

Десять опасных уязвимостей авторизации в Node.js: анализ и методы устранения

Саухин Сергей Иванович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассматриваются наиболее распространённые и опасные уязвимости авторизации в серверных веб-приложениях, разработанных с использованием платформы Node.js. Проанализированы типовые ошибки реализации механизмов аутентификации и контроля доступа, приводящие к несанкционированному доступу и компрометации данных. На основе анализа научных публикаций и практических рекомендаций представлены основные способы выявления и устранения данных уязвимостей. Результаты исследования могут быть использованы при разработке и аудите безопасных Node.js-приложений.

Ключевые слова: Node.js, безопасность, авторизация, аутентификация, уязвимости, контроль доступа, JWT.

Ten dangerous authorization vulnerabilities in Node.js: analysis and mitigation methods

Saukhin Sergey Ivanovich

Sholom-Aleichem Priamursky State University

Student

Abstract

This article analyzes the most common and dangerous authorization vulnerabilities in server-side web applications developed using the Node.js platform. Typical mistakes in the implementation of authentication and access control mechanisms that lead to unauthorized access and data breaches are examined. Based on the analysis of scientific publications and best practices, methods for detecting and mitigating these vulnerabilities are proposed.

Keywords: Node.js, security, authorization, authentication, vulnerabilities, access control, JWT.

1 Введение

1.1 Актуальность

В условиях активной цифровизации и широкого распространения веб-сервисов вопросы информационной безопасности приобретают особую значимость. Современные веб-приложения обрабатывают значительные объёмы персональных и корпоративных данных, что делает их

привлекательной целью для злоумышленников. Одним из наиболее критичных аспектов безопасности таких систем является корректная реализация механизмов авторизации, обеспечивающих разграничение прав доступа пользователей к ресурсам приложения.

Платформа Node.js получила широкое распространение при разработке серверных приложений и REST API благодаря высокой производительности, асинхронной модели выполнения и развитой экосистеме сторонних библиотек. Вместе с тем данные особенности усложняют контроль безопасности, поскольку ошибки в логике авторизации и аутентификации нередко возникают на уровне прикладного кода и не всегда выявляются автоматизированными средствами анализа.

Согласно отчётам проекта OWASP, уязвимости, связанные с нарушением контроля доступа и некорректной аутентификацией, стабильно входят в число наиболее опасных угроз веб-приложений [9]. Для Node.js-приложений данные риски усиливаются активным использованием npm-зависимостей, среди которых могут присутствовать компоненты с известными уязвимостями, влияющими на безопасность механизмов авторизации [3, 4].

В этой связи актуальной является задача анализа уязвимостей авторизации в Node.js-приложениях, их классификации и выработки практических рекомендаций по предотвращению несанкционированного доступа. Результаты такого анализа позволяют повысить уровень защищённости серверных приложений и снизить вероятность успешных атак на информационные системы.

1.2 Обзор исследований

Вопросы безопасности серверных приложений, разработанных на платформе Node.js, широко рассматриваются в современных научных исследованиях. В работе И. Вербицкий проведён анализ архитектурных особенностей Node.js с точки зрения информационной безопасности. Автор отмечает, что асинхронная модель выполнения и активное использование сторонних библиотек могут приводить к логическим ошибкам в реализации механизмов аутентификации и авторизации, что повышает риск несанкционированного доступа [1].

Исследование К. Нтантогиан и соавторов посвящено выявлению и эксплуатации уязвимостей серверного JavaScript-кода в Node.js-приложениях. Авторами предложен инструмент NodeXP, позволяющий обнаруживать инъекционные уязвимости и нарушения контроля доступа. В работе подчёркивается, что ошибки в проверке пользовательских прав и недостаточная фильтрация входных данных могут использоваться злоумышленниками для обхода механизмов авторизации [2].

В ряде работ М. Альфадел и соавторов рассматривается проблема уязвимых зависимостей в экосистеме npm. Авторы показывают, что значительная часть Node.js-приложений использует библиотеки с известными уязвимостями, что негативно сказывается на безопасности механизмов

аутентификации и авторизации [3]. В более позднем исследовании анализируется обнаруживаемость таких уязвимостей в реальных проектах, где делается вывод о недостаточной информированности разработчиков и ограниченных возможностях автоматических средств анализа [4].

