



OWASP API Security Top 10 2019

Десять наиболее критичных рисков безопасности API

Содержание

Содержание.....	2
Предисловие.....	3
Введение.....	4
Анонс выпуска.....	5
Риски безопасности API.....	6
OWASP API Security Top 10 - 2019.....	7
API1:2019 Некорректная Авторизация на Уровне Объектов.....	8
API2:2019 Некорректная Аутентификация Пользователей.....	10
API3:2019 Предоставление Излишних Данных. .12	
API4:2019 Отсутствие Ограничений на Количество Запросов и Потребляемые Ресурсы. .14	
API5:2019 Некорректная Авторизация на Уровне Функций.....	16
API6:2019 Массовое Переназначение Параметров	18
API7:2019 Ошибки Настроек Безопасности.....	20
API8:2019 Инъекции.....	22
API9:2019 Ненадлежащее Управление Активами	24
API10:2019 Недостаточное Логирование и Мониторинг.....	26
Дальнейшие шаги для разработчиков.....	28
Дальнейшие шаги для DevSecOps.....	29
Методология и данные.....	30
Благодарность.....	31

ОБ OWASP

Open Web Application Security Project (OWASP) - открытое сообщество, нацеленное на предоставление возможности организациям разрабатывать, покупать и поддерживать приложения и API, которым можно доверять.

В OWASP вы найдёте открытые и бесплатные:

- Инструменты и стандарты по безопасности приложений.
- Полноценные книги по тестированию безопасности, разработке безопасного кода и код ревью на безопасность.
- Презентации и [видео](#).
- [Списки рекомендаций \(Cheat sheets\)](#) на разные темы.
- Стандартизированные меры безопасности и библиотеки.
- [Локальные отделения по всему миру](#).
- Современнейшие исследования.
- Масштабные [конференции по всему миру](#).
- [Списки рассылки](#).

Узнайте больше на <https://www.owasp.org>.

Все инструменты, документы, видео, презентации и отделения OWASP бесплатны и открыты для всех, кто заинтересован в улучшении безопасности разработки приложений.

Мы выступаем за подход к безопасности к разработке приложений как к проблеме на уровне людей, процессов и технологий, поскольку наиболее эффективные методы обеспечения безопасности приложений требует улучшения на всех этих уровнях.

OWASP - это новый вид организации. Наша свобода от давления коммерческих организаций позволяет нам предоставлять беспристрастную, практичную и экономически оправданную информацию о безопасности приложений.

OWASP не аффилирован ни с какими технологическими компаниями, однако мы поддерживаем информированное использование коммерческих технологий по безопасности. OWASP создаёт множество материалов коллективным, прозрачным и открытым способом.

OWASP Foundation - некоммерческая организация, обеспечивающая долговременный успех проекта. Почти все участники OWASP - волонтеры, включая исполнительный комитет OWASP, руководителей отделений, лидеров и участников проектов. Мы поддерживаем инновационные исследования по безопасности грантами и инфраструктурой.

Присоединяйтесь к нам!

Предисловие

Программный интерфейс приложений (API) - фундаментальный элемент инноваций в современном, подвижном приложениями мире. API - важная составляющая современных мобильных, SaaS и веб приложений, используемая в клиентских, партнерских и внутренних приложениях от банковской сферы, сфер розничных продаж и логистики до интернета вещей, автономных автомобилей и умных городов.

По своей природе API раскрывают логику приложения и критичные данные, например, персональные данные, именно поэтому API все чаще становятся целью злоумышленников. Стремительные инновации невозможны без безопасных API.

Несмотря на то, что более обширный Web Application Security Risks Top 10 по-прежнему актуален, ввиду специфики API, необходим отдельный список рисков безопасности специфичных для API. Безопасность API фокусируется на стратегиях и решениях, направленных на понимание и предотвращение уникальных уязвимостей и рисков безопасности, связанных с использованием API.

Если вы уже знакомы с [OWASP Top 10 Project](#), то наверняка заметите сходства с настоящим документом, их основная цель - улучшение читаемости и повышение частоты использования настоящего документа. Если вы не знакомы с семейством OWASP Top 10, то рекомендуем сначала ознакомиться с секциями [Риски Безопасности API](#) и [Методология и Данные](#) перед погружением в список Top 10.

Вы можете поспособствовать OWASP API Security Top 10 своими вопросами, комментариями и идеями в нашем репозитории GitHub:

- <https://github.com/OWASP/API-Security/issues>
- <https://github.com/OWASP/API-Security/blob/master/CONTRIBUTING.md>

Вы можете ознакомиться с OWASP API Security Top 10 здесь:

- https://www.owasp.org/index.php/OWASP_API_Security_Project
- <https://github.com/OWASP/API-Security>

Мы хотим поблагодарить всех участников проекта, которые своим вкладом и стараниями помогли проекту свершиться. Все они перечислены в секции [Благодарность](#). Спасибо вам!

Добро пожаловать в OWASP API Security Top 10 - 2019!

Добро пожаловать в первую версию OWASP API Security Top 10. Если вы знакомы с семейством OWASP Top 10, то вы заметите сходства, их основная цель - улучшение читаемости и повышение частоты использования настоящего документа. В противном случае посетите страницу [OWASP API Security Project wiki](#) перед погружением в наиболее критичные риски безопасности API.

API играют важную роль в архитектуре современных приложений. Поскольку информированность в сфере безопасности и инновации движутся с разной скоростью, важно сфокусироваться на общих недостатках безопасности API.

Основная цель OWASP API Security Top 10 - предоставить информацию людям, вовлечённым в разработку и поддержание API, например, разработчикам, дизайнерам, архитекторам, менеджерам или целым организациям.

В секции [Методология и Данные](#) вы можете более подробно ознакомиться с тем, как была создана текущая версия. В будущих версиях мы хотим вовлечь индустрию безопасности путём публичного сбора данных. А сейчас мы призываем всех внести свой вклад вопросами, комментариями и идеями в нашем [GitHub репозитории](#) или [Списке рассылки](#).



Анонс выпуска

Это первая версия OWASP API Security Top 10, которую мы планируем обновлять каждые три-четыре года.

В будущих версиях, в отличие от этой, мы планируем публичный сбор данных, вовлекая индустрию безопасности в эту активность. В секции [Методология и Данные](#) вы можете найти подробности о том, как эта версия была создана. Для дополнительной информации о рисках безопасности обратитесь к секции [Риски безопасности API](#).

Необходимо понимать, что за последние несколько лет архитектура приложений значительно изменилась. В настоящий момент API играют очень важную роль в новой архитектуре микросервисов, одностраничных приложений (SPA), мобильных приложений, интернете вещей и так далее.

OWASP API Security Top 10 - попытка повысить информированность о проблемах безопасности современных API. Он был возможен благодаря труду волонтеров, перечисленных в секции [Благодарность](#). Спасибо вам!

Риски безопасности API

Для анализа рисков была использована [Методология оценки рисков OWASP](#).

Терминология, относящаяся к оценкам риска, приведена в таблице ниже.

Источники угроз	Сложность эксплуатации	Распространенность	Сложность обнаружения	Технические последствия	Последствия для бизнеса
Зависит от API	Просто: 3	Широкая 3	Просто 3	Значительные 3	Зависит от бизнеса
	Средне: 2	Обычная 2	Средне 2	Средние 2	
	Сложно: 1	Редкая 1	Сложно 1	Незначительные 1	

Примечание: Этот подход не принимает во внимание источник угроз, а также различные технические детали вашего конкретного приложения. Любой из этих факторов может значительно повлиять на общую вероятность обнаружения и эксплуатации злоумышленником конкретной уязвимости. Эта система оценки не принимает во внимание фактические последствия для вашего бизнеса. Вашей организации необходимо решить, какие риски безопасности, создаваемые приложениями и API, она готова принять с учётом её культуры безопасности, индустрии и требований регуляторов. Проведение анализа рисков за вас не является целью OWASP API Security Top 10.

