БЕЗОПАСНОСТЬ ЦИФРОВЫХ ТЕХНОЛОГИЙ. – 2025. – № 3 (118). – 23–39

МЕТОДЫ И СИСТЕМЫ ЗАЩИТЫ ИНФОРМАЦИИ, ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

УДК 004 DOI: 10.17212/2782-2230-2025-3-23-39

РАЗРАБОТКА СЕРВЕРНОГО ПРИЛОЖЕНИЯ ДЛЯ ЗАЩИТЫ ОТ SQLI-АТАК В POSTGRESQL*

Ю.А. КОТОВ[1], А.Р. ВТЮРИН[2]

[1] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, кандидат физико-математических наук, доцент, доцент кафедры защиты информации. E-mail: kotov@corp.nstu.ru*
[2] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, лаборант кафедры защиты информации. E-mail: a.vtyurin.2021@stud.nstu.ru*

В статье обсуждается актуальная проблема защиты баз данных PostgreSQL от атак типа SQL-инъекций (SQLi), представляющих серьезную угрозу целостности и конфиденциальности данных. Для решения проблемы предложен метод серверного расширения, интегрируемого на уровне ядра СУБД. Расширение использует механизм перехвата SQL-запросов через системные хуки (ProcessUtility_hook, ExecutorStart_hook) и выполняет их верификацию с применением криптостойкого алгоритма хеширования ГОСТ Р 34.11-2018 («Стрибог»).

Рассмотрены ключевые аспекты реализации:

- очистка SQL-запросов от переменных данных (значений атрибутов) для анализа структуры;
- сравнение хешей очищенных запросов с «белым списком» разрешенных шаблонов;
- реализация функций управления политиками (добавление хешей, отключение проверки);
- детальное логирование операций.

Решение обеспечивает защиту на уровне ядра СУБД без модификации прикладного ПО, соответствуя требованиям российских стандартов информационной безопасности. Разработанное расширение предназначено для систем с повышенными требованиями к безопасности данных.

Ключевые слова: PostgreSQL, SQL-инъекция, защита базы данных, триггеры событий PostgreSQL, расширение PostgreSQL, хеш-функция, ГОСТ Р 34.11-2018, информационная безопасность

* Статья получена 01 июля 2025 г.

ВВЕДЕНИЕ

Базы данных – фундаментальный компонент современных информационных систем, обеспечивающий структурированное хранение и управление данными [1]. Их повсеместное использование, от мобильных устройств до корпоративных систем, формирует критическую зависимость общества от их надежности и бесперебойной работы.

Для бизнеса базы данных являются стратегическим активом, определяющим конкурентоспособность. Их безопасность трансформировалась из технической задачи в элемент бизнес-стратегии, так как сбои или компрометация данных способны парализовать работу компании и привести к значительным убыткам и репутационным рискам [2].

Актуальность защиты баз данных постоянно возрастает. Угрозы, включая внешние атаки (такие как SQL-инъекции) и внутренние риски, становятся всё сложнее [3]. Одновременно растет давление правовых требований (GDPR, HIPAA, PCI DSS, 152-ФЗ, 187-ФЗ) и стандартов ФСТЭК/ФСБ, невыполнение которых влечет серьезные санкции.

Таким образом, обеспечение безопасности баз данных – неотъемлемая часть общей стратегии информационной безопасности любой организации и актуальная научно-практическая задача.

Объектом исследования настоящей работы являются механизмы защиты баз данных от атак с использованием SQL-инъекций. Работа направлена на создание инструмента, который позволит снизить риски безопасности, возникающие в процессе взаимодействия пользователей с базой данных.

Предметом исследования является разработка серверного приложения для защиты баз данных от атак с использованием SQL-инъекций.

Целью работы является разработка серверного приложения, снижающего риски атак с использованием SQL-инъекций на защищаемую базу.

Для достижения поставленной цели были выделены следующие задачи:

- разработка системы перехвата запросов пользователей;
- создание системы мониторинга и контроля SQL-запросов, включая проверку на наличие некорректных или потенциально вредоносных запросов;
- реализация системы логирования всех действий для последующего анализа и аудита.

1. МЕТОДЫ И СРЕДСТВА КОНТРОЛЯ SQL-ИНЪЕКЦИЙ В РЕЛЯЦИОННЫХ БД

1.1. SQL-ИНЪЕКЦИИ И ИХ ВИДЫ

SQL-инъекция – это метод атаки на базы данных, при котором злоумышленник внедряет вредоносный код в SQL запрос, отправляемый к базе данных, для манипулирования базой данных и получения доступа к потенциально ценной информации. Атаки подобного типа позволяют неавторизованным пользователям получить доступ к информации, содержащейся в базе данных, а также к ее структуре. Подобные атаки могут быть нацелены на любые сайты и веб-приложения, которые взаимодействуют с базами данных [1].

