УДК 004.67:004.054

Анализ распространённых ошибок сериализации, приводящих к CWE-502, на примере REST- и gRPC-микросервисов

Н.Б. Алдабергенов
студент 1 курса аспирантуры НИЯУ МИФИ, Москва
E-mail: nurzofix12@yandex.ru
И.В. Прохоров
доцент кафедры финансового мониторинга НИЯУ МИФИ, Москва
E-mail: iprohorov@mephi.ru

Аннотация: *В работе рассматриваются распространённые ошибки сериализации данных в микросервисных архитектурах, которые приводят к уязвимости CWE-502 «Deserialization of Untrusted Data». Анализируются особенности REST- и gRPC-взаимодействий, исследуются типичные сценарии возникновения небезопасной десериализации и демонстрируется, как неправильное обращение с входящими объектами в распределённых системах может привести к выполнению произвольного кода или нарушению целостности данных. Представлены рекомендации по предотвращению CWE-502 в сервис-ориентированных приложениях.*

Ключевые слова: *CWE-502, десериализация, REST, gRPC, микросервисы, безопасность данных, сериализация.*

Analysis of common serialization errors leading to CWE-502 in REST and gRPC microservices

N.B. Aldabergenov
2nd year master’s student at NRNU MEPhI, Moscow
E-mail: nurzofix12@yandex.ru
I.V. Prokhorov
Associate Professor, Department of Financial Monitoring,
NRNU MEPhI, Moscow
E-mail: iprohorov@mephi.ru

Abstract: *This paper examines common data serialization errors in microservice architectures that lead to the CWE-502 vulnerability known as “Deserialization of Untrusted Data.” The study analyzes the specifics of REST and gRPC communication, explores typical scenarios in which unsafe*

deserialization occurs, and demonstrates how improper handling of incoming objects in distributed systems may result in arbitrary code execution or compromise of data integrity. Recommendations for preventing CWE-502 in service-oriented applications are provided.

Keywords: *CWE-502, deserialization, REST, gRPC, microservices, data security, serialization.*

Введение

Микросервисные архитектуры стали доминирующим подходом к построению современных распределённых систем, поскольку обеспечивают гибкость, независимость компонентов и возможность горизонтального масштабирования. Однако декомпозиция приложения на отдельные сервисы неизбежно увеличивает количество точек взаимодействия и расширяет поверхность атаки. Во всех формах коммуникации между сервисами центральную роль играет сериализация данных, позволяющая преобразовывать сложные структуры в формат, пригодный для передачи по сети или хранения. Ошибки в механизмах сериализации и десериализации приводят к серьёзным угрозам безопасности, среди которых одной из наиболее опасных является CWE-502 - десериализация недоверенных данных.

Уязвимость CWE-502 возникает, когда сервис получает внешние данные и восстанавливает из них объект, не применяя достаточных ограничений, фильтрации или проверки структуры входного сообщения. Такая ситуация характерна не только для старых технологий, использующих бинарные форматы сериализации с поддержкой выполнения кода, но и для современных REST-сервисов, работающих с JSON, и gRPC-микросервисов, основанных на протоколе protobuf. Несмотря на то что protobuf считается более безопасным и типобезопасным, неправильное использование его расширенных возможностей также создаёт условия для возникновения CWE-502.

Опасность CWE-502 заключается в том, что атака может приводить к выполнению произвольного кода, нарушению целостности данных или изменению логики приложения. Для микросервисных архитектур такие последствия особенно критичны, поскольку один скомпрометированный компонент способен стать точкой входа для дальнейшего распространения атаки по всей системе. В условиях высокой связанности сервисов ошибки сериализации могут проявляться в самых разных сценариях: от взаимодействия REST-API до обмена сообщениями через очереди Kafka или RabbitMQ.

Актуальность исследования обусловлена тем, что разработчики нередко ошибочно воспринимают сериализацию как нейтральную и безопасную процедуру, не отдавая себе отчёта в том, какие побочные эффекты она

может иметь. В результате API-эндпоинты и gRPC-методы начинают принимать неконтролируемые структуры данных, которые в определённых условиях становятся вектором атаки. Данная работа направлена на анализ наиболее распространённых ошибок сериализации, встречающихся при разработке REST- и gRPC-микросервисов, и исследование механизмов, приводящих к уязвимости CWE-502.

