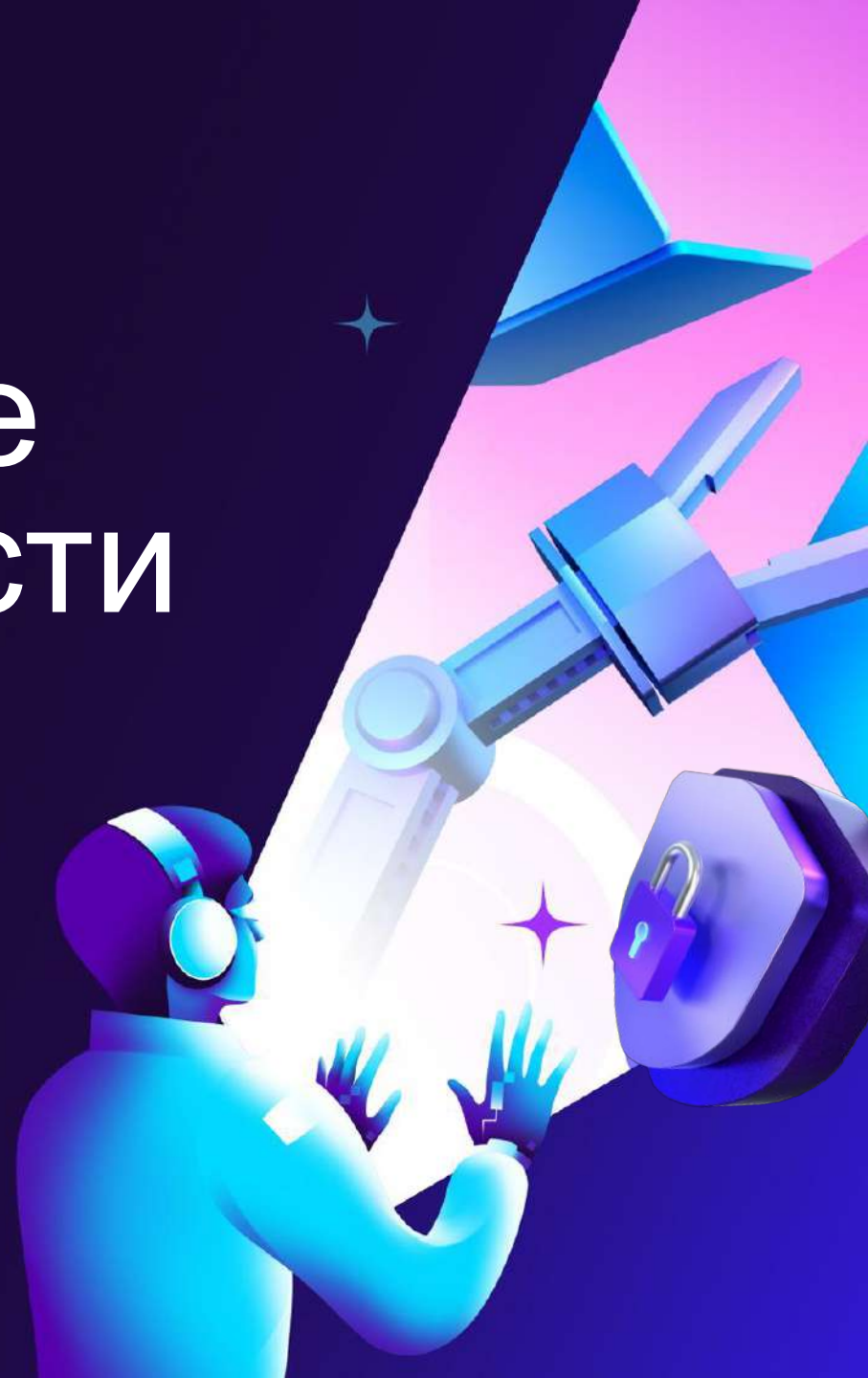


Алексей Смирнов

CEO & Founder @CodeScoring

Моделируем влияние проверок безопасности

на *разных* этапах разработки





- / Основал платформу безопасной разработки **CodeScoring**
- / Делаем душевные ИТ-конференции **DUMP, PyCon, EkbPy, RustCon**
- / Спикер, ПК, ведущий AppSec-корпоративов
- / Пишу в **@codemining**
- / Внезапные факты: астроном, преподавал python 15 лет