Чаше всего атаки формата SQL-инъекции реализуются посредством прямой вставки кода в пользовательские входные данные.

Существует различные виды атак, основанных на внедрении SQL-кода:

- SQL-инъекция на основе объединения (Классическая SQL-инъекция);
- SQL-инъекция на основе ошибок (Error-based SQL инъекция);
- слепая SQL-инъекция (Blind SQLi);
- SQL-инъекция на основе логических значений;
- атаки SQL-инъекций, основанные на времени [4, 5].

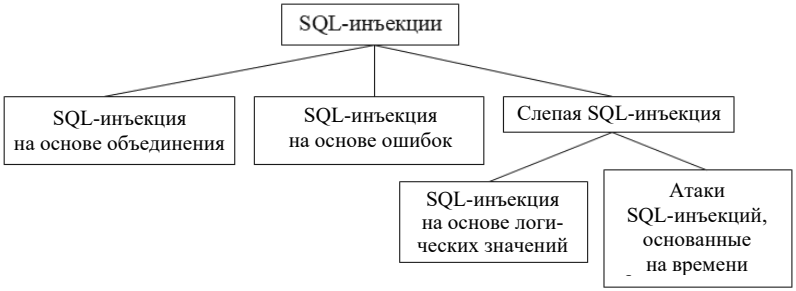

Рис. 1. Виды SQL-инъекций

В настоящей работе будет рассматриваться защита от атак типа **Классическая SQL-инъекция** и **SQL-инъекция на основе ошибок**.

Классическая SQL-инъекция

Классическая SQL-инъекция представляет собой один из наиболее известных и распространенных видов атак на веб-приложения. Такая уязвимость возникает в случае, когда вводимые пользователем данные напрямую

включаются в SQL-запрос без предварительной проверки и экранирования. Это позволяет злоумышленнику изменять структуру запроса и внедрять в него произвольный SQL-код [5].

Одним из характерных примеров является атака типа UNION SELECT, при которой атакующий добавляет к исходному запросу дополнительные команды. Подобные действия могут привести к раскрытию информации, содержащейся в других таблицах базы данных, а в некоторых случаях – к модификации или удалению данных из таблиц [3].

Основная причина такой уязвимости заключается в отсутствии надлежащей обработки вводимых пользователем данных. Классическая SQL-инъекция может привести к серьезным последствиям, включая компрометацию базы данных, утечку данных и нарушения целостности хранимой в ней информации [2].

SQL-инъекция на основе ошибок

SQL-инъекция на основе ошибок – это метод атаки, при котором злоумышленник намеренно вызывает ошибки в работе базы данных с целью получения дополнительной информации о ее структуре или содержимом. Суть такого подхода заключается в том, что СУБД, возвращая сообщения об ошибках, может непреднамеренно раскрыть сведения, не предназначенные для пользователей [5].

Примером подобной атаки является модификация SQL-подобным образом:

```
'OR 1=CONVERT(int, (SELECT TOP 1 table_name FROM INFORMATION_SCHEMA.TABLES)) --'
```

Такой запрос может привести к ошибке преобразования типов, в сообщении которой будет указано название таблицы, что позволяет злоумышленнику поэтапно «собирать» информацию о структуре базы данных, даже не имея к ней прямого доступа.

Эта уязвимость возникает из-за некорректной обработки исключений и открытого отображения сообщений об ошибках всем пользователям. Надежной мерой защиты в данном случае будет скрытие технических деталей ошибок, а также использование механизмов обработки исключений и валидации пользовательского ввода [3].

1.2. МЕТОДИКИ ЗАЩИТЫ ОТ SQL-ИНЪЕКЦИЙ

Анализируя существующие виды SQL-инъекций, можно сделать вывод, что основной причиной их возникновения являются недостаточная проверка данных, поступающих от пользователя, а также ошибки, допущенные при

проектировании и разработке логики взаимодействия с базой данных. Для защиты от подобных уязвимостей можно использовать следующие методы.

Проверка входных данных

Одним из наиболее эффективных и универсальных методов защиты от SQL-инъекций является реализация строгого контроля пользовательского ввода [3].

Для реализации такого подхода можно использовать, черные и белые списки значений:

- **белый список значений** допускает к выполнению только те запросы, которые соответствуют заранее определенным безопасным значениям;
- **черный список символов** блокирует выполнение запроса, если он содержит потенциально опасные символы или значения.

В настоящей работе рассматривалась реализация белых списков, так как их использование считается более надежным подходом благодаря строгим рамкам ввода [3].

Хеширование запросов

Современным и перспективным методом предотвращения SQL-инъекций является хеширование SQL-запросов [6]. Этот подход предполагает, что каждый SQL-запрос перед выполнением проверяется на соответствие заранее определенному шаблону [8].