Постановка задачи

Исследование распространённых ошибок сериализации в микросервисных архитектурах требует формального определения целей и задач, которые позволяют структурировать анализ и сформировать практическую ценность работы. Поскольку уязвимость CWE-502 проявляется тогда, когда сервис без должного контроля десериализует объекты из внешнего источника, первостепенной задачей становится выявление тех архитектурных и логических решений, которые делают такую ситуацию возможной. В условиях REST- и gRPC-взаимодействий механизмы сериализации присутствуют на каждом этапе обработки запроса, и несоответствие входящих данных ожиданиям сервиса нередко приводит к нарушению его работы. Поэтому важно последовательно исследовать, где именно в цепочке обработки сообщения возникает риск некорректной десериализации.

Первым шагом является определение набора механизмов сериализации, наиболее часто используемых в современных микросервисных системах. Это включает анализ JSON-парсеров, применяемых в REST-сервисах, и особенностей protobuf-структур, лежащих в основе gRPC-коммуникаций. Понимание различий между этими форматами необходимо для объяснения того, почему оба подхода, несмотря на свою кажущуюся безопасность, могут стать источником защиты недостаточного уровня и привести к CWE-502.

Следующей задачей является выявление причин, по которым сервис начинает принимать данные в формате, который разработчик не предусмотрел. В микросервисных архитектурах эта проблема усугубляется тем, что сервисы часто обновляются независимо друг от друга. Различия версий контрактов, неполное применение схем валидации данных либо отключённые защитные механизмы при отладке создают множество ситуаций, в которых данные проходят вглубь системы в неверном виде. Подобные несоответствия могут сопровождаться ошибками маппинга полей, непредусмотренными преобразованиями типов или загрузкой структур, обходящих контроль доступа.

Дополнительная задача связана с исследованием тех случаев, когда небезопасная десериализация возникает не из-за внешнего злоумышленника, а в результате внутреннего поведения системы. Примеры включают обмен бинарными объектами через очереди

сообщений, использование дополнительных библиотек сериализации, динамическую загрузку типов, применение полей расширения в protobuf или передачу универсальных структур типа Any. Такие ситуации характерны для высоконагруженных микросервисов, где разработчики стремятся сохранить производительность, но при этом невольно вводят механизмы, допускающие неконтролируемую десериализацию.

Заключительная задача формулируется как разработка рекомендаций, позволяющих минимизировать вероятность возникновения CWE-502 в реальных REST- и gRPC-системах. Она предполагает анализ методологий безопасной разработки, архитектурных подходов к валидации данных, требований к строгим контрактам и правил проектирования микросервисов. Ожидаемый результат решения этих задач - создание комплексного, но практичного представления о том, как ошибки сериализации возникают, как они могут быть обнаружены и какие меры позволяют эффективно ограничить их влияние.

Основная часть

Анализ распространённых ошибок сериализации в REST- и gRPC-микросервисах требует рассмотрения как технических особенностей форматов передачи данных, так и практических аспектов разработки распределённых систем. Поскольку каждая архитектурная модель по-разному обрабатывает данные, важно понять, какие именно свойства форматов сериализации создают условия для возникновения CWE-502. Несмотря на то что REST и gRPC часто воспринимаются как современные и безопасные решения, ошибки сериализации в них присутствуют так же часто, как и в более старых технологиях.

Начать следует с REST-сервисов, в которых сериализация данных чаще всего осуществляется в формате JSON. JSON обладает гибкостью и не требует строгих схем, что делает его удобным инструментом для быстрой разработки, но именно отсутствие жёсткой типизации приводит к множеству ошибок. Парсеры JSON нередко допускают расширенные конструкции и автоматически преобразуют строки в числа, массивы в объекты или наоборот. Когда сервис получает структуру, которая не соответствует ожидаемой, десериализация может привести к созданию объекта с непредсказуемым набором полей. Некоторые фреймворки, такие как Jackson или Gson, по умолчанию пытаются восстановить объект максимально близко к оригиналу, что нередко приводит к появлению побочных эффектов, особенно если разработчик не отключил динамическую загрузку типов или не ограничил набор допустимых классов. В ситуациях, когда приложение принимает входящий JSON от внешнего источника, эта гибкость становится фактором риска, поскольку позволяет злоумышленнику передать структуру данных, несоответствующую контракту.