Ссылки

OWASP

- [OWASP Risk Rating Methodology](#)
- [Article on Threat/Risk Modeling](#)

Внешние

- [ISO 31000: Risk Management Std](#)
- [ISO 27001: ISMS](#)
- [NIST Cyber Framework \(US\)](#)
- [ASD Strategic Mitigations \(AU\)](#)
- [NIST CVSS 3.0](#)
- [Microsoft Threat Modeling Tool](#)

OWASP API Security Top 10 - 2019

API1:2019 - Некорректная Авторизация на Уровне Объектов	API зачастую предоставляют точки входа, оперирующие идентификаторами объектов, тем самым создавая широкую поверхность атаки на уровне управления доступом. Необходимо принимать во внимание проверки авторизации на уровне объектов в каждой функции, которая обращается к источнику данных, используя пользовательский ввод.
API2:2019 - Некорректная Аутентификация Пользователей	Механизмы аутентификации часто функционируют некорректно, позволяя злоумышленникам компрометировать аутентификационные токены или эксплуатировать несовершенства в реализации механизма с целью временного или постоянного присвоения учетной записи пользователя. Компрометация системы идентификации пользователей или клиентов компрометирует API в целом.
API3:2019 - Предоставление Излишних Данных	Будучи нацеленными на универсальную реализацию функций API, разработчики зачастую раскрывают все свойства объектов, не учитывая критичность каждого из них и рассчитывая, что клиентское приложение отфильтрует данные перед отображением пользователю.
API4:2019 - Отсутствие ограничений на количество запросов и потребляемые ресурсы	Зачастую API не устанавливают ограничений на размер или количество ресурсов, запрашиваемых клиентским приложением или пользователем. Это может повлиять на производительность API, приводя к отказу в обслуживании, а также открывает возможность атак перебором на механизм аутентификации.
API5:2019 - Некорректная авторизация на уровне функций	Сложные политики контроля доступа с различными иерархиями, группами и ролями, а также нечеткое разделение административного и пользовательского функционала приводят к ошибкам механизма авторизации. Эксплуатируя эти ошибки, злоумышленник может получить доступ к ресурсам других пользователей и административному функционалу.
API6:2019 - Массовое переназначение параметров (Mass assignment)	Присвоение поступивших от пользователя данных, например в формате JSON, модели данных без надлежащей фильтрации параметров на базе белого списка обычно приводит к массовому переназначению параметров. Злоумышленник может изменить свойства объектов, к которым он не должен иметь доступ, угадав эти свойства, найдя их в других точках входа API или документации, или же отправив запрос с дополнительными свойствами.
API7:2019 - Ошибки настроек безопасности	Как правило, ошибки настроек безопасности - результат небезопасных настроек по умолчанию, неполных или временных настроек, незащищенного облачного хранилища, некорректно настроенных заголовков HTTP, излишних методов HTTP, нестрогой политики разделения ресурсов между источниками (CORS) и детальных сообщений об ошибках, содержащих критичные данные.
API8:2019 - Инъекции	Инъекции, такие как SQL, NoSQL, инъекции команд на операционной системе и другие, случаются, когда недоверенные данные отправляются в интерпретатор как часть команды или запроса. Злоумышленник может вынудить интерпретатор выполнить команду или получить данные в обход проверок авторизации.
API9:2019 - Ненадлежащее управление активами	API обычно имеют больше точек входа по сравнению с традиционным веб приложениями, что приводит к необходимости постоянного ведения и обновления документации. Надлежащая инвентаризация хостов (доменов и серверов, на которых функционирует API) и развернутых версий API играет важную роль в предотвращении проблем, таких как публикация устаревших версий API и точек входа, используемых для отладки.
API10:2019 - Недостаточное логирование и мониторинг	Недостаточное логирование и мониторинг вместе с отсутствующим или неэффективным реагированием на инциденты позволяет злоумышленникам атаковать и оставаться в системе, атаковать другие системы, доступные из скомпрометированной, а также извлекать или уничтожать данные. Большинство исследований произошедших атак показывает, что время обнаружения атак превышает 200 дней, а также что обычно атаки обнаруживаются извне, а не внутренними процессами или мониторингом.

API1:2019 Некорректная Авторизация на Уровне Объектов

Зависит от API	Сложность эксплуатации: 3	Распространенность: 3	Сложность обнаружения: 2	Технические последствия: 3	Зависит от бизнеса
<p>Злоумышленник может проэксплуатировать уязвимые к некорректной авторизации на уровне объектов точки входа API, манипулируя идентификатором объекта, отправляемого в запросе. Это может привести к неавторизованному доступу к критичной информации. Это очень распространенная ситуация среди приложений базирующихся на API, потому что серверная часть не полностью отслеживает состояние клиента, а вместо это полагается на отправляемые клиентом параметры, например, идентификатор объекта, чтобы принять решение, к какому объекту осуществляется доступ.</p>		<p>Данная атака на API наиболее распространена и несет наибольшие последствия. Авторизация и механизмы контроля доступа в современных приложениях повсеместны и зачастую запутаны. Даже если проверки авторизации реализованы в приложении корректно, разработчики могут забыть добавить эти проверки перед доступом к критичным объектам. Автоматизированное статическое и динамическое тестирование, как правило, не обнаруживает проблемы в управлении доступом.</p>		<p>Неавторизованный доступ может привести к потере или манипуляции данными, а также разглашению данных лицам, не имеющим права доступа к ним. Кроме того неавторизованный доступ может привести к получению полного контроля над учетными записями.</p>	

Как определить, является ли API уязвимым?

Авторизация на уровне объектов - это механизм управления доступом, реализуемый на уровне кода и проверяющий, что пользователь может получить доступ только к тем объектам, к которым у него должен быть доступ.

Каждая точка входа API, получающая идентификатор объекта и выполняющая любое действие с объектом, должна провести проверку авторизации на уровне объекта. Проверка должна удостовериться в том, что текущий пользователь действительно имеет право осуществить запрошенное действие над запрошенным объектом.

Некорректная работа этого механизма обычно приводит к неавторизованному разглашению или изменению данных, а также к их уничтожению.

Пример сценария атаки

Сценарий #1

Торговая онлайн платформа для онлайн магазинов предоставляет страницу с графиками доходов для каждого размещенного на платформе магазина. Злоумышленник, проанализировав отправляемые браузером запросы, может найти точки входа API, предоставляющие данные для графиков и определить формат запросов к ним `/shops/{shopName}/revenue_data.json`. Используя другую точку входа API, злоумышленник может получить список названий всех магазинов, размещенных на платформе. Используя простой скрипт, заменяющий `{shopName}` в URL запроса на названия магазинов, злоумышленник может получить доступ к данным о продажах тысяч онлайн магазинов.

Сценарий #2

Злоумышленник анализирует сетевой трафик носимого устройства, и HTTP PATCHN запрос, содержащий нестандартный HTTP заголовок X-User-Id: 54796, привлекает его внимание. Изменив значение заголовка X-User-Id на 54795, злоумышленник получает ответ сервера, означающий успешную обработку запроса. Это означает, что злоумышленник может изменять данные учётных записей других пользователей.

