УДК 004.056
DOI: 10.31673/2409-7292.2022.012733

Прус К.В.

АНАЛІЗ ЗАГРОЗ WEB-ДОДАТКІВ НА ОСНОВІ АРХІТЕКТУРИ REST

В роботі проведено аналіз проблеми забезпечення кібербезпеки Web-додатків, визначено мету та завдання для забезпечення безпеки Web-додатків. Проведено аналіз уразливостей Web-додатків. На основі досліджень проведених в роботі розроблено рекомендації щодо застосування методів захисту Web-додатків.

Ключові слова: кібербезпека, інформаційна система, rest, інформаційна система, web-додаток.

Вступ

Сьогодні важко уявити своя життя без Інтернету, соціальних мереж, веб-порталів та блогів. Більша частино повсякденного життя базується на використанні інформаційних технологій: оплата проїду в метро, новини, пошта, месенджери. Більшість з них спроектовані саме на REST архітектурі. Тобто backend відокремлений від інтерфейсу користувача. Це дозволяє використовувати одне й те ж програмне забезпечення для різних додатків, таких як Web-додатки, мобільні додатки, комп'ютерні додатки.

В умовах швидкого розвитку інформаційних технологій також з’являються нові ризики та загрози інформаційній безпеці. Тому важливо самого початку проектувати інформаційну систему враховуючи ці ризики, загрози та можливі вразливості. За останній рік зросла кількість експлойтів, що доводить значимість «безпечної» розробки Web-додатків, бо саме через неправильно спроектовану систему та помилки при написанні коду, вони експлойти стають можливими в своїй реалізації зловмисниками та порушують нормальну роботу системи.

Сучасні методи розробки Web-додатків на основі REST архітектури

REST, або Representational State Transfer – це архітектурний стиль для реалізації стандартів для комунікації між комп’ютерними системами в мережі Інтернеті, що полегшує взаємодію систем між собою. Системи, сумісні з REST, які часто називають системами RESTful, характеризуються тим, що вони не зберігають свій стан між запитами та розділяють проблеми клієнта та сервера. Далі буде детальніше розглянуто, що означають ці терміни та їх переваги під час взаємодії між інформаційними системами в Інтернеті.

Типовий протокол передачі даних, такий як SOAP (Simple Object Access Protocol), пропонує великі можливості з точки зору безпеки та цілісності даних. Більше того, SOAP реалізує можливість повторного запиту при виникненні помилок з’єднання з мережею. Але такі протоколи тяжче реалізовувати та використовувати. REST – це простіша альтернатива, яка розвинулася в геометричній прогресії за останні кілька років. Люди часто плутаються щодо стандартів REST. У порівнянні із SOAP, старими веб-службами, REST є більш гнучким та простим у впровадженні.

При розробці RESTful Web-додатків, реалізація клієнта (frontend) та реалізація сервера (backend) можуть здійснюватися незалежно, не знаючи один про одного. Це означає, що мову програмування та фреймворки на стороні клієнта можна змінити в будь-який час, не впливаючи на роботу сервера, так само і з серверною частиною. Ця перевага дозволяє проводити розробку паралельно, що дозволяє пришвидшити розробку, і тим самим, дає більше часу на забезпечення безпеки та тестування.

Для того щоб незалежні компоненти, які можуть мати декілька реалізацій, Web-додатку мали змогу обмінюватися інформацією, потрібно дотримуватися певного формату повідомлень, щоб вони були модульними та окремими. Відокремлення користувальницького інтерфейсу від методів зберігання даних, дозволяє масштабувати сервіс, спрощуючи серверні компоненти. Прикладом є мікросервісна архітектура, яка базується на REST. Крім того, поділ дозволяє кожній частині розвиватися незалежно. Також, такий підхід, дає змогу розгортати декілька різних серверів та за допомогою сервісу балансування навантаження розділяти запити клієнту, тим самим зменшуючи навантаження на апаратне забезпечення.

Використовуючи REST API, різні клієнти потрапляють в однакові endpoint, виконують однакові дії та отримують однакові відповіді.

Додатки, які дотримуються парадигми REST, не мають стану, тобто серверу не потрібно надавити інформацію про стан клієнта між запитами, і навпаки. Таким чином, і сервер, і клієнт можуть оброблювати будь-які отримані повідомлення, не опираючись на попередніх запитах. Ця концепція реалізується використанням використанням ресурсів, які описують той чи інший об’єкт, документ або будь які інші данні. На рисунку 1 зображено приклад взаємодії компонентів клієнт-серверної архітектури [1].

