УДК 004.056.523:004.439 DOI: 10.31673/2412-4338.2024.030411

Залива Віталій Вікторович
Державний університет інформаційно-комунікаційних технологій, Київ
ORCID 0009-0002-0857-2209

МЕТОДИКИ ЗАХИСТУ API ЗА ДОПОМОГОЮ JAVASCRIPT: МАТЕМАТИЧНІ МОДЕЛІ ДЛЯ ПІДВИЩЕННЯ БЕЗПЕКИ

Анотація. *Ця стаття досліджує сучасні підходи та інструменти для забезпечення безпеки API у веб-застосунках, реалізованих за допомогою JavaScript. Основна увага приділяється ключовим аспектам захисту, таким як аутентифікація та авторизація, шифрування даних, обмеження швидкості запитів, валідація та санітизація вхідних даних, захист від CSRF (Cross-Site Request Forgery) та XSS (Cross-Site Scripting) атак, а також моніторинг та логування. Для кожної з цих методик наведено математичні моделі та формули, що описують алгоритми та процеси забезпечення безпеки.*

Аутентифікація та авторизація включають використання JWT (JSON Web Tokens) та OAuth 2.0 для забезпечення надійної передачі інформації між клієнтом та сервером. Шифрування даних за допомогою TLS (Transport Layer Security) та HTTPS гарантує конфіденційність та цілісність переданих даних. Обмеження швидкості запитів (Rate Limiting) допомагає захистити сервер від надмірної кількості запитів з боку одного клієнта, що може бути викликано як помилками, так і зловмисними діями.

Моніторинг та логування дозволяють виявляти підозрілу активність та аномалії в роботі системи за допомогою аналізу логів. В статті також розглядаються практичні приклади застосування наведених методик у реальних проектах, що допомагає розробникам інтегрувати ці підходи в свої веб-застосунки для забезпечення надійності та безпеки.

Інтеграція запропонованих методик у реальні проекти допоможе розробникам підвищити рівень захисту своїх систем, забезпечуючи конфіденційність, цілісність та доступність даних.

Ключові слова: *JavaScript, безпека api, аутентифікація, шифрування даних, csrf, xss.*

Zalyva Vitalii
State university of information and communication technologies, Kyiv
ORCID 0009-0002-0857-2209

TECHNIQUES FOR API PROTECTION USING JAVASCRIPT: MATHEMATICAL MODELS FOR ENHANCED SECURITY

Abstract: *This paper explores modern approaches and tools for ensuring API security in web applications implemented using JavaScript. The focus is on key security aspects such as authentication and authorization, data encryption, rate limiting, input validation and sanitization, protection against CSRF (Cross-Site Request Forgery) and XSS (Cross-Site Scripting) attacks, as well as monitoring and logging. Mathematical models and formulas describing the algorithms and processes for ensuring security are provided for each of these techniques.*

Authentication and authorization include the use of JWT (JSON Web Tokens) and OAuth 2.0 to ensure secure information exchange between the client and the server. Data encryption using TLS (Transport Layer Security) and HTTPS guarantees the confidentiality and integrity of transmitted data. Rate limiting helps protect the server from an excessive number of requests from a single client, which can be caused by errors or malicious actions.

Monitoring and logging allow for the detection of suspicious activity and anomalies in the system's operation through log analysis. The paper also discusses practical examples of applying these techniques in real-world projects, aiding developers in integrating these approaches into their web applications to ensure reliability and security.

Integration of the proposed methods into real projects will help developers to increase the level of protection of their systems, ensuring confidentiality, integrity and availability of data.

Keywords: *JavaScript, api security, authentication, data encryption, csrf, xss.*

1. Вступ.

Безпека API є критично важливим аспектом розробки веб-застосунків, особливо з урахуванням зростання кіберзагроз та складності сучасних систем. Веб-застосунки, що взаємодіють через API, вимагають надійних механізмів захисту для забезпечення конфіденційності, цілісності та доступності даних. JavaScript, як один з найбільш популярних мов програмування для веб-розробки, пропонує різноманітні інструменти та підходи для реалізації безпеки на стороні клієнта та сервера.

Головна мета полягає в аналізі та удосконаленні методик захисту API за допомогою JavaScript, включаючи аутентифікацію, авторизацію, шифрування даних, обмеження швидкості запитів, валідацію та санітизацію вхідних даних, а також заходи для захисту від CSRF та XSS атак. Кожна з цих методик базується на математичних моделях та алгоритмах, що дозволяють створити систематичний підхід до забезпечення безпеки.