Как предотвратить

- Надлежащим образом внедрить механизм авторизации, основывающийся на иерархии и политиках пользователей.
- Использовать механизм авторизации для проверки того, что текущий аутентифицированный пользователь имеет право доступа к запрошенному действию над записью в базе данных в каждой функции, использующей пользовательский ввод для доступа к записи.
- Использовать случайно сгенерированные значения, например, GUID в качестве идентификаторов записей в базе данных.
- Использовать тесты, проверяющие корректность работы механизма авторизации. Не пропускать уязвимые изменения, которые не проходят тесты.

Ссылки

Внешние

- [CWE-284: Improper Access Control](#)
- [CWE-285: Improper Authorization](#)
- [CWE-639: Authorization Bypass Through User-Controlled Key](#)

<div>Источники угроз</div> 		<div>Векторы атак</div>		<div>Недостатки безопасности</div>		<div>Последствия</div>	
<div>Зависит от API</div>	<div>Сложность эксплуатации: 3</div>	<div>Распространенность: 2</div>	<div>Сложность обнаружения: 2</div>	<div>Технические последствия: 3</div>	<div>Зависит от бизнеса</div>		
<div>Аутентификация в API - сложный и запутанный механизм. Разработчики и инженеры безопасности могут иметь неверное представление о том, что входит в понятие аутентификации, и как правильно её внедрить. Кроме того, механизм аутентификации - простая цель для злоумышленников, поскольку он общедоступен. Все это приводит к тому, что механизм аутентификации потенциально содержит большое число уязвимостей.</div>		<div>Механизм аутентификации подвержен двум основным проблемам: 1. Отсутствие механизмов защиты: разработчики должны обращать особое внимание на точки входа API, отвечающие за аутентификацию, и внедрять дополнительные уровни защиты. 2. Некорректная реализация механизма: механизм используется или реализован, не принимая во внимание основные векторы атак, или используется механизм, не подходящий под текущую ситуацию (например, механизм аутентификации для IoT устройств зачастую не подходит для веб приложений).</div>		<div>Злоумышленник может получить контроль над учетными записями других пользователей в системе, получить доступ к их персональным данным, или осуществить критичные действия от их имени, например, отправить денежные переводы или персональные сообщения.</div>			

Как определить, является ли API уязвимым?

Точки входа, отвечающие за аутентификацию и процесс аутентификации, - это активы, требующие защиты. Необходимо относиться к функционалу восстановления пароля аналогично механизму аутентификации.

API уязвим, если:

- Позволяет [перебор учётных данных](#), при условии, что у злоумышленника есть списки существующих логинов и паролей.
- Позволяет злоумышленнику подбирать пароль к одной и той же учетной записи путём перебора, не требуя ввода CAPTCHA или не блокируя учётную запись.
- Допускает слабые пароли.
- Передаёт критичные аутентификационные данные в URL, например, аутентификационные токены или пароли.
- Не проверяет подлинность токенов.
- Допускает JWT токены не содержащие подпись ("alg":"none") или использующие уязвимые алгоритмы подписи, не проверяет срок действия токена.
- Хранит пароли в открытом виде или в хэшированном виде с использованием слабых алгоритмов хеширования.
- Использует слабые ключи шифрования.

Примеры сценариев атаки

Сценарий #1

[Перебор учётных данных](#) с использованием [списка известных логинов и паролей](#) - распространённая атака. Если в приложении отсутствуют автоматизированные меры защиты от угроз или перебора учётных данных, то оно может быть использовано для определения валидности учётных данных.

Сценарий #2

Злоумышленник начинет восстановление пароля, отправив POST запрос в точку входа `/api/system/verification-codes` и указав имя пользователя в теле запроса. Затем одноразовый пароль из 6 цифр отправляется на телефон жертвы. Поскольку API не ограничивает количество запросов, злоумышленник может за несколько минут подобрать корректный одноразовый пароль, перебирая все возможные пароли с помощью скрипта, работающего в многопоточном режиме и отправляющего запросы на `/api/system/verification-codes/{smsToken}`.

Как предотвратить?

Идентифицируйте все возможные способы аутентификации в API (для мобильных и веб клиентов, deep links, обеспечивающих аутентификацию в одно нажатие, и так далее).

- Спросите у разработчиков, какие способы аутентификации вы пропустили.
- Изучите используемые механизмы аутентификации. Изучите, что они из себя представляют и как используются. OAuth не используется для аутентификации пользователей, так же как и API ключи.
- Не изобретайте велосипед, когда речь идёт об аутентификации, генерации токенов и хранении паролей. Используйте стандарты.
- Внедрите защиту от перебора, ограничение на количество одновременных запросов и временную блокировку учётных записей на точках входа, отвечающих за восстановление учётных данных и пароля, аналогично мерам защиты на точках входа, используемых для аутентификации.
- Ознакомьтесь с [OWASP Authentication Cheatsheet](#).
- Используйте многофакторную аутентификацию, где это возможно.
- Используйте механизмы защиты от перебора учётных данных, перебора по словарю и перебора всех возможных значений на точках входа, отвечающих за аутентификацию. Эти механизмы должны использовать более строгие правила по сравнению с механизмом ограничивающим количество запросов в остальных точках входа API.
- Внедрите [блокировку учётных записей](#) или CAPTCHA для предотвращения перебора аутентификационных данных, направленного на единичных пользователей. Внедрите защиту от слабых паролей.
- Не используйте API ключи для аутентификации пользователей. Они должны использоваться для [аутентификации приложений и проектов, являющихся клиентами API](#).

Ссылки

OWASP

- [OWASP Key Management Cheat Sheet](#)
- [OWASP Authentication Cheatsheet](#)
- [Credential Stuffing](#)

Внешние

- [CWE-798: Use of Hard-coded Credentials](#)

Зависит от API	Сложность эксплуатации: 3	Распространенность: 2	Сложность обнаружения: 2	Технические последствия: 2	Зависит от бизнеса
Предоставление излишних данных легко эксплуатировать. Для этого нужно перехватить трафик и проанализировать ответы API на предмет критичных данных, которые API не должно возвращать пользователю.		API рассчитывают, что клиентское приложение отфильтрует отображаемые пользователю данные. Поскольку API используются в качестве источника данных, иногда разработчики пытаются сделать его универсальным для всех клиентов, не думая о критичности возвращаемых пользователю данных. Автоматизированные инструменты зачастую не обнаруживают эту уязвимость, поскольку без детального понимания логики приложения очень трудно определить, должно ли API возвращать те или иные данные.		Предоставление излишних данных обычно приводит к разглашению конфиденциальных данных.	

Как определить, является ли API уязвимым?

API уязвим, если он спроектирован так, что возвращает критичные данные клиентскому приложению, которое в свою очередь фильтрует их перед отображением пользователю, то злоумышленник с лёгкостью может перехватить трафик и увидеть критичные данные.

Примеры сценариев атаки

Сценарий #1

Команда мобильного приложения использует точку входа `/api/articles/{articleId}/comments/{commentId}` для отображения метаданных комментариев в представлении статей. Злоумышленник перехватывает трафик от мобильного приложения и находит в ответе дополнительные критичные данные об авторе комментария. Точка входа реализована так, что использует стандартный метод `toJSON()` на объекте модели `User`, содержащем персональные данные, для сериализации этого объекта.

Сценарий #2

Система видеонаблюдения, базирующаяся на IoT, позволяет администраторам создавать пользователей с различными привилегиями. Администратор создал учётную запись для нового охранника, которая должна иметь доступ только к определенным зданиям на объекте. Когда охранник использует мобильное приложение, оно отправляет запрос в точку входа `/api/sites/111/cameras`, чтобы получить данные о доступных камерах и отобразить их на панели управления. Ответ API содержит список данных о камерах в следующем формате: `{"id":"xxx","live_access_token":"xxxx-bbbbb","building_id":"yyy"}`. Клиентское приложение показывает только те камеры, к которым охранник имеет доступ, однако ответ API содержит полный список камер на объекте.

