку сайту шкідливого коду, який виконується на клієнтському пристрої, може змінювати сторінку та передавати інформацію зловмиснику.

При перевірці необхідно налаштувати перетворення спеціальних символів, які можуть бути інтерпретованими як частина команди. При динамічній генерації HTML-коду рекомендується використовувати спеціальні функції для зміни та отримання значень атрибутів, а також шаблонізатори, які автоматично виконують екранізацію спеціальних символів.

Перевірка вхідних даних. Не можна обробляти дані, що не відповідають очікуванням системи, тому їх необхідно контролювати як на стороні клієнта, так і на стороні сервера. Перевірка повинна включати синтаксичний та семантичний аспекти. Наприклад, якщо очікується

отримання значення дати, то система повинна обробляти лише запити у визначеному форматі (MM-dd-yyyy, dd.MM.yyyy та інші) та ігнорувати букви, інші символи та формати. Відсутність перевірки на стороні сервера приводить до використання зловмисником ін'єкцій та інших видів атак.

Захист «цифрової особистості». Дане поняття означає сукупність даних про користувача в рамках сеансів роботи з системою. Рекомендується використовувати сильні паролі та засоби подвійної або криптографічної автентифікації (пароль у поєднанні з кодом зі спеціального токену).

Управління доступом. Даний механізм включає процеси авторизації та автентифікації. Система повинна визначати, якими правами на отримання інформації володіє кожен користувач, додаток або процес.

Всебічний захист інформації. Забезпечення безпеки необхідно розділити на три частини – класифікація даних для коректної обробки кожного типу, шифрування передачі інформації та баз даних.

Для шифрування передачі інформації слугує протокол HTTPS (HyperText Transfer Protocol Secure) – розширення HTTP, яке підтримує шифрування та захищає дані при передачі в мережі.

Паролі також варто зберігати у вигляді хешу, причому рекомендується використовувати алгоритми одностороннього хешування, наприклад, SHA – у такому випадку для авторизації користувача порівнюються хешовані значення. Якщо зловмисник отримає доступ до хешованих паролів, збитки будуть знижені за рахунок того, що хеш має невідворотну дію та отримати з нього вихідні дані практично неможливо.

Моніторинг та ведення журналів безпеки. Контроль подій дозволяє своєчасно виявити ознаки кібератаки: відправка нехарактерних запитів та команд, спроби редагувати незмінні дані, підозріла поведінка користувачів.

Коректне опрацювання помилок та виключень. При розробці системи необхідно перевірити, як вона реагує на непередбачені кодом команди та ситуації. В деяких випадках непередбачувана помилка може знизити продуктивність системи або призвести до її збою. Такі ситуації потрібно відстежувати та модернізувати алгоритм дій системи.

Також слід приділяти увагу тому, що відображається в повідомленнях про помилку. Клієнт повинен отримувати інформацію про помилки в максимально стислій формі, яка виключає наявність будь-якої технічної інформації. Детальні відомості слід зберігати в лог-файлах системи, оскільки маючи повну інформацію, зловмиснику легше виконати комплексні атаки на кшталт SQL-ін'єкції.

CSP та CORS

Серед сучасних технологій безпеки веб-ресурсів у першу чергу слід звернути увагу на CSP та CORS.

Політика безпеки вмісту (CSP – Content Security Policy) – стандарт безпеки, що дозволяє розпізнавати та усувати визначені типи атак, такі як XSS та атаки впровадження даних [2].

Основна мета створення CSP полягає в усуненні XSS атак та зборі даних про спроби. CSP дозволяє адміністраторам серверів знизити або повністю усунути шляхи, за якими зловмисники можуть провести XSS за допомогою визначення доменів, які браузер клієнта повинен вважати довіреними джерелами виконуваних скриптів. У такому випадку сумісний з CSP браузер буде виконувати тільки скрипти, отримані зі списку дозволених джерел та ігнорувати інші. У якості крайньої міри захисту сайти, які хочуть заборонити виконання скриптів, можуть налаштувати цю поведінку глобально за допомогою відповідної опції.

Спільне використання ресурсів з різних джерел (CORS – Cross-Origin Resource Sharing) – механізм, що використовує додаткові HTTP-заголовки для надання агенту користувача можливості отримання дозволу на використання даних з домену, відмінного від поточного [3].

