УДК 004.056:004.774

АНАЛИЗ МЕТОДОВ ЗАЩИТЫ ВЕБ-ПРИЛОЖЕНИЙ ОТ АТАК ТИПА SQL-ИНЪЕКЦИЯ

Сальникова В.А., студент гр 361402

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Мокеров В.С. – ассистент каф. ЗИ

Аннотация. В статье проводится теоретический анализ методов защиты веб-приложений от SQL-инъекций. Рассмотрены параметризованные запросы, хранимые процедуры, валидация, экранирование, ORM, WAF и принцип наименьших привилегий. Выполнено сравнение методов по надёжности, универсальности и ограничениям. Обоснована необходимость эшелонированной защиты (defense-in-depth). Наиболее эффективной признана комбинация параметризованных запросов, валидации по белому списку, WAF и минимизации привилегий СУБД.

Ключевые слова. SQL-инъекция, параметризованные запросы, WAF, эшелонированная защита, валидация, prepared statements.

SQL (Structured Query Language)-инъекции остаются одной из ключевых угроз информационной безопасности веб-приложений, входя в топ-3 наиболее критических уязвимостей по классификации OWASP (Open Web Application Security Project) Top 10 за 2021 год, с сохранением актуальности в 2024-2026 годах. По данным OWASP, более 67% современных веб-приложений содержат уязвимости к инъекциям, что приводит к утечкам миллионов записей персональных данных ежегодно; в России и Беларуси в 2024 году через SQLi скомпрометировано свыше 3 млн записей, а глобальные потери от таких атак превышают 6 трлн долларов. В Республике Беларусь государственные учреждения подверглись атакам в 22% случаев, промышленные предприятия – в 14%, что подтверждает высокую актуальность проблемы для национальной ИТ-инфраструктуры [1].

Виды атак SQL-инъекций включают in-band (union-based, error-based), blind (boolean-based, time-based) и out-of-band методы, различающиеся по способу эксфильтрации данных и техническим условиям эксплуатации. Эти атаки позволяют злоумышленникам обходить аутентификацию, извлекать конфиденциальные данные, модифицировать или уничтожать информацию в базах данных, что особенно опасно для корпоративных и государственных систем [1].

SQL-инъекция (SQLi) представляет собой метод атаки на веб-приложения, при котором злоумышленник манипулирует SQL-запросами через пользовательский ввод, внедряя вредоносный код в строки запросов к базе данных. Это происходит из-за отсутствия должной проверки и санитизации входных данных в приложении, что позволяет атакующему изменить логику запроса, например, добавить условия типа «OR 1=1 --», обходя аутентификацию. В результате возможны несанкционированный доступ, кража данных или их модификация [2].

SQL-инъекции появились в конце 1990-х, когда веб-приложения на PHP (Hypertext Preprocessor), ASP (Active Server Pages) и других платформах стали массово использоваться для работы с базами данных. С ростом популярности веб-технологий разработчики часто пренебрегали правильной обработкой пользовательского ввода, что открыло путь для атак на SQL-запросы. Сегодня это одна из наиболее частых и разрушительных угроз для веб-приложений: успешная атака дает злоумышленнику полный контроль над базой данных [3].

В зависимости от способа получения доступа к данным бэкенд-сервера и потенциальных масштабов ущерба SQL-инъекции можно разделить на три категории: внутриполосная атака, инференциальная атака и внеполосная атака (Out-of-band SQLi).

Внутриполосная атака (In-band SQLi) – это самый простой вид атаки для злоумышленников, так как для реализации атаки и сбора результатов используется один и тот же канал связи. Этот тип SQLi-атак разделяют на два подвида: атака на основе ошибок (Error-based SQLi) и атака на основе объединения (Union-based SQLi).

При Error-based SQLi действия злоумышленника приводят к тому, что база данных генерирует сообщение об ошибке. На основе полученных сообщений об ошибках злоумышленник пытается сформировать представление об инфраструктуре базы данных.

В случае атаки на основе объединения (Union-based SQLi) атакующий получает необходимые данные путем объединения нескольких инструкций SELECT в единый ответ HTTP (HyperText Transfer Protocol) с помощью SQL-оператора UNION.

Инференциальная атака (Inferential SQLi, также известна как «слепая SQL-инъекция») – это атака, при которой злоумышленники изучают ответы и поведение сервера после отправки наборов данных, чтобы узнать больше о структуре базы данных. При этом никакие записи из базы данных веб-сайта не передаются злоумышленнику, и он не видит их в том же канале связи, как в случае

внутриполосной атаки (этим и объясняется название «слепая SQL-инъекция»). Такие атаки разделяют на два подвида: слепая атака, основанная на времени, и булевая слепая атака.