Как предотвратить

- Не рассчитывайте, что клиентская часть приложения отфильтрует критичные данные.
- Проверьте, что ответы API содержат только те данные, которые отображаются клиентским приложением.
- Разработчики серверной части должны всегда задаваться вопросом "Кто получит данные?" перед публикацией новых точек входа API.
- Избегайте использования стандартных методов, например, `to_json()` или `to_string()`. Вместо этого вручную выбирайте свойства объектов, которые вы возвращаете в ответе.
- Классифицируйте критичную информацию и персональные данные, с которыми работает приложение, путём анализа всех вызовов API, возвращающих подобные данные, чтобы определить, несут ли эти ответы риски безопасности.
- Внедрите механизм валидации, базирующийся на проверке данных по схеме, в качестве дополнительного уровня защиты. В рамках этого механизма определите данные возвращаемые каждой точкой входа (в том числе в ошибках) и обеспечьте, что только эти данные возвращаются пользователю.

Ссылки

Внешние

- [CWE-213: Intentional Information Exposure](#)

Источники угроз		Векторы атак		Недостатки безопасности		Последствия	
Зависит от API	Сложность эксплуатации: 2	Распространенность: 3	Сложность обнаружения: 3	Технические последствия: 2	Зависит от бизнеса		
Для эксплуатации необходимы простые запросы к API. Аутентификация не требуется. Злоумышленник может одновременно отправить большое количество запросов, используя локальный компьютер или облачную систему вычислений.		Часто API не ограничивают количество запросов, или эти ограничения настроены некорректно.		Эксплуатация может привести к отказу в обслуживании, выражающемся в долгом времени ответа или полной недоступности API.			

Как определить, является ли API уязвимым?

Запросы к API потребляют ресурсы, например, пропускную способность канала, процессорное время, оперативную память и место в хранилище данных. Количество ресурсов, потребляемых для ответа на запрос к API, во многом зависит от пользовательского ввода и бизнес логики точки входа. Кроме того, нужно принимать во внимание то, что запросы от различных клиентов API используют ресурсы совместно. API уязвимо, если хотя бы одно из следующих ограничений отсутствует или имеет некорректное значение (например, слишком высокое или низкое):

- Максимальное время ожидания выполнения
- Максимальный объем выделяемой памяти
- Количество файловых дескрипторов
- Количество процессов
- Размер полезной нагрузки запроса (например, размер загружаемого файла)
- Количество запросов на одного клиента или ресурс
- Количество записей из базы данных, возвращаемых в ответе на один запрос

Примеры сценариев атаки

Сценарий #1

Злоумышленник загружает большое изображение, отправив POST запрос на `/api/v1/images`. После завершения загрузки, API создаёт миниатюры изображения разного размера. Во время создания миниатюр приложение использует всю доступную память и перестаёт отвечать на запросы из-за большого размера загруженного изображения.

Сценарий #2

Рассмотрим приложение, отображающее список пользователей в пользовательском интерфейсе с ограничением 200 штук на страницу. Для получения списка пользователей приложение отправляет следующий запрос на сервер: `/api/users?page=1&size=200`. Злоумышленник увеличивает `size` до 200 000, что приводит к проблемам производительности в базе данных. API перестаёт отвечать на запросы и больше не может обработать запросы текущего или любого другого клиента (отказ в обслуживании).

Отсутствие Ограничений на Количество Запросов и Потребляемые Ресурсы

Аналогичный сценарий может быть использован для обнаружения ошибок переполнения буфера или целочисленного переполнения.

Как предотвратить

- Docker легко позволяет ограничить [объем памяти](#), [процессорное время](#), [количество перезапусков](#), [файловые дескрипторы](#) и [процессы](#).
- Установите ограничение на частоту вызовов метода API одним клиентом в заданный промежуток времени.
- Уведомите клиента, когда ограничение превышено, предоставив ему значение ограничения и время, когда ограничение будет сброшено.
- Добавьте соответствующие проверки параметров строки запроса и тела запроса на стороне сервера, особенно важно контролировать количество записей, возвращаемых в запросе.
- Определите и контролируйте максимальный размер данных, содержащихся во всех входных параметрах и полезных нагрузках, например, максимальную длину строки или максимальное количество элементов массива.

Ссылки

OWASP

- [Blocking Brute Force Attacks](#)
- [Docker Cheat Sheet - Limit resources \(memory, CPU, file descriptors, processes, restarts\)](#)
- [REST Assessment Cheat Sheet](#)

Внешние

- [CWE-307: Improper Restriction of Excessive Authentication Attempts](#)
- [CWE-770: Allocation of Resources Without Limits or Throttling](#)
- “Rate Limiting (Throttling)” - [Security Strategies for Microservices-based Application Systems](#), NIST

Зависит от API	Сложность эксплуатации: 3	Распространенность: 2	Сложность обнаружения: 1	Технические последствия: 2	Зависит от бизнеса
<p>Эксплуатация уязвимости предполагает, что злоумышленник может успешно отправить запросы в точки входа API, к которым у него не должно быть доступа. Эти точки входа могут быть доступны любому неаутентифицированному пользователю или аутентифицированному пользователю, не имеющему достаточных привилегий. Подобную ошибку легче обнаружить в API (по сравнению с традиционными приложениями), поскольку API более структурированы, а порядок доступа к определенным функциям более предсказуем (например, изменив метод HTTP запроса с GET на PUT, или изменив строку "users" в URL запроса на "admins").</p>		<p>Проверки авторизации к функции или ресурсу обычно определяются на уровне конфигурации, иногда на уровне кода. Проведение проверок надлежащим образом - непростая и неоднозначная задача, поскольку современные приложения могут использовать много типов ролей и групп, а также иметь сложную иерархию пользователей (например, под-пользователей или пользователей с несколькими ролями).</p>		<p>Подобные ошибки позволяют злоумышленнику получить доступ к функционалу, минуя авторизацию. Административный функционал - ключевая цель атак этого типа.</p>	

Как определить, является ли API уязвимым?

Лучший способ найти проблемы с некорректной авторизацией на уровне объектов - провести глубокий анализ механизма авторизации, учитывая иерархию пользователей, различные роли и группы внутри приложения, а также задав себе следующие вопросы:

- Может ли обычный пользователь получить доступ к административным точкам входа?
- Может ли пользователь совершить критичные действия (например, создать, изменить или удалить объект), к которым у него не должно быть доступа, просто изменив метод HTTP запроса (например, с GET на DELETE)?
- Может ли пользователь из группы X получить доступ к точке входа, доступной только пользователям из группы Y, просто угадав URL и параметры этой точки входа (например, /api/v1/users/export_all)?

Неверно предполагать, что точка входа API является обычной или административной только на основании пути URL.

Зачастую разработчики открывают доступ к административным точкам входа по определённому относительному пути, например, api/admins. Однако очень часто административные точки входа находятся по другим относительным путям вместе с обычными точками входа, например, api/users.

Примеры сценариев атаки

Сценарий #1

В ходе процесса регистрации в приложении, которое позволяет регистрироваться только приглашённым пользователям, мобильное приложение отправляет следующий запрос к API GET /api/invites/{invite_guid}. Ответ содержит JSON с деталями приглашения, включая роль пользователя и его электронную почту.

Злоумышленник может дублировать запрос, изменив HTTP метод и точку входа на POST /api/invites/new. Только администраторы должны иметь доступ к этой точке входа через интерфейс администрирования, однако он не проводит проверки авторизации на уровне функций.

