GUARD CONF

Исключение и принятие рисков ИБ без владельца и коммитментов — это бэкдор

Как вести исключения из проверок и не тормозить релизы

Кейсы перехода от разового аудита к контролю на каждый PR



Илья Шмаков

Руководитель направления AppSec
СберСпасибо

Спасибо

FIN DEV SEC OPS



Продукт становится частью контура с «историческими» недостатками

тысячи

репозиториев под одним гейтом

сотни

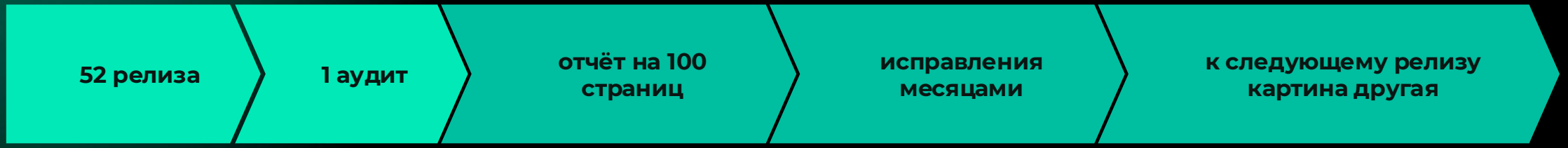
релизов в год

~1-3

команд AppSec на весь портфель

- Финтех-вендор для регулируемых заказчиков: инцидент у вендора — инцидент у клиента и штраф по договору (при должной осмотрительности и доказуемости)
- Каждый день код уходит в прод — проверки не успевают за релизами, ведут к принятию рисков, отсутствия мер митигации и возможности отслеживания (из-за массовости)
- Цель: перестать узнавать о дырах в моменте времени и научиться контролируемым допущениям/ ограничениям

Один аудит в год не успевает за релизами



- Находка приходит не автору и не вовремя — в квартальный отчёт, а не в PR
- У долга нет владельца: решения по каждой находке никто не принимает, либо давит AppSec вызывая негатив у команд или «неприязнь» к лишним доработкам
- Пока чинили старое, релизы обрастали новыми «историческими» недостатками

Безопасность как событие, а не процесс — «стрельнет, тогда решим»

Один полный прогон — ~300 сработок и ~120 критичных

~312

сработок на секреты в одном репозитории

92% — UUID и кэш-ключи тестовых фикстур,

8% — значения по форме учёток: значение в истории = учётка у всех, кто клонировал (даже исторических)

~119

критичных и высоких из ~360 зависимостей одного сервиса

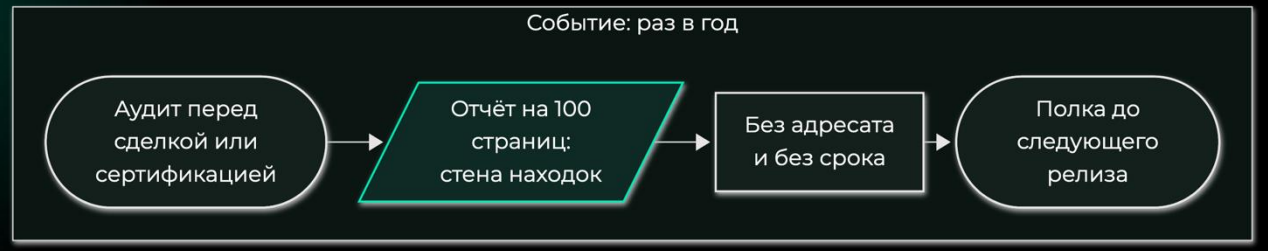
119 находок без владельца = 119 решений, которых никто не примет

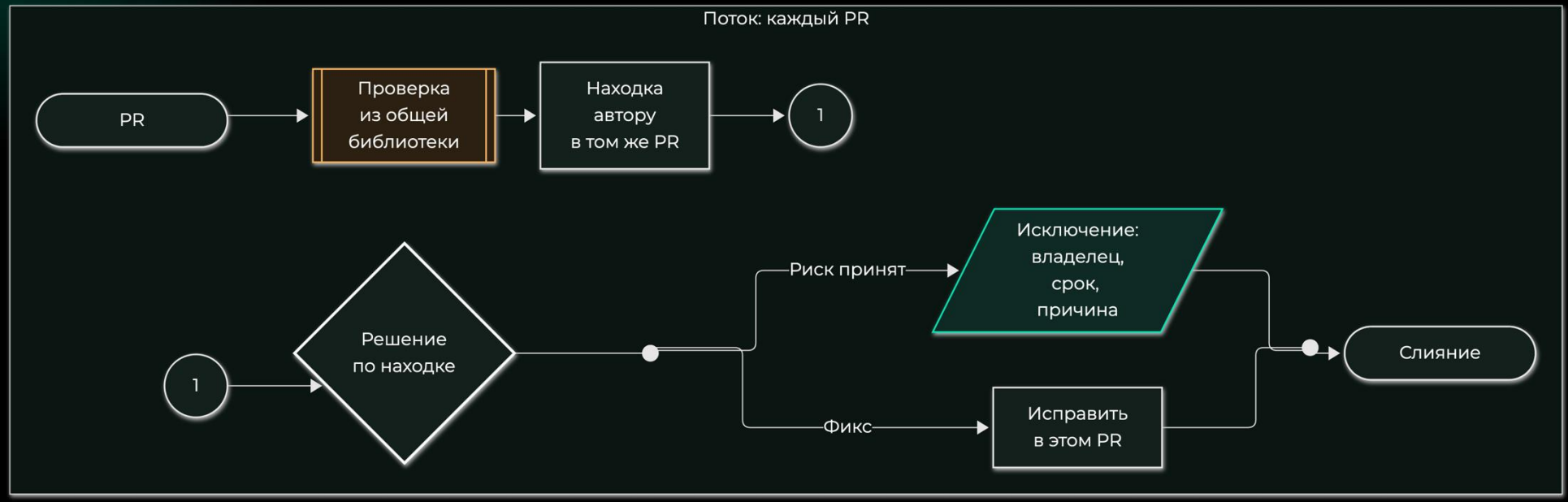
3 из 4

Python-проектов без lock-файла, у Gradle — ни одного

Без lock-файла два билда одного коммита собираются из разных зависимостей — SBOM невоспроизводим

