Уязвимости веб-приложений возникают тогда, когда разработчики добавляют небезопасный код в веб-приложение. Это может происходить как на этапе разработки, так и на этапе доработки или исправления найденных ранее уязвимостей. Недостатки часто классифицируются по степени критичности и их распространенности.

# Инъекции (Внедрение/Injection)

Уязвимости, связанные с внедрением SQL, NoSQL, OS и LDAP скриптов в приложению.

**SQL-injection** – злоумышленник использует ввод данных, который позволяет выполнить вредоносные SQL-запросы к бд.



**Command Injection –** Злоумышленник внедряет вредоносные команды в системные вызовы или команды оболочки операционной системы. Это может привести к выполнению нежелательных операций на сервере, таких как удаление файлов или получение несанкционированного доступа к системе.

**LDAP Injection**: Злоумышленник внедряет вредоносный код в LDAP-запросы, используемые для поиска и извлечения информации из каталогов.

Уязвимо если:

* вводимые пользователем данные не проверяются, не фильтруются или не очищаются;
* В коде используются динамические запросы и непараметризированные вызовы без экранирования.
* вредоносные данные используются в поисковых параметрах объектно-реляционного отображения для извлечения дополнительной, критичной информации;

Для защиты:

* Используйте параметризованные запросы и подготавливаемые выражения для предотвращения SQL-инъекций.
* Валидируйте входные данные на серверной стороне.
* Экранируйте специальные символы, чтобы предотвратить инъекции (HTML, JavaScript, SQL, XML и т. д.)  
  Примечание: элементы SQL названия таблиц или столбцов, нельзя экранировать, поэтому предоставляемые пользователями названия представляют опасность. Это обычная проблема программ для составления отчетов.
* Используйте безопасный API, исключающий применение интерпретатора или предоставляющий параметризованный интерфейс, либо используйте инструменты объектно-реляционного отображения (ORM).
* Включите механизмы мониторинга и регистрации, чтобы обнаружить и отследить попытки SQL-инъекций.

**Примеры сценариев атак**

**Сценарий №1** Приложение использует недоверенные данные при создании следующего уязвимого SQL-вызова:

String query = "SELECT \* FROM accounts WHERE custID='" + request.getParameter("id") + "'";

**Сценарий №2** Безоговорочное доверие приложений к фреймворкам

может привести к появлению уязвимых запросов (например, в

языке запросов HQL):

Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");

В обоих случаях злоумышленник изменяет в своем браузере значение параметра "id" для отправки ' or '1'='1. Например: http://example.com/app/accountView?id=' or '1'='1

Изменение обоих запросов позволяет получить все записи из таблицы учетных данных. Более серьезные атаки позволяют изменить или удалить данные, а также вызвать хранимые процедуры.

# Недостатки аутентификации (Broken Authentication and Session Management)

Неправильная реализация механизмов аутентификации и сессий, позволяющая злоумышленникам скомпрометировать пароли, ключи, токены сессий или другие уязвимости реализации для установления контроля над аккаунтами других пользователей.

API уязвим если:

* Позволяет злоумышленникам брутфорсить пароли.
* Разрешает слабые пароли.
* Отправляет конфиденциальные данные проверки подлинности, такие как токены проверки подлинности и пароли в URL-адресе.
* Позволяет пользователям изменять свой адрес электронной почты, текущий пароль или выполнять любые другие конфиденциальные операции без запроса подтверждения пароля.
* Не проверяет подлинность токенов.
* Не проверяет дату истечения срока действия JWT.
* Использует простой текст, незашифрованные или слабо хешированные пароли.
* Отсутствует многофакторная аутентификация. Злоумышленнику будет достаточно только учетных данных.