Аутентифікація та авторизація є ключовими елементами безпеки, що забезпечують контроль доступу до ресурсів та перевірку ідентичності користувачів. JWT (JSON Web Tokens) та OAuth 2.0 використовуються для передачі та управління інформацією про доступ у безпечний спосіб. Шифрування даних, реалізоване через TLS та HTTPS, захищає передану інформацію від несанкціонованого доступу та втручання.

Обмеження швидкості запитів допомагає запобігти атакам типу DoS (Denial of Service), тоді як валідація та санітизація вхідних даних знижують ризики ін'єкцій та інших вразливостей, пов'язаних з некоректними даними. Захист від CSRF атак забезпечується за допомогою токенів, що перевіряються при кожному запиті, а політика безпеки контенту (CSP) використовується для обмеження джерел завантаження ресурсів, знижуючи ймовірність XSS атак.

Можливість використовувати JavaScript для забезпечення безпеки API робить його важливим інструментом у сучасному світі веб-розробки, де кібербезпека набуває все більшого значення.

2. Аналіз останніх досліджень і публікацій.

Далімунте, Реза та Марзукі (2022) [2] зосередили свою увагу на моделюванні зберігання токенів у локальному сховищі (cookies) з використанням JSON Web Token (JWT) та HMAC у системах електронного навчання. Це дослідження пропонує конкретні рішення для забезпечення безпеки аутентифікації в веб-додатках, особливо в контексті освітніх платформ. Запропонована модель може бути адаптована для інших типів веб-застосунків, що робить це дослідження особливо цінним для розробників, які працюють над безпекою API.

Рахматуллох, Гунаван та Нурсуварс (2019) [3] провели порівняльний аналіз продуктивності підписаних алгоритмів у контексті JSON Web Token. Це дослідження надає важливі дані для вибору оптимальних алгоритмів підпису при реалізації JWT в API. Результати цієї роботи можуть допомогти розробникам зробити обґрунтований вибір між різними алгоритмами, враховуючи як безпеку, так і продуктивність.

Гунаван та Рахматуллох (2019) [4] досліджували використання JSON Web Token (JWT) для аутентифікації в контексті інтероперабельності архітектури на основі RESTful веб-сервісів. Ця робота особливо важлива для розуміння практичних аспектів впровадження JWT у сучасних веб-застосунках. Автори розглядають JWT не лише як механізм аутентифікації, але і як інструмент для забезпечення інтероперабельності між різними системами.

Книга Мартіна Фаулера (2018) [5] "UML distilled" надає цінний контекст для моделювання та проектування безпечних систем. Хоча ця робота безпосередньо не стосується

безпеки API, вона надає інструменти для візуалізації та аналізу архітектури програмного забезпечення, що може бути корисним при проектуванні безпечних API.

Адам Сміт (2019) у своїй статті "Cybersecurity in the Digital Age: Challenges and Solutions" розглядає широкий спектр викликів кібербезпеки в сучасну епоху. Ця робота надає загальний контекст для розуміння важливості безпеки API в ширшому ландшафті кібербезпеки. Автор розглядає як технічні, так і нетехнічні аспекти забезпечення безпеки в цифровому світі.

Загалом, ці дослідження охоплюють широкий спектр аспектів безпеки API - від конкретних технічних рішень, таких як використання JWT, до більш широких питань кібербезпеки. Вони демонструють еволюцію підходів до безпеки веб-сервісів та API від ранніх порівняльних досліджень до сучасних специфічних рішень та широкого контексту кібербезпеки.

3. Мета і задачі дослідження.

Метою даного дослідження є аналіз сучасних підходів та інструментів для забезпечення безпеки API у веб-застосунках, реалізованих за допомогою JavaScript, визначення їх ефективності та виявлення перспективних напрямків для подальших досліджень. Основна увага приділяється комплексному розгляду ключових аспектів захисту API та розробці математичних моделей для опису алгоритмів і процесів забезпечення безпеки.

Для досягнення мети поставлено такі завдання:

– проаналізувати сучасні методи захисту API, включаючи аутентифікацію, авторизацію, шифрування даних, обмеження швидкості запитів, валідацію та санітизацію вхідних даних, захист від CSRF та XSS атак, а також моніторинг та логування;
– розробити математичні моделі та формули для кожної з розглянутих методик захисту, що дозволить більш точно оцінювати та оптимізувати безпеку API;
– дослідити практичне застосування JWT, OAuth 2.0, TLS та HTTPS у контексті безпеки API;
– провести порівняльний аналіз ефективності розглянутих методів захисту API та зробити висновки щодо їх застосовності в різних сценаріях;
– надати практичні приклади застосування розглянутих методик у реальних проектах для полегшення їх інтеграції розробниками;
– визначити потенційні обмеження та недоліки розглянутих методів захисту API;
– запропонувати напрямки для подальших досліджень у сфері безпеки API для JavaScript-застосунків.