Суть метода:

1) создается массив, содержащий хеш-значения допустимых запросов (например, SELECT, INSERT, DELETE, ALTER) в их абстрактной форме;

2) перед выполнением SQL-запроса его структура анализируется: из него удаляются переменные данные (значения атрибутов) с использованием регулярных выражений или других алгоритмов обработки;

3) полученный запрос преобразуется в хеш с использованием криптографического алгоритма;

4) хеш вычисленного запроса сравнивается с ранее сохраненными хешами допустимых команд:

- если хеш совпадает, запрос выполняется;
- в противном случае запрос отклоняется как потенциально вредоносный.

Этот подход обеспечивает высокий уровень безопасности, так как вредоносные SQL-запросы не пройдут проверку хеш-значений. Даже если атакующий скомпрометирует базу данных, использование хешей делает практически невозможным выявление оригинальных данных или легитимных запросов [7].

1.3. МЕТОДЫ ПЕРЕХВАТА SQL-ЗАПРОСОВ В POSTGRESQL

Триггеры событий в PostgreSQL

ProcessUtility_hook

ProcessUtility_hook – это низкоуровневый механизм расширения, встроенный в ядро PostgreSQL. Он предназначен для перехвата и обработки административных (DLL) SQL-запросов, таких как CREATE, DROP, ALTER, GRANT и т. д. [9]. Хук срабатывает при выполнении административных запросов.

Основные возможности ProcessUtility_hook:

1) **перехват административных команд**: ProcessUtility_hook позволяет перехватывать административные SQL-команды до их выполнения. Это дает возможность анализировать запросы и принимать решение о дальнейшем их выполнении, изменении или блокировке;

2) **изменение логики выполнения команд**: после перехвата запроса возможно изменить его структуру. Это предоставляет широкие возможности по внедрению собственных правил исполнения запросов;

4) **аудит и логирование**: с помощью ProcessUtility_hook можно вести детальное логирование всех перехваченных административных запросов. Это особенно актуально в системах, предъявляющих высокие требования к безопасности и учету действий пользователей.

ExecutorStart_hook

ExecutorStart_hook – это механизм расширения PostgreSQL, предназначенный для перехвата SQL-запросов непосредственно перед их выполнением. Он позволяет обрабатывать команды, относящиеся к классу DML (Data Manipulation Language), включая SELECT, INSERT, UPDATE, DELETE и т. д. [9]. Данный хук предоставляет разработчику возможность вмешиваться в процесс выполнения запроса, осуществляя его анализ, модификацию, логирование или блокировку выполнения.

Основные возможности ExecutorStart_hook:

1) **перехват DML-запросов**: ExecutorStart_hook позволяет перехватывать команды SELECT, INSERT, UPDATE, DELETE и другие DML-запросы;

2) **аудит и логирование запросов**: благодаря доступу к тексту и структуре запроса до его выполнения ExecutorStart_hook широко используется для журналирования пользовательской активности. Это позволяет вести аудит операций с данными, отслеживать потенциально опасные действия и формировать отчетность;

3) **модификация запросов**: внутри хука возможно изменение параметров исполнения запроса (например, можно динамически добавлять фильтры,

ограничивать выборку, перенаправлять доступ или изменять контекст выполнения);

4) **контроль безопасности**: с помощью данного механизма можно внедрять контекстно-зависимые политики доступа, запрещать выполнение отдельных типов запросов для определенных пользователей или ограничивать доступ к чувствительным данным.

2. АРХИТЕКТУРА И СТРУКТУРА СЕРВЕРНОГО ПРИЛОЖЕНИЯ

2.1. ВЫБОР ПОДХОДА РАЗРАБОТКИ

В ходе исследования было принято решение реализовать приложение, обеспечивающее защиту от атак с использованием SQL-инъекций (SQLi), в виде расширения к системе управления базами данных PostgreSQL. В качестве языка программирования для реализации выбран язык C, что обусловлено необходимостью прямого взаимодействия с внутренними механизмами СУБД.

Для написания расширения была выбрана модульная архитектура. Модульная архитектура предполагает разделение системы на независимые компоненты (модули или плагины), которые можно добавлять, удалять или заменять без изменения основной системы. Каждый модуль решает конкретную задачу и взаимодействует с ядром через четко определенные интерфейсы.

Для реализации механизма защиты используются триггеры событий **ProcessUtility_hook** и **ExecutorStart_hook**, которые обеспечивают перехват SQL-запросов на этапе их обработки и выполнения. Это позволяет внедрить дополнительную проверку запросов без необходимости модификации исходного кода PostgreSQL, что упрощает обновление и поддержку базы данных.

В качестве основной меры защиты применяется механизм **хеширования запросов** с использованием криптографического алгоритма **ГОСТ Р 34.11-2018 (Стрибог)**. Этот алгоритм выбран благодаря его устойчивости к современным атакам, высокой степени криптостойкости и соответствию отечественным стандартам безопасности [10, 11]. Хеширование позволяет сравнивать полученные SQL-запросы с предварительно вычисленными и доверенными шаблонами. Если хеш нового запроса не совпадает с известными значениями, его выполнение блокируется.