Дополнительную сложность создают ситуации, когда REST-сервис взаимодействует с другими компонентами через брокеры сообщений или очереди. В таких системах разработчики нередко передают данные между сервисами без строгой проверки соответствия схемам, особенно если взаимодействие осуществляется через внутренние каналы связи. Когда очереди используются как промежуточный уровень доставки, десериализация осуществляется автоматически, и ошибки могут возникать на этапе получения даже легитимного сообщения. Злоумышленник, получив доступ к продюсеру сообщений, может сгенерировать структуру, вызывающую некорректные преобразования в консьюмере, что создаёт условия для эксплуатации CWE-502. Таким образом, даже в тех случаях, когда сервисы не принимают данные напрямую от внешнего клиента, риск остаётся актуальным.

В gRPC-микросервисах ситуация выглядит иначе: протокол основан на бинарном формате protobuf, который обеспечивает строгую типобезопасность и требует предварительной компиляции схем. Такая архитектура значительно снижает вероятность ошибок сериализации, но полностью проблему не устраняет. В реальных системах protobuf используется совместно с расширенными механизмами, такими как динамические сообщения, универсальные контейнеры, поля расширений и структура Any. Последняя, в частности, позволяет передавать произвольные типы объектов, что фактически создаёт возможность десериализовать неизвестный тип без контроля. Если разработчик не предусмотрел фильтрацию, сервис может попытаться восстановить объект, который ему не предназначался. Подобные случаи особенно часто возникают в системах, где необходимо поддерживать обратную совместимость или интеграцию с legacy-сервисами.

Особого внимания требуют ошибки, возникающие при обновлении версий protobuf-схем. В условиях микросервисной архитектуры различные команды обновляют сервисы независимо, что приводит к временному несоответствию контрактов. Когда один сервис обновляет структуру сообщения, а другой продолжает использовать старую схему, может возникнуть ситуация, при которой входящие данные будут иметь значения, которые принимающая сторона не ожидает. Несмотря на то что protobuf спроектирован таким образом, чтобы старые сервисы игнорировали неизвестные поля, ошибки возникают тогда, когда разработчики опираются на строго определённую структуру и не учитывают возможность появления новых значений. В результате десериализация может приводить к инициализации объекта в неверном состоянии, что становится начальной точкой для эксплуатации уязвимости.

Рассмотрим практический пример. REST-сервис, принимающий JSON-объект для создания нового пользователя, ожидает минимальный набор

полей: имя, пароль и роль. Если в обработчике данных отсутствует строгая валидация, злоумышленник может добавить дополнительные поля, которые будут интерпретированы фреймворком в качестве параметров конфигурации или системных свойств. Аналогичная ситуация может возникнуть и в gRPC-сервисе, если используется поле Any. Такие ошибки демонстрируют, что даже современные инструменты сериализации, ориентированные на безопасность, могут стать источником CWE-502 при неправильном использовании.

Важно подчеркнуть, что большинство ошибок сериализации связано не с особенностями самих технологий, а с неправильными архитектурными решениями. Разработчики нередко стремятся упростить логику обработки данных, передавая объекты напрямую, без уточнения контрактов или проверки структуры входящих сообщений. В условиях высокой степени автоматизации сериализации и десериализации многие фреймворки скрывают от разработчика детали процесса, что создаёт ложное ощущение безопасности. На практике это приводит к тому, что сервис начинает принимать данные, не соответствующие ожиданиям, и тем самым создаёт предпосылки для возникновения CWE-502.

Таким образом, анализ распространённых ошибок показывает, что возникновение CWE-502 в современных микросервисных архитектурах является следствием сочетания гибких форматов сериализации, недостаточной валидации данных и архитектурных особенностей распределённых систем. REST- и gRPC-микросервисы по-разному подвержены таким ошибкам, однако в обоих случаях неправильное использование механизмов сериализации создаёт критическую угрозу безопасности.

