safe_dev

Выпуск 0.0.7

июн. 17, 2026

Содержание

1 Общие положения

В Global ERP безопасная разработка является неотъемлемой частью инженерного процесса. Раздел устанавливает обязательные требования и принципы безопасной разработки для сотрудников и контрагентов, участвующих в создании, интеграции, тестировании, выпуске и сопровождении прикладных решений и компонентов платформы Global ERP.

Цель раздела — формализация процесса безопасной разработки, снижение риска включения в поставку уязвимого прикладного кода и непроверенных сторонних компонентов, а также обеспечение единого подхода к контролю безопасности на всех этапах жизненного цикла разработки. Раздел систематизирует реализованные в системе меры, применяемые в рамках подхода *security by design*, а также процесс централизованного учета и устранения выявленных уязвимостей.

1.1 Область применения

Раздел применяется ко всем подразделениям, сотрудникам и внешним контрагентам, участвующим в:

- разработке, интеграции и сопровождении прикладных решений и компонентов платформы Global ERP;
- включении сторонних библиотек и компонентов в поставочный контур;
- сопровождении и выпуске релизов системы, в рамках которых изменяются код, зависимости или конфигурация;
- тестировании, анализе и устранении уязвимостей на всех этапах жизненного цикла.

1.2 Подход к безопасной разработке

Подход к безопасной разработке учитывает архитектуру платформы Global ERP: система использует СУБД PostgreSQL / Postgres Pro, сервер приложений и браузерный клиент, а прикладные решения и интеграционные компоненты поставляются в составе общей системы. В связи с этим контроль безопасности строится вокруг трех основных направлений:

- проверка прикладного кода;
- проверка сторонних зависимостей;
- проверка поведения развернутых веб-интерфейсов и API.

1.3 Основные принципы

Внутренний процесс безопасной разработки строится на следующих принципах:

- проверка безопасности выполняется на нескольких этапах жизненного цикла: при разработке, при сборке, перед выпуском и после устранения выявленных замечаний;
- для прикладного кода применяется обязательный статический анализ;
- изменения прикладного кода проходят рецензирование перед включением в поставочный контур;
- для сторонних библиотек применяется отдельный контроль состава зависимостей;
- для веб-интерфейсов и HTTP/API-контуров применяется динамическая проверка на выделенном стенде;
- для отдельных компонентов и значимых изменений может применяться фаззинг-тестирование как дополнительный метод проверки устойчивости к некорректным и специально искаженным входным данным;
- все новые внешние библиотеки проходят архитектурное согласование до включения в доверенный контур сборки.

Точные правила срабатывания, внутренние профили сканирования, наборы правил проверки и настройки доверенных репозиториев не раскрываются в открытой эксплуатационной документации.

1.4 Ответственность участников процесса

Распределение ответственности между ролями не исключает совместного участия в процессе безопасной разработки. Все участники процесса обязаны соблюдать требования раздела в пределах своей компетенции.

Разработчик

Осуществляет разработку и изменение прикладного кода, обеспечивает соблюдение требований безопасной разработки, прохождение статического анализа и устранение выявленных уязвимостей, а также обосновывает использование сторонних библиотек.

Архитектор

Осуществляет согласование использования сторонних компонентов, оценивает архитектурные и технологические риски и принимает решение о включении библиотек в доверенный контур сборки.

Инженер по тестированию

Участвует в обеспечении безопасности за счет выполнения проверок в процессе тестирования, проведения динамического анализа и верификации исправлений уязвимостей.

Специалист по информационной безопасности

Обеспечивает методологическую поддержку процесса, участвует в анализе уязвимостей и инцидентов, проводит адресные проверки и формирует рекомендации по снижению рисков.

Ответственный за релиз

Контролирует прохождение обязательных проверок, учитывает результаты анализа и принимает решение о допуске версии к поставке либо о возврате на доработку.

1.5 Нормативная база

Процесс безопасной разработки реализуется с учетом следующих нормативных документов:

- Приказ ФСТЭК России от 25.12.2017 № 239 «Об утверждении Требований по обеспечению безопасности значимых объектов критической информационной инфраструктуры Российской Федерации» в части требований к безопасной разработке и жизненному циклу программного обеспечения;
- внутренние нормативные документы ООО «Бизнес Технологии» по информационной безопасности и качеству разработки;
- руководство по безопасной эксплуатации и сопровождению компонентов;
- рекомендации OWASP.