2.2. АРХИТЕКТУРА СЕРВЕРНОГО ПРИЛОЖЕНИЯ

Для описания архитектуры разрабатываемого расширения было принято решение использовать нотацию C4 [12]. Нотация C4 предназначена для последовательного представления архитектуры программного обеспечения на различных уровнях детализации. Она охватывает четыре уровня: контекст (Context), контейнеры (Containers), компоненты (Components) и код (Code) [13]. Такой подход позволяет отобразить как общую картину архитектуры, так и ее отдельные детали вплоть до исходного кода.

C1 (контекст)

На этом уровне система рассматривается как «черный ящик» в окружении внешних сущностей [13].

Цель данного уровня – показать, кто и как взаимодействует с системой [12].

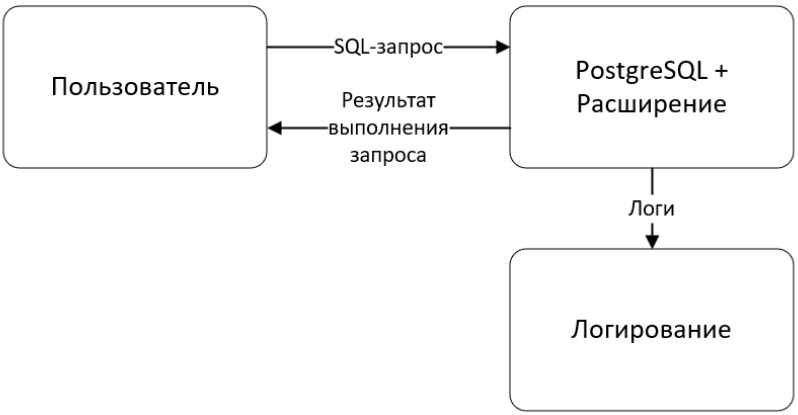

Рис. 2. Диаграмма архитектуры расширения уровня C1

C2 (Контекст)

На этом уровне детализируются сущности, рассмотренные на уровне C1 [13].

Цель данного уровня – показать, из каких частей состоит система и как они взаимодействуют [12].