Алексей Смирнов

CEO @CodeScoring



В предыдущих сериях

- ✦ Узнали *среднее количество* компонентов и уязвимостей на приложение
- ✦ *до 86% всех уязвимостей* находится в транзитивных зависимостях и могут находиться на поверхности атаки
- ✦ *23 минуты* тратит опытный триажер известных уязвимостей *на одну уязвимость*
- ✦ посмотрели на *CodeScoring Мегаграф* на примере log4shell: глубины залегания и реакция сообщества по выпуску патча
- ✦ рассмотрели *алгоритмы исправления транзитивных* уязвимостей

Алексей Смирнов
CEO & Founder Profiscope /CodeScoring

phd 2 Positive Hack Days Fest
от positive technologies

Проблемы отцов и детей

Аналитика и триаж транзитивных зависимостей

Доклад: «Проблемы отцов и детей: Аналитика и триаж транзитивных зависимостей», PHD, 2024
<https://youtu.be/dqjGQBe2yTY>

TLDR: Моделируем влияние проверок безопасности на разных этапах разработки



*нет-нет
да и да*

Сегодня попытаемся ответить на следующие *вопросы*

- ❖ Что из себя представляет проверка Open Source в жизненном цикле разработки ПО?
- ❖ Как можно «накрутить» проверку во имя эффективности?
- ❖ Насколько затратными могут быть эти проверки?
- ❖ Полезные числа про триаж и скорость обнаружения уязвимостей
- ❖ Влияние сдвига проверок «влево» на общий объем трудозатрат и кейсы сопровождения



01



Жизненный цикл

И место защиты цепочки поставки в нём

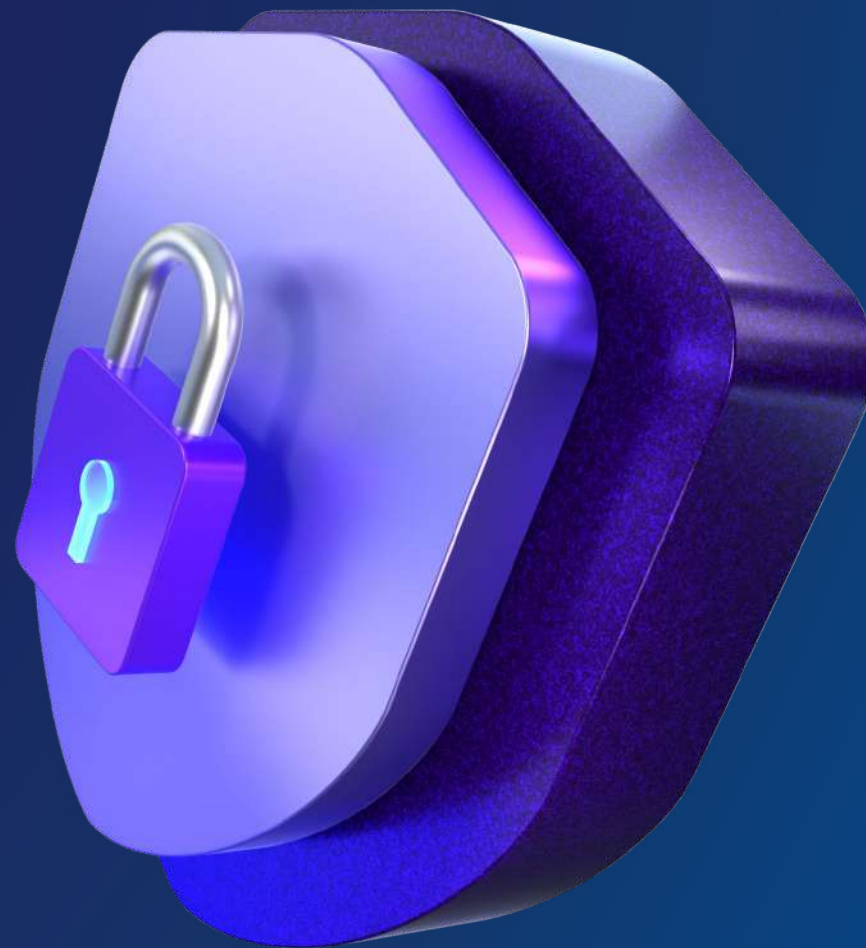
Безопасность OSS — два *комплиментарных* подхода