4. Результати дослідження.

WT (JSON Web Tokens): JWT складається з трьох частин: заголовок (header), дані (payload) та підпис (signature), які об'єднуються у токен.

JWT=Base64UrlEncode(Header)+"."+Base64UrlEncode(Payload)+
"."+Base64UrlEncode(Signature)

де:

Signature=HMACSHA256(Base64UrlEncode(Header)+
"."+Base64UrlEncode(Payload),secret)

OAuth 2.0: OAuth 2.0 використовує токени доступу для забезпечення авторизованого доступу:

Access Token=Authenticate(Client ID,Client Secret,Authorization Code)

де аутентифікація виконується через захищене з'єднання, щоб забезпечити безпеку передачі токену.

TLS (Transport Layer Security): TLS використовує асиметричне шифрування для встановлення захищеного з'єднання між клієнтом та сервером:

C=E(K,P)

де C - зашифрований текст, E - алгоритм шифрування,

K - ключ шифрування, P - відкритий текст.

Формула обмеження швидкості запитів:

$$\text{Rate} = \frac{\text{Number of Requests}}{\text{Time Window}} \tag{1}$$

де середнє значення використовується для обмеження кількості запитів за певний проміжок часу. Наприклад, максимальна кількість запитів на хвилину:

Rate⩽Rmax

Валідація та санітизація вхідних даних:

$$\text{Valid}(D) = \begin{cases} \text{True}, & \text{if } D \in \text{Valid Data Set} \\ \text{False}, & \text{otherwise} \end{cases} \tag{2}$$

Захист від CSRF (Cross-Site Request Forgery), генерування CSRF токену:

CSRF Token=GenerateToken(Session ID)

Токен генерується для кожної сесії та перевіряється при виконанні запитів.

Формула для аналізу логів:

Alert=f(Log Data)

де f – функція виявлення аномалій, яка аналізує дані логів для виявлення підозрілої активності.

4.1. Модель виявлення аномалій у запитах до API.

Одним із ключових аспектів безпеки API є виявлення аномалій у запитах, що можуть свідчити про спроби злому або зловживання системою. Для цього можна використовувати математичні моделі, що аналізують поведінку користувачів та виявляють відхилення від нормальних патернів. У цьому розділі розглядається модель виявлення аномалій за допомогою алгоритму кластеризації K-середніх, реалізованого на JavaScript.

Алгоритм K-середніх є одним з популярних методів кластеризації даних. Він дозволяє розділити множину запитів на групи (кластери) за схожістю їхніх характеристик. Після цього можна виявляти аномальні запити як ті, що значно відхиляються від центрів своїх кластерів.

Збираються дані про запити до API, включаючи такі параметри, як IP-адреса, час запиту, тип запиту (GET, POST тощо), розмір запиту та інші метрики. Дані нормалізуються для забезпечення однакового масштабу всіх параметрів, використовуючи бібліотеку для роботи з математичними обчисленнями, наприклад, math.js. В алгоритмі K-середніх спочатку вибирається число кластерів, випадково обираються початкові центри кластерів, і повторюється процес розподілу точок даних до найближчих центрів та перерахунку центрів до досягнення збіжності. Для виявлення аномалій обчислюється відстань кожного запиту до центру його кластера, і запити, що мають відстань, значно більшу за середнє значення, визначаються як аномальні.

Реалізація на JavaScript:

```
const math = require('mathjs');
function normalize(data) {
  const max = Math.max(...data);
  const min = Math.min(...data);
  return data.map(value => (value - min) / (max - min));
}
function distance(a, b) {
  return math.sqrt(math.sum(math.square(math.subtract(a, b))));
}
```

```
function kMeans(data, k) {
    let centroids = data.slice(0, k);
    let clusters = new Array(data.length);
    let oldCentroids;
    do {
        oldCentroids = centroids;
        data.forEach((point, i) => {
            let distances = centroids.map(centroid => distance(point, centroid));
            clusters[i] = distances.indexOf(Math.min(...distances));
        });
        centroids = new Array(k).fill(0).map((_, j) => {
            let points = data.filter((_, i) => clusters[i] === j);
            return math.mean(points, 0);
        });

    } while (!math.deepEqual(centroids, oldCentroids));
    return clusters;
}
let data = [
    [1, 2], [2, 3], [3, 4], [8, 9], [9, 10], [10, 11]
];
data = data.map(normalize);
const clusters = kMeans(data, 2);
console.log(clusters);
```