При слепой атаке, основанной на времени (Time-based SQLi) атакующие направляют SQL-запрос к базе данных, вынуждая ее сделать задержку на несколько секунд, прежде чем она подтвердит или опровергнет полученный запрос.

При булевой слепой атаке (Boolean SQLi) атакующие делают SQL-запрос к базе данных, ожидая получить результат в виде утвердительного или отрицательного ответа [4].

Внеполосная атака (Out-of-band SQLi) – это атака, при которой данные извлекаются через DNS (Domain Name System)-запросы, HTTP-вызовы или электронную почту. Происходит она в двух случаях:

– когда атакующие не могут провести атаку и собрать данные через один и тот же канал связи;

– когда сервер работает слишком медленно или нестабильно, чтобы достичь нужного результата.

Методы защиты веб-приложений от SQL-инъекций традиционно классифицируются по нескольким основаниям: по уровню внедрения (код приложения, серверная инфраструктура, СУБД), по принципу действия (профилактические, детектирующие, блокирующие) и по степени охвата (точечные, комплексные).

Методы на уровне приложения являются первичным и наиболее критичным барьером защиты, поскольку непосредственно контролируют формирование SQL-запросов.

Параметризованные запросы (prepared statements) признаны наиболее надёжным методом защиты на уровне кода приложения. Сущность метода заключается в предварительной компиляции шаблона SQL-запроса с плейсхолдерами (место держателями), после чего пользовательские данные передаются отдельно и интерпретируются исключительно как данные, но не как исполняемый код.

Теоретическая основа метода базируется на принципе разделения кода и данных. При использовании параметризованных запросов СУБД чётко различает структуру команды и значения параметров, что делает невозможным изменение синтаксиса запроса злоумышленником.

Хранимые процедуры представляют собой набор предварительно скомпилированных SQL-инструкций, хранящихся непосредственно на сервере базы данных. При корректной реализации хранимые процедуры могут обеспечить защиту, аналогичную параметризованным запросам, поскольку пользовательский ввод передаётся в процедуру через параметры.

Однако, как отмечают исследователи, эффективность хранимых процедур напрямую зависит от качества их реализации: если внутри процедуры динамически конкатенируются параметры, защитный эффект полностью нивелируется. Кроме того, избыточное использование хранимых процедур может привести к смешению бизнес-логики на уровне СУБД, что затрудняет сопровождение и масштабирование приложения.

Валидация входных данных – это процесс проверки пользовательского ввода на соответствие ожидаемому формату, типу, длине или диапазону значений. Санитизация (очистка) – процесс удаления или модификации потенциально опасных символов. В контексте защиты от SQL-инъекций эти методы играют вспомогательную, но важную роль, поскольку не могут полностью гарантировать безопасность, но значительно снижают риск успешной атаки.

Валидация базируется на принципе «белого списка» (whitelist), когда допускаются только строго определённые значения (например, цифры для поля возраста, формат email), в отличие от «чёрного списка» (blacklist), который пытается отфильтровать известные опасные последовательности. «Чёрные списки» считаются ненадёжными, так как злоумышленник может использовать обходные техники (вариации регистра, кодирование, комментарии SQL).

Основные подходы к валидации:

– типизация данных (type checking): принудительное преобразование в целое число (intval(), (int)), число с плавающей точкой, булевый тип;

– проверка по регулярным выражениям (regex validation): для полей с фиксированным форматом (дата, телефон, почтовый индекс);

– проверка по длине (length validation): ограничение количества символов, что затрудняет внедрение длинных полезных нагрузок;

– семантическая валидация: проверка на допустимые значения (например, месяц от 1 до 12).

Ограничением валидации является то, что она не может применяться ко всем типам данных (например, текстовые поля произвольного содержания, комментарии пользователей). В таких случаях валидация неэффективна, и требуется использование параметризованных запросов или экранирования.

Экранирование – это метод, при котором специальные символы, имеющие значение в SQL (например, одинарная кавычка «'», двойная кавычка «"», обратный слеш «\», символ процента «%», подчёркивание «_»), предваряются escape-символом (обычно обратным слешом) или заменяются на безопасные последовательности.