OSA

Фаерволинг нехороших компонентов, попадающих в руки разработчиков. Как правило, реализуется на уровне доп. проверок в хранилищах артефактов

SCA

Проверка компонентов ПО на стадии разработки, сборки-сборки-сборки и пост-релиза. Короче, может участвовать в процессах на всех этапах

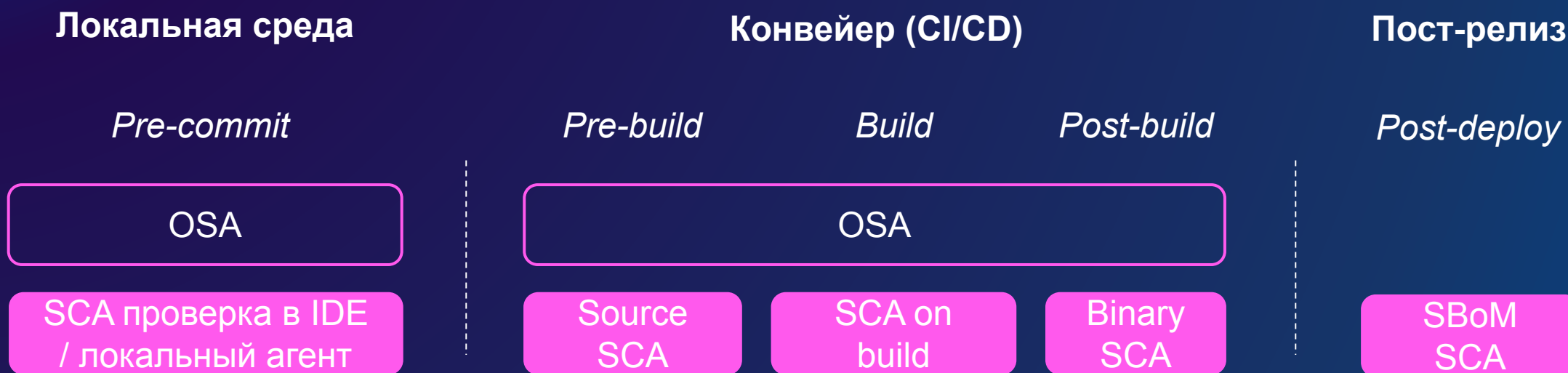


Процесс *разработки*



<https://androidarts.com/smb/SMB.htm>

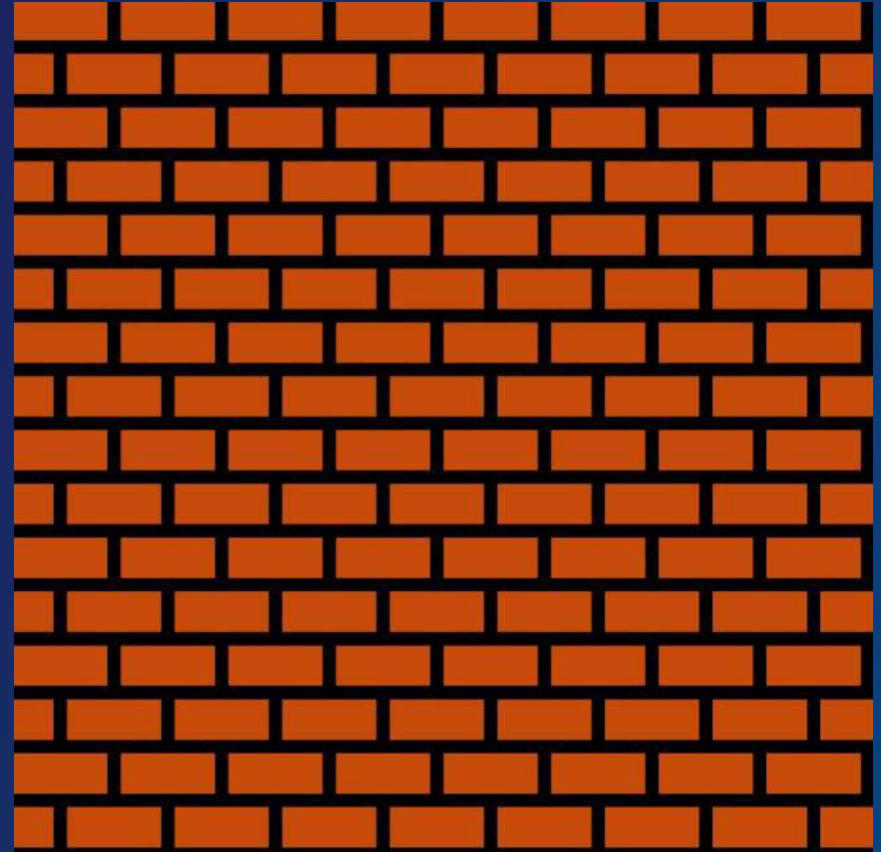
Разложим на стадии */stages*



OSA — как обычно работает



- ❖ Компоненты проверяются в момент попытки использования из репозитория артефактов
- ❖ Политики пресекают использование вредоносных или особоуязвимых компонентов
- ❖ Отсекать компоненты на этой стадии по Vuln Score — не лучшая практика или когда нужен «чистый» репозиторий
- ❖ Проверке подлежат и пакеты и докер-образы