«У находок нет адресата и срока»

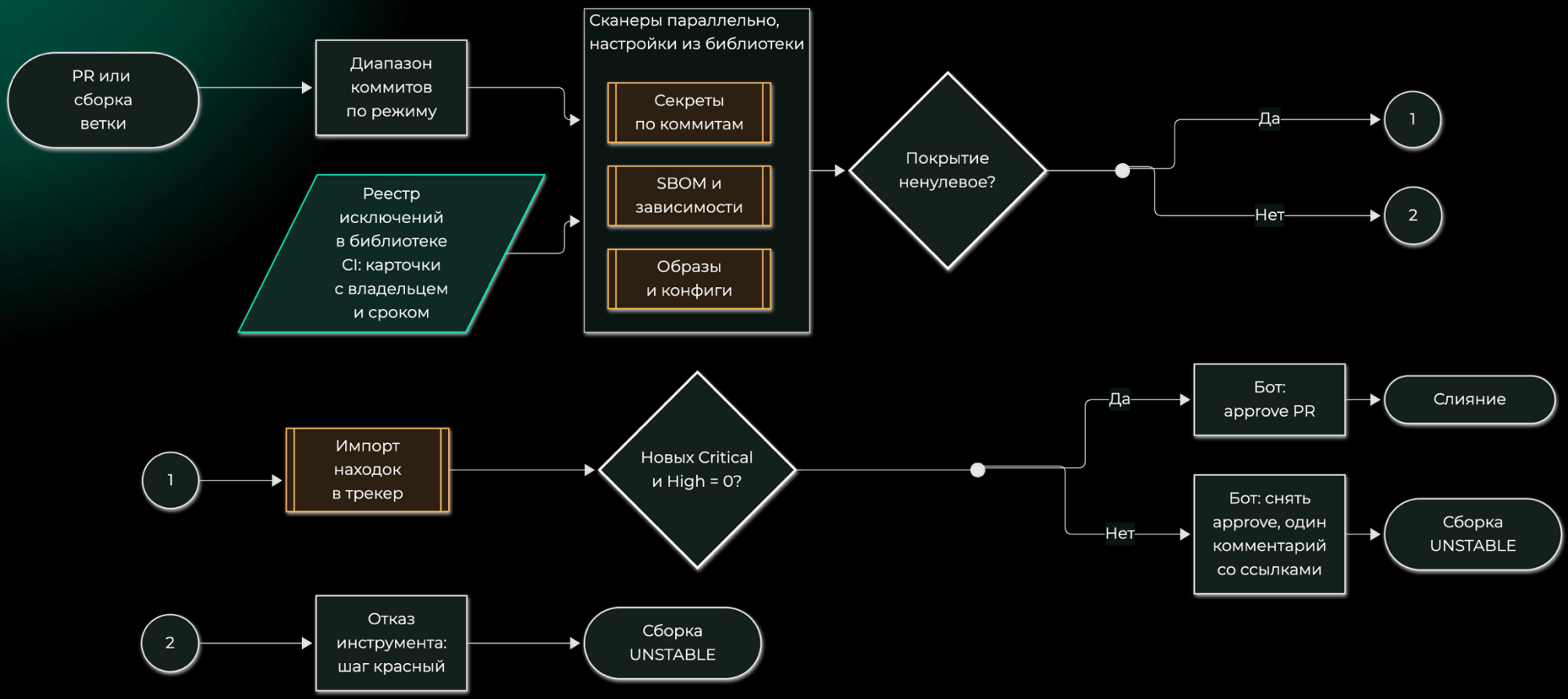




Нужен поток: проверка на каждый PR и решение по каждой находке, так как:

- **не автору и не вовремя**
- **92% фолсов убивают доверие и «погасить» дешевле, чем разобраться**