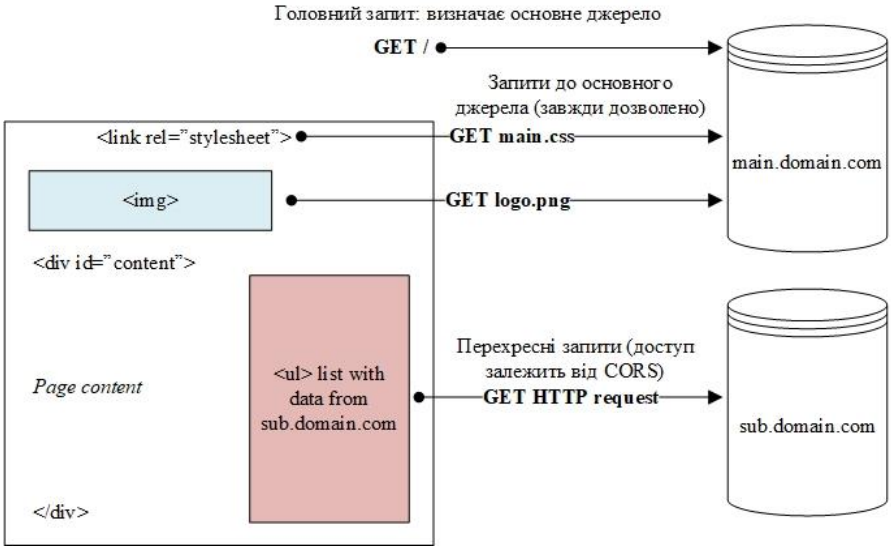

Рис. 1. Приклад сценарію застосування перехресних запитів

З міркувань безпеки клієнти припиняють всі перехресні HTTP-запити, здійснені кодом скриптів. Це означає, що веб-додаток, отриманий з певного джерела не може виконувати запити до HTTP-ресурсів з відмінного джерела (виконання можливе, якщо відповідь міститиме належні CORS-заголовки). На рис. 1 демонструється сценарій застосування перехресних запитів.

Механізм CORS надає можливість виконання безпечних перехресних запитів та передавання даних між клієнтами та серверами. Стандарт CORS працює за допомогою додавання нових HTTP-заголовків, які дозволяють серверам описувати набір джерел, яким дозволено читати запитувану інформацію.

Проектування API-сервісів

При розробці функціоналу веб-додатку досить поширеним є використання API-сервісів. Досить часто для обміну даними між різними системами та базами даних використовується REST API (наприклад, для передачі даних із системи електронного документообігу до системи бухгалтерського обліку). Тому, при розробці такого функціоналу теж варто дотримуватись деяких правил.

REST API (Representational State Transfer – передача стану представлення) – архітектурний стиль взаємодії компонентів розподіленого додатку в мережі.

Принципи REST:

– незалежність від стану (Statelessness) – дані, що повертаються певним викликом API не повинні залежати від виконаних раніше викликів;

– багаторівнева архітектура (Layered System) – клієнт не знає, чи є відповідаючий сервер кінцевою точкою обслуговування ресурсу, що є прекрасним принципом для забезпечення балансування навантаження та надання спільних кешів;

– єдиний уніфікований інтерфейс;

– кешована архітектура (Cacheable) – відповідь сервера може кешуватись на певний період часу та використовуватись без нових запитів до сервера;

– зручне представлення даних – у якості представлення даних об’єкта передаються дані у форматі JSON або XML.

API-сервіси використовують HTTP методи для виконання операцій CRUD (create, read, update, delete – створення, читання, оновлення, видалення). Для цих операцій в REST API зазвичай використовуються наступні HTTP методи:

– GET – використовується для отримання інформації (read);

– POST – найчастіше використовується для створення ресурсів (create);

– PUT – використовується для оновлення ресурсів (update);

– DELETE – використовується для видалення ресурсів (detete).

Окрім перерахованих, використовуються також такі методи, як OPTIONS (отримання клієнтом різних представлень ресурсу) та HEAD (запит ресурсу з сервера). Однак, використання цих методів потребує врахування особливостей кожного з них. Загалом HTTP методи визначаються двома характеристиками:

– безпека – метод вважається безпечним, якщо його виклик не змінює стан даних;

– ідемпотентність – метод вважається ідемпотентним, коли отримується одна і та ж відповідь при кожному однаковому запиті.

Не усі методи є безпечними та ідемпотентними. У табл. 1 наведено перелік HTTP методів, які використовуються в REST API, та їх характеристики [4].

Таблиця 1

Характеристики HTTP методів

Метод	Характеристика	
	Безпечний	Ідемпотентний
GET	+	+
POST	-	-
PUT	-	+
DELETE	-	+
OPTIONS	+	+
HEAD	+	+

На основі наведеної інформації видно, що метод GET не повинен змінювати стан ресурсу, до якого застосовується. Методи PUT та DELETE можуть змінювати стан ресурсу, проте їх можна повторювати якщо немає впевненості, що попередній запит виконався. Але метод POST не є безпечним та ідемпотентним – окрім зміни стану ресурсу багатократне повторення запиту буде створювати ефект, залежний від кількості повторень.

Висновки

Працюючи над захистом веб-систем варто враховувати сферу використання, від якої залежить який підхід краще обрати. Перед впровадженням певних змін для забезпечення захищеності слід також враховувати наскільки погіршиться якість системи для користувацького використання. При виборі заходів для підвищення рівня безпеки системи слід проаналізувати, наскільки це доцільно, оскільки такі заходи можуть зробити вихідний код складнішим та подальшу підтримку важчою. Є ряд вразливостей, які необхідно уникати для веб-додатків та веб-систем будь-якого рівня – SQL-ін’єкції, XSS, блокування користувацького контенту, та інші.

1. *Anton K., Manico J., Bird J.* 10 Critical Security Areas That Software Developers Must Be Aware Of // OWASP Top Ten Proactive Controls Project. 2018. – URL: https://www.owasp.org/images/b/bc/OWASP_Top_10_Proactive_Controls_V3.pdf (дата звернення: 22.04.2019).
2. Content Security Policy (CSP). // MDN Web Docs. – URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP (дата звернення: 20.04.2019).
3. Cross-Origin Resource Sharing (CORS). // MDN Web Docs. – URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (дата звернення: 20.04.2019).
4. *Srinivasan K.* REST API Best Practices. // JAVABEAT. 2016. – URL: https://javabeat.net/rest-api-best-practices/ (дата звернення: 16.04.2019).

http://doi.org/10.5281/zenodo.3859657

Поступила 3.10.2019р.