Экранирование имеет ряд недостатков. Например, зависимость от кодировки: при некорректно настроенной кодировке возможен обход экранирования через многобайтовые символы (атака

«экранирование через мультибайт»). Также экранирование защищает только строковые литералы, но не другие части запроса (имена таблиц, столбцов, ключевые слова, операторы). При таком методе защите присутствует большой риск человеческого фактора: разработчик может забыть применить экранирование к какому-либо полю ввода.

В современной практике экранирование не рекомендуется как основной метод защиты. Оно может использоваться лишь в legacy-системах, где невозможно внедрение параметризованных запросов.

ORM (Object-Relational Mapping) – это технология, позволяющая работать с базой данных через объекты языка программирования, а не через написание SQL-запросов вручную. Современные ORM (Entity Framework для C#, Hibernate для Java, SQLAlchemy для Python, Eloquent для PHP) автоматически параметризуют запросы, что обеспечивает защиту от SQL-инъекций при условии корректного использования. ORM генерирует SQL-запросы на основе методов, вызванных разработчиком. Внутренняя реализация этих методов, как правило, использует подготовленные выражения или безопасное связывание параметров.

Преимущества ORM с точки зрения защиты:

– минимизация ручного написания SQL-кода, снижение вероятности ошибок;

– единая точка управления экранированием/параметризацией;

– дополнительные механизмы валидации на уровне моделей.

Потенциальные риски:

– использование запросов (raw queries) внутри ORM, которые обходят встроенную защиту;

– уязвимости в самом ORM (редко, но возможны);

– неправильная конфигурация, отключающая параметризацию.

Таким образом, применение ORM является эффективным методом защиты, особенно в сочетании с другими практиками безопасной разработки.

Принцип наименьших привилегий для учётных записей БД заключается в том, что учётная запись, под которой веб-приложение подключается к СУБД, должна иметь только необходимые привилегии (SELECT, INSERT, UPDATE, DELETE для конкретных таблиц) и не должна обладать правами на создание, удаление таблиц, выполнение системных хранимых процедур, запись в файлы и т.д. Если злоумышленник всё же сумеет внедрить SQL-код, ограниченные привилегии не позволят ему выполнить деструктивные действия (DROP TABLE, xp_cmdshell, INTO OUTFILE). Это принцип «защиты в глубину» (defense-in-depth).

Инфраструктурные и серверные методы защиты реализуются вне кода приложения, а на уровне веб-сервера, межсетевого экрана, специализированных устройств или программных модулей. Они обеспечивают дополнительный барьер, который может блокировать атаки даже при наличии уязвимостей в коде.

Web Application Firewall (WAF) – это специализированное программное или аппаратное средство, анализирующее HTTP-трафик между пользователем и веб-приложением и блокирующее подозрительные запросы на основе заданных правил. WAF может функционировать в различных режимах: блокирующий, регистрирующий, пассивный (только мониторинг).

Методами анализа в WAF являются сигнатурный анализ, поведенческий анализ и анализ на основе грамматики SQL.

Сигнатурный анализ (signature-based) – это сопоставление запросов с базой известных атакующих шаблонов. Преимуществом является высокая скорость и точность для известных атак. Недостатком – невозможность обнаружить новые (zero-day) атаки или модифицированные payloads.

Поведенческий анализ (behavioral analysis) – это построение профиля нормального трафика и выявление аномалий (например, резкое увеличение длины запроса, появление необычных символов). Преимуществом является способность обнаруживать неизвестные атаки, а недостатком – возможность ложных срабатываний.

Анализ на основе грамматики SQL – это парсинг предполагаемого SQL-запроса с проверкой соответствия грамматике. Позволяет выявить попытки изменения структуры запроса.

Преимущества WAF:

– независимость от кода приложения: защита может быть применена к legacy-системам без модификации исходного кода;

– централизованное управление политиками безопасности;

– возможность быстрого реагирования на новые угрозы через обновление сигнатур.

Недостатки WAF:

– не защищает от атак, использующих обход сигнатур (например, кодирование, фрагментация);

– может быть обойдён при наличии уязвимостей в самом WAF;

– ложные срабатывания могут блокировать легитимных пользователей.

Системы обнаружения вторжений (IDS (Intrusion Detection System)) и предотвращения вторжений (IPS (Intrusion Prevention System)) могут работать на сетевом уровне и уровне хоста. В контексте SQL-инъекций IPS способен анализировать SQL-трафик между приложением и СУБД,

выявляя аномалии (например, необычно большое количество запросов, подозрительные команды). Однако на практике специализированные WAF более эффективны для веб-приложений.