В работе А. Анупам и соавторов проведён сравнительный анализ открытых инструментов сканирования уязвимостей Node.js-приложений. Результаты исследования показывают, что большинство существующих сканеров ориентированы на выявление конфигурационных и синтаксических ошибок, в то время как логические уязвимости авторизации и контроля доступа часто остаются необнаруженными и требуют ручного анализа [5].

Таким образом, анализ научных публикаций свидетельствует о высокой актуальности проблемы уязвимостей авторизации в Node.js-приложениях и необходимости комплексного подхода к их предотвращению.

1.3 Цель исследования

Целью данного исследования является комплексный анализ уязвимостей авторизации в серверных веб-приложениях, разработанных с использованием платформы Node.js, а также разработка практических рекомендаций по их выявлению и устранению.

Для достижения поставленной цели в работе сформулированы и решены следующие задачи:

- анализ научных публикаций и нормативных рекомендаций, посвящённых безопасности Node.js-приложений;
- выявление наиболее распространённых уязвимостей авторизации и аутентификации;
- классификация типовых ошибок реализации механизмов контроля доступа;
- анализ потенциальных последствий эксплуатации выявленных уязвимостей;
- формирование рекомендаций по повышению уровня безопасности авторизации в Node.js-приложениях.

Решение указанных задач позволило систематизировать существующие подходы к обеспечению безопасности авторизации и определить наиболее критичные направления для их совершенствования.

2 Материалы и методы

В ходе проведения исследования использовался комплекс теоретических и практических методов, направленных на анализ уязвимостей авторизации в Node.js-приложениях. Теоретическую основу исследования составили научные публикации, посвящённые вопросам безопасности серверных приложений и платформы Node.js, а также исследования в области анализа уязвимостей веб-приложений, представленные в рецензируемых журнальных изданиях и материалах профильных конференций [1–5].

В качестве нормативной и методической базы использовались рекомендации проекта OWASP, в частности отчёты OWASP Top 10, а также

официальная документация платформы Node.js, спецификация языка JavaScript (ECMAScript) и документация фреймворка Express.js [6–9]. Указанные источники применялись для формирования требований к безопасной реализации механизмов аутентификации и контроля доступа.

Для анализа механизмов авторизации применялись методы логического и структурного анализа архитектуры серверных приложений, а также сравнительный анализ различных подходов к реализации аутентификации и контроля доступа, описанных в научных и прикладных источниках [1–5].

Практическая часть исследования включала моделирование типовых сценариев атак на механизмы авторизации, анализ возможных векторов обхода контроля доступа и оценку последствий эксплуатации выявленных уязвимостей. Дополнительно использовались элементы статического анализа кода и обзор возможностей существующих инструментов автоматического выявления уязвимостей Node.js-приложений, описанных в специализированных исследованиях и технической документации [2, 5].

Использование совокупности указанных методов позволило получить обоснованные выводы о характере и степени опасности уязвимостей авторизации и сформировать рекомендации, ориентированные на практическое применение в процессе разработки и сопровождения веб-приложений.

3 Результаты и обсуждение

В результате проведённого анализа были выявлены наиболее распространённые уязвимости авторизации, характерные для Node.js-приложений. Установлено, что большинство из них связано с логическими ошибками реализации контроля доступа, а также с некорректной работой с токенами и пользовательскими сессиями. Ниже приведён детальный анализ ключевых уязвимостей с примерами.

3.1 Отсутствие проверки прав доступа

Одной из наиболее распространённых уязвимостей является отсутствие проверки прав доступа пользователя при обращении к защищённым ресурсам. В ряде случаев разработчики ограничиваются проверкой факта аутентификации, не анализируя роль пользователя (рис. 1).

```
app.get('/admin', authenticateUser, (req, res) => {  
    res.send('Admin panel');  
});
```

Рис 1. Пример уязвимого кода

В данном примере доступ к административному ресурсу получают все аутентифицированные пользователи, независимо от их роли (рис. 2).

```
app.get('/admin', authenticateUser, checkRole('admin'), (req, res) => {  
  res.send('Admin panel');  
});
```

Рис. 2. Безопасный вариант

3.2 Неправильная реализация JWT

Ошибки при работе с JSON Web Token являются частой причиной обхода механизмов авторизации. К ним относятся отсутствие проверки подписи токена и срока его действия (рис. 3).

```
const payload = jwt.decode(token);
```

Рис 3. Пример небезопасной проверки

Функция decode не выполняет верификацию подписи, что позволяет злоумышленнику подделать токен (рис. 4).

```
const payload = jwt.verify(token, process.env.JWT_SECRET);
```

Рис. 4. Корректная реализация

Проверка подписи и срока действия токена является обязательным элементом безопасной авторизации.