Кроме того, микросервис уязвим, если:

* Другие микросервисы могут получить к нему доступ без аутентификации.
* Использует слабые или предсказуемые токены для обеспечения аутентификации

## **Сценарий №1**

Чтобы выполнить аутентификацию пользователя, клиент должен отправить запрос API, подобный приведенному ниже, с учетными данными пользователя:

POST /graphql

{

"query":"mutation {

login (username:\"<username>\",password:\"<password>\") {

token

}

}"

}

Если учетные данные действительны, то возвращается токен аутентификации, который должен быть предоставлен в последующих запросах для идентификации пользователя. Попытки входа в систему подлежат жесткому ограничению скорости: разрешено только три запроса в минуту.

Чтобы взломать вход с учетной записью жертвы, злоумышленники используют пакетную обработку запросов GraphQL, чтобы обойти ограничение скорости запросов, ускоряя атаку:

POST /graphql

[

{"query":"mutation{login(username:\"victim\",password:\"password\"){token}}"},

{"query":"mutation{login(username:\"victim\",password:\"123456\"){token}}"},

{"query":"mutation{login(username:\"victim\",password:\"qwerty\"){token}}"},

...

{"query":"mutation{login(username:\"victim\",password:\"123\"){token}}"},

]

## **Сценарий №2**

Чтобы обновить адрес электронной почты, связанный с учетной записью пользователя, клиенты должны отправить запрос API, подобный приведенному ниже:

PUT /account

Authorization: Bearer <token>

{ "email": "<new\_email\_address>" }

Поскольку API не требует, чтобы пользователи подтверждали свою личность, предоставляя свой текущий пароль, злоумышленники, способные украсть токен аутентификации, могут получить доступ к учетной записи жертвы, запустив рабочий процесс сброса пароля после обновления электронной почты. адрес аккаунта потерпевшего.

**Сценарий №3**

Атака на учетные записи, с использованием списков известных паролей, является очень распространенной. Если в приложении нет защиты от автоматизированных атак или атак на учетные записи, то оно может быть использовано для определения действующих учетных данных.

**Сценарий №4**

Большинство атак на аутентификацию связано с использованием исключительно паролей. Ранее считавшиеся хорошими требования к смене пароля и его сложности способствуют использованию и переиспользованию пользователями ненадежных паролей. Организациям рекомендуется отказаться от подобной практики (см. NIST 800-63) и внедрить многофакторную аутентификацию.

**Сценарий №5**

Тайм-ауты сессий настроены некорректно. Люди используют общедоступные компьютеры для доступа к приложению, а вместо "выхода из приложения" просто закрывают вкладку и уходят. Злоумышленник может открыть тот же самый браузер, спустя час, и воспользоваться все еще действующей аутентификацией пользователя.

**Как предотвратить:**

* Использование сильных паролей и регулярное обновление паролей.
* Внедрение многофакторной аутентификации.
* Защита сессионных файлов и токенов аутентификации с использованием шифрования и безопасных протоколов.
* Восстановления пароля следует тоже защищать от грубой силы. Следует ограничивать количество попыток.
* Требовать повторной аутентификации для конфиденциальных операций (например, изменение адреса электронной почты владельца учетной записи/номера телефона 2FA).
* Внедрите механизмы защиты от грубой силы, чтобы смягчить заполнение учетных данных, атаки по словарю и атаки методом грубой силы на ваши конечные точки аутентификации. Этот механизм должен быть более строгим, чем обычные механизмы ограничения скорости в ваших API. Блокируйте сессии или аккаунты при многократных попытках входа.

# Разглашение конфиденциальных данных

Вместо взлома механизмов шифрования злоумышленники крадут ключи, проводят атаки по принципу “Человек посередине”. Они могут получать данные в незашифрованном виде в процессе их передачи от клиента пользователя. Ранее полученные базы данных паролей могут быть взломаны методом подбора с использованием графических процессоров.

Часто отсутствует шифрование конфиденциальных данных, либо используются ненадежные алгоритмы, протоколы, методы хранения хешированных паролей и методы создания и управления ключами.