Заключение

Проведённый анализ показал, что ошибки сериализации в REST- и gRPC-микросервисах остаются одной из ключевых причин возникновения уязвимости CWE-502, несмотря на широкое распространение современных инструментов и фреймворков, ориентированных на безопасность. Исследование позволило выявить, что источником большинства проблем является не сама технология сериализации, а особенности взаимодействия микросервисов, стремительное развитие архитектуры приложения и упрощённые подходы к обработке данных. В условиях высокой степени автоматизации разработчики зачастую полагаются на стандартные механизмы десериализации, не учитывая возможность передачи непредвиденных структур, что открывает путь к возникновению уязвимостей.

REST-сервисы, использующие JSON в качестве основного формата обмена данными, подвержены рискам из-за отсутствия строгой схемы и гибкости структуры. При отсутствии проверок входящих данных сервис

может некорректно интерпретировать переданные значения, что приводит к созданию объектов в нежелательном состоянии. gRPC-микросервисы в целом обладают более строгими механизмами контроля благодаря protobuf-схемам, однако использование расширенных возможностей вроде универсальных структур и динамических типов также создаёт условия для возникновения CWE-502, особенно в проектах, где различные компоненты поддерживаются независимыми командами и обновляются асинхронно.

Полученные результаты подчёркивают необходимость комплексного подхода к защите микросервисных систем. Надёжность сериализации должна обеспечиваться как на уровне архитектурных решений, так и на уровне конкретных реализаций сервисов. Строгая валидация входящих данных, использование жёстко определённых контрактов, отказ от небезопасных механизмов сериализации и контроль за версиями схем - всё это является обязательным условием предотвращения CWE-502. Дополнительные меры, такие как анализ поведения сервисов, ограничение размеров сообщений и контроль глубины структур, также повышают устойчивость системы к ошибкам десериализации.

Таким образом, можно сделать вывод, что безопасная сериализация является фундаментальной частью разработки микросервисных архитектур. Понимание механизмов возникновения CWE-502 и знание распространённых ошибок позволяет разработчикам и специалистам по безопасности создавать более устойчивые сервисы, снижать риски эксплуатации уязвимостей и обеспечивать надёжность распределённых систем. Продолжение исследований в этой области остаётся актуальным, поскольку расширение микросервисных платформ и усложнение взаимосвязей между сервисами неизбежно приводят к появлению новых форм небезопасной обработки данных.

Список использованных источников:

1. CWE-502: Deserialization of Untrusted Data [Электронный ресурс]. – Режим доступа: https://cwe.mitre.org/data/definitions/502.html (дата обращения: 13.11.2025).

2. MITRE. 2024 CWE Top 25 Most Dangerous Software Weaknesses [Электронный ресурс]. – Режим доступа: https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html (дата обращения: 12.11.2025).

3. OWASP Foundation. Deserialization Cheat Sheet [Электронный ресурс]. – Режим доступа: https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html (дата обращения: 25.10.2025).

4. Gauthier F., Bae S. Runtime Prevention of Deserialization Attacks [Электронный ресурс] // arXiv. – Режим доступа: https://arxiv.org/abs/2204.09388 (дата обращения: 18.10.2025).

5. Cao S., He B., Sun X., Ouyang Y., Zhang C., Wu X., Li B. ODDFUZZ: Discovering Java Deserialization Vulnerabilities via Structure-Aware Directed Greybox Fuzzing [Электронный ресурс] // arXiv. – Режим доступа: https://arxiv.org/abs/2304.04233 (дата обращения: 13.10.2025).

6. Sayar I., Bartel A., Bodden E., Le Traon Y. An In-depth Study of Java Deserialization Remote-Code Execution Exploits and Vulnerabilities [Электронный ресурс] // arXiv. – Режим доступа: https://arxiv.org/abs/2208.08173 (дата обращения: 4.11.2025).

7. Unsafe Deserialization in Python (CWE-502) [Электронный ресурс] // Codiga Blog. – Режим доступа: https://www.codiga.io/blog/python-unsafe-deserialization/ (дата обращения: 8.11.2025).

8. Waratek. Best Practices – Unsafe Deserialization of Untrusted Data (CWE-502) [Электронный ресурс]. – Режим доступа: https://support.waratek.com/knowledge/best-practices-unsafe-deserialization-of-untrusted-data (дата обращения: 8.11.2025).