ТЕХНИЧЕСКИЕ НАУКИ

УДК 004.056

Современные вызовы и угрозы информационной безопасности REST API и способы их предотвращения

Е.Н. Садовая
Донской государственный технический университет (г. Ростов-на-Дону, Российская Федерация)

Аннотация. На основе анализа стандартов OWASP и CWE определены всевозможные актуальные уязвимости, угрозы и слабые места, существующие в системах, использующих REST API. Выявлены наиболее важные проблемы безопасности, возникающие в подавляющем большинстве реализуемых REST API. Проанализированы рассматриваемые уязвимости на абстрактных примерах их возможного возникновения, а также предложены варианты их решения.

Ключевые слова: API, REST API, OWASP, CWE, уязвимости, угрозы, безопасность REST API, безопасность.

Modern Challenges and Threats of REST API Security and Ways to Prevent Them

Ekaterina N Sadovaya
Don State Technical University (Rostov-on-Don, Russian Federation)

Abstract. Based on the analysis of the well-established OWASP and CWE standards, the article identifies all possible vulnerabilities, threats, and weaknesses that exist in systems using REST API. The most important security problems that arise in the vast majority of REST APIs implemented have been identified. The considered vulnerabilities are analyzed on abstract examples of their possible occurrence, as well as options for their solution are proposed.

Keywords: API, REST API, OWASP, CWE, vulnerabilities, threats, REST API security, security

Введение. API-интерфейсы являются одним из основных механизмов взаимодействия программных компонентов. Они служат связующим элементом современных мобильных, десктопных и веб-приложений, возможности которых нашли применение в таких сферах деятельности, как розничная торговля, транспорт, социальные сети, экономика, бухгалтерия и др.

REST (Representational State Transfer) — это архитектура программного обеспечения для взаимодействия по протоколу HTTP. REST API (Application Programming Interface) представляет собой механизм запроса от клиентского приложения к серверному ПО с использованием принципов REST. Такой подход обеспечивает возможность свободного взаимодействия между веб-клиентом и сервером по большому массиву распределенных веб-ресурсов.

Архитектура REST API — это решение, пользующееся широкой популярностью в области организации взаимодействия между различными программами, что обусловлено поддержкой HTTP-протокола, реализуемого во всех операционных системах и языках программирования, в отличие от протоколов, которые принадлежат авторам или правообладателям и не соответствуют принципам свободного программного обеспечения (ПО). Простота реализации и такие преимущества REST API, как гибкость, независимость и возможность масштабирования, определили востребованность REST API при построении микросервисных серверных приложений, организации связи между клиентскими и серверными приложениями, предоставлении программам сторонних разработчиков. В то же время API-интерфейсы раскрывают логику приложения, конфиденциальные данные, в том числе личную информацию. Вследствие чего логичен и интерес злоумышленников к такого рода системам. Без REST API инновации, ведущие за собой создание новых технических и технологических идей, подходов, методов в любой сфере деятельности были бы невозможны, поэтому обеспечение безопасности REST API имеет решающее значение.

В большинстве научных публикаций, относящихся к теме данного исследования [1–3], анализируются угрозы, которые присутствуют в работе микросервисов или веб-приложений. Однако в представленных научных

трудах не нашли отражения вопросы, которые касаются актуальных уязвимостей, встречающихся при разработке REST API, оказывающего существенное влияние на взаимодействие приложений в сети.

Таким образом, целью настоящей работы является анализ наиболее опасных уязвимостей и поиск решений обеспечения безопасности REST API и всей системы, в которой он используется.

Объектом исследования является REST API.

Предметом исследования является безопасность REST API.

Согласно поставленной цели исследования сформулированы следующие задачи:

– найти и определить возможные уязвимости, угрозы и слабые места, существующие в различных системах, которые используют REST API;

– найти и рассмотреть подходящие способы устранения выявленных уязвимостей REST API.