- ❖ Не должно тормозить разработку (пока-пока транзитивы!)



SCA — как обычно работает

- ❖ Компоненты проверяются *в контексте приложения*: инвентаризация - проверка - действие
- ❖ Композиционный анализ — это *процесс*: на каждой стадии работают разные политики
- ❖ Процесс должен быть *оптимальным*, чтобы не засыпать разработчика кучей срабатываний и частым падением сборок



SCA — крутилки эффективности



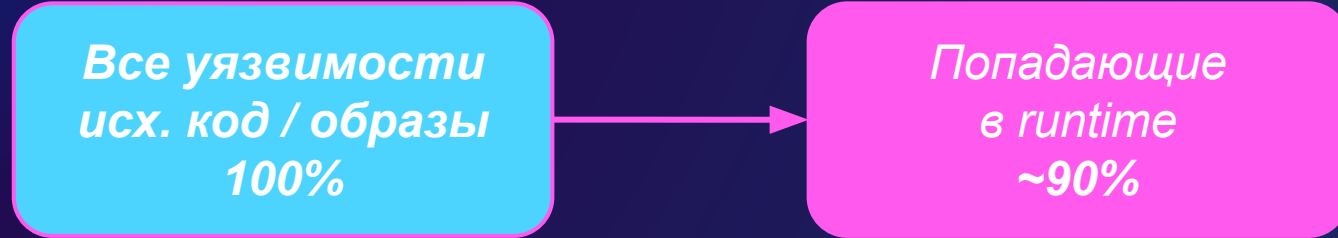
- ✦ *Возможность управлять политиками безопасности по стадиям и чтобы агент реагировал*
- ✦ *Глубина политик и скоринг: критичность, наличие фикса, эксплойта, признак транзитивности, достижимости, возраст и многое иное*
- ✦ *Объем базы знаний решает. Приходите завтра на доклад «[Новый log4shell, о котором вам забыли рассказать](#)»*