Один гейт на PR, настройки в одном месте



Настройки проверок — в общей библиотеке CI, не в репозитории команды

Гейт один: новых критичных и высоких — 0

Находка приходит автору в тот же PR

Lock-файлы везде

Воспроизводимая сборка и SBOM на каждый релиз

Свежие версии с выдержкой

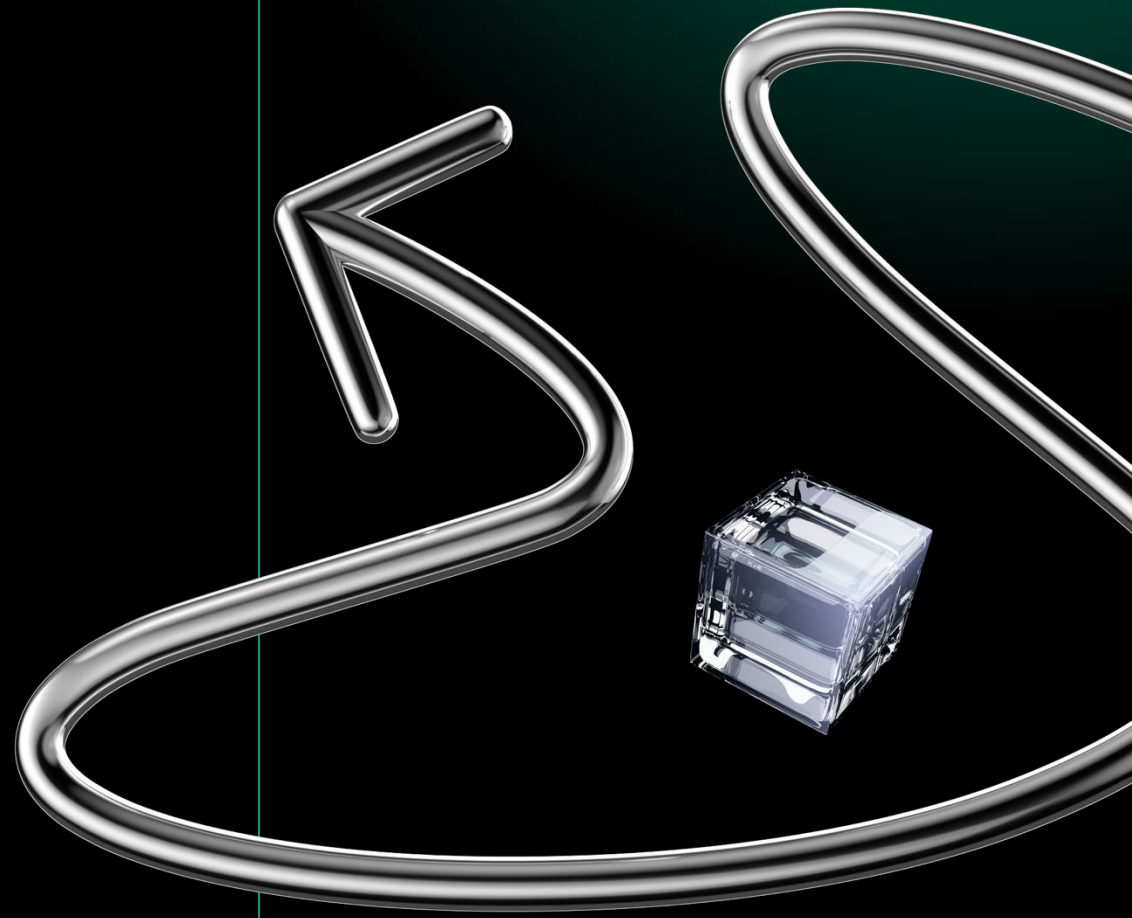
Обновление через несколько дней после выхода: защита от подмены пакета (тагитирование и хэш)

Шаблоны и авто-PR

Шаблоны исправлений и автоматические PR обновлений зависимостей

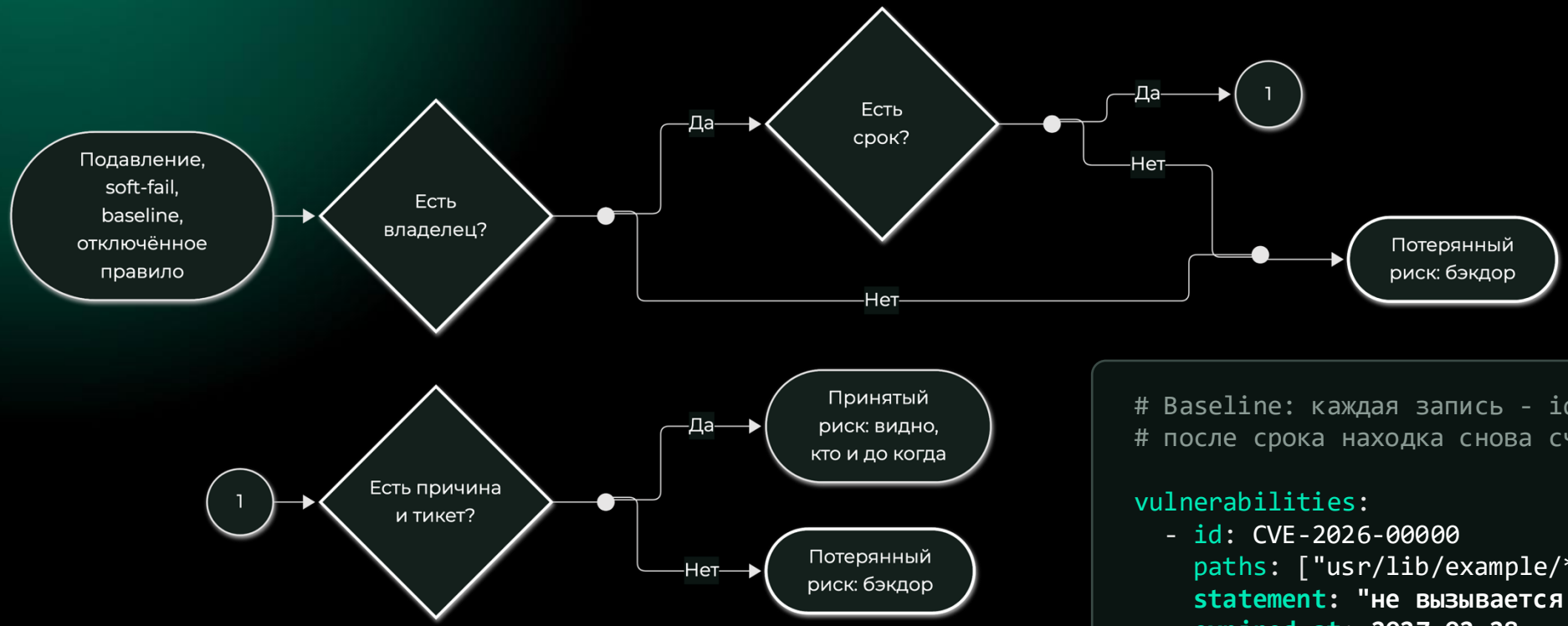
Единый реестр артефактов с гейтом

Вместо случайных источников



«Исправление дешевле всего там, где находка появилась» (с)