Основная часть. Обзор существующих угроз. Для того чтобы сформулировать достаточный список требований, которые могут быть применены к безопасности REST API, понадобится несколько стандартов: OWASP [4] и CWE [5].

Open Web Application Security Project (OWASP) — это открытый проект, с помощью материалов которого можно обеспечить безопасность приложений. Использование рекомендаций данного проекта давно стало стандартом обеспечения безопасности как веб-приложений, так и API.

Всевозможные списки рисков в разных программных технологиях — это то, чем известен OWASP. Для анализа подойдет OWASP Top 10 — информационный документ, перечисляющий основные проблемы приложений или API. Он обновляется ежегодно, и регулярно выкладываются 10 актуальных серьезных рисков, с которыми могут столкнуться пользователи.

В последнем отчете OWASP (по состоянию на декабрь 2022 года. Список не обновлялся с 2019 года) перечислены 10 основных уязвимостей API: API1:2019 Broken Object Level Authorization (недостатки контроля доступа к объектам), API2:2019 Broken User Authentication (недостатки аутентификации пользователей), API3:2019 Excessive Data Exposure (разглашение конфиденциальных данных), API4:2019 Lack of Resources & Rate Limiting (отсутствие проверок и ограничений), API5:2019 Broken Function Level Authorization (недостатки контроля доступа на функциональном уровне), API6:2019 Mass Assignment (небезопасная десериализация), API7:2019 Security Misconfiguration (некорректная настройка параметров безопасности), API8:2019 Injection (внедрение, инъекции), API9:2019 Improper Assets Management (недостатки управления API), API10:2019 Insufficient Logging & Monitoring (недостатки журналирования и мониторинга) [5, С. 7].

В организации Common Weakness Enumeration (CWE) есть список уязвимостей CWE Top 25 Most Dangerous Software Errors, который также является стандартом обеспечения безопасности [6]. Однако в этом списке были выделены только те ошибки, которые относятся именно к API: CWE-79 Cross-Site Scripting (XSS) (межсайтовое выполнение сценариев), CWE-352 Cross-Site Request Forgery (CSRF) (межсайтовая подмена запросов) [6].

В рамках данной статьи будут рассмотрены наиболее распространенные уязвимости, с которыми сталкиваются все разработчики при работе с REST API:

– API1:2019 Broken Object Level Authorization (недостатки контроля доступа к объектам);

– API2:2019 Broken User Authentication (недостатки аутентификации пользователей) [5, С. 7];

– CWE-352 Cross-Site Request Forgery (CSRF) (межсайтовая подмена запросов) [6].

Исследование и решение уязвимости API1:2019 Broken Object Level Authorization (недостатки контроля доступа к объектам). Используя данную уязвимость, злоумышленник может получить данные объекта по идентификатору через запрос API и изменить его данные, тем самым обойдя весь механизм авторизации.

Например, веб-приложение магазина приложений для каждой компании, которая разместила свои продукты на платформе, предоставляет страницу с графиком доходов. Без решения уязвимости контроля доступа к объектам злоумышленник может проанализировать запросы, которые отправляет браузер для предоставления данных графиков и найти точки входа API, а также определить формат запросов к ним. Злоумышленник может использовать другую точку входа API и, заменив часть с названием компании в URL запроса, получить список всех компаний, которые размещены на платформе. Затем злоумышленник может получить доступ к данным о продажах всех компаний платформы, количество которых может перевалить за тысячу.

Во избежание таких проблем необходимо предпринять несколько действий. Во-первых, при каждом запросе требуется проверять права доступа к объектам, во-вторых, следует проверять авторизованного пользователя на наличие доступа только к разрешенным объектам, в-третьих, стоит учитывать, что ID объектов должны быть в виде UUID, а не в виде простой последовательности чисел, которую несложно подобрать.

OWASP рекомендует следующие модели обеспечения контроля доступа в своем документе Authorization Cheat Sheet [7]:

– Attribute-Based Access Control (ABAC) — разграничение доступа на основе атрибутов доступа;

– Role-Based Access Control (RBAC) — управление доступом на основе ролей;

– Relationship-Based Access Control (ReBAC) — организация разрешений на основе отношений между ресурсами [7].