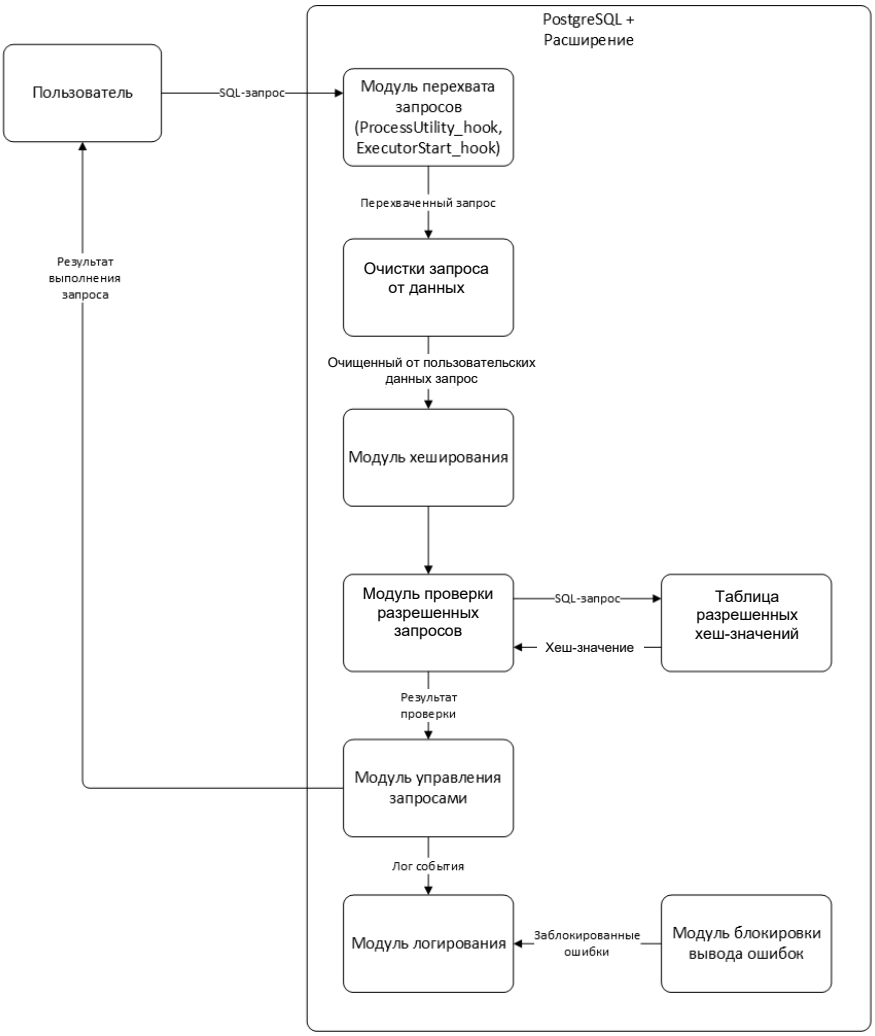

Рис. 3. Диаграмма архитектуры расширения уровня С2

3. АРХИТЕКТУРА И СТРУКТУРА СЕРВЕРНОГО ПРИЛОЖЕНИЯ

3.1. СРЕДСТВА И ИНСТРУМЕНТЫ РАЗРАБОТКИ

В качестве платформы для разработки серверного расширения, предназначенного для защиты от SQL-инъекций, была выбрана система управления базами данных PostgreSQL. Это решение обусловлено развитым механизмом расширения функциональности на уровне ядра, включая поддержку подключения внешних модулей, написанных на языке C, а также возможностью доступа к внутренним хукам, позволяющим перехватывать выполнение SQL-команд. Также PostgreSQL отличается высоким уровнем стабильности, активной поддержкой сообщества.

Для реализации расширения была выбрана СУБД PostgreSQL. Такое решение обусловлено развитым функционалом для расширения.

В качестве версии PostgreSQL использовалась актуальная на момент разработки версия 16.6, поддерживающая все необходимые интерфейсы для взаимодействия с расширениями на C, включая ProcessUtility_hook и ExecutorStart_hook.

Как ранее упоминалось, для разработки будет использоваться язык программирования C. Для компиляции и сборки применялись стандартные инструменты GNU: компилятор gcc, система сборки make, а также PostgreSQL-пакеты разработки, в частности postgresql-server-dev-16, обеспечивающие доступ к заголовочным файлам и Makefile-инфраструктуре PostgreSQL.

3.2. РЕАЛИЗАЦИЯ КОМПОНЕНТОВ РАСШИРЕНИЯ

Разработка расширения производилась на языке программирования C с использованием внутренних интерфейсов PostgreSQL. Реализация включает в себя несколько ключевых модулей: перехват SQL-запросов, очистку и хеширование запроса, сравнение хешей, логирование и управление режимами работы, а также систему управления разрешенными хеш-значениями.

Инициализация и подключение

Подключение расширения реализовано через функцию _PG_init(), которая вызывается при старте PostgreSQL, для корректной работы расширения необходимо будет указать в shared_preload_libraries параметре, расположенном в конфигурационном файле PostgreSQL (/etc/postgresql/16/main/postgresql.conf). Функция _PG_init() инициализирует ProcessUtility_hook и ExecutorStart_hook.

Также при инициализации расширения, до запуска проверки SQL-запросов проверяется наличие таблицы query_hashes, в случае ее отсутствия она будет создана, а также регистрируются SQL-функции, необходимые для дальнейшего функционирования расширения.

Перехват SQL-запросов

Для перехвата административных SQL-команд используется ProcessUtility_hook, а для запросов SELECT, INSERT, UPDATE, DELETE и других – ExecutorStart_hook.

Логика работы хуков реализована в функциях:

- static void custom_ProcessUtility(PlannedStmt *pstmt, const char *queryString, bool readOnlyTree, ProcessUtilityContext context, ParamListInfo params, QueryEnvironment *queryEnv, DestReceiver *dest, QueryCompletion *completionTag) [12];
- static void custom_ExecutorStart(QueryDesc *queryDesc, int eflags) [12].

Очистка запроса от переменных данных

Целью функции очистки запроса является удаление значений, не влияющих на структуру запроса (константы, литералы) [14], для получения стабильного представления запроса при последующем хешировании.

Функция очистки запроса представлена в файлах sanitize_sql_query.c и sanitize_sql_query.h.

Пример работы функции:

- входной запрос: ‘SELECT * FROM users WHERE id = 42;’;
- вывод функции: ‘SELECT * FROM users WHERE id = ?;’.

После очистки запрос будет передан для хеширования.

Хеширование и сравнение

Очищенный запрос передается в хеш-функцию, реализованную с использованием ГОСТ Р 34.11-2018 и представленную в файлах: stribog.c, stribog.h, gost_3411_2018_const.h, gost_3411_2018_calc.h, gost_3411_2018_calc.c [15]. Полученный хеш в результате работы функции GetHashString() сравнивается с содержимым таблицы query_hashes с использованием SPI. При отсутствии совпадения выполнение запроса отменяется с соответствующим сообщением (если не отключена проверка запросов).

Пример работы функции хеширования:

- входной запрос: ‘SELECT 1;’;
- вывод функции:

‘aa89c74c7c1038845dfc1f7f2625d2787180073d3b1909ee7ec6b4303f1a4b59’.