Исключение без владельца и срока — потерянный риск



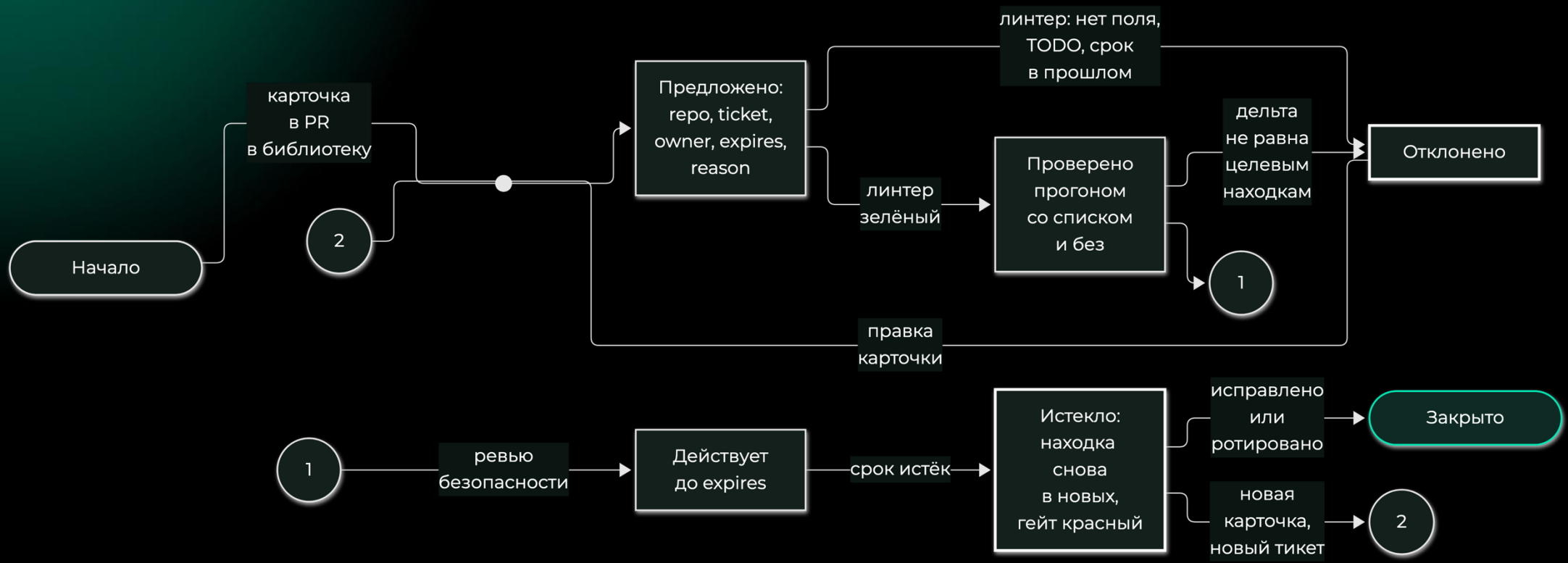

```yaml
# Baseline: каждая запись - id, причина, задача и срок
# после срока находка снова считается новой

vulnerabilities:
  - id: CVE-2026-00000
    paths: ["usr/lib/example/*"]
    statement: "не вызывается из образа, TICKET-123"
    expired_at: 2027-02-28
misconfigurations: []
```

Исключение без владельца и срока — это не риск, который приняли, а риск, который потеряли из вида:

- **Легитимные возможности, которые становятся бэкдором - подавление по маске пути, soft-fail «пока» и т.д.**
- **Файл исключений в репозитории команды**
- **«Временно отключим правило»**

Исключение — это код: владелец, срок, причина, одно место



- **Один реестр вне репозитория команды**
- **Карточка над каждым блоком**
- **Истёк срок — линтер не пропускает**

Карточка исключения

```
# repo: team/service | ticket: TICKET-123 | owner: team
# expires: 2027-02-28 | reason: кэш-ключи провайдера в тестовых фикстурах, не креды

[[allowlists]]
description = "TICKET-123: provider cache keys in test fixtures"
condition   = "AND"
commits     = ["3f2a9c1e4b7d…", "8b7d04aa91c3…"]
paths       = ['''^service/src/test/resources/.*\.json$''']
```

- **Два вида исключений**

  Класс значений (форма, которая секретом не бывает) — бессрочно;
  точечное — коммиты и пути, со сроком