3.3 Хранение паролей без хэширования

Использование открытого текста для хранения паролей остаётся критической ошибкой безопасности (рис. 5).

```
user.password = req.body.password;
```

Рис 5. Уязвимый подход

В случае компрометации базы данных злоумышленник получает доступ ко всем учетным данным пользователей (рис. 6).

```
const bcrypt = require('bcrypt');  
user.password = await bcrypt.hash(req.body.password, 10);
```

Рис. 6. Рекомендуемое решение

Применение криптографических хэш-функций существенно снижает риск компрометации паролей.

3.4 Отсутствие защиты от перебора паролей

При отсутствии ограничений на количество попыток входа возможна реализация атак типа brute force (рис. 7).

```
app.post('/login', loginHandler);
```

Рис 7. Пример отсутствия защиты

Установить время. (рис. 8).

```
const rateLimit = require('express-rate-limit');

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100
});

app.post('/login', limiter, loginHandler);
```

Рис. 8. Использование ограничения запросов

Данный подход позволяет существенно снизить вероятность успешного подбора паролей.

3.5 Использование слабых секретных ключей

Недостаточно стойкие секретные ключи упрощают подбор подписи JWT-токенов (рис. 9).

```
const secret = "12345";
```

Рис 9. Небезопасный пример

Секретные ключи должны быть криптографически стойкими и храниться в переменных окружения.

3.6 Утечка токенов через URL

Передача токенов в URL является небезопасной практикой, поскольку они сохраняются в логах сервера и истории браузера (рис. 10).

```
GET /profile?token=eyJhbGciOi...
```

Рис 10. Небезопасный пример

Есть другой способ передачи наиболее надежный (рис. 11).

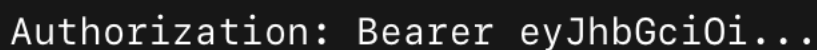
A black rectangular box containing the text "Authorization: Bearer eyJhbGciOi..." in a white monospaced font.

Рис. 11. Рекомендуемый способ передачи

3.7 Отсутствие контроля времени жизни сессии

Бессрочные сессии увеличивают риск несанкционированного доступа при компрометации учетных данных (рис. 12).

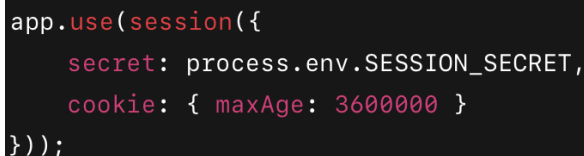
A black rectangular box containing a code snippet in a light-colored monospaced font: `app.use(session({
 secret: process.env.SESSION_SECRET,
 cookie: { maxAge: 3600000 }
}));`

Рис 12. Рекомендуемая настройка

Ограничение времени жизни сессии снижает последствия возможных атак.

3.8 Избыточные данные в токенах

Хранение конфиденциальной информации внутри JWT увеличивает риск её утечки (рис. 13).

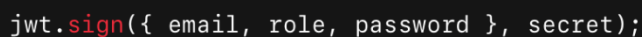
A black rectangular box containing a code snippet in a light-colored monospaced font: `jwt.sign({ email, role, password }, secret);`

Рис 13. Небезопасный пример

Лучше использовать меньше данных (рис. 14).

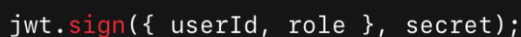
A black rectangular box containing a code snippet in a light-colored monospaced font: `jwt.sign({ userId, role }, secret);`

Рис. 14. Безопасный подход

3.9 Ошибки реализации OAuth

Некорректная обработка параметров `redirect_uri` и `state` создаёт условия для атак перенаправления.