Проексплуатируя уязвимость, злоумышленник может отправить себе приглашение с ролью администратора:

```
POST /api/invites/new
{"email": "hugo@malicious.com", "role": "admin"}
```

Сценарий #2

API содержит точку входа, которая должна быть доступна только администраторам GET /api/admin/v1/users/all. Эта точка входа возвращает данные всех пользователей и не проводит проверки авторизации на уровне функции. Злоумышленник, изучив структуру API, подбирает URL и получает доступ к точке входа, которая возвращает критичные данные пользователей приложения.

Как предотвратить

В вашем приложении должен быть согласованный и легко анализируемый модуль авторизации, вызываемый всеми бизнес функциями. Зачастую такая защита предоставляется одной или несколькими компонентами вне кода приложения.

- Механизм, обеспечивающий выполнение проверок авторизации, должен запрещать весь доступ по умолчанию и требовать наличия определённых ролей для доступа к каждой из функций.
- Проверьте все точки входа API на предмет некорректной авторизации на уровне функций, принимая во внимание бизнес логику приложения и иерархию групп.
- Убедитесь, что все административные контроллеры наследуют абстрактный административный контроллер, в котором реализованы проверки авторизации на базе пользовательских групп и ролей.
- Убедитесь, что административные функции внутри обычных контроллеров проводят проверки авторизации на базе пользовательских групп и ролей.

Ссылки

OWASP

- [OWASP Article on Forced Browsing](#)
- [OWASP Top 10 2013-A7-Missing Function Level Access Control](#)
- [OWASP Development Guide: Chapter on Authorization](#)

Внешние

- [CWE-285: Improper Authorization](#)

Источники угроз		Векторы атак		Недостатки безопасности		Последствия	
Зависит от API	Сложность эксплуатации: 2	Распространенность: 2	Сложность обнаружения: 2	Технические последствия: 2	Зависит от бизнеса		
Для эксплуатации зачастую требуется понимание бизнес логики, связей между объектами и структуры API. Эксплуатация массового переназначения параметров проще реализуема в API, поскольку они изначально предусматривают общедоступность внутренней реализации API и названий свойств объектов.		Современные фреймворки предлагают разработчикам функции, которые автоматически присваивают переменным и внутренним объектам значения соответствующих параметров из пользовательского ввода. Злоумышленник может использовать эту методологию, чтобы обновить или переназначить критичные свойства объектов, которые разработчик не намеревался делать доступными для пользователя.		Эксплуатация уязвимости может привести к повышению привилегий, злонамеренному изменению данных, обходу механизмов защиты и так далее.			

Как определить, является ли API уязвимым?

Объекты в современных приложениях могут иметь большое количество свойств. Некоторые из них могут быть изменены напрямую клиентом (например, `user.first_name` или `user.address`), в то время как изменение других не должно быть доступно (например, флаг `user.is_vip`).

Конечная точка API уязвима, если она автоматически присваивает предоставленные клиентом параметры свойствам внутренних объектов, не учитывая критичность и уровень доступности этих свойств. Это может позволить злоумышленнику изменить свойства объектов, к которым у него не должно быть доступа.

Примеры критичных свойств:

- **Свойства, относящиеся к привилегиям:** `user.is_admin`, `user.is_vip` должны устанавливаться только администраторами.
- **Свойства, зависящие от процесса:** `user.cash` должны устанавливаться только внутри кода после проверки платежа.
- **Внутренние свойства:** `article.created_time` должны устанавливаться только внутри кода самим приложением.

Примеры сценариев атаки

Сценарий #1

Приложение для совместных поездок позволяет пользователю редактировать базовую информацию своего профиля. В ходе редактирования отправляется следующий запрос `PUT /api/v1/users/me` с корректным JSON объектом в теле запроса:

```
{"user_name":"inons","age":24}
```

Запрос к `GET /api/v1/users/me` включает в себя дополнительное свойство `credit_balance`:

```
{"user_name":"inons","age":24,"credit_balance":10}.
```


Злоумышленник дублирует первый запрос со следующим телом запроса:

```
{"user_name":"attacker","age":60,"credit_balance":99999}
```

Поскольку точка входа уязвима к массовому переназначению параметров, злоумышленник зачисляет деньги на свой баланс, не совершив платежа.

Сценарий #2

Портал для обмена видео позволяет пользователям загружать и скачивать материалы в разных форматах. Злоумышленник исследует API и обнаруживает, что точка входа GET /api/v1/videos/{video_id}/meta_data возвращает JSON объект с параметрами видео. Один из параметров "mp4_conversion_params":"-v codec h264" даёт понять, что приложение использует консольную команду для конвертации видео.

Злоумышленник также обнаружил, что точка входа POST /api/v1/videos/new уязвима к массовому переназначению параметров и позволяет клиенту установить значение любого свойства объекта видео. Злоумышленник устанавливает следующее значение параметра: "mp4_conversion_params":"-v codec h264 && format C:". Это значение приведёт к инъекции команды операционной системы, как только злоумышленник скачает видео в формате MP4.

Как предотвратить

- Если возможно, избегайте использования функций, которые автоматически присваивают переменным и внутренним объектам соответствующие значения из пользовательского ввода.
- Добавляйте в белые списки только те свойства, которые могут быть изменены клиентом.
- Используйте встроенный функционал по добавлению в чёрный список свойств, к которым клиенты не могут иметь доступ.
- Если это возможно, явно определите схемы входящих данных и проверяйте входящие данные по ним.

Ссылки

Внешние

- [CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes](#)

Источники угроз		Векторы атак		Недостатки безопасности		Последствия	
Зависит от API	Сложность эксплуатации: 3	Распространённость: 3	Сложность обнаружения: 3	Технические последствия: 2	Зависит от бизнеса		
Злоумышленники часто пытаются найти незакрытые уязвимости, распространённые точки входа или незащищённые файлы и папки, чтобы получить информацию о системе или неавторизованный доступ к ней.		Ошибка настроек безопасности может произойти на любом уровне API: от сетевого уровня до уровня приложения. Существуют автоматизированные инструменты для обнаружения и эксплуатации таких ошибок конфигурации, как ненужные (забытые) сервисы или использование устаревших параметров.		Ошибки настроек безопасности могут не только раскрыть конфиденциальные данные пользователей, но и данные о системе, что потенциально может привести к её полной компрометации.			

Как определить, является ли API уязвимым?

API уязвим, если:

- Должные настройки безопасности отсутствуют на каком-либо уровне приложения, а также если права доступа к облачным сервисам некорректно настроены.
- Используется устаревшая система, или не установлены новейшие исправления по безопасности.
- Активен излишний функционал (например, неиспользуемые HTTP методы).
- Не используется протокол TLS (Transport Layer Security).
- Директивы безопасности не отправляются клиентским приложениям (например, [Заголовки Безопасности](#)).
- Политика разделения ресурсов между источниками (Cross-Origin Resource Sharing) отсутствует или некорректно настроена.
- Сообщения об ошибках включают детальную информацию или раскрывают критичные данные.

Примеры сценариев атаки

Сценарий #1

Злоумышленник в корне директории сервера находит файл .bash_history, содержащий команды, которые использовала команда DevOps для доступа к API:

```
$ curl -X GET 'https://api.server/endpoint/' -H 'authorization: Basic Zm9vOmJhcg=='
```

Злоумышленник также может найти другие незадокументированные точки входа API, используемые только командой DevOps.