- **Свой конфиг сканера в сканируемом репозитории**

- **Линтер двусторонний**

  Карточка без блока, TODO, истёкший срок

- **Любая правка реестра**

  Прогон «со списком и без»: разность равна целевым находкам

Было выявлено

~1900

сработок, 7 коммитов, 12 путей

→

Стало

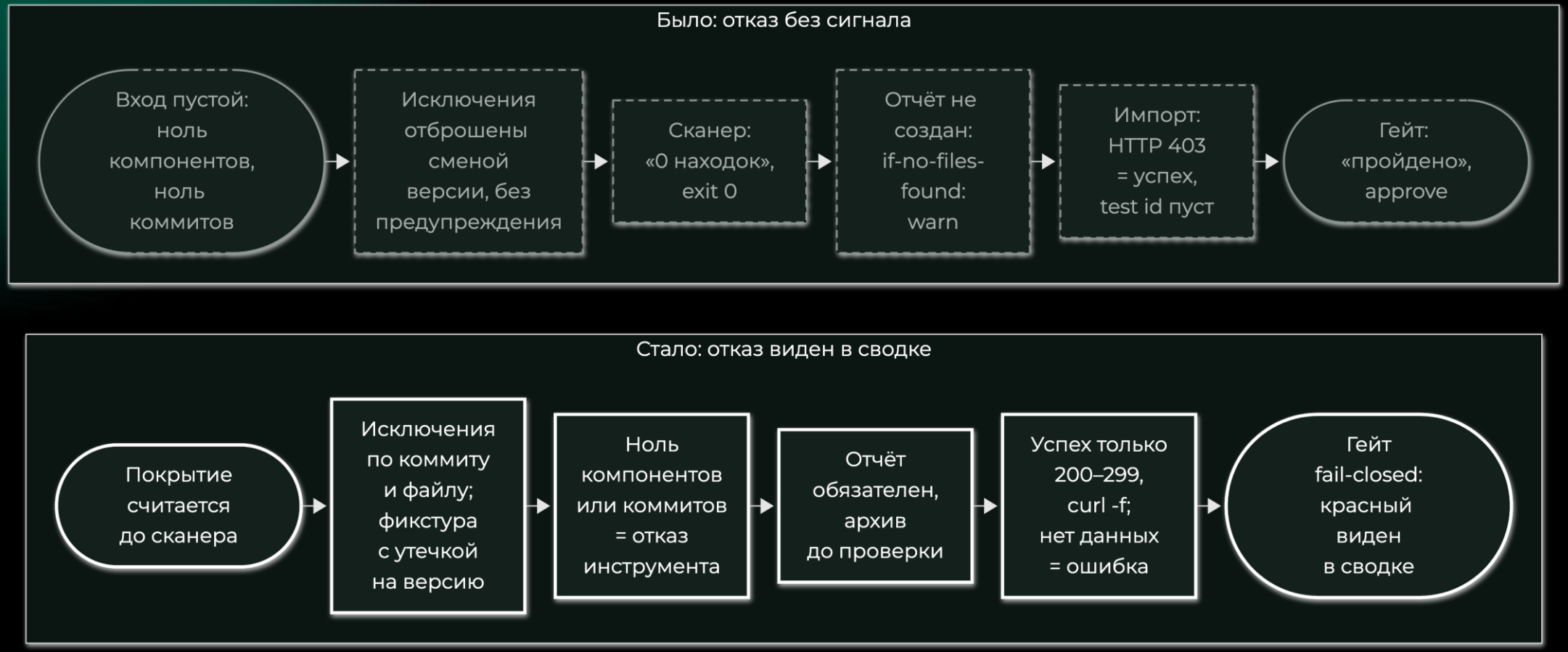
0

в отчёте, правила включены

- 92% — один блок по коммитам и путям, бессрочно: класс тестовых фикстур
- 8% — три блока до даты и тикет команде: подтвердить контур, ротировать, заменить плейсхолдерами
- После срока блоки перестают действовать — гейт снова красный
- Прогон «со списком и без»: первый блок снимает ровно 1771, остальные ровно 146, разность с отчётом — ноль