1.6 Термины и сокращения

В разделе используются следующие термины и сокращения:

- **CI/CD** — конвейер автоматизированной сборки, проверки и поставки программного обеспечения.
- **SCA** (*Software Composition Analysis*) — анализ состава сторонних компонентов и зависимостей на наличие известных уязвимостей, устаревших версий и лицензионных ограничений.
- **DAST** (*Dynamic Application Security Testing*) — динамический анализ безопасности развернутых веб-интерфейсов, HTTP-сервисов и прикладных API.
- **SBOM** (*Software Bill of Materials*) — сведения о составе зависимостей и компонентов поставки.
- **CVSS** — система оценки критичности уязвимостей.
- **Фаззинг-тестирование** — метод проверки устойчивости компонента к некорректным, неожиданным или специально искаженным входным данным.
- **Доверенный контур сборки** — контролируемый набор репозиториев, зависимостей и процедур, используемых для формирования поставочных сборок.

2 Защита при разработке

2.1 Рецензирование кода

В процессе безопасной разработки применяется обязательное рецензирование изменений прикладного кода перед включением их в поставочный контур. Рецензирование выполняется как дополнительный уровень контроля поверх автоматических проверок и направлено на выявление дефектов, которые не всегда обнаруживаются средствами статического анализа, анализа зависимостей и типовыми тестами.

При рецензировании кода проверяются:

- корректность реализации логики и отсутствие очевидных дефектов безопасности;
- соблюдение внутренних требований к безопасной работе с пользовательским вводом, запросами к БД, сервисными интерфейсами и механизмами авторизации;
- корректность использования сторонних библиотек и внутренних API;
- отсутствие обходных решений, ослабляющих штатные механизмы безопасности платформы;
- соответствие согласованным архитектурным решениям и принятому стилю реализации.

При необходимости к рецензированию привлекаются специалисты смежных направлений, включая архитекторов, разработчиков платформенных компонентов и специалистов по информационной безопасности. Для изменений, затрагивающих механизмы аутентификации, авторизации, интеграционные API, SQL-логику и иные чувствительные участки, рецензирование выполняется с повышенным вниманием.

Результаты рецензирования учитываются при принятии решения о включении изменения в поставочную сборку. Изменения, по которым выявлены существенные замечания, подлежат доработке и повторному рассмотрению.

2.2 Контроль сторонних библиотек и доверенного контура сборки

Использование сторонних библиотек в прикладной разработке допускается только в рамках контролируемого процесса. Разработчик может подготовить изменение с новой зависимостью, однако включение такой зависимости в поставочный контур возможно только после архитектурного и технического рассмотрения.

На этапе рассмотрения оцениваются:

- необходимость библиотеки;
- совместимость с технологическим стеком Global ERP;
- наличие известных уязвимостей и история их устранения;
- риски лицензирования и сопровождения.

После согласования библиотека включается во внутренний доверенный репозиторий артефактов, используемый для поставочных сборок. Это означает, что релизные и контролируемые сборки формируются только из проверенного набора зависимостей. Использование неразрешенного внешнего компонента в поставочном контуре должно приводить к отклонению такой сборки до прохождения процедуры согласования.

Такой подход позволяет одновременно ограничить попадание непроверенных компонентов в релиз и поддерживать единый управляемый состав библиотек, подлежащих последующему анализу на уязвимости.

2.3 Статический анализ прикладного кода

Для прикладных модулей основной обязательной проверкой является статический анализ на этапе компиляции. В документации Global Framework предусмотрен отдельный раздел, посвященный статическому анализу, включая использование инструмента, добавление правил и разработку собственных правил.

Базовым инструментом статического анализа прикладного Scala-кода является **wartremover**. Сам инструмент позиционируется как гибкий Scala-linter, предназначенный для повышения безопасности и корректности кода; нарушения правил могут приводить к ошибке компиляции.

В рамках процесса безопасной разработки статический анализ решает следующие задачи:

- выявление небезопасных и нежелательных конструкций на раннем этапе;
- контроль соблюдения внутренних правил написания прикладного кода;
- предотвращение включения в релиз кода, не соответствующего принятому набору ограничений;
- поддержка единообразного инженерного стиля в крупной кодовой базе.