При этом каждая компания выбирает модель обеспечения контроля доступа исходя из своих нужд и потребностей системы.

Исследование и решение уязвимости API2:2019 Broken User Authentication (недостатки аутентификации пользователей). Механизм аутентификации общедоступен, вследствие чего он является простой целью для злоумышленников.

Используя уязвимость, связанную с аутентификацией пользователей, злоумышленник может получить доступ к персональным данным других пользователей, размещенных в системе, а также взять контроль над их учетными записями. Получив эти данные, злоумышленник может, к примеру, отправить денежные переводы или персональные сообщения от лица других пользователей.

Для решения данной проблемы была рассмотрена авторизация на основе токена.

Token-Based Authentication работает с помощью подписанного сервером токена, так называемого bearer token, который передается клиентом на сервер в заголовке Authorization HTTP, используя ключевое слово Bearer или в теле запроса. Пример такого заголовка представлен на рис. 1.

```
Authorization: Bearer
ya29.GlOOBArTXfuhWoGhRmbbH3yVulTdcm8EGaDv2U1RfCiRnD1VxHqRNhJztHuXopze4YHOOdwHC-f9bJsjDvYe5AjXD1hSzEuzMi
b-veSPireP9OBMr3LMHOvY2wu_T5s
```

Рис. 1. Пример Bearer token в заголовке Authorization

Затем, когда сервер получает токен, он должен проверить его на валидность, то есть проверить существование пользователя и истечение времени пользования токена, и т. д.

Для данной уязвимости необходимо учитывать, что при любом способе аутентификации следует использовать протокол HTTPS, обеспечивающий шифрование данных, URL и HTTP заголовков.

Исследование и решение CWE-352 Cross-Site Request Forgery (CSRF) (межсайтовая подмена запросов).

CSRF является опаснейшей атакой, при которой злоумышленник пытается выполнить некоторые действия от имени других авторизованных пользователей без их согласия. Атака основывается на том, что большинство веб-приложений, в том числе использующих REST API, работают с сессионными cookie, хранение которых происходит в браузере.

Например, злоумышленник может создать поддельную форму на своём сайте, выполняющую запрос на действие на целевом сайте от имени пользователя, который посещает этот поддельный сайт и взаимодействует со вредоносной формой. Абстрактный пример формы злоумышленника приведен на рис. 2.

```
<form method="POST" action="http://victim-site.com/user/password_change">
  <input type="password" name="new_password" value="пароль123" style="display:none;">
  <button type="submit" style="display:none;"></button>
</form>
<script>document.forms[0].submit();</script>
```

Рис. 2. Пример вредоносной формы злоумышленника

Когда пользователь посещает эту страницу, последняя строчка скрипта выполняет отправку формы с новым паролем на целевой сайт в незаметном режиме, таким образом изменение пароля пользователя будет выполнено без его согласия.

Для решения данной уязвимости в REST API было найдено следующее решение: Cross-origin resource sharing (CORS) (кросс-доменное использование ресурсов) — это механизм браузера, который позволяет веб-страницам запрашивать данные с другого домена. Если на сервере, предоставляющем REST API, не настроен CORS, то браузеры блокируют запросы из других источников. Чтобы разрешить запросы из других источников, веб-сервер должен отправить заголовки CORS в ответе на запрос. На рис. 3 показан пример использования CORS, написанного на .NET Core.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseRouting();
    app.UseAuthorization();
    app.UseCors();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

Рис. 3. Подключение CORS в REST API на .NET Core

При установке на сервере заголовка WithOrigins: * доступ к API не будет ограничен. Однако, если API является закрытым, необходимо указать источники (Origin), которым разрешен доступ к API, используя параметр WithOrigins (https://example.com).