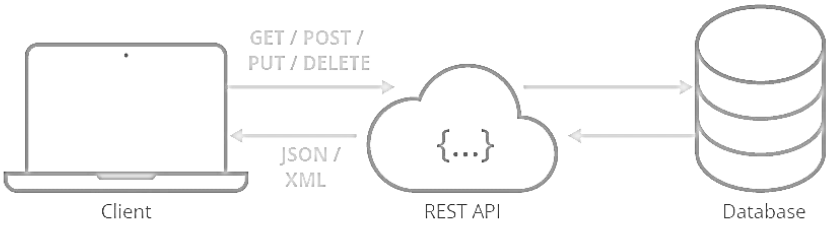

Рис. 1. Приклад архітектури клієнт-серверу

Основні концепції REST архітектури

Для того щоб веб-додаток вважався RESTful, він повинен дотримуватися шести основних концепцій:

- Єдиний інтерфейс;
- Клієнт-сервер;
- Відсутність стану;
- Рівні абстракцій;
- Кеш;
- Код по запиту.

Єдиний інтерфейс використовується для відокремлення клієнта від реалізації сервера.

Запит до сервера повинен містити ідентифікатор ресурсу (рисунок 2). В більшості випадків це id, що відображає унікальне число, яке записано в базі даних.

https://api.rapidlink.piraeusbank.gr/api/v1/accounts/34

Рис. 2. Представлення ресурсу у вигляді посилання

Відповідь, яку повертає сервер, повинна містити достатньо інформації, щоб клієнт міг змінити ресурс.

Кожен запит до API повинен містити всю інформацію, необхідну серверу для виконання запиту, а кожна відповідь, має містити всю інформацію, необхідну клієнту для коректної обробки відповіді.

Наявність гіпермедіа. Під гіпермедіа мається на увазі гіперпосилання або просто посилання, які сервер може включити у відповідь.

Клієнт і сервер діють незалежно, кожен сам по собі, і взаємодія між ними відбувається лише у формі запитів, ініційованих лише клієнтом, та відповідей, які сервер надсилає клієнту лише як реакцію на запит. Але тут також є виключення. RESTful додатки можуть перші відсилати відповідь клієнту, наприклад, коли потрібно виконати певну роботу на сервері, яка потребує деякого часу. І щоб мережеве з’єднання не використовувало ресурсів мережі, цю роботу, сервер виконує в паралельному потоці. А клієнта повідомляють, що сервер почав виконувати задачу [3].

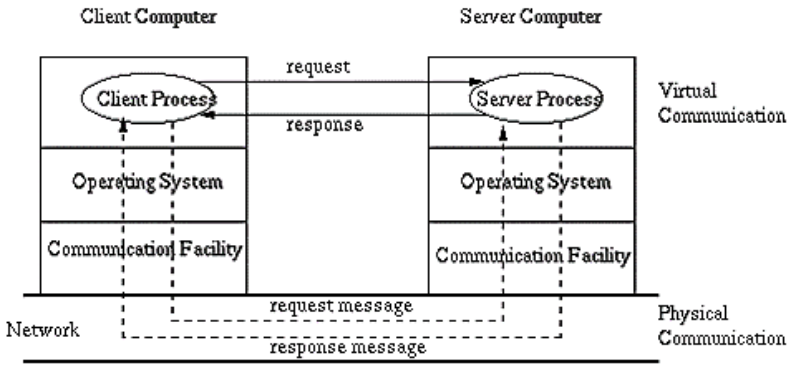

Рис. 3. Модель взаємодії клієнт-сервер

Відсутність стану допомагає службам бути масштабованими та більш надійними. Відсутність стану, в контексті REST, означає, що всі запити клієнта на сервер містять всю інформацію, яка необхідна для коректного оброблення запиту сервером. Він розглядає їх як незалежні. Запити клієнта залишають сервер незалежним від будь-якого зберігання контексту між запитами. Зберігати стан сеансу на стороні клієнта важливо для управління цим обмеженням. На рисунку 4 показано, що клієнт та стан сервісу незалежні та керуються відповідно клієнтом та сервером [3].