**Для защиты:**

* Используйте протоколы передачи данных с поддержкой шифрования (https с TLS вместо http). Это касается и внутреннего трафика, например, между балансировщиками нагрузки и внутренними системами.
* Стоит шифровать даже резервные копии.
* Не используйте ненадежные или устаревшие алгоритмы шифрования. Меняйте алгоритмы, установленные во фреймворках по умолчанию.
* Не используйте шифроключи созданные по умолчанию. Применяйте механизмы контроля и смены ключей.
* Не храните конфиденциальные данные без необходимости. Сразу удаляйте их или используйте токенезацию или усечение.
* Отключите кэширование ответов, содержащих конфиденциальные данные.
* Сохраняйте пароли с помощью надежных, адаптивных функций кэширования с солью.

**Примеры сценариев атак**

**Сценарий №1**: Приложение шифрует номера кредитных карт в базе данных, используя автоматическое шифрование БД. Однако эти данные автоматически расшифровываются при извлечении, позволяя с помощью внедрения SQL-кода получить данные кредитных карт в незашифрованном виде.

**Сценарий №2**: Сайт не использует TLS для всех страниц или поддерживает

ненадежное шифрование. Злоумышленник может просмотреть сетевой

трафик (например, в небезопасной беспроводной сети), переключить

соединение с HTTPS на HTTP, перехватить запросы и похитить сессионные

куки. После этого он может использовать полученные куки для перехвата

сессии пользователя (прошедшего аутентификацию), изменив личные

данные пользователя. Также злоумышленник может изменить все

передаваемые данные , например, получателя денежного перевода.

**Сценарий №3**: Для сохранения паролей в базе данных не используется соль

или используется простой алгоритм хеширования. Уязвимость в загрузке

файлов позволяет злоумышленнику получить БД паролей. Все хеш-

значения без соли могут быть восстановлены с помощью радужной таблицы

предварительно рассчитанных хешей. Хеш-значения, рассчитанные с

использованием простых или быстрых хеш-функций, могут быть взломаны с

помощью графических процессоров, даже если для них использовалась

соль.

# Внешние сущности XML (XXE)

Злоумышленники могут эксплуатировать уязвимые обработчики XML через загрузку XML и внедрения вредоносного контента в XML-документы.

Большинство старых обработчиков XML позволяет задавать внешние сущности. URI, которые разыменовываются и вычисляются при обработке XML.

Подобные уязвимости могут использоваться для получения данных, выполнения удаленных запросов с сервера, сканирования внутренней системы, провоцирования отказа в обслуживании и др.

**Инъекция внешних сущностей XML** (также известная как XXE) — это уязвимость веб-безопасности, которая позволяет злоумышленнику вмешиваться в обработку XML-данных приложением. Оно часто позволяет злоумышленнику просматривать файлы в файловой системе сервера приложений и взаимодействовать с любыми серверными или внешними системами, к которым может получить доступ само приложение.

**Приложение уязвимо если:**

* XML-обработчик использует DTD.
* если приложение уязвимо для XXE-атак, то злоумышленник может также вызвать отказ в обслуживании или осуществить атаку с использованием миллиона XML-сущностей (Billion Laughs).

**Как защититься:**

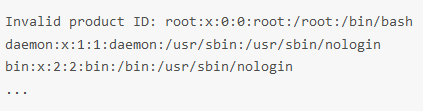
* использовать, по возможности, более простые форматы данных, например, JSON, и избегать сериализации критичных данных;
* Отключить обработку внешних сущностей XML и DTD во всех XML-обработчиках приложения
* реализовать на сервере (по белым спискам) проверку, фильтрацию или очистку (экранирование) входных данных для предотвращения попадания вредоносных данных в XML-документы, заголовки или узлы;
* удостовериться, что функция загрузки XML или XSL проверяет входящие файлы с использованием XSD или другой подобной методики;

**Примеры атак**

**Сценарий №1**: Злоумышленник пытается получить содержимое файлов с сервера:



В этом случае вместо id будет передаваться содержимое файла passwd. Это конечно же приведет к ошибке в приложении, и в сообщении об ошибке мы сможем увидеть содержимое файла.