<https://www.sound-objects.com/sound-objects>

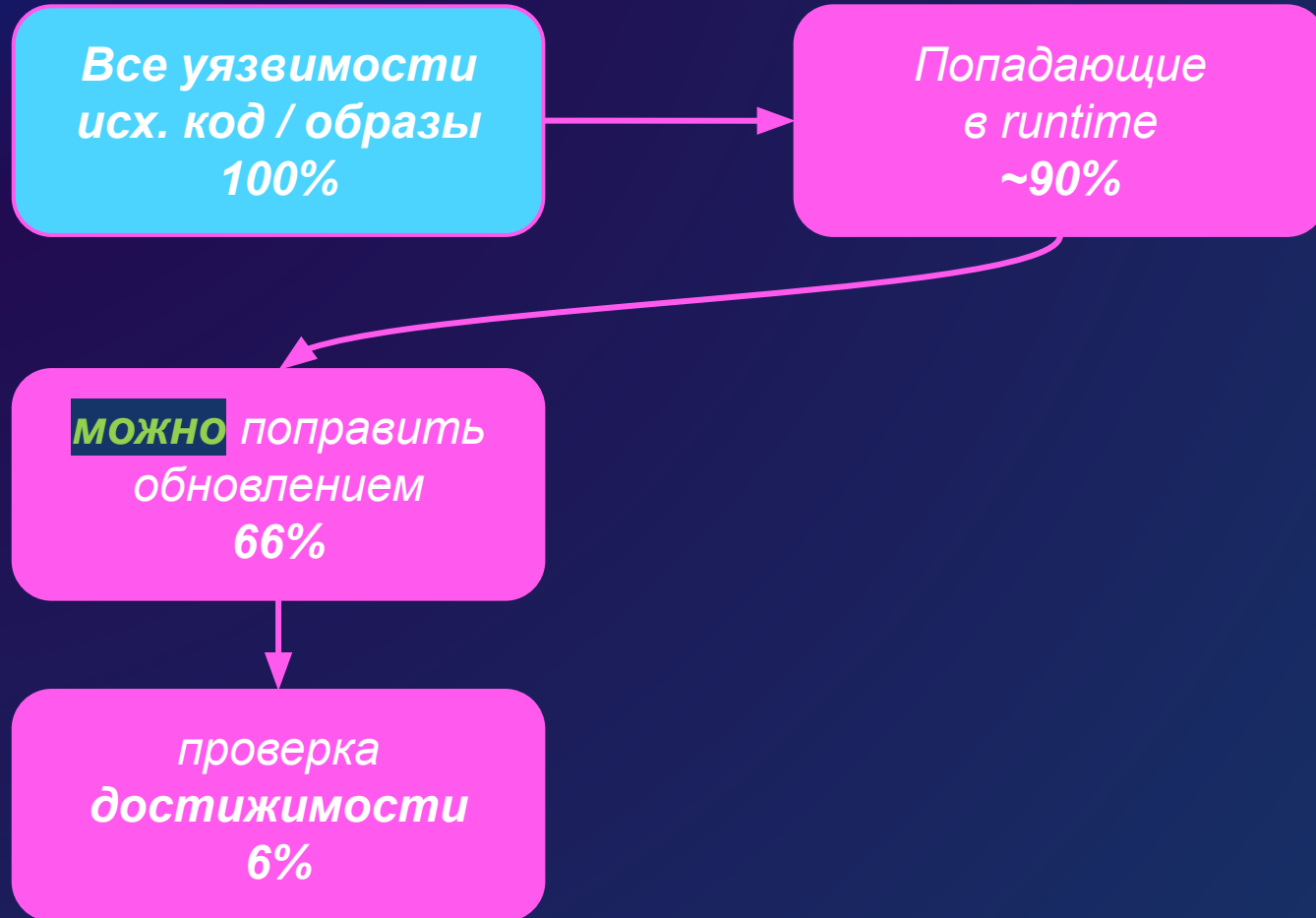
Крутилки воронки уязвимостей

/числа средние - по популярным технологиям