Ограничение прав доступа к файловой системе и сети включает в себя:

– запрет на использование «LOAD_FILE()», «INTO OUTFILE» и других файловых функций СУБД путём отключения соответствующих привилегий;

– настройку сетевых экранов для ограничения доступа к СУБД только с IP (Internet Protocol)-адресов веб-серверов;

– использование безопасных конфигураций СУБД.

Эшелонированная защита (defense-in-depth) предполагает одновременное применение методов на всех уровнях: приложения, СУБД, инфраструктуры. Поскольку все методы действуют независимо, общая устойчивость стремится к нулю, если хотя бы один из коэффициентов близок к нулю. Таким образом, наиболее надёжной признаётся комбинация методов, перекрывающих слабые места друг друга [5] [6].

Проведём теоретический анализ рассмотренных методов защиты, сравнивая их по степени надёжности, универсальности, наличию принципиальных ограничений и роли в общей системе защиты.

Параметризованные запросы (prepared statements) признаются наиболее надёжным методом на уровне приложения. Их ключевое преимущество перед всеми остальными методами заключается в принципиальном разделении кода и данных: SQL-запрос компилируется до передачи пользовательских параметров, что делает невозможным интерпретацию ввода как исполняемого SQL-кода. В отличие от экранирования, параметризация не зависит от кодировок и не подвержена мультибайтовым атакам. В отличие от валидации, она применима к любым типам данных, включая текстовые поля произвольного содержания. Однако параметризованные запросы имеют одно принципиальное ограничение: они не работают для динамических идентификаторов (имён таблиц, столбцов, ключевых слов SQL), тогда как валидация по белому списку или экранирование в этой узкой области могли бы применяться.

Хранимые процедуры теоретически могут обеспечить уровень защиты, сравнимый с параметризованными запросами, но только при жёстком условии: внутри процедуры не должно быть динамической конкатенации параметров. Если это условие нарушается, защитный эффект полностью исчезает, что делает хранимые процедуры менее надёжными, чем параметризованные запросы, поскольку контроль за безопасностью ложится на разработчика процедуры. Кроме того, хранимые процедуры менее универсальны: их поддержка и синтаксис различаются между СУБД, а избыточное использование ведёт к смешению бизнес-логики на уровне базы данных.

Валидация по белому списку занимает вспомогательное место. В отличие от параметризованных запросов, она не даёт абсолютной гарантии даже для полей, где применяется, так как злоумышленник может найти неучтённый вариант допустимого значения. По сравнению с экранированием, валидация безопаснее, так как не зависит от кодировок, но её область применения уже: она принципиально неприменима к текстовым полям произвольного содержания (комментарии, описания, сообщения), тогда как экранирование и параметризация работают и там. Таким образом, валидация – не самостоятельный метод предотвращения SQL-инъекций, а дополнительный барьер для полей с фиксированным форматом.

Экранирование – наименее надёжный метод среди всех рассматриваемых. По сравнению с параметризованными запросами, оно обладает фундаментальными недостатками: зависимость от корректной настройки кодировки, защита только строковых литералов (не распространяется на имена таблиц и столбцов), высокий риск пропуска разработчиком какого-либо поля ввода. В отличие от валидации, экранирование не требует знания формата данных, но платит за это снижением надёжности. В современной практике экранирование уступает параметризованным запросам по всем критериям и рекомендуется только для legacy-систем, где невозможно переписать код на подготовленные выражения.

ORM (объектно-реляционное отображение) по своей защитной эффективности близка к параметризованным запросам, поскольку современные ORM автоматически параметризуют генерируемые запросы. Однако ORM имеет специфическое ограничение, которого нет у прямых параметризованных запросов: разработчик может использовать запросы (raw queries) в обход ORM, и тогда защита исчезает. Таким образом, ORM не добавляет принципиально новой защиты по сравнению с prepared statements, но снижает вероятность человеческой ошибки за счёт минимизации ручного написания SQL. С точки зрения накладных расходов ORM несколько тяжелее из-за абстракции и генерации кода.

WAF (межсетевой экран уровня веб-приложений) принципиально отличается от всех перечисленных методов тем, что работает вне кода приложения. В отличие от параметризованных запросов и ORM, WAF не требует изменения исходного кода, что позволяет защищать legacy-системы. Однако по полноте защиты WAF уступает параметризованным запросам: сигнатурный WAF бесполезен против новых или модифицированных атак, а поведенческий может давать ложные срабатывания. Более того, WAF может быть обойдён через кодирование, фрагментацию или

комментарии SQL, тогда как параметризованные запросы от таких обходов защищают принципиально. Следовательно, WAF –это дополнительный, а не основной барьер.