XML можно использовать для подделки запроса на стороне сервера (SSRF).

**SSRF (Server-Side Request Forgery)** - это атака на серверную сторону, при которой злоумышленник может заставить сервер отправить запросы на другие ресурсы или системы, на которые у него нет прямого доступа.

В атаке SSRF злоумышленник обманывает сервер, чтобы он выполнил запросы на различные внутренние или внешние ресурсы. Это может включать запросы к локальным файлам, внутренним сетям, другим веб-сайтам или даже защищенным системам, к которым сервер имеет доступ.



# Недостатки контроля доступа (Broken Access Control)

**Приложение уязвимо если:**

* Возможно изменять первичный ключ в запросах для доступа к записям других пользователей, включая просмотр или редактирование чужой учетной записи.
* Повышение привилегий. Выполнение операций с правами пользователя, не входя в систему, или с правами администратора, войдя в систему с правами пользователя;
* Манипуляция с метаданными: например подмена токенов контроля доступа JWT или куки файлов, изменение скрытых полей для повышения привелегий.
* несанкционированный доступ к API из-за некорректной настройки междоменного использования ресурсов (CORS);
* контролировать доступ к моделям, используя владение записями,
* Можно запрашивать файлы, которые лежат в каталоге на сервере.
* Отсутствует контроль доступа на уровне функций.

**Как защититься:**

* запрещать доступ по умолчанию, за исключением открытых ресурсов.
* регистрировать сбои контроля доступа и уведомлять администраторов при необходимости
* ограничивать частоту доступа к API и контроллерам для минимизации ущерба от инструментов автоматизации атак;
* аннулировать токены JWT на сервере после выхода из системы.

**Сценарий №1**: Приложение использует непроверенные данные в SQL-вызове, который обращается к информации об учетной записи:



Злоумышленник изменяет в браузере параметр 'acct' для отправки желаемого номера учетной записи. Без должной проверки атакующий может получить доступ к учетной записи любого пользователя.

**Сценарий №2**: Злоумышленник задает в браузере целевой URL. Для доступа к странице администрирования требуются права администратора. 

Уязвимость существует, если пользователь без аутентификации может получить доступ к этим страницам или если пользователь без прав администратора может получить доступ к странице администрирования.

# !!!!!!!!!Некорректная настройка параметров безопасности

Злоумышленники часто пытаются эксплуатировать неисправленные уязвимости, настроенные по умолчанию учетные записи, неиспользуемые страницы, незащищенные файлы и каталоги для получения несанкционированного доступа или информации о системе.

**Приложение уязвимо, если:**

* любой из компонентов приложения недостаточно защищен или разрешения облачных сервисов некорректно настроены;
* включены или присутствуют лишние функции (например, неиспользуемые порты, службы, страницы, учетные записи или привилегии);
* учетные записи и пароли, создаваемые по умолчанию, используются без изменений;
* обработка ошибок позволяет осуществить трассировку стека или получить слишком подробные сообщения об ошибках;

**Как предотвратить:**

* Использовать минимально необходимый набор функций, библиотек. Удалите или не устанавливайте лишние компоненты и фреймворки.

**Примеры сценариев атак**

**Сценарий №1** Сервер приложений поставляется с образцами приложений,

которые не удаляются с рабочего сервера. Эти приложения содержат

известные уязвимости, позволяющие злоумышленникам скомпрометировать

сервер. Если одно из этих приложений является консолью администратора, а

стандартные учетные записи не менялись, то атакующий может войти в

приложение и перехватить контроль над ним, используя стандартный пароль.