и проектам разного размера



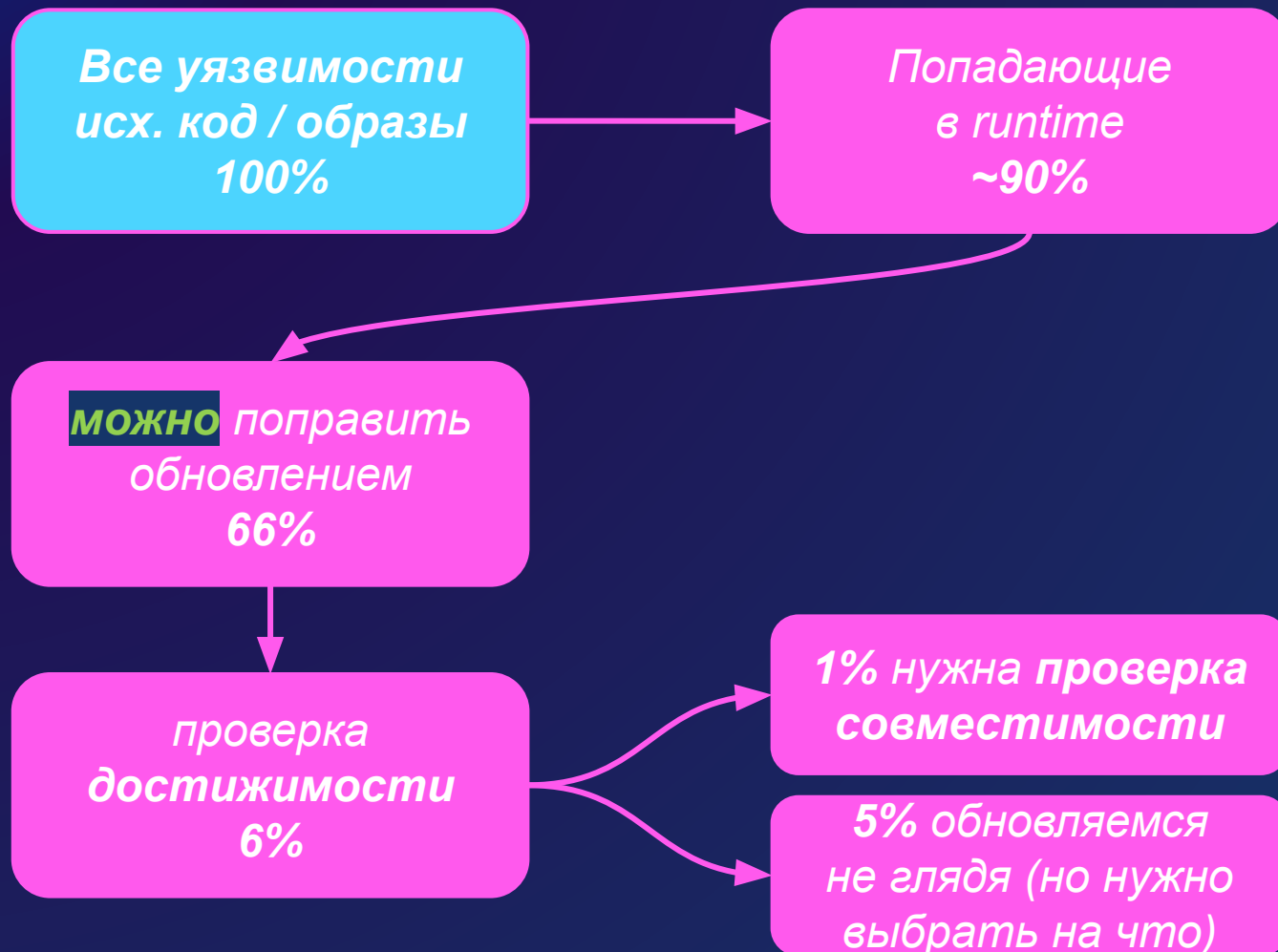
Крутилки воронки уязвимостей

/числа средние - по популярным технологиям и