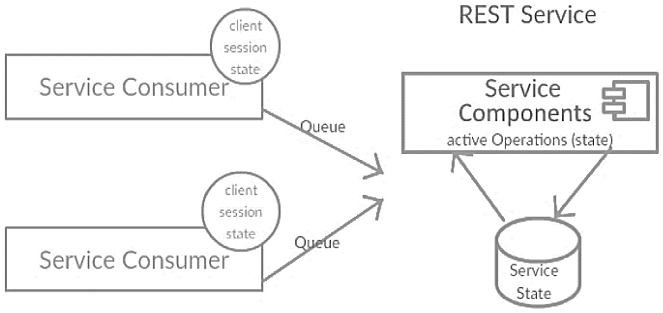

Рис. 4. Принцип керування станом на стороні клієнта та сервера

Загалом, багаторівнева система складається із рівнів з різним функціоналом. Основними характеристиками таких систем є те, що кожен рівень взаємодіє за допомогою заздалегідь визначених інтерфейсів і взаємодіє лише із рівнем, що знаходиться вище або нижче знизу, з точки зору логіки програми. Рівні можна додавати, видаляти, модифікувати або міняти місцями (по вертикалі) згідно з розвитком та вимогами архітектури. На рисунку 5 зображено приклад такої ієрархії.

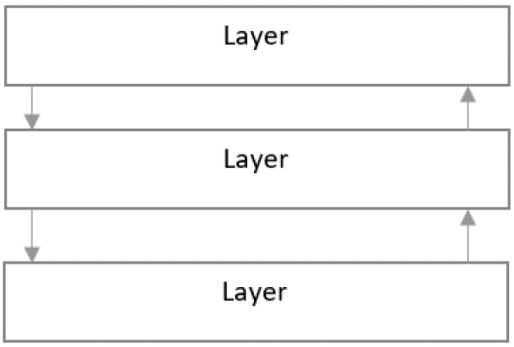

Рис. 5. Ієрархія сервісних рівнів

Кешування – це концепція, яка дозволяє зберігати часто доступні дані для обробки запитів клієнта. Ніколи не потрібно генерувати однакову відповідь більше одного разу, поки цього не потребується. Правильно налаштований кеш усуває часткову або повну взаємодію клієнт-сервер і в той же час, забезпечує клієнта очікуваною відповіддю. Очевидно, що кешування покращує продуктивність завдяки швидшому часу відгуку та зменшенню навантаження на сервер.

У комп'ютерних системах COD – це будь-яка технологія, яка дозволяє серверу надсилати програмний код, як правило, у формі скрипту, коли відповідь у форматі HTML, клієнтам, що в свою чергу, запускається на клієнтському комп'ютері [1].

Аналіз та визначення проблематики захисту Web-додатків

Більшість організацій, які використовують RESTful API сьогодні, покладаються на своїх розробників як для розробки додатку, так і для подальшої його підтримки. Хоча 33% використовують спеціальні технології для підтримки та менеджменту своїх сервісів, 90% респондентів використовують свої команди розробників або зовнішні ресурси, щоб реалізувати API з нуля.

Згідно зі State of API за 2020 рік від Postman, більше розробників звертають увагу надійність – 71,6%, потім на безпеку, продуктивність та документацію, (в межах 70%). Зручність використання та масштабованість 59,4% та 57,2% відповідно [4].

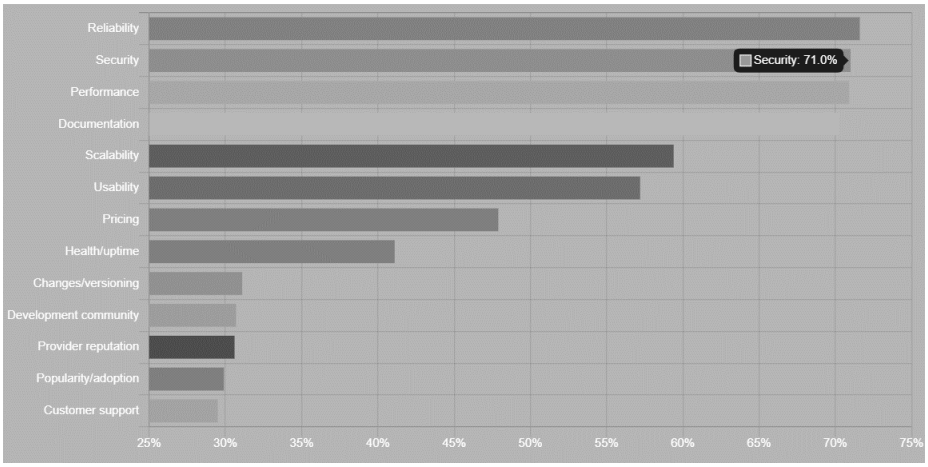

Рис. 6. Статистичні дані актуальності проблем при розробці

Оскільки REST, як правило, використовує HTTP як базовий протокол, який має звичайний набір проблем безпеки:

потенційний зловмисник має повний контроль над кожним HTTP запитом або відповіддю. RESTful API зазвичай використовуються для обміну інформацією, яка зберігається та, можливо, оброблюється на багатьох серверах, це може призвести до багатьох прихованих порушень та витоків інформації;

зловмисник може перебувати на стороні клієнта, або на стороні сервера де він запускає зловмисну програму;

для додатку, який використовує REST в якості клієнта або сервера, інша сторона, як правило, має повний контроль над станом ресурсу і може підпадати під ін'єкції шкідливого коду для атаки на обробку ресурсів (наприклад, отримання довільного коду Java або виконання системної команди).

В архітектурі REST, end-to-end передбачає послідовність потенційно вразливих операцій:

під час зіставлення HTTP повідомлень та URL-адресу ресурсу (controller mapping);

коли об’єкт, що представляє цільовий ресурс (певна інформація про об’єкт), створюється та виконується певна операція над цим ресурсом, згідно з логікою додатку (виклик служб з контролера);

під час доступу або модифікації даних на сервері, що містять стан ресурсу (звертання до бази даних або до якось іншого сховища).

Багаторівнева структурах REST означає, що одна слабка ланка в ланцюжку може зробити вашу програму вразливою [5].

Аналіз загроз та засобів захисту Web-додатків. SQL injection

Ця атака може бути виконана передаючи ненадійні дані в запиті до API. Вхідні дані згодом виконуються інтерпретатором програмного коду, що може призвести до того, що зловмисник отримає несанкціонований доступ до інформації або спричинить інші збитки.

Згідно з OWASP Top 10 зловмисник може використати недолік написання коду. Приклад зображено на рисунку 7.

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

Рис. 7. SQL запит

Подібним чином, сліпа довіра програми до фреймворків може призвести до запитів, які все ще є вразливими (наприклад, Hibernate Query Language).

Запобігання SQL ін’єкціям вимагає написання коду, що дозволяє відокремити команди від запитів:

найкращим варіантом є використання безпечного API, який уникає використання інтерпретатора на пряму повністю або забезпечує параметризований інтерфейс, або використання інструментів реляційного відображення об’єктів (ORM);

використання валідації даних на стороні сервера;

для будь-яких динамічних запитів потрібно уникати спеціальні символи, використовуючи конкретний синтаксис екранування для цього інтерпретатора мови програмування серверу;

використання LIMIT та інших елементів керування SQL у запитах для запобігання масового розкриття записів на випадок SQL ін’єкцій.

Security Misconfiguration

Неправильна конфігурація безпеки визначається як неможливість реалізувати всі засоби управління безпекою для сервера або веб-програми або реалізація засобів контролю безпеки, але при цьому з помилками.

Програма може бути вразливою, якщо:

відсутні відповідні посилення безпеки в будь-якій частині стеку програм або неправильно налаштовані дозволи на хмарні служби;

увімкнено або встановлено непотрібні функції (наприклад, непотрібні порти, служби, сторінки, облікові записи або привілеї);

облікові записи за замовчуванням та їх паролі все ще ввімкнені та незмінені;

обробка помилок виявляє для користувачів сліди стека або інші надмірно інформативні повідомлення про помилки;

для оновлених систем найновіші функції безпеки вимкнені або не налаштовані надійно;

сервер не надсилає заголовки або директиви безпеки, або вони не встановлені як захищені;

програмне забезпечення застаріле або вразливе.

Способи протидії:

мінімальна платформа без будь-яких непотрібних функцій, компонентів, документації та зразків. Видаліть або не встановлюйте невикористані функції та рамки;

перегляд та оновлення конфігурацій серверу та дозволів хмарного сховища (наприклад, дозволи сервісу AWS S3);

сегментована архітектура додатків;

автоматизований процес для перевірки ефективності конфігурацій і налаштувань у всіх середовищах [6].

Cross-site scripting

Атаки міжсайтових сценаріїв (XSS) – це різновид ін'єкції, при якій шкідливі сценарії виконуються як легітимні та надійні Web-сайти. Атаки XSS трапляються, коли зловмисник використовує Web-додаток для надсилання зловмисного коду, як правило, у формі сценарію на стороні браузера, іншому кінцевому користувачеві. Недоліки, які дозволяють цим атакам мати успіх, досить широко розповсюджені і трапляються скрізь, де веб-програма використовує вхідні дані користувача у вихідних даних, які він генерує, без перевірки та кодування. Таке часто можна зустріти у блогах та коментарях.