Для компонентов, разрабатываемых на иных языках, по решению ответственной команды могут применяться дополнительные локальные средства контроля качества и безопасности. При этом для прикладной разработки Global ERP основной обязательный акцент делается на прикладной сборке и правилах статического анализа Scala-кода.

Статический анализ выполняется как часть обычного инженерного цикла разработки и сборки, включая процесс поставки программного обеспечения, и доступен для использования в рамках проектной разработки. Если нарушение относится к категории блокирующих правил, изменение не считается готовым к включению в поставочный контур до устранения замечаний. За счет этого наиболее типовые дефекты и небезопасные конструкции устраняются до релизной стадии.

2.4 Анализ зависимостей и уязвимостей сторонних компонентов

Отдельный контур контроля в процессе безопасной разработки связан с составом используемых сторонних компонентов, библиотек, а также прямых и транзитивных зависимостей. Цель такого контроля — своевременно выявлять включение в поставочный контур компонентов, содержащих известные уязвимости, устаревшие версии, неподтвержденные источники происхождения либо иные технологические риски. Для этой задачи применяется анализ зависимостей класса SCA (*Software Composition Analysis*).

Проверка состава зависимостей выполняется в рамках внутреннего процесса контроля поставки и распространяется на компоненты, включаемые в релизный контур, а также на новые или измененные внешние зависимости, вводимые в продукт. Анализ ориентирован на сопоставление используемого состава библиотек с актуальными сведениями о выявленных уязвимостях, статусе сопровождения компонентов и ограничениях их допустимого применения в корпоративной среде.

Проверка состава зависимостей выполняется в автоматизированном режиме в составе CI/CD-конвейера и является частью процесса сборки и поставки программного обеспечения. В рамках автоматизированного SCA-тестирования обеспечивается:

- выявление уязвимостей в используемых библиотеках и зависимостях;
- контроль версий компонентов;
- проверка соблюдения лицензионных ограничений.

Анализ зависимостей не подменяет архитектурное согласование библиотек, а дополняет его. Архитектурное согласование определяет допустимость включения компонента в доверенный контур, а SCA-проверка используется для контроля уязвимостей на момент выпуска версии.

В Global ERP такой анализ применяется:

- для релизных и предрелизных сборок;
- для крупных выпусков и значимых технологических изменений;
- при включении новых внешних библиотек в доверенный репозиторий;
- при разборе инцидентов и адресной проверке конкретного компонента.

Для решения задач данного класса применяются специализированные средства анализа состава зависимостей и известных уязвимостей в сторонних компонентах. В качестве примера могут использоваться инструменты класса *Software Composition Analysis*, в том числе OWASP Dependency-Check для JVM-контуров, а также аналогичные средства для иных технологических стеков.

При необходимости заказчик может реализовать аналогичный SCA-анализ в своём контуре: платформа предоставляет информацию о составе зависимостей (SBOM).

Сведения о фактически применяемых внутри средствах анализа, их версиях, профилях сканирования, внутренних источниках данных, правилах классификации результатов и параметрах выполнения проверки в эксплуатационной документации не раскрываются, так как относятся ко внутренним процедурам ограниченного доступа.

2.5 Динамический анализ веб-интерфейсов и API

Для проверки безопасности развернутых веб-интерфейсов, HTTP-сервисов и прикладных API в Global ERP применяется динамический анализ на выделенных тестовых контурах. Его задача — выявление уязвимостей, проявляющихся не в исходном коде, а в фактическом поведении приложения при обработке запросов, аутентификации, авторизации, навигации, пользовательских сценариев и интеграционных вызовов.

Динамическая проверка используется для компонентов, для которых существенны сетевое взаимодействие, веб-доступ, прикладные точки входа и бизнес-сценарии, зависящие от состояния развернутой системы. Такой анализ проводится при значительных изменениях пользовательских интерфейсов и API, при изменении механизмов доступа и аутентификации, а также в рамках адресной проверки исправлений, связанных с информационной безопасностью.

Проверки выполняются на специально подготовленных стендах и могут сочетать регламентные сценарии анализа с дополнительной ручной верификацией наиболее критичных участков. Это позволяет контролировать не только типовые технические уязвимости, но и особенности поведения конкретных модулей в реальном прикладном контексте.