проектам разного размера



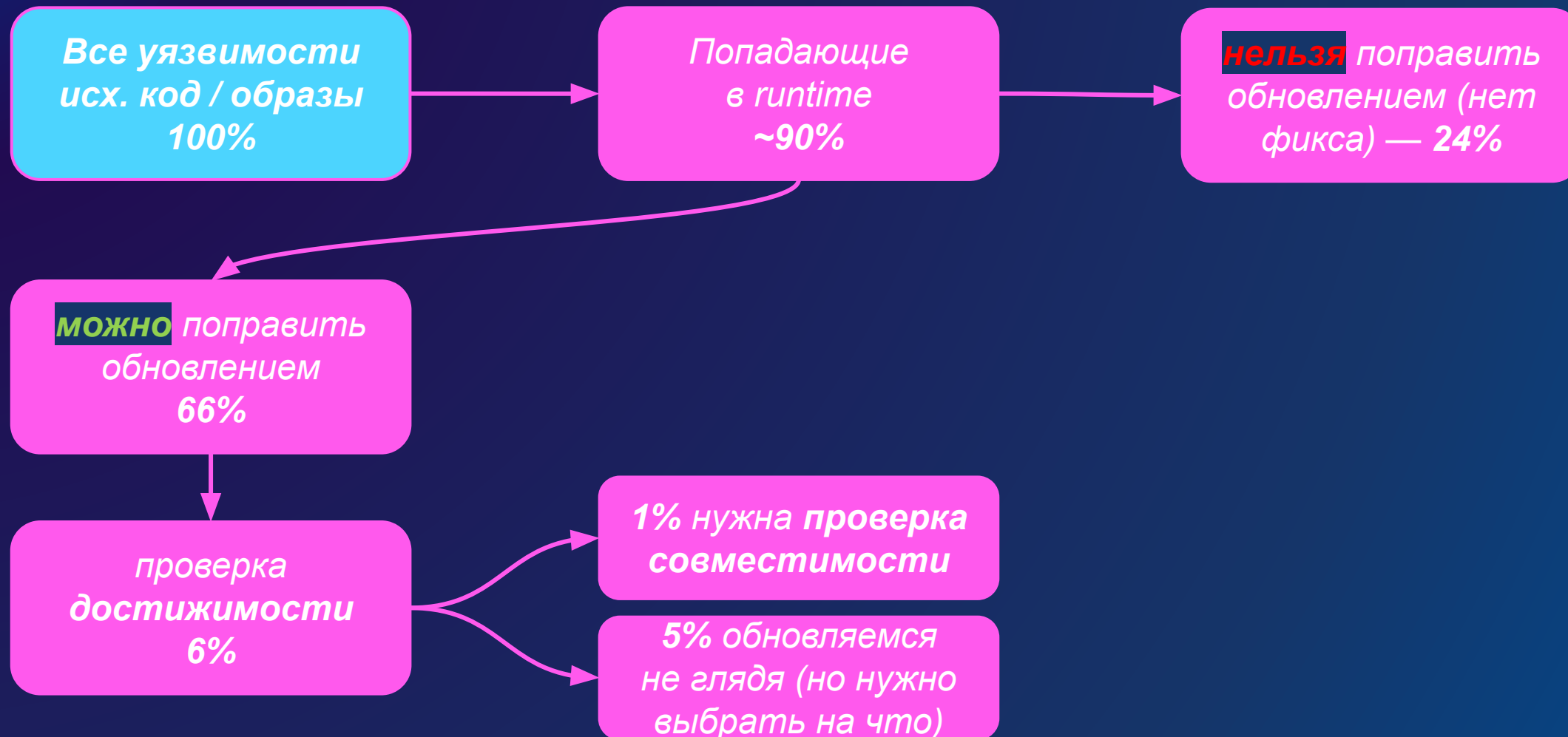
Крутилки воронки уязвимостей

/числа средние - по популярным технологиям и проектам разного размера