Зловмисник може отримати доступ до будь-яких файлів cookie, токенів безпеки або іншої конфіденційної інформації, що зберігається браузером і використовується на цьому веб-сайті. Ці скрипти можуть навіть переписати вміст HTML-сторінки.

Якщо програма не перевіряє вхідні дані, зловмисник може легко викрасти cookie у автентифікованого користувача. Все, що повинен зробити зловмисник – це розмістити код, зображений на рисунку 8, у будь-якій частині сайту, де контент береться з користувацького вводу (тобто: дошки оголошень, приватні повідомлення, профілі користувачів).

```
<SCRIPT type="text/javascript">
var adr = '../evil.php?cakemonster=' + escape(document.cookie);
</SCRIPT>
```

Рис. 8. Зловмисний код для викрадення cookie

Вищезазначений код відправить захищений вміст файлу cookie (відповідно до вмісту RFC, перед тим як надсилати його за протоколом HTTP методом GET), до скрипта evil.php у змінній «cakemonster». Потім зловмисник перевіряє результати свого скрипта evil.php (сценарій викрадення файлів cookie зазвичай записує cookie у файл) і використовує його.

Для того щоб захистити Web-додаток від XSS можна використовувати методи, які будуть описані нижче.

Заголовок Content-Type повинен бути application/json, а не text/html. Це зобов'язує браузер не виконувати вбудований скрипт.

Ненадійні URL-адреси, що включають JavaScript: виконуватимуть код JavaScript при використанні в розташуваннях DOM URL-адрес, таких як атрибути href тегу прив'язки або розташування iFrame src. Потрібно перевіряти всі ненадійні URL-адреси, щоб вони містили лише безпечні параметри, такі як HTTPS.

Встановлення HTTPOnly на файлах cookie та будь-які користувацькі файли cookie, до яких не має доступу жоден JavaScript код.

Content Security Policy. Це механізм на стороні браузера, який дозволяє створювати списки дозволів для джерел клієнтських ресурсів Web-додатку, наприклад JavaScript, CSS, зображення, тощо. CSP за допомогою спеціального заголовка HTTP вказує браузеру виконувати або відображати ресурси лише з цих джерел (рисунок 9).

```
Content-Security-Policy: default-src: 'self'; script-src: 'self' static.domain.tld
```

Рис. 9. Content-Security-Policy заголовок

Таке значення заголовку доручить Web-браузеру завантажувати всі ресурси лише з серверу на якому знаходиться frontend та файлів вихідного коду JavaScript додатково з static.domain.tld [7].

Висновок

У статті зроблено огляд поширених загроз Web-додатків на основі REST архітектури. Виявлена проблематика захисту Web-додатків від загроз в інформаційних системах. Проаналізовано теоретичні відомості про REST архітектуру. Виявлено основні концепції та ознаки REST Web-додатків. Також було запропоновано вирішення деяких проблем захисту Web-додатків.

Перелік посилань

1. REST API Definition: What is a REST API (RESTful API)? [Електронний ресурс] // Tehreem Naeem. – 2021. – Режим доступу до ресурсу: https://www.astera.com/type/blog/rest-api-definition.
2. Why Should We Choose REST (Client-Server) Model to Develop Web Apps? [Електронний ресурс] // Audira Zuraida. – 2018. – Режим доступу до ресурсу: https://medium.com/@audira98/why-should-we-choose-rest-client-server-model-to-develop-web-apps-c3bb2451b13a.
3. What is REST [Електронний ресурс] // Code Academy. – 2020. – Режим доступу до ресурсу: https://www.codecademy.com/articles/what-is-rest.
4. State of the API Report // Postman. – 2020.
5. What is REST – A Simple Explanation for Beginners, Part 2: REST Constraints [Електронний ресурс] // Shif Ben Avraham. – 2017. – Режим доступу до ресурсу: https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-2-rest-constraints-129a4b69a582.
6. OWASP Top 10: Injection Security Vulnerability Practical Overview [Електронний ресурс] // ImmuniWeb. – 2018. – Режим доступу до ресурсу: https://www.immuniweb.com/blog/OWASP-SQL-injection-attack.html.
7. Cross Site Scripting Prevention Cheat Sheet [Електронний ресурс] // Owasp. – 2020. – Режим доступу до ресурсу: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.

Надійшла: 18.01.2022 Рецензент: д.т.н., професор Гайдур Г.І.