Управление разрешениями

В расширение добавлены SQL-функции для администрирования:

- allow_query_hash(text) – добавляет разрешенный хеш в таблицу;
- enable_query_hash_check() / disable_query_hash_check() – включает/отключает проверку хешей.

Каждая из этих функций реализована на C и регистрируется при установке расширения.

Логирование действий

Все события, включая перехваченные запросы, хеши и статус проверки, записываются в файл /var/log/postgresql/extension_log.txt. Это позволяет проводить аудит выполненных действий и отслеживать причины блокировок.

ЗАКЛЮЧЕНИЕ

В ходе реализации серверного расширения для защиты СУБД PostgreSQL от атак с применением SQL-инъекций была достигнута основная цель – разработан механизм фильтрации SQL-запросов на уровне ядра, не требующий модификации пользовательских приложений или middleware-слоя.

Применение внутренних хуков PostgreSQL (ProcessUtility_hook и ExecutorStart_hook) позволило реализовать глубокую интеграцию расширения в процессы анализа и исполнения запросов. Использование алгоритма хеширования ГОСТ Р 34.11–2018 обеспечивает детерминированную и устойчивую к подделке проверку структуры SQL-запросов. Механизм очистки запросов от данных позволяет абстрагироваться от конкретных значений и сосредоточиться на анализе их логической структуры.

Особое внимание было уделено стабильности и устойчивости расширения: внедрен механизм инициализационного флага, предотвращающий перехват системных запросов при установке; реализованы управляющие функции (enable_query_hash_check, и др.), позволяющие гибко изменять поведение фильтра.

Результаты тестирования подтвердили работоспособность и корректность решения: допустимые запросы успешно исполняются, в то время как неизвестные или измененные – блокируются. Ведение логов в отдельном файле обеспечивает возможность аудита и мониторинга.

Таким образом, предложенное расширение может служить основой для построения гибкой и независимой системы защиты баз данных [6] от SQL-инъекций, ориентированной на применение в системах с повышенными требованиями к безопасности.

СПИСОК ЛИТЕРАТУРЫ

1. Юрченко А.С., Яшаров Д.А., Ефименко К.Н. Отдельные аспекты безопасности информационных систем // Программная инженерия: методы и технологии разработки информационно-вычислительных систем (ПИИВС-2018): сборник научных трудов II Международной научно-практической конференции (студенческая секция). В 2 т. Т. 2. – Донецк: Донец. нац. техн. ун-т, 2018. – С. 218–222. – EDN AJEVJE.

2. *Бурмистров А.В., Белов Ю.С.* Недостатки реляционных баз данных // Электронный журнал: наука, техника и образование. – 2015. – № 3 (3). – С. 25–34. – EDN VZCABF.

3. *Бакиров А.М., Рычков В.А., Рычкова В.И.* Анализ уязвимостей систем управления базами данных и как их обнаружить на основе атаки SQL // МСИ: 10 лет подготовки кадров для международной системы ПОД/ФТ: материалы IX Международной научно-практической конференции Международного сетевого института в сфере ПОД/ФТ, Москва, 22–24 ноября 2023 года. – М.: МИФИ, 2023. – С. 128–133. – EDN NQNGVA.

4. *Бурлевич М.В.* Утилита для обнаружения и эксплуатации слепых SQL-инъекций // Студенческая научная весна: сборник тезисов докладов Всероссийской студенческой конференции, посвященной 110-летию со дня рождения академика В.Н. Челомея, Москва, 01–30 апреля 2024 года. – М.: Научная библиотека, 2024. – С. 214–215. – EDN TKULDR.

5. *Мисбахов Н.И., Лукьянов Э.Р., Степанов М.О.А.А.М.* Исследование методов эксплуатации SQL-инъекций // Актуальные проблемы науки и образования в условиях современных вызовов: сборник материалов XXVIIII Международной научно-практической конференции, Москва, 01 марта 2024 года. – СПб., 2024. – С. 59–62. – EDN AJEOHL.

6. Адаптивное обнаружение вредоносных запросов в веб-атаках / Е.Н. Успенский, А.С. Стариков, Г.В. Ромашкина, А.Н. Норкина // Актуальные проблемы менеджмента, экономики и экономической безопасности: сборник материалов Международной научной конференции, Костанай, 27–29 мая 2019 года. – Чебоксары: Среда, 2019. – С. 308–311. – DOI: 10.31483/r-32922. – EDN KBXFRR.

7. Предупреждение атак, базирующихся на SQL-инъекциях / Н.В. Яковенко, Е.М. Чижова, Л.А. Тремасова, Д.Д. Гадасин // REDS: Телекоммуникационные устройства и системы. – 2023. – Т. 13, № 3. – С. 41–48. – EDN QUTEJS.