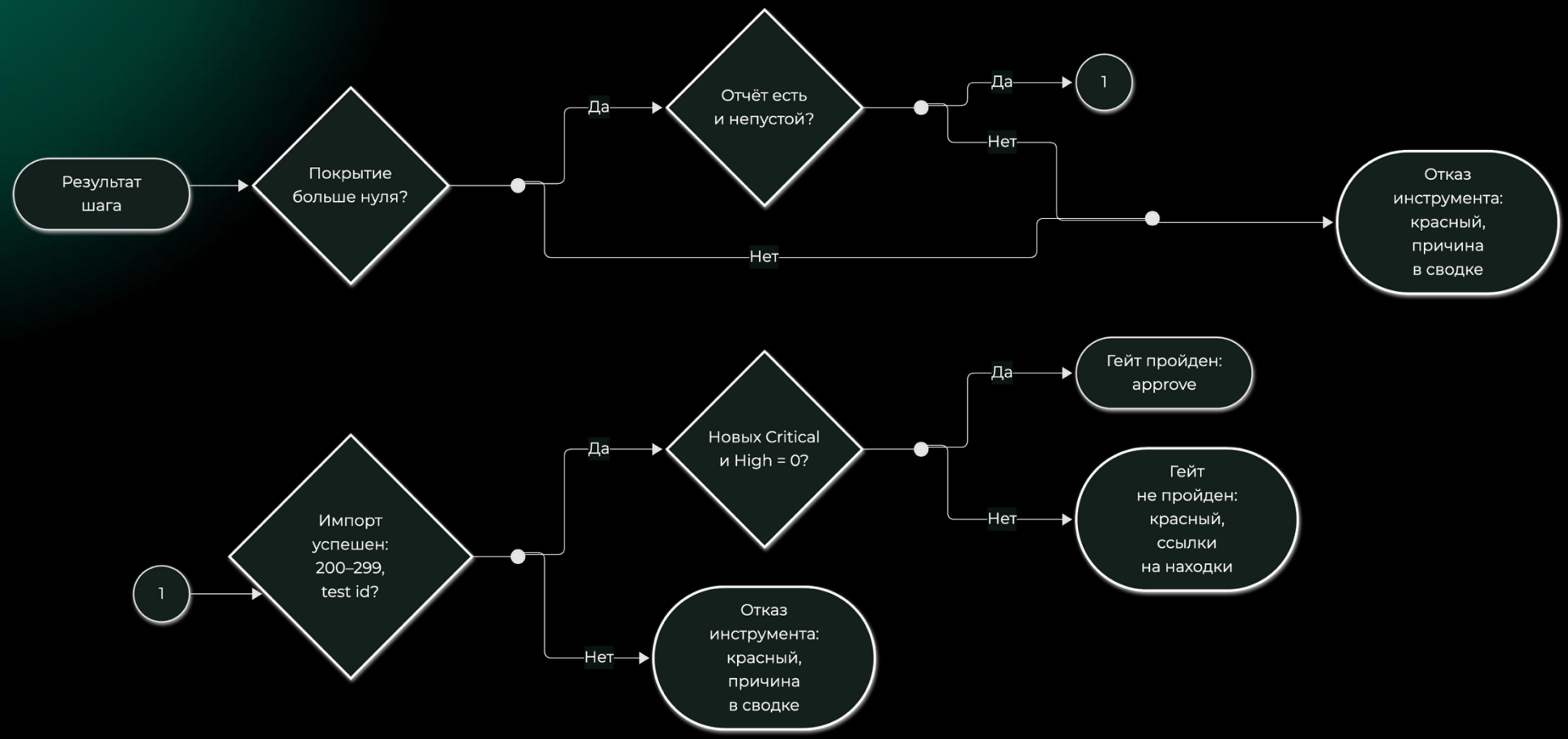
- Пустой SBOM и вложенные компоненты (24 и 39 уязвимостей «в SBOM, но вложены»)
- Смена версии сканера секретов отбросила исключения без предупреждения
- Гейт fail-open — просроченный токен делал все PR зелёными
- Сценарии зелёные в тестах, красные у команд

Отказ без сигнала — главный риск



Отчёт есть, статус зелёный — а проверки не было (просто не сработало)

Проверка обязана доказать, что проверяла



Нет данных — значит «не проверено», а не «чисто»: гейт fail-closed

Покрытие считается до сканера — ноль объектов является отказом инструмента

```python
def die(message: str) -> None:
    raise SystemExit(f"[gate] {message}")

def require_count(what: str, n: int) -> int:
    """Ноль объектов - отказ инструмента, а не чистый результат."""
    if n <= 0:
        die(f"{what}: 0 - инструмент ничего не проверил")
    print(f"покрытие: {what} = {n}")
    return n

require_count("коммитов в диапазоне", len(commits))
require_count("компонентов SBOM", len(bom.get("components") or []))
require_count("пакетов ОС в образе", coverage["os_packages"])
```

- Сканер секретов — число коммитов диапазона
- SBOM — компонентов с идентификатором пакета, до сканера
- Сканер уязвимостей — наличие базы и число результатов
- Стат анализ — файлы и правила
- Отчёт в пару сотен байт — пустой каркас как факт его существования
- Конфигурации — пройденные проверки

ИИ — ещё один источник уверенных ошибок

«Проверено» без проверки

То есть «проверено без интерпретатора», на стенде упало

Документация из прошлого

Описано поведение до правок: статусы, порядок проверок

Комментарий против кода

«нет результатов – считаем успешно», а код возвращает отказ

Версия, которую блокирует реестр

Критичная уязвимость в транзитивной зависимости, — переопределений не видно

Инструмент не о том

Модуль «про безопасные workflow» оказался руководством по смарт-контрактам

Ошибается не в синтаксисе — в утверждениях «проверено»

«Утверждение без команды и её вывода — гипотеза»

- **Гипотеза, не факт** — ответ ИИ проверяют выводом команды, тестом или чтением кода
- **Ревью, не доверие** — код с ИИ ревьюится как человеческий, скидок на автора нет
- **Сверка, не ощущение** — документацию сверяют с кодом прицельно, расхождение — дефект
- **Эксперимент, не чтение** — скептик ищет, где правка сломается: права, сеть, песочница CI
- **Гейт, не исключение** — конфиги и плагины ИИ под тем же гейтом: пины, права, секреты

Долг стал видимым - получил владельца и срок

Было	Стало
Находки раз в год, отчёт на 100 страниц	Находки в PR автору, гейт на каждый PR
Baseline: 33 записи без владельца и срока, два user:pass открытым текстом	19 блоков с карточками, истёкших 0, замечаний линтера 21 → 0
Один сервис: ~119 критичных и высоких, владельца нет	Долг виден, у него владелец и срок
SBOM npm: 873 пакета, 180 уязвимостей	1021 пакет, 204 уязвимости — вложенные развёрнуты
«Да» в анкете заказчика	Факты: статистика гейта, реестр исключений, SBOM

- **Шаг SCA на фронтенд-репозитории — 2 мин 7 с**
- **Дорогие операции — ночным job'ом: 97 запросов вместо 2128**