**Сценарий №2** На сервере не отключен вывод списка файлов в каталогах, что

позволяет злоумышленнику найти и выгрузить скомпилированные Java-

классы, после декомпиляции и обратного анализа которых можно

просмотреть исходный код. В результате атакующий может обнаружить

уязвимости и получить доступ к приложению

**Сценарий №3** Сервер приложений настроен на отправку подробных сообщений об ошибках, включая данные о трассировке стека. Это может привести к разглашению важной информации, например, о версии компонента, содержащей известные уязвимости.

# Межсайтовое выполнение сценариев (XSS)

XSS (англ. Cross-Site Scripting — «межсайтовый скриптинг»). Ее суть довольно проста, злоумышленнику удается внедрить на страницу JavaScript-код, который не был предусмотрен разработчиками. Этот код будет выполняться каждый раз, когда жертвы будут заходить на страницу приложения.

Злоумышленник может :

получить авторизационные данные пользователя и войти в его аккаунт.

Незаметно для жертвы перенаправить его на другую страницу. Эта страница может выглядеть идентично, и пользователь может вводить на ней какие-то авторизационные данные и т.п.

Регестриовать нажатия клавиш.

Заменять или подменять DOM узлы.

Украсть авторизационные куки. Куки хранятся на клиенте. JS код может без труда их достать и отправить злоумышленнику (например через AJAX-запрос).

**Как заразить:**

В инпут можно ввести вредоносный скрипт, который затем выводится на странице (например коментарий).

Когда другие пользователи зайдут на эту же страницу, вместе с текстом они загрузят и JavaScript-код злоумышленника.

Если страница использует какие-то get-параметры. Например при поиске, ссылка изменяется на



А при неудачном поиске страница выводит “сущность gdfg не найдена”. Значит в URL в параметр search злоумышленник может написать <script>вредоносный код</script>. В этом случае пользователя нужно заставить перейти по ссылке.

**Типы:**

**отраженное(Reflected)** - Когда модифицируется url, и пользователь переходит по ссылке.

**Межсайтовое -**  приложение сохраняет необработанные входные данные в бд. Затем эти данные передаются другим пользователям которые посещают эту страницу (комментарии на форумах).

**DOM XSS –** этой атаке подвержены JS-фреймворки, одностраничные приложения, которые динамически обновляют DOM. С помощью них также можно внедрить. (Класический XSS выполняется на стороне сервера, теже JSP). Если веб-приложение динамически создает DOM-элементы на основе пользовательского ввода без должной фильтрации, это может привести к возможности внедрения вредоносного кода.

**Примечание**: злоумышленник может использовать XSS для обхода защиты от межсайтовой подмены запросов (CSRF), используемой в приложении.

**Как защититься:**

* Экранировать пользовательский ввод.
* Белый список (Whitelisting): Рекомендуется использовать подход "белого списка" для разрешения только определенных тегов, атрибутов или стилей. Таким образом, все остальные теги или атрибуты будут отфильтрованы, и только разрешенные элементы будут разрешены в пользовательских данных.
* Использовать фреймворки с автоматическим преобразованием данных.
* От кражи куки можно установить для cookie атрибут “http only” и к куке нельзя будет обратиться через JS.

**Небезопасная десериализация**

Приложение уязвимо если оно осуществляет десериализацию вредоносных или модифицированных объектов.

**Два основных типа атак**:

атаки, связанные со структурой объектов и данных, когда злоумышленник изменяет логику приложения или удаленно выполняет произвольный код при наличии доступных приложению классов, поведение которых может меняться во время или после десериализации;

атаки с подменой данных, например, связанные с управлением доступом, когда используются существующие структуры данных, но изменяется содержимое.

**Для защиты**:

* Проверять подлинность данных и целостность сериализованных данных, например с помощью цифровых подписей.
* Валидация и фильтрация данных.
* Ограничение доступа к сериализованным данным, чтобы предотвратить их модификацию и злоупотребление.
* Использовать сериализацию только примитивных типов данных.
* Изоляция и запуск кода, осуществляющего десериализацию в среде с минимальными привилегиями, если это возможно.
* Журналирование исключений и ошибок десериализации, например, непредусмотренных типов данных.