Для усиления безопасности и защиты от нежелательного использования можно ограничить разрешенные HTTP методы, используя параметр AllowWithMethods (GET, POST, DELETE, PUT), а также задать список допустимых заголовков, которые сервер может принимать, используя параметр AllowAnyHeaders (Origin, Content-Type, Authorization). Применение этих параметров повышает безопасность API и способствует защите от возможных угроз.

Для более чёткого понимания на рис. 4 представлен вариант использования CORS, написанного на .NET Core.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors(option => option.AddDefaultPolicy(policy =>
    {
        policy.WithOrigins("*");
        //policy.AllowAnyOrigin();
        policy.WithHeaders("Origin", "Content - Type", "Authorization");
        //policy.AllowAnyHeader();
        policy.WithMethods("GET", "POST", "DELETE", "PUT");
        //policy.AllowAnyMethod();
    }));
}
```

Рис. 4. Использование CORS в REST API на .NET Core

В результате реализации механизма защиты на основе кросс-доменного использования ресурсов можно эффективно настроить CORS для REST API и разрешить запросы из других источников безопасным и контролируемым способом.

Заключение. Таким образом, в ходе данного исследования были проанализированы стандарты OWASP и CWE и определены актуальные угрозы и слабые места, существующие в системах, использующих REST API. API-интерфейсы раскрывают логику приложения, конфиденциальные данные, поэтому обеспечение их безопасности имеет решающее значение. Было выяснено, что для решения уязвимости API1:2019 Broken Object Level Authorization следует тщательно проверять права доступа к объектам авторизованного и неавторизованного пользователя, а также использовать UUID. При обнаружении угрозы API2:2019 Broken User Authentication следует использовать авторизацию на основе токена. Во избежание межсайтовой подмены запросов CWE-352 Cross-Site Request Forgery можно использовать кросс-доменное использование ресурсов.

Знание актуальных уязвимостей и наличие вариантов их устранения помогут в дальнейшем усилить защищенность каждой системы, использующей REST API, от различного рода атак и утечек конфиденциальных, медицинских, финансовых и личных данных.

Библиографический список

1. Ковалев В.В., Храмков В.В., Семенова Е.И. *Актуальность использования архитектуры REST для обмена данными между клиент-серверными приложениями*. В: Сборник материалов V Международной научно-практической конференции, посвященной Дню космонавтики «Актуальные проблемы авиации и космонавтики». В 3-х томах. Том 2. Красноярск; 2019. С. 339–340.

2. Baker O., Nguyen Q. *A Novel Approach to Secure Microservice Architecture from OWASP vulnerabilities*. In: Proceedings of the 10th Annual CITRENZ Conference. Wellington; 2019. p. 56-61.

3. Нестеренко В.Р., Маслова М.А. Современные вызовы и угрозы информационной безопасности публичных облачных решений и способы работы с ними. *Научный результат. Информационные технологии*. 2021;6(1):48–54. doi: 10.18413/2518-1092-2021-6-1-0-6

4. OWASP API Security Project. *OWASP*. URL: https://owasp.org/www-project-api-security (accessed: 10.02. 2023).

5. *OWASP API Security Project. Top 10.* OWASP. URL: https://github.com/OWASP/API-Security/blob/master/2019/en/dist/owasp-api-security-top-10.pdf (accessed 10.02.2023).

6. *2022 CWE Top 25 Most Dangerous Software Weaknesses*. Common Weakness Enumeration. URL: https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html (accessed 02.02.2023).

7. *Authorization Cheat Sheet*. Owasp. Cheat. Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html (accessed 10.02.2023)

Об авторе:

Садовая Екатерина Николаевна, студент кафедры «Информационные системы и технологии» Донского государственного технического университета (344003, РФ, г. Ростов-на-Дону, пл. Гагарина, 1), katya_ns@mail.ru

About the Author:

Ekaterina N Sadovaya, student of the Information Systems and Technologies Department, Don State Technical University (1, Gagarin Sq., Rostov-on-Don, 344003, RF), katya_ns@mail.ru