Сценарий #2

Для атаки на конкретный сервис злоумышленник использует популярный поисковик, чтобы найти компьютеры, напрямую доступные из сети Интернет. Злоумышленник находит сервер, на котором запущена популярная система управления базой данных, доступная на стандартном порте. На сервере используется стандартная конфигурация, не предполагающая аутентификации, что позволяет злоумышленнику получить доступ к миллионам записей с персональными данными, личными предпочтениями и аутентификационными данными.

Сценарий #3

Анализируя трафик мобильного приложения, злоумышленник обнаруживает, что не весь HTTP трафик защищён (например, с помощью TLS). В частности, не защищено скачивание изображений профиля. Поскольку взаимодействие пользователя с приложением бинарно (да или нет, свайп влево или вправо, и так далее), несмотря на шифрование трафика, злоумышленник может найти закономерности в

параметрах ответов API (например, размер ответа на свайп влево больше, чем на свайп вправо), которые он в свою очередь может использовать для отслеживания действий и предпочтений пользователя.

Как предотвратить

Жизненный цикл API должен включать в себя:

- Повторяемый процесс усиления настроек безопасности, ведущий к более быстрому и простому развёртыванию должным образом защищённого окружения.
- Задачу по обзору и обновлению конфигурации на всех уровнях API. Обзор должен включать в себя файлы оркестрации, компоненты API и облачных сервисов (например, права доступа в S3 bucket).
- Защищённый канал связи при доступе к статическим ресурсам.
- Автоматизированный процесс, проводящий постоянную оценку эффективности настроек и параметров во всех окружениях.

Кроме того:

- Для предотвращения отправки злоумышленникам подробных сообщений об ошибках и другой критичной информации, если это возможно, определите схемы данных всех ответов API и обеспечьте проверку этих ответов по схемам, включая сообщения об ошибках.
- Убедитесь, что API доступно только с использованием заданного списка HTTP методов. Любые другие методы HTTP должны быть отключены (например, HEAD).
- API, клиентами которых подразумеваются браузерные клиентские приложения, должны иметь корректно настроенную политику разделения ресурсов между источниками (Cross-Origin Resource Sharing).

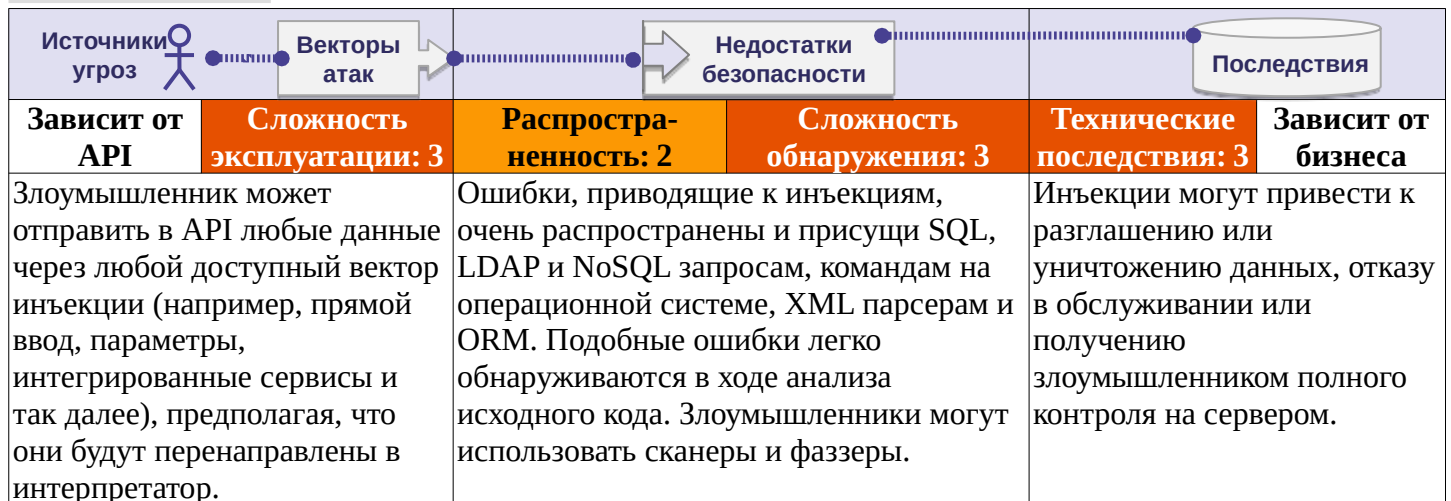
Ссылки

OWASP

- [OWASP Secure Headers Project](#)
- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Testing Guide: Test Cross Origin Resource Sharing](#)

Внешние

- [CWE-2: Environmental Security Flaws](#)
- [CWE-16: Configuration](#)
- [CWE-388: Error Handling](#)
- [Guide to General Server Security](#), NIST
- [Let's Encrypt: a free, automated, and open Certificate Authority](#)



Как определить, является ли API уязвимым?

API уязвим к инъекциям, если:

- Данные, поступившие от пользователя, не валидируются, не фильтруются или не очищаются на стороне API.
- Данные, поступившие от пользователя, конкатенируются или используются в неизменном виде в SQL/NoSQL/LDAP запросах, командах на операционной системе, XML парсерах, ORM (Object Relational Mapping) или ODM (Object Document Mapper).
- Данные поступающие из внешних систем (например, интегрированных систем) не валидируются, не фильтруются или не очищаются на стороне API.

Примеры сценариев атаки

Сценарий #1

Прошивка устройства контроля за детьми предоставляет точку входа /api/CONFIG/restore, которая ожидает запроса с multipart параметром appId. Используя декомпилятор, злоумышленник обнаруживает, что appId передаётся непосредственно в вызов команды на операционной системе без предварительной очистки:

```
snprintf(cmd, 128, "%srestore_backup.sh /tmp/postfile.bin %s %d",
        "/mnt/shares/usr/bin/scripts/", appid, 66);
system(cmd);
```

Следующая команда позволяет злоумышленнику отключить любое устройство с уязвимой прошивкой:

```
$ curl -k "https://${deviceIP}:4567/api/CONFIG/restore" -F
'appid=$(/etc/pod/power_down.sh)'
```

Сценарий #2

Рассмотрим приложение с базовым CRUD функционалом для операций с бронированиями. Злоумышленник обнаружил NoSQL инъекцию через параметр bookingId в запросе на удаление бронирования. Запрос выглядит следующим образом: DELETE /api/bookings?bookingId=678.

Сервер API обрабатывает запросы на удаление с помощью следующей функции:

```
router.delete('/bookings', async function (req, res, next) {
  try {
    const deletedBooking = await Bookings.findOneAndRemove({_id' : req.query.bookingId});
    res.status(200);
  } catch (err) {
    res.status(400).json({
      error: 'Unexpected error occured while processing a request'
    });
  }
});
```

Злоумышленник перехватывает запрос и изменяет параметр bookingId, как продемонстрировано ниже. В этом случае злоумышленник может удалить бронирование, принадлежащее другому пользователю:

```
DELETE /api/bookings?bookingId[$ne]=678
```

Как предотвратить

Для предотвращения инъекций необходимо отделять данные от команд и запросов.

- Валидируйте данные, используя одну доверенную и активно поддерживаемую библиотеку.
- Валидируйте, фильтруйте и очищайте все данные, получаемые от клиентов или интегрированных систем.
- Специальные символы должны быть экранированы, используя синтаксис целевого интерпретатора.
- Отдайте предпочтение безопасному API, предоставляющему параметризованный интерфейс.
- Всегда ограничивайте количество возвращаемых записей, чтобы предотвратить массовую утечку данных в случае инъекции.
- Валидируйте входящие данные с помощью надлежащих фильтров, допуская только подходящие значения каждого из входящих параметров.
- Определите тип данных и строгую модель данных для всех строковых параметров.

