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 сторонею
* 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 злоумыщленникам скомпрометивать 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.

**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.

**Недостатки контроля доступа (Broken Access Control)**

**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.