Для решения задач данного класса используются специализированные DAST-инструменты и прокси-средства анализа веб-приложений, например OWASP ZAP и иные аналогичные инструменты, поддерживающие автоматизированные и ручные сценарии проверки веб-контуров.

Динамический анализ не рассматривается как непрерывная проверка каждой промежуточной сборки. Его применение определяется характером изменений, уровнем риска и профилем выпускаемой версии. Динамический анализ применяется:

- перед крупными выпусками;
- при существенных изменениях пользовательского интерфейса, REST/SOAP-сервисов или механизмов аутентификации;
- при адресной проверке исправлений, связанных с информационной безопасностью;
- при дополнительной верификации подозрительных участков по итогам внутреннего или внешнего аудита.

2.6 Дополнительные методы проверки безопасности

Помимо статического анализа, анализа зависимостей и динамической проверки веб-интерфейсов, в процессе безопасной разработки могут применяться дополнительные методы проверки безопасности. Такие проверки используются избирательно, в зависимости от характера изменений, критичности затрагиваемого компонента, архитектурного риска и профиля поставки.

К дополнительным методам проверки относятся:

- проверка реакции системы на ошибочные, неполные и недопустимые действия пользователей и внешних систем;
- ручная верификация сценариев аутентификации, авторизации и разграничения доступа;
- проверка устойчивости прикладных и интеграционных интерфейсов к некорректным, неполным и противоречивым запросам;
- регрессионная проверка ранее исправленных уязвимостей и связанных участков логики;
- адресная проверка изменений, затрагивающих обработку файлов, сериализацию, XML/JSON-обмен и механизмы генерации запросов.

Состав и глубина таких проверок определяются характером выпускаемой версии и не регламентируются как единый обязательный набор для каждой промежуточной сборки.

2.7 Фаззинг-тестирование

Для отдельных компонентов системы может применяться фаззинг-тестирование как дополнительный метод проверки устойчивости к некорректным, неожиданным или специально искаженным входным данным.

Фаззинг-тестирование используется избирательно и, как правило, не выполняется на постоянной основе для каждой сборки. Его применение целесообразно в следующих случаях:

- при значимых изменениях логики обработки внешних данных;
- при изменении форматов обмена, REST/SOAP-интерфейсов, механизмов сериализации и десериализации;
- при доработках, затрагивающих загрузку файлов, XML/JSON-структур и иные потенциально чувствительные точки входа;
- при адресной проверке устойчивости компонента после исправления дефектов безопасности;
- при дополнительной проверке критичных компонентов перед крупными выпусками.

Фаззинг рассматривается как вспомогательный инструмент углубленной проверки и применяется по решению ответственной команды для тех участков системы, где такой метод дает практическую ценность с точки зрения выявления скрытых ошибок обработки данных и дефектов устойчивости.

2.8 Управление уязвимостями

В процессе безопасной разработки и сопровождения используется централизованный подход к управлению выявленными уязвимостями. Он охватывает результаты внутренних и внешних проверок безопасности и обеспечивает полный цикл учета, анализа, устранения и повторной проверки выявленных замечаний.

Все уязвимости, выявленные в результате статического анализа, анализа зависимостей, динамических проверок, дополнительных методов тестирования, внутреннего рецензирования и внешних работ по анализу защищенности, подлежат регистрации во внутреннем реестре уязвимостей.

Для каждой выявленной уязвимости фиксируются:

- идентификатор записи;
- дата регистрации;
- источник обнаружения;
- краткое описание проблемы;
- затронутый компонент, модуль или поставка;
- уровень критичности;
- техническое описание, включая предполагаемый сценарий эксплуатации и потенциальные последствия;
- идентификатор задачи и сведения об ответственном;
- сведения о повторной проверке;
- статус обработки;
- релиз компонента с исправлениями.

Уязвимости классифицируются по уровню критичности в соответствии со стандартом CVSS:

- низкий уровень;

- средний уровень;
- высокий уровень;
- критический уровень.

Критические и высокие уязвимости рассматриваются в приоритетном порядке. Уязвимости среднего и низкого уровня подлежат регистрации, учету и устранению в соответствии с внутренними регламентами сопровождения. Решение о допуске версии к поставке принимается с учетом наличия или отсутствия незакрытых уязвимостей недопустимого уровня риска.