Ссылки

OWASP

- [OWASP Injection Flaws](#)
- [SQL Injection](#)
- [NoSQL Injection Fun with Objects and Arrays](#)
- [Command Injection](#)

Внешние

- [CWE-77: Command Injection](#)
- [CWE-89: SQL Injection](#)

Источники угроз		Векторы атак		Недостатки безопасности		Последствия	
Зависит от API	Сложность эксплуатации: 3	Распространенность: 3	Сложность обнаружения: 2	Технические последствия: 2	Зависит от бизнеса		
Старые версии API обычно не содержат всех исправлений и могут быть с лёгкостью скомпрометированы без необходимости обходить новейшие механизмы безопасности, которые с высокой вероятностью используются для защиты последних версий API.		Устаревшая документация усложняет поиск и исправление уязвимостей. Отсутствие инвентаризации активов и стратегии вывода из эксплуатации приводит к функционированию не обновленных систем, которые могут быть использованы для кражи критичных данных. При использования современных подходов типа микросервисов, которые позволяют легко разворачивать независимые приложения (например, облачные сервисы или kubernetes) хосты API (домены и серверы, на которых функционирует API) зачастую без необходимости доступны извне.		Злоумышленник может получить доступ к критичным данным или даже получить контроль над сервером через старую, не обновленную версию API, использующую одну и ту же базу данных.			

Как определить, является ли API уязвимым?

API может быть уязвимым, если:

- Назначение API хоста неясно, а также нет чётких ответов на следующие вопросы:
 - В каком окружении запущен API (например, production, staging, test, development)?
 - Каким должен быть сетевой доступ к API (например, общедоступным, внутренним, для партнёров)?
 - Какая версия API запущена?
 - Какие данные собираются и обрабатываются API (например, персональные данные)?
 - Каков поток движения данных?
- Документация отсутствует или не обновляется.
- Отсутствует план вывода из эксплуатации предыдущих версий API.
- Инвентаризация хостов не проводится, или её результаты устарели.
- Инвентаризация интегрированных внутренних или сторонних сервисов не проводится, или её результаты устарели.
- Старые или предыдущие версии API функционируют без обновлений.

Примеры сценариев атаки

Сценарий #1

После переработки своих приложений локальный поисковый сервис оставил доступ к старой версии API (api.someservice.com/v1) без надлежащих мер защиты и с доступом к базе данных пользователей. Тестируя одну из последних выпущенных версий приложения, злоумышленник нашёл адрес API (api.someservice.com/v2). Заменив v2 на v1 в URL, злоумышленник получил доступ к старому незащищённому API, предоставляющему доступ к персональным данным более 100 миллионов пользователей.

Сценарий #2

Социальная сеть внедрила механизм ограничения количества запросов, не позволяющий злоумышленнику подобрать токен для сброса пароля. Этот механизм не был внедрён непосредственно в код API, а использовался в качестве отдельного компонента между клиентом и официальным API (www.socialnetwork.com). Исследователь нашёл бета версию API (www.mbasic.beta.socialnetwork.com), использующую тот же API, включая механизм сброса пароля, но без механизма ограничения количества запросов. Исследователь смог сбросить пароль любого пользователя, перебирая все возможные варианты кода из 6 цифр.

Как предотвратить




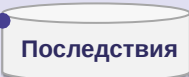
- Проводите инвентаризацию хостов API и документируйте важные аспекты каждого из них: окружение (например, production, staging, test, development), каким должен быть сетевой доступ (например, общедоступным, внутренним, для партнёров) и версию API.
- Проводите инвентаризацию интегрированных сервисов и документируйте важные аспекты каждого из них: роль в системе, какие данные участвуют в обмене (поток данных), какая степень критичности этих данных.
- Документируйте все аспекты API: аутентификацию, ошибки, перенаправления, ограничение количества запросов, политику разделения ресурсов между источниками (CORS) и точки входа, включая параметры, запросы и ответы.
- Создавайте документацию автоматически, используя общедоступные стандарты. Включайте создание документации в CI/CD.
- Предоставьте документацию тем, кто имеет право доступа к API.
- Используйте внешние меры защиты, например, API security firewalls на всех доступных версиях API, а не только на текущей версии в production.
- Избегайте использования данных с production системы в API на базе не production окружения. Если такого использования невозможно избежать, защищайте этот API аналогично используемым в production.
- Когда новая версия API включает улучшения, связанные с безопасностью, проводите анализ рисков для принятия решения о действиях по снижению рисков в старых версиях, например, переносу улучшения в старые версии без нарушения совместимости, или отключению старой версии и переводу всех клиентов на новую.

Ссылки

Внешние

- [CWE-1059: Incomplete Documentation](#)
- [OpenAPI Initiative](#)

API10:2019 Недостаточное Логирование и Мониторинг

Источники угроз 		Векторы атак 		Недостатки безопасности 		Последствия 	
Зависит от API	Сложность эксплуатации: 2	Распространенность: 3	Сложность обнаружения: 1	Технические последствия: 2	Зависит от бизнеса		
Злоумышленник пользуется отсутствием логирования и мониторинга для незаметной эксплуатации уязвимостей системы.		Отсутствующие или недостаточные логирование и мониторинг не позволяют отследить подозрительную активность и своевременно отреагировать на неё.		Без наблюдения за происходящей подозрительной активностью у злоумышленника есть достаточно времени для полной компрометации системы.			

Как определить, является ли API уязвимым?

API уязвим, если:

- Не пишутся логи, уровень логирования некорректно установлен, или сообщения в логах недостаточно детальны.
- Не обеспечивается целостность логов (например, [Инъекция в логи](#)).
- Логи не подвергаются постоянному мониторингу.
- API не подвергается постоянному мониторингу.

Примеры сценариев атаки

Сценарий #1

Ключ доступа к административному API утекли через общедоступный репозиторий. Владелец репозитория был уведомлен о потенциальной утечке по электронной почте, но отреагировал на инцидент более чем через 48 часов, в связи с чем утечка ключей могла привести к получению доступа к критичным данным. Из-за недостаточного логирования компания не в состоянии оценить, к каким данным злоумышленники смогли получить доступ.

Сценарий #2

Платформа обмена видео подверглась масштабной атаке по перебору учётных данных. Не смотря на логирование неуспешных попыток входа, уведомление об атаке не последовало в течение всего хода атаки. Логи были проанализированы и атака обнаружена только во время анализа обращения пользователя. Компании пришлось публично попросить пользователей сменить пароли и отправить отчёт об атаке в регулирующие органы.

Как предотвратить

- Логируйте все неудачные попытки входа, отказы в доступе и ошибки валидации входящих данных.
- Логи должны быть представлены в формате, позволяющем обрабатывать их с помощью систем управления логами, и должны включать достаточное количество деталей, позволяющих идентифицировать злоумышленника.
- Логи должны считаться критичными данными, а их целостность должна быть обеспечена при передаче и хранении.
- Настройте систему мониторинга для постоянного контроля инфраструктуры, сети и функционирующих API.
- Используйте систему управления информацией и событиями безопасности (SIEM), чтобы агрегировать и управлять логами всех компонентов на всех уровнях и хостах API.
- Настройте персональные уведомления и панели управления для скорейшего обнаружения и реагирования на подозрительную активность.

Ссылки

OWASP

- [OWASP Logging Cheat Sheet](#)
- [OWASP Proactive Controls: Implement Logging and Intrusion Detection](#)
- [OWASP Application Security Verification Standard: V7: Error Handling and Logging Verification Requirements](#)

Внешние

- [CWE-223: Omission of Security-relevant Information](#)
- [CWE-778: Insufficient Logging](#)

Дальнейшие шаги для разработчиков

Задача по созданию и поддержке ПО в безопасном состоянии, или исправлению уже существующего ПО может быть сложной. То же верно и для API.