Рекомендация: использовать строгий список допустимых `redirect URI` и обязательную проверку параметра `state`.

3.10 Отсутствие многофакторной аутентификации

Использование только пароля в качестве фактора аутентификации снижает общий уровень безопасности системы.

Внедрение многофакторной аутентификации (например, TOTP) позволяет существенно повысить защищённость пользовательских учетных записей.

4 Выводы

В ходе проведённого исследования были проанализированы наиболее распространённые уязвимости авторизации, характерные для серверных веб-приложений, разработанных с использованием платформы Node.js. Установлено, что значительная часть выявленных уязвимостей обусловлена не техническими ограничениями платформы, а ошибками проектирования и реализации механизмов аутентификации и контроля доступа.

Показано, что наиболее критичными являются уязвимости, связанные с отсутствием проверки прав доступа, некорректной реализацией JSON Web Token, а также использованием слабых секретных ключей и небезопасных методов хранения паролей. Эксплуатация данных уязвимостей может привести к несанкционированному доступу, эскалации привилегий и компрометации конфиденциальных данных пользователей.

На основе анализа научных публикаций и практических примеров были сформулированы рекомендации по повышению уровня безопасности авторизации в Node.js-приложениях. К ним относятся внедрение ролевой модели доступа, корректная проверка подписи и срока действия токенов, использование криптографических методов защиты учетных данных, а также применение механизмов ограничения попыток аутентификации и многофакторной аутентификации.

Результаты исследования подтверждают необходимость комплексного подхода к обеспечению безопасности авторизации, включающего как использование автоматизированных средств анализа уязвимостей, так и проведение ручного аудита логики контроля доступа. Полученные выводы могут быть использованы при разработке, сопровождении и аудите серверных веб-приложений на платформе Node.js.

Библиографический список

1. Verbitskiy I. Node.js security // Cyber Security: A Peer-Reviewed Journal. 2017. Т. 1. №. 2. С. 175–186. URL: <https://www.ingentaconnect.com/content/hsp/jcs/2017/00000001/00000002/art00009> (дата обращения: 09.01.2026).
2. Ntantogian C., Malliaros S., Xenakis C. NodeXP: Node.js server-side JavaScript injection vulnerability detection and exploitation // Journal of Information Security and Applications. 2021. Т. 58. С. 102752. URL: <https://www.sciencedirect.com/science/article/pii/S221421262100003X> (дата обращения: 09.01.2026).
3. Alfadel M., Costa D., Zaidman A. On the threat of npm vulnerable dependencies in Node.js applications // arXiv preprint arXiv:2009.09019. 2020. URL: <https://arxiv.org/abs/2009.09019> (дата обращения: 09.01.2026).

4. Alfadel M., Zaidman A., Adams B. On the discoverability of npm vulnerabilities in Node.js projects // ACM Transactions on Software Engineering and Methodology. 2023. Т. 32. №. 4. С. 1–27. URL: <https://dl.acm.org/doi/abs/10.1145/3571848> (дата обращения: 09.01.2026).
5. Anupam A., Kumar R., Singh S. Analysis of open source Node.js vulnerability scanners // International Research Journal of Engineering and Technology (IRJET). 2020. Т.7. №5. С.5449. URL: <https://www.academia.edu/download/64607920/IRJET-V7I51045.pdf> (дата обращения: 09.01.2026).
6. Node.js Foundation. Node.js Official Documentation // Node.js. 2026. URL: <https://nodejs.org/en/docs> (дата обращения: 09.01.2026).
7. ECMA International. ECMAScript® Language Specification // ECMA International. 2026. URL: <https://tc39.es/ecma262/> (дата обращения: 09.01.2026).
8. Express.js Team. Express — Node.js web application framework // Express.js. 2026. URL: <https://expressjs.com/> (дата обращения: 09.01.2026).
9. Open Web Application Security Project (OWASP). OWASP Top 10 — Web Application Security Risks // OWASP. 2026. URL: <https://owasp.org/www-project-top-ten/> (дата обращения: 09.01.2026).
10. Auth0. JSON Web Token Introduction // Auth0. 2026. URL: <https://jwt.io/introduction> (дата обращения: 09.01.2026).