Начать с lock-файлов, одного гейта и реестра

1. Lock-файлы и SBOM

lock-файлы везде, SBOM на каждый релиз

2. Один гейт

По критичным и высоким в общем CI; начать с секретов и зависимостей

3. Реестр исключений

С владельцем и сроком; локальные исключения в репозитории роняют сборку

4. Доказанное покрытие

Пустой результат и ошибка чтения — падение

5. Правило для ИИ

Нет вывода команды — не «проверено»

6. Пакет артефактов

SBOM, описание процесса, статистика гейта, реестр исключений, контакт для инцидентов

Инструмент для первого шага: cdxgen-distroless

- **Образ генератора SBOM cdxgen на distroless**
  SBOM CycloneDX по манифестам и lock-файлам, каталог только на чтение, сеть контейнеру не нужна
- **Внутри нет оболочки, apt, npm, pip**
  Только Node.js, cdxgen, uv и stdlib Python; пользователь 65532, базы по digest
- **Python без lock-файла**
  Сначала uv pip compile — без установки и сборки пакетов; установка зависимостей ради SBOM исполняла бы чужой код на агенте
- **Публикация**
  Docker Hub и GHCR, amd64 и arm64, provenance, SBOM образа и подпись cosign keyless; перед публикацией — gitleaks, hadolint, Trivy

```
IMAGE=ghcr.io/geminishkv/cdxgen-distroless:12.9.0
docker pull "$IMAGE"

# SBOM по lock-файлам без сети: каталог проекта только на чтение, результат забираем из контейнера

docker create --name sbom --network none -v "$PWD":/app:ro -w /app "$IMAGE" \
  -t python --spec-version 1.6 --no-install-deps -o /tmp/bom.json /app
docker start -a sbom && docker cp sbom:/tmp/bom.json bom.json && docker rm sbom
```

github.com/geminishkv/cdxgen-distroless

Версия образа = версия cdxgen (12.9.0) и в пайплайнах закреплять digest

«Не знаю» на любой из трёх вопросов — бэкдор

- Автоматический by-pass и расхождение с валидацией руками - одна и та же дыра
- Закрывает её не инструмент, а владелец, срок и доказательство покрытия

1

Кто владелец каждого исключения и когда оно истекает?

2

Что ваш сканер доказал, что проверил?

3

Что в отчёте о проверке написано без вывода команды?

Если ответ «не знаю» — у вас есть бэкдор, и вы нашли его бесплатно

«Карта, не предписание»

ГОСТ Р 56939-2024

Безопасная разработка; действует с 20.12.2024

ЦБ 683-П / 719-П / 757-П

Анализ уязвимостей по ОУД4 либо сертификация ФСТЭК

NIST SSDF v1.2, SLSA v1.2

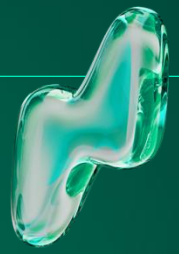
SSDF — драфт 17.12.2025; SLSA v1.2 — уровни целостности сборки

EU CRA, CISA SBOM 2025

CRA — отчётность об эксплуатируемых уязвимостях с 11.09.2026; CISA Minimum Elements for SBOM 2025

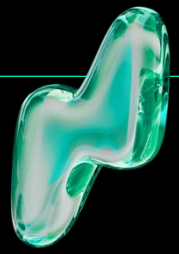
- **Применимость у каждого своя**
- **Общий вектор — от разовой к непрерывному контролю**

Карточка риска и исключение из проверки — разные записи



Принятие риска

- Актив, угроза, вероятность, ущерб, владелец риска, срок пересмотра, решение и кто принял
- ISO/IEC 27001:2022 — остаточные риски принимают владельцы
- NIST RMF — POA&M со сроками
- DefectDojo Risk Acceptance — владелец, дата истечения, реактивация по сроку



Исключение из проверки

- Техническая запись — коммит и путь или класс значений, ссылка на тикет и срок
- Ссылается на принятый риск, а не заменяет его

Реестр держится на линтере, fail-closed и двойном прогоне

- **Карточка**
  repo, ticket, owner, expires, reason
- **Двусторонний линтер**
  без карточки, без блока, TODO
- **Fail-closed**
  локальный конфиг сканера в репозитории — ошибка шага
- **Прогон «со списком и без»**
  на PR в библиотеку: дельта ровно целевые находки
- **Синхронизация в wiki**
  для управляемости

```
CARD = re.compile(r"^# repo: (?P<repo>[^|]+) \| ticket: (?P<ticket>[^|]+)"
                  r" \| owner: (?P<owner>[^|]+) \| expires: (?P<expires>[^|]+)"
                  r" \| reason: (?P<reason>.+)$")

def check(lines: list[str], today: date) -> list[str]:
    problems = []
    for i, line in enumerate(lines):
        if line.strip() != "[[allowlists]]":
            continue
        card = CARD.match(lines[i - 1])
        if not card or "TODO" in card.group(0):
            problems.append(f"строка {i + 1}: нет карточки или TODO")
        elif card["expires"] != "бессрочно" and date.fromisoformat(card["expires"]) < today:
            problems.append(f"строка {i + 1}: срок {card['expires']} истёк")
    return problems
```