Принцип наименьших привилегий занимает особое место. Он не предотвращает саму SQL-инъекцию – в этом его коренное отличие от параметризованных запросов, валидации, экранирования, ORM и WAF. Однако он ограничивает последствия успешной атаки, запрещая деструктивные действия. Ни один другой метод не даёт такой защиты «на случай неудачи». Поэтому принцип наименьших привилегий является обязательным дополнением к любому из вышеперечисленных методов, но не может их заменить.

Эшелонированная защита (defense-in-depth) представляет собой комбинацию методов разных уровней. Сравнительный анализ показывает, что ни один из методов не является абсолютно надёжным изолированно. Параметризованные запросы не работают для динамических идентификаторов – эту нишу закрывает валидация по белому списку. WAF защищает от атак, вызванных человеческими ошибками (например, забытый prepared statement). Принцип наименьших привилегий минимизирует ущерб, если все предыдущие рубежи пройдены. Таким образом, комбинирование методов даёт синергетический эффект, недостижимый при использовании любого из них по отдельности.

По итогам анализа получено, что наиболее надёжной теоретической моделью признаётся сочетание параметризованных запросов (как основного метода на уровне кода), валидации по белому списку (для динамических идентификаторов), WAF (как дополнительного барьера) и принципа наименьших привилегий (для ограничения ущерба). Экранирование и хранимые процедуры (при небезопасной реализации) уступают по надёжности и не рекомендуются в качестве основных методов.

Список использованных источников:

1. CODEBY.NET [Электронный ресурс]. – Режим доступа: https://codeby.net/threads/10-kriticheskikh-uyazvimostei-veb-infrastruktury-ot-idor-do-sql-injection-polnoye-rukovodstvo-2025.72703/#-2-sql-injection-topovaya-ugroza-baz-dannykh – Дата доступа: 06.04.2026.

2. Security Vision [Электронный ресурс]. – Режим доступа: https://www.securityvision.ru/blog/chto-takoe-sql-inektsiya-klassifikatsiya-sql-inektsiy-sqli-ataki-metody-zashchity/ – Дата доступа: 06.04.2026.

3. FoxmindEd [Электронный ресурс]. – Режим доступа: https://foxminded.ua/ru/sql-inekcii/ – Дата доступа: 06.04.2026.

4. Kaspersky [Электронный ресурс]. – Режим доступа: https://www.kaspersky.ru/resource-center/definitions/sql-injection – Дата доступа: 06.04.2026.

5. OWASP [Электронный ресурс]. – Режим доступа: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_ Prevention_Cheat_Sheet.html – Дата доступа: 06.04.2026

6. IDS00-J. Prevent SQL injection [Электронный ресурс]. – Режим доступа: https://cmu-sei.github.io/secure-coding-standards/sei-cert-oracle-coding-standard-for-java/rules/input-validation-and-data-sanitization-ids/ids00-j/ – Дата доступа: 06.04.2026.

8. System Binary Proxy [Электронный ресурс]. – Режим доступа: https://attack.mitre.org/techniques/T1218/005/ – Дата доступа: 03.04.2026.

9. Rules Syntax [Электронный ресурс]. – Режим доступа: https://documentation.wazuh.com/current/user-manual/ruleset/ruleset-xml-syntax/rules.html – Дата доступа: 03.04.2026.

10. Regular Expression Syntax [Электронный ресурс]. – Режим доступа: https://documentation.wazuh.com/current/user-manual/ruleset/ruleset-xml-syntax/regex.html – Дата доступа: 03.04.2026.

11. Sysmon версии 15.2 [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/sysinternals/downloads/sysmon – Дата доступа: 03.04.2026.

UDC 004.056:004.774

ANALYSIS OF WEB APPLICATION PROTECTION METHODS AGAINST SQL INJECTION ATTACKS

Salnikova V.A., student gr 361402

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Mokerov V.S. – Assistant Professor, Department of Information Security

Annotation. The article provides a theoretical analysis of methods for protecting web applications against SQL injection attacks. The following methods are considered: parameterized queries (prepared statements), stored procedures, input validation, escaping, ORM, WAF, and the principle of least privilege. A comparison of the methods is made in terms of reliability, universality, and limitations. The necessity of defense-in-depth is substantiated. The combination of parameterized queries, whitelist validation, WAF, and database privilege minimization is recognized as the most effective approach.

Keywords. SQL injection, parameterized queries, WAF, defense-in-depth, validation, prepared statements.