8. *Левин Л.А.* Односторонние функции // Проблемы передачи информации. – 2003. – Т. 39, № 1. – С. 103–117. – EDN DJXDMX.

9. PostgreSQL Global Development Group. *Server Programming Interface* // PostgreSQL Documentation. – URL: https://www.postgresql.org/docs/current/spi.html (accessed: 02.09.2025).

10. *Лёвин В.Ю.* О повышении криптостойкости однонаправленных хеш-функций // Фундаментальная и прикладная математика. – 2009. – Т. 15, № 5. – С. 171–179. – EDN MXSFIX.

11. *Котов Ю.А.* Приложения шифров. Криптоанализ. – Новосибирск: Изд-во НГТУ, 2019. – 76 с.

12. *Brown S.* The C4 model for visualising software architecture. – URL: https://c4model.com/ (accessed: 05.05.2025).

13. Криптоанализ хэш-функции ГОСТ Р 34.11–2012. – URL: https://habr.com/ru/post/210684/ (дата обращения: 02.09.2025).

14. *Гадасия Д.В., Шведов А.В., Пантелеева К.А.* Предобработка информации для систем машинного обучения // Актуальные проблемы и перспективы развития экономики: труды XXI Международной научно-практической конференции, Симферополь – Гурзуф, 20–22 октября 2022 года. – Симферополь, 2022. – С. 268–269. – EDN QVIOMF.

15. *Дроботун Е.* Импортозамещенное шифрование глазами программиста. Хэшируем по ГОСТ 34.11–2012. – URL: https://xakep.ru/2016/07/20/hash-gost-34-11-2012/ (дата обращения: 02.09.2025).

Котов Юрий Алексеевич, кандидат физико-математических наук, доцент, доцент кафедры защиты информации Новосибирского государственного технического университета. Основное направление научных исследований – информационная и компьютерная безопасность, криптография и криптоанализ, математическое обеспечение вычислительных систем. Имеет более 35 публикаций. E-mail: kotov@corp.nstu.ru

Втюрин Александр Романович, лаборант кафедры защиты информации Новосибирского государственного технического университета. Область научных интересов – разработка серверного и десктопного программного обеспечения для защиты информации. E-mail: a.vtyurin.2021@stud.nstu.ru

DOI: 10.17212/2782-2230-2025-3-23-39

Developing a server application to protect against SQLi attacks in PostgreSQL *

Yu.A. Kotov[1], A.R. Vtyurin[2]

[1] *Novosibirsk State Technical University, 20 Karl Marx Prospekt, Novosibirsk, 630073, Russian Federation, candidate of physical and mathematical sciences, docent, docent of the information security department. E-mail: kotov@corp.nstu.ru*
[2] *Novosibirsk State Technical University, 20 Karl Marx Prospekt, Novosibirsk, 630073, Russian Federation, Laboratory Assistant of the Information Security Department. E-mail: a.vtyurin.2021@stud.nstu.ru*

The article addresses the critical problem of protecting PostgreSQL databases against SQL injection (SQLi) attacks, which pose a serious threat to data integrity and confidentiality. To solve this problem, we propose a server extension method integrated at the DBMS kernel level. The extension employs a query interception mechanism via system hooks (ProcessUtility_hook, ExecutorStart_hook) and verifies queries using the GOST R 34.11-2018 "Streebog" cryptographic hashing algorithm.
Key implementation aspects are examined:

- Sanitization of SQL queries by removing variable data (attribute values) to analyze structure;
- Comparison of sanitized query hashes against an "allowlist" of permitted patterns;
- Implementation of policy management functions (hash whitelisting, verification toggle);
- Detailed operation logging.

The solution provides kernel-level protection without modifying application code, complying with Russian information security standards (FSTEC, 152-FZ). The developed extension is designed for data-sensitive systems with heightened security requirements.

Keywords: PostgreSQL, SQL injection, database security, PostgreSQL event triggers, PostgreSQL extension, hash function, GOST R 34.11-2018, information security

REFERENCES

1. Yurchenko A.S., Yasharov D.A., Efimenko K.N. [Some aspects of information system security]. *Programmnaya inzheneriya: metody i tekhnologii razrabotki informatsionno-vychislitel'nykh sistem (PIIVS-2018)* [Software Engineering: Methods and Technologies for Developing Information and Computing Systems (PIIVS-2018)]. Proceedings of the II International Scientific and Practical Conference (Student Section). In 2 vol. Vol. 2. Donetsk, 2018. Vol. 2. Donetsk, 2018, pp. 218–222. (In Russian).

2. Burmistrov A.V., Belov Yu.S. Nedostatki relyatsionnykh baz dannykh [Disadvantages of relational databases]. *Elektronnyy zhurnal: nauka, tekhnika i obra-*

* *Received 01 July 2025.*

zovanie = Electronic Journal: Science, Technology and Education, 2015, no. 3 (3), pp. 25–34.