Локальный конфиг сканера в репозитории роняет шаг

- **Сканер сам читает trivy.yaml и .trivyignore из каталога**

  одним файлом команда отключала SCA

- **Исключения приходят только из библиотеки**

  реестр с карточками, вне репозитория команды

- **Теперь это ошибка шага**

  любой из трёх файлов в сканируемом репозитории — шаг красный

- **Конфиг репозитория отключён**

  --config /dev/null и ignore-файл библиотеки

```
// Trivy сам читает trivy.yaml и .trivyignore из каталога репозитория:
// локальные настройки и исключения запрещены, реестр - только в библиотеке

for (String name : ['trivy.yaml', '.trivyignore', '.trivyignore.yaml']) {
    if (steps.fileExists(name)) {
        steps.error "[Trivy] найден ${name}: локальные исключения не допускаются"
    }
}
steps.sh "trivy sbom ${bom} --config /dev/null --ignorefile trivy-ignore.yaml --format json --output ${report}"
```

В Jenkinsfile команды нет ни порога, ни исключения

- **Три сканера параллельно**
  секреты по коммитам, SBOM и зависимости, образ и конфиги
- **Импорт находок в трекер**
  продукт в трекере ищется по имени репозитория
- **Покрытие считается до гейта**
  0 просмотренных объектов — отказ инструмента, шаг красный
- **Один гейт: Critical и High = 0**
  approve от бота; иначе снятие, комментарий, UNSTABLE

```
// Jenkinsfile команды целиком: только вызовы шагов общей библиотеки CI.
// Настройки сканеров, пороги гейта и реестр исключений - в библиотеке

@Library('security-ci') _
node {
    stage('Security') {
        checkout scm  // trivy.yaml или .trivyignore в репозитории: шаг красный
        parallel secrets: { scanSecrets(range: commitRange()) },
                 sbom:    { scanSbom() },
                 image:   { scanImage() }
        def results = collectResults()   // 0 объектов = отказ инструмента
        importFindings(results)          // продукт в трекере = имя репозитория
        securityGate(results)            // новых Critical и High = 0: approve
    }
}
```

Стадия Security: команда вызывает, библиотека решает





У команды — вызов стадии и запрет своих конфигов сканера; сканеры, покрытие, трекер и гейт — в библиотеке CI

Benchmark рисков ИБ

- **Проверка регулятора** — Нарушение законодательства в области ИБ, выявленное регулятором в рамках проверки, которое привело к штрафам и предписаниям на срочное устранение
- **Ошибка персонала** — Ненамеренное несоблюдение сотрудниками требований эксплуатации систем (включая требования ИБ) в связи с недостаточным уровнем контроля (незнание, неинформированность, недостаточность технического или организационного контроля и т.д.), которое привело к недоступности систем, прерыванию работы процессов или ухудшению качества оказания услуг клиентам/заказчику
- **Атака на компанию** — Киберпреступления, совершенные третьими лицами, в отношении компьютерных систем компании, которые привели к утечке конфиденциальной информации, недоступности информационных систем (включая ransomware) и/или мошенническим операциям
- **Атака на клиентов** — Киберпреступления, совершенные третьими лицами, в отношении компьютерных систем клиентов/заказчиков компании, которые привели к утечке конфиденциальной информации, недоступности систем (включая ransomware)
- **Внутренний саботаж** — Саботаж внутренних ИТ-систем компании сотрудниками, который привел к прерыванию работы процессов или ухудшению качества оказания услуг клиентам/заказчикам

Безопасный проект / продукт

Зрелость ИБ

Построение процесса DevSecOps

↓

Цель

- Безопасность текущей поставки ПО в PROD
- Безопасный функционал web-приложения
- Безопасный инкремент поставки ПО
- Безопасная архитектура приложения

Виды работы AppSec

База

- обследование технологического стека
- изучение сопроводительной документации
- разбор технического решения и архитектуры приложения
- разбор бизнес-логики приложения
- подготовка и настройка pipeline для проведения ручного сканирования
- подбор и настр. анализаторов SAST, SCA, Container Security, Secret Management под стек проекта
- подготовка сценариев тестирования
- проведение первичного (повторного) сканирования (верхнеуровнево)
- проведение анализа рисков от уязвимостей и их группировка по критичности
- подготовка отчетов для разработчиков и менеджмента с уязвимостями

Рекомендовано

- проведение ручного DAST сканирования
- написание сценариев тестирования DAST
- автоматическое тестирование API endpoints
- тестирование API приложения, его детализация по методам запросов, ошибок
- углубленное тестирование
- подготовка pipeline для самостоятельного тестирования командами

Повышение зрелости

- автоматизация по инкременту поставки
- отрисовка UML схемы pipeline
- проработка механизмов безопасных версий open-source (golden repository)
- проработка stage job для pipeline команд или управляющего конфига ИБ
- реализация Quality Gate на разных этапах сканирований
- реализация контролирующего Security Gate для поставки
- Jira Issue Workflow для автоматизации постановки задач устранения уязвимостей
- подготовка метрик оценки безопасности поставок кода
- подготовка требования ИБ к архитектуре приложения
- подготовка технического мануала для автоматизированного сервиса ИБ
- адаптация и оптимизация шагов сканирований ИБ (сдерживание роста TTM)

Группы услуг могут быть подобраны индивидуально и содержать часть из описанных видов работ, например, отдельно можно заказать DAST

Необходимо вовлечение команды для реализации работ по ИБ и устранения уязвимостей, эти трудозатраты считаются отдельно

Командам необходимо поддерживать в актуальном состоянии автоматизацию сканирований, включая версии инструментов ИБ

В зависимости от решений могут потребоваться лицензии и мощности для развертывания инструментов

Нетиповые работы являются дополнением построения процесса DevSecOps

Повышение зрелости ИБ включает все перечисленные работы

Спасибо
за внимание!



Илья Шмаков

Руководитель направления
AppSec СберСпасибо