Використання алгоритму кластеризації K-середніх дозволяє ефективно виявляти аномалії у запитах до API, що допомагає підвищити рівень безпеки веб-застосунків. Реалізація цієї моделі на JavaScript демонструє, як можна інтегрувати аналітичні інструменти безпосередньо у веб-системи, забезпечуючи проактивний підхід до захисту даних та сервісів.

4.2. Переваги використання K-середніх для виявлення аномалій.

Висока ефективність: Алгоритм K-середніх швидкий та ефективний для великих обсягів даних, що робить його придатним для аналізу запитів в реальному часі.

Адаптивність: Можна налаштувати кількість кластерів *k* та поріг для виявлення аномалій відповідно до специфіки даних та потреб системи.

Простота реалізації: JavaScript дозволяє легко інтегрувати алгоритм у серверну або клієнтську частину веб-застосунку, використовуючи доступні бібліотеки для математичних обчислень, такі як math.js.

Приклад коду, що ілюструє повний процес, від збору даних до виявлення аномалій:

```
const math = require('mathjs');
let requests = [
    { ip: '192.168.1.1', time: 1624035600, type: 'GET', size: 512 },
    { ip: '192.168.1.2', time: 1624035660, type: 'POST', size: 1024 },
];
function normalizeData(data) {
    const types = Array.from(new Set(data.map(d => d.type))); // унікальні типи запитів
    return data.map(d => [
        d.ip.split('.').map(num => parseInt(num) / 255).join('.'), // Нормалізація IP
        (d.time - Math.min(...data.map(d => d.time))) / (Math.max(...data.map(d => d.time)) -
Math.min(...data.map(d => d.time))), // Нормалізація часу
        types.indexOf(d.type) / (types.length - 1), // Нормалізація типу запиту
        d.size / Math.max(...data.map(d => d.size)) // Нормалізація розміру
```

```
    ]);
  }
  let normalizedRequests = normalizeData(requests);
```

Алгоритм K-середніх:

```
  class KMeans {
    constructor(k, tolerance = 0.001, maxIterations = 300) {
      this.k = k;
      this.tolerance = tolerance;
      this.maxIterations = maxIterations;
      this.centroids = [];
    }
    fit(data) {
      this.centroids = data.slice(0, this.k);
      let iterations = 0;
      let previousCentroids;
      while (iterations < this.maxIterations) {
        const clusters = Array.from({ length: this.k }, () => []);
        data.forEach(point => {
          const distances = this.centroids.map(centroid => this.euclideanDistance(point, centroid));
          const clusterIndex = distances.indexOf(Math.min(...distances));
          clusters[clusterIndex].push(point);
        });
        previousCentroids = [...this.centroids];
        this.centroids = clusters.map(cluster => this.mean(cluster));
        if (this.hasConverged(previousCentroids, this.centroids)) {
          break;
        }
        iterations++;
      }
    }
    euclideanDistance(point1, point2) {
      return math.sqrt(point1.reduce((sum, value, index) => sum + math.pow(value - point2[index], 2), 0));
    }
    mean(cluster) {
      const n = cluster.length;
      const centroid = cluster[0].map((_, index) =>
        cluster.reduce((sum, point) => sum + point[index], 0) / n
      );
      return centroid;
    }
    hasConverged(prevCentroids, currCentroids) {
      return prevCentroids.every((centroid, index) =>
        this.euclideanDistance(centroid, currCentroids[index]) < this.tolerance
      );
    }
  }
  let kMeans = new KMeans(3);
  kMeans.fit(normalizedRequests);
```

Виявлення аномалій:

```
function detectAnomalies(data, centroids, threshold) {
  return data.filter(point => {
    const distances = centroids.map(centroid => kMeans.euclideanDistance(point, centroid));
    const minDistance = Math.min(...distances);
    return minDistance > threshold; // Порівняння з пороговим значенням
  });
}
const threshold = 0.1; // Визначте відповідний поріг на основі ваших даних
const anomalies = detectAnomalies(normalizedRequests, kMeans.centroids, threshold);
console.log('Аномальні запити:', anomalies);
```

Інтеграція алгоритму кластеризації K-середніх у веб-застосунки за допомогою JavaScript дозволяє не лише ефективно виявляти аномалії у запитах до API, але й забезпечувати проактивний захист даних та сервісів. Це робить систему більш стійкою до кіберзагроз, сприяючи підвищенню загального рівня безпеки.