3. Bakirov A.M., Rychkov V.A., Rychkova V.I. [Analysis of database management system vulnerabilities and how to detect them based on SQL injection attack]. *MSI: 10 let podgotovki kadrov dlya mezhdunarodnoi sistemy POD/FT* [MSI: 10 years of training personnel for the international AML/CFT system]. Proceedings of the IX International scientific and practical conference of the International Network Institute in the Field of AML/CFT. Moscow, MIFI Publ., 2023, pp. 128–133. (In Russian).

4. Burlevich M.V. [Utility for detecting and exploiting blind SQL injections]. *Studencheskaya nauchnaya vesna* [Student Scientific Spring]. Proceedings of the All-Russian student conference dedicated to the 110th anniversary of Academician V.N. Chelomey. Moscow, 2024, pp. 214–215. (In Russian).

5. Misbakhov N.I., Lukyanov E.R., Stepanov M.O.A.A.M. [Research on methods for exploiting SQL]. *Aktual'nye problemy nauki i obrazovaniya v usloviyakh sovremennykh vyzovov* [Actual problems of science and education in the context of modern challenges]. Proceedings of the XXVIIII International scientific and practical conference, Moscow, March 01, 2024. St. Petersburg, 2024, pp. 59–62. (In Russian).

6. Uspenskiy E.N., Starikov A.S., Romashkina G.V., Norkina A.N. [Adaptive detection of malicious queries in web attacks]. *Aktual'nye problemy menedzhmenta, ekonomiki i ekonomicheskoi bezopasnosti* [Actual problems of management, economics and economic security]. Proceedings of the International scientific conference, Kostanay, May 27–29, 2019. Cheboksary, 2019, pp. 308–311. DOI: 10.31483/r-32922.

7. Yakovenko N.V., Chizhova E.M., Tremasova L.A., Gadasin D.D. Preduprezhdenie atak, baziruyushchikhsya na SQL-in"ektsiyakh [Prevention of SQL injection-based attacks]. *REDS: Telekommunikatsionnye ustroistva i sistemy = REDS: Telecommunications Devices and Systems*, 2023, vol. 13, no. 3, pp. 41–48.

8. Levin L.A. Odnostoronnie funktsii [The tale of one-way functions]. *Problemy peredachi informatsii = Problems of Information Transmission*, 2003, vol. 39, no. 1, pp. 103–117. (In Russian).

9. PostgreSQL Global Development Group. Server Programming Interface. *PostgreSQL Documentation*. Available at: https://www.postgresql.org/docs/current/spi.html (accessed 02.09.2025).

10. Levin V.Yu. O povyshenii kriptostoikosti odnonapravlennykh kheshfunktsii [The increasing of hash functions security]. *Fundamental'naya i prikladnaya matematika = Fundamental and Applied Mathematics*, 2009, vol. 15, no. 5, pp. 171–179.

11. Kotov Yu.A. *Prilozheniya shifrov. Kriptoanaliz* [Applications of Ciphers. Cryptanalysis]. Novosibirsk, NSTU Publ., 2019. 76 p.

12. Brown S. *The C4 model for visualising software architecture*. Available at: https://c4model.com/ (accessed 05.05.2025).

13. Cryptanalysis of GOST R 34.11-2012 Hash Function. (In Russian). Available at: https://habr.com/ru/post/210684/ (accessed 02.09.2025).

14. Gadasiya D.V., Shvedov A.V., Panteleeva K.A. [Data preprocessing for machine learning systems]. *Aktual'nye problemy i perspektivy razvitiya ekonomiki* [Actual problems and development prospects of the economy]. Proceedings of the XXI International scientific and practical conference. Simferopol, 2022, pp. 268–269. (In Russian).

15. Drobotun E. *Importozameshchennoe shifrovanie glazami programmista. Kheshiruem po GOST 34.11–2012* [Import-substituted encryption through a programmer's eyes. Hashing according to GOST 34.11–2012]. Available at: https://xakep.ru/2016/07/20/hash-gost-34-11-2012/ (accessed 02.09.2025).

Для цитирования:

Котов Ю.А., Втюрин А.Р. Разработка серверного приложения для защиты от SQLi-атак в PostgreSQL // Безопасность цифровых технологий. – 2025. – № 3 (118). – С. 23–39. – DOI: 10.17212/2782-2230-2025-3-23-39.

For citation:

Kotov Yu.A., Vtyurin A.R. Razrabotka servernogo prilozheniya dlya zashchity ot SQLi-atak v PostgreSQL [Developing a server application to protect against SQLi attacks in PostgreSQL]. *Bezopasnost' tsifrovykh tekhnologii = Digital Technology Security*, 2025, no. 3 (118), pp. 23–39. DOI: 10.17212/2782-2230-2025-3-23-39.