После устранения уязвимости выполняется повторная проверка соответствующего компонента или сценария. При необходимости проверка охватывает не только исправленный участок, но и связанные с ним функции, чтобы исключить повторное проявление дефекта в смежной логике.

Примеры классов уязвимостей, выявляемых и устраняемых в рамках данного процесса, приведены в таблице:

Класс уязвимости	Уровень риска	Выявленные сценарии	Реализованные меры
Внедрение SQL-кода (SQL Injection)	Высокий	Возможность влияния пользовательского ввода на формируемые SQL-запросы и получение данных из БД	Параметризация запросов, централизованная валидация и экранирование пользовательских данных, аудит ORM-механизмов. Практические правила защиты от SQL-инъекций приведены в разделе SQL-инъекции
Недостаточная авторизация	Высокий	Выполнение пользователем действий, не соответствующих его привилегиям	Строгая серверная проверка прав доступа, централизованная ролевая модель, контроль доступа к REST- и SOAP-интерфейсам
Внедрение внешних сущностей XML (XXE)	Высокий / Средний	Обработка XML-данных с поддержкой внешних сущностей и DTD	Отключение обработки DTD и внешних сущностей, ограничение используемых форматов данных
Раскрытие информации в сообщениях об ошибках	Средний	Раскрытие установочных путей и внутренней структуры приложения	Централизованная обработка ошибок, вывод минимальной информации пользователю, регистрация подробностей только во внутренних журналах
Подделка запроса со стороны сервера (SSRF)	Средний	Возможность выполнения HTTP-запросов к внутренним сервисам	Ограничение допустимых схем и адресов, фильтрация целевых ресурсов, контроль сетевых взаимодействий
Использование ПО с известными уязвимостями	Средний	Использование уязвимых версий сторонних компонентов	Регулярный аудит зависимостей, контроль версий, регламент обновления программного обеспечения
Подбор учетных данных	Средний	Многократные попытки аутентификации через точки входа системы	Ограничение количества попыток входа, отказ от небезопасных схем аутентификации, применение защищенных механизмов входа. Раздел Защита от подбора паролей
Межсайтовое выполнение сценариев (XSS)	Средний	Отображение неэкранированных пользовательских данных в интерфейсе	Обязательное экранирование пользовательского ввода перед отображением. Практические правила обработки HTML-ввода приведены в разделе Защита от HTML-инъекций

2.9 Принятие решения о выпуске

Решение о допуске новой версии системы к поставке принимается по совокупности результатов контроля, включая:

- успешное прохождение прикладной сборки и обязательных статических проверок;
- отсутствие неразрешенных нарушений по блокирующим правилам статического анализа;
- завершение проверки состава зависимостей для релизной сборки;
- выполнение динамической проверки для тех компонентов, для которых она предусмотрена регламентом выпуска или профилем изменений.

Если по итогам проверки выявлены уязвимости или существенные замечания, они подлежат регистрации, классификации, устранению и повторной проверке.

Для поставочных сборок действует принцип недопустимости выпуска версии с неустраненными уязвимостями высокого и критического уровня риска. Конкретный порог приемки определяется внутренним регламентом выпуска и требованиями конкретного проекта, однако по умолчанию релиз должен быть очищен от незакрытых уязвимостей, требующих обязательного устранения до передачи заказчику. Исключения допускаются только по формализованной процедуре с указанием компенсирующих мер и ответственных лиц.

2.10 Учет результатов проверок

Результаты проверок учитываются при анализе готовности изменений к поставке. При выявлении замечаний выполняется их устранение и, при необходимости, повторная проверка соответствующих компонентов. Результаты выполненных проверок и устранения выявленных замечаний также используются для совершенствования архитектуры *security by design* и применяемых защитных механизмов.

2.11 Обучение и осведомленность

Все сотрудники, участвующие в разработке, интеграции и сопровождении компонентов Global ERP, обязаны:

- проходить обучение по безопасной разработке, методам контроля кода и управления зависимостями;
- поддерживать актуальный уровень знаний о применимых угрозах и уязвимостях;
- применять полученные знания при выполнении задач, включая разработку, тестирование и приемку релизов.

Обучение проводится при приеме на работу, при обновлении используемых инструментов или технологий, а также по результатам выявленных инцидентов безопасности. Сотрудники информируются о новых внутренних правилах, изменениях профилей сканирования и актуальных рекомендациях по безопасной разработке.