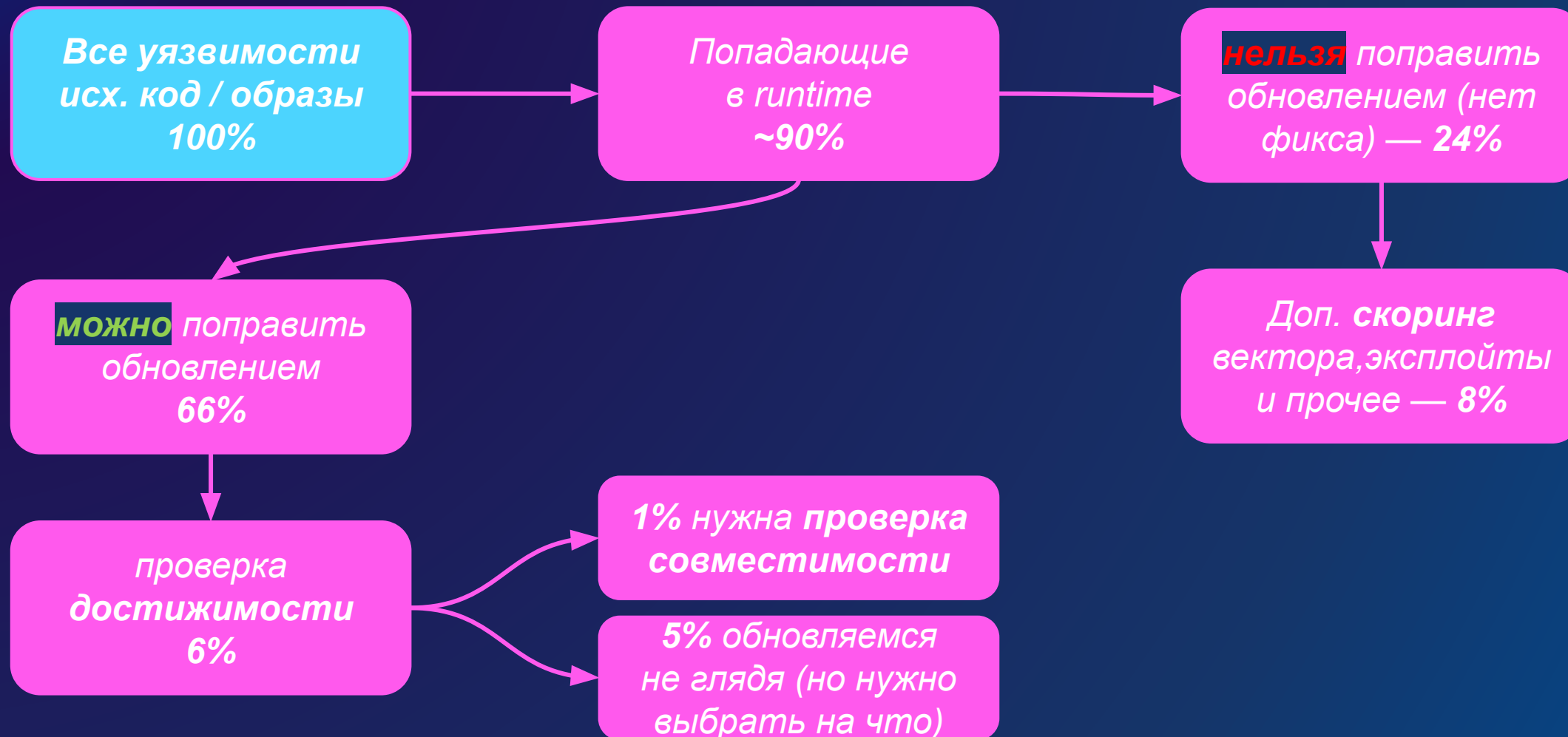
Крутилки воронки уязвимостей

/числа средние - по популярным технологиям и проектам разного размера