# Использование компонентов с известными уязвимостями

**Приложение уязвимо если:**

* ПО содержит уязвимости, не поддерживается или устарело. Сюда относятся ОС, веб-серверы, серверы приложений, СУБД, приложения, API, а также все компоненты, среды исполнения и библиотеки;
* своевременно не устанавливаются исправления или обновления для используемых платформ, фреймворков и зависимостей

**Для защиты:**

* Удалите неиспользуемые зависимости, а также лишние функции, компоненты, файлы и сведения из документации;
* регулярно проверяйте актуальность версий клиентских и серверных компонентов (например, фреймворков и библиотек), а также их зависимостей. Для упрощения этого процесса можно использовать инструменты автоматического обновления зависимостей, такие как **Renovate**, **Greenkeeper** или **Dependabot**.
* загружайте компоненты из официальных источников по безопасным ссылкам.
* Использование утилит сканирования уязвимостей: Существуют различные утилиты, такие как **OWASP Dependency-Check**, **Retire.js**, **Snyk** и другие, которые позволяют сканировать зависимости и компоненты проекта на наличие известных уязвимостей. Эти инструменты могут автоматически анализировать используемые библиотеки и предоставлять отчеты о выявленных уязвимостях.

# Недостатки журналирования и мониторинга

При совершении атак злоумышленники полагаются на отсутствие контроля и своевременного реагирования на инциденты.

**Приложение уязвимо если:**

* не журналируются события удачных и неудачныхх попыток входа в систему.
* Предупреждения или ошибки не регистрируются или регестрируются неккоректно.
* журналы приложений и API не проверяются на предмет подозрительной активности;
* Приложение не может определять, реагировать или предупреждать об атаках в реальном или почти реальном времени
* Журналируются конфиденциальные данные. Если злоумышленник получит к ним доступ, то скомпрометирует их.
* Социальная инженерия: Злоумышленники могут использовать информацию из журналов и данных мониторинга для проведения атак социальной инженерии. Например, они могут использовать информацию о событиях или действиях пользователей для создания поддельных сообщений или для маскировки своей активности.

**Как предотвратить:**

* регистрировать все ошибки входа, доступа и проверки данных на стороне сервера с указанием контекста, достаточного для выявления подозрительных или вредоносных действий, а также хранить их для последующего анализа;
* регистрировать события в формате, наиболее подходящем для обработки централизованной службой журналирования;
* использовать контроль целостности журналов аудита важных транзакций для предотвращения подмены или удаления данных, например, с помощью доступных только для добавления таблиц БД;

**Примеры сценариев атак**

**Сценарий №1**: Форум открытого проекта, используемый небольшой

командой, был взломан через уязвимость в его ПО. Злоумышленники

удалили внутренний репозиторий, содержащий следующую версию

продукта, а также все содержимое форума. Несмотря на возможность

восстановления источника, отсутствие мониторинга, журналирования

или оповещений привело к более серьезным последствиям. Из-за

инцидента программный проект с форума более не развивается.

**Сценарий №2**: Злоумышленник может использовать один стандартный

пароль для проверки доступа ко всем учетным записям, к некоторым

из них он может подойти. Для остальных будет зарегистрирована лишь

неудачная попытка входа. Через несколько дней попытка может

повториться, но уже с другим паролем.

**Сценарий №3**: В крупной торговой сети имеется песочница для

внутреннего анализа вредоносных вложений. Средства песочницы

обнаружили потенциально вредоносное ПО, но никто не обращал

внимания на получаемые от песочницы предупреждения, пока взлом

не обнаружили в связи с мошенническими транзакциями по

банковским картам от стороннего банка