Мы уверены, что обучение и осведомлённость являются ключевыми факторами написания безопасного ПО. Все остальное, необходимое для достижения этой цели, зависит от **налаженного и повторяемого процесса безопасности и стандартных мер защиты**.

OWASP включает большое количество бесплатных открытых ресурсов по безопасности с самого начала проекта. Пожалуйста ознакомьтесь со [страницей OWASP Projects](#), чтобы получить полный список доступных проектов.

Обучение	Вы можете начать изучать материалы OWASP Education Project , в соответствии со своей профессией и интересами. Для получения практических знаний мы добавили crAPI – Completely Ridiculous API в наш план развития . Тем временем вы можете попрактиковаться в безопасности веб приложений, используя OWASP DevSlop Pixi Module - уязвимое веб приложение с API сервисом, направленное на обучение пользователей тестированию современных веб приложений и API на предмет ошибок безопасности и написанию более безопасных API в будущем. Вы также можете принять участие в практических сессиях конференции OWASP AppSec или присоединиться в вашему локальному отделению OWASP .
Требования по безопасности	Безопасность должна быть частью любого проекта с самого начала. На этапе сбора требований необходимо определить роль безопасности в проекте. OWASP рекомендует использовать OWASP Application Security Verification Standard (ASVS) в качестве руководства по постановке требований безопасности. Если вы пользуетесь услугами внешних разработчиков, рассмотрите проект OWASP Secure Software Contract Annex и адаптируйте его в соответствии с локальными законами и нормативными требованиями.
Безопасная архитектура	Безопасность должна получить должное внимание на всех этапах проекта. OWASP Prevention Cheat Sheets - хорошая отправная точка по проектированию механизмов безопасности на этапе архитектуры приложения. Среди прочего вы можете ознакомиться с REST Security Cheat Sheet и REST Assessment Cheat Sheet .
Стандартные меры безопасности	Внедрение стандартных мер безопасности снижает риск ошибок, связанных с безопасностью, в ходе написания логики приложения. Несмотря на то, что многие современные фреймворки включают в себя стандартные меры безопасности, OWASP Proactive Controls предоставляет хороший обзор мер безопасности, которые стоит включить в ваш проект. OWASP также предоставляет библиотеки и инструменты, которые могут вам пригодиться, например, механизмы валидации.
Жизненный цикл разработки безопасного ПО	Вы можете использовать OWASP Software Assurance Maturity Model (SAMM) для улучшения процессов создания API. Несколько других проектов OWASP могут помочь вам на различных этапах разработки API, например, OWASP Code Review Project .

Дальнейшие шаги для DevSecOps

Создание безопасных API критически важно из-за их роли в архитектуре современных приложений. Безопасностью нельзя пренебрегать, она должна быть частью всего жизненного цикла разработки. Уже недостаточно проводить сканирование и тестирование на проникновение раз в год.

DevSecOps должны присоединиться к работе по разработке и обеспечить непрерывное тестирование безопасности на всем жизненном цикле разработки ПО. Их цель - усовершенствовать процесс разработки, автоматизировав проверки безопасности, не влияя на скорость разработки.

Если вы сомневаетесь, ознакомьтесь с [DevSecOps Manifesto](#) и следите за его обновлениями.

Понимание модели угроз	Приоритеты при тестировании исходят из модели угроз. Если у вас её нет, рассмотрите использование OWASP Application Security Verification Standard (ASVS) и OWASP Testing Guide для начала её составления. Включение команды разработки может помочь им быть более осведомленными в сфере безопасности.
Понимание жизненного цикла разработки ПО	Объединитесь с командой разработки для лучшего понимания жизненного цикла разработки ПО. Ваше содействие в части непрерывного тестирования безопасности должно гармонизировать с людьми, процессами и инструментами. Все должны быть согласны с процессом для предотвращения ненужных разногласий и сопротивления.
Стратегии тестирования	Поскольку ваша работа не должна влиять на скорость разработки, вам необходимо продуманно выбрать наилучшие (простые, быстрые и наиболее точные) подходы к проверке требований по безопасности. OWASP Security Knowledge Framework и OWASP Application Security Verification Standard - отличные источники функциональных и нефункциональных требований по безопасности. Существуют также другие отличные проекты и инструменты , аналогичные предлагаемому сообществом DevSecOps .
Достижение покрытия и точности	Вы соединяете команды разработки и эксплуатации. Чтобы достичь покрытия нужно сфокусироваться не только на функциональности, но и на оркестрации. Работайте совместно с командами разработки и эксплуатации с самого начала, чтобы оптимизировать свои трудозатраты. Вы должны стремиться к состоянию, когда основы безопасности непрерывно проверяются.
Чётко описывайте найденные уязвимости	Приносите пользу, избегая конфликтов. Передавайте команде разработки информацию о найденных уязвимостях своевременно, с помощью инструментов, используемых командой разработки. Помогите команде разработки разобрать найденные уязвимости. Постарайтесь обучить их, чётко объяснив выявленные недостатки и процесс их эксплуатации, включая реальный сценарий атаки.

Обзор

Поскольку индустрия AppSec не сфокусирована главным образом на новейшей архитектуре приложений, в которой API играют важную роль, составление списка из десяти наиболее критичных рисков безопасности API на базе публичного сбора данных - сложная задача. Несмотря на отсутствие публичного сбора данных, получившийся список все равно основан на общедоступной информации, вкладе экспертов по безопасности и открытых дискуссиях сообщества по безопасности.

Методология

На первом этапе группа экспертов по безопасности собрала, проанализировала и категоризировала публичные данные об инцидентах безопасности в API. Эти данные были собраны с площадок bug bounty площадок и баз данных уязвимостей за последний год. Этот временной промежуток выбран для удобства расчёта статистики.

Затем эксперты по безопасности с опытом тестирования на проникновение составили свой список 10-ти наиболее критичных рисков.

[Методология ранжирования рисков OWASP](#) была использована в ходе анализа рисков. Эксперты по безопасности проанализировали результаты. Чтобы узнать о рассуждениях на эту тему обратитесь к секции [Риски безопасности API](#).

Первый черновик OWASP API Security Top 10 2019 был составлен на базе статистических результатов первого этапа и списков, созданных экспертами по безопасности. Этот черновик был отправлен для проверки и анализа другой группе экспертов по безопасности с достаточным опытом в сфере безопасности API.

OWASP API Security Top 10 2019 впервые был представлен на конференции OWASP Global AppSec Tel Aviv в мае 2019 года. С тех пор он доступен на GitHub для публичного обсуждения и содействия.

Список участников, внёсших свой вклад, доступен в секции [Благодарность](#).



Благодарность

Благодарность участникам

Мы хотели бы поблагодарить всех участников, кто публично участвовал в проекте на GitHub или другими способами:

- 007divyachawla
- Abid Khan
- Adam Fisher
- anotherik
- bkimminich
- caseysoftware
- Chris Westphal
- dsopas
- DSotnikov
- emilva
- ErezYalon
- flascelles
- Guillaume Benats
- IgorSasovets
- Inonshk
- JonnySchnittger
- jmanico
- jmdx
- Keith Casey
- kozmic
- LauraRosePorter
- Matthieu Estrade
- nathanawmk
- PauloASilva
- pentagramz
- philippederyck
- pleothaud
- r00ter
- Raj kumar
- Sagar Popat
- Stephen Gates
- thomaskonrad
- xycloops123