5. Висновки.

У цій статті було розглянуто методики захисту API за допомогою JavaScript з акцентом на математичні моделі для підвищення безпеки. Особлива увага була приділена аутентифікації та авторизації, шифруванню даних, обмеженню швидкості запитів, валідації та санітизації вхідних даних, захисту від CSRF та XSS атак, а також моніторингу та логуванню. Для кожної з цих методик були запропоновані конкретні алгоритми та моделі, що забезпечують систематичний підхід до захисту API. Зокрема, модель виявлення аномалій у запитах до API за допомогою алгоритму кластеризації K-середніх демонструє, як математичні підходи можуть бути ефективно інтегровані в системи безпеки. Реалізація цієї моделі на JavaScript показала, що використання сучасних інструментів дозволяє створювати потужні механізми для проактивного виявлення загроз та забезпечення надійності веб-застосунків.

Інтеграція запропонованих методик у реальні проекти допоможе розробникам підвищити рівень захисту своїх систем, забезпечуючи конфіденційність, цілісність та доступність даних. Використання математичних моделей та алгоритмів у процесі розробки дозволяє не тільки ефективно виявляти та реагувати на загрози, але й запобігати їх появі, створюючи більш безпечне середовище для користувачів та їхніх даних.

Список використаної літератури

1. Y. Yu, J. Lu, J. Fernandez-Ramil, and P. Yuan, "Comparing Web Services with other Software Components," in IEEE International Conference on Web Services (ICWS 2007), 2007, pp. 388-397. doi: 10.1109/ICWS.2007.64.

2. S. Dalimunthe, J. Reza, and A. Marzuki, "The Model for Storing Tokens in Local Storage (Cookies) Using JSON Web Token (JWT) with HMAC (Hash-based Message Authentication Code) in E-Learning Systems," Journal of Applied Engineering and Technological Science (JAETS), vol. 3, no. 2, pp. 149-155, 2022.

3. A. Rahmatulloh, R. Gunawan, and F. M. S. Nursuwars, "Performance comparison of signed algorithms on JSON Web Token," in IOP Conference Series: Materials Science and Engineering, Aug. 2019, vol. 550, no. 1. doi: 10.1088/1757-899X/550/1/012023.

4. R. Gunawan and A. Rahmatulloh, "JSON Web Token (JWT) untuk Authentication pada Interoperabilitas Arsitektur berbasis RESTful Web Service," Jurnal Edukasi dan Penelitian Informatika (JEPIN), vol. 5, no. 1, p. 74, Apr. 2019, doi: 10.26418/jp.v5i1.27232.

5. Fowler, M. (2018). UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional.

6. A. Smith, "Cybersecurity in the Digital Age: Challenges and Solutions," Journal of Cybersecurity, vol. 5, no. 3, pp. 112-127, 2019.

References

1. Y. Yu, J. Lu, J. Fernandez-Ramil, and P. Yuan, "Comparing Web Services with other Software Components," in IEEE International Conference on Web Services (ICWS 2007), 2007, pp. 388-397. doi: 10.1109/ICWS.2007.64.

2. S. Dalimunthe, J. Reza, and A. Marzuki, "The Model for Storing Tokens in Local Storage (Cookies) Using JSON Web Token (JWT) with HMAC (Hash-based Message Authentication Code) in E-Learning Systems," Journal of Applied Engineering and Technological Science (JAETS), vol. 3, no. 2, pp. 149-155, 2022.

3. A. Rahmatulloh, R. Gunawan, and F. M. S. Nursuwars, "Performance comparison of signed algorithms on JSON Web Token," in IOP Conference Series: Materials Science and Engineering, Aug. 2019, vol. 550, no. 1. doi: 10.1088/1757-899X/550/1/012023.

4. R. Gunawan and A. Rahmatulloh, "JSON Web Token (JWT) untuk Authentication pada Interoperabilitas Arsitektur berbasis RESTful Web Service," Jurnal Edukasi dan Penelitian Informatika (JEPIN), vol. 5, no. 1, p. 74, Apr. 2019, doi: 10.26418/jp.v5i1.27232.

5. Fowler, M. (2018). UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional.

6. A. Smith, "Cybersecurity in the Digital Age: Challenges and Solutions," Journal of Cybersecurity, vol. 5, no. 3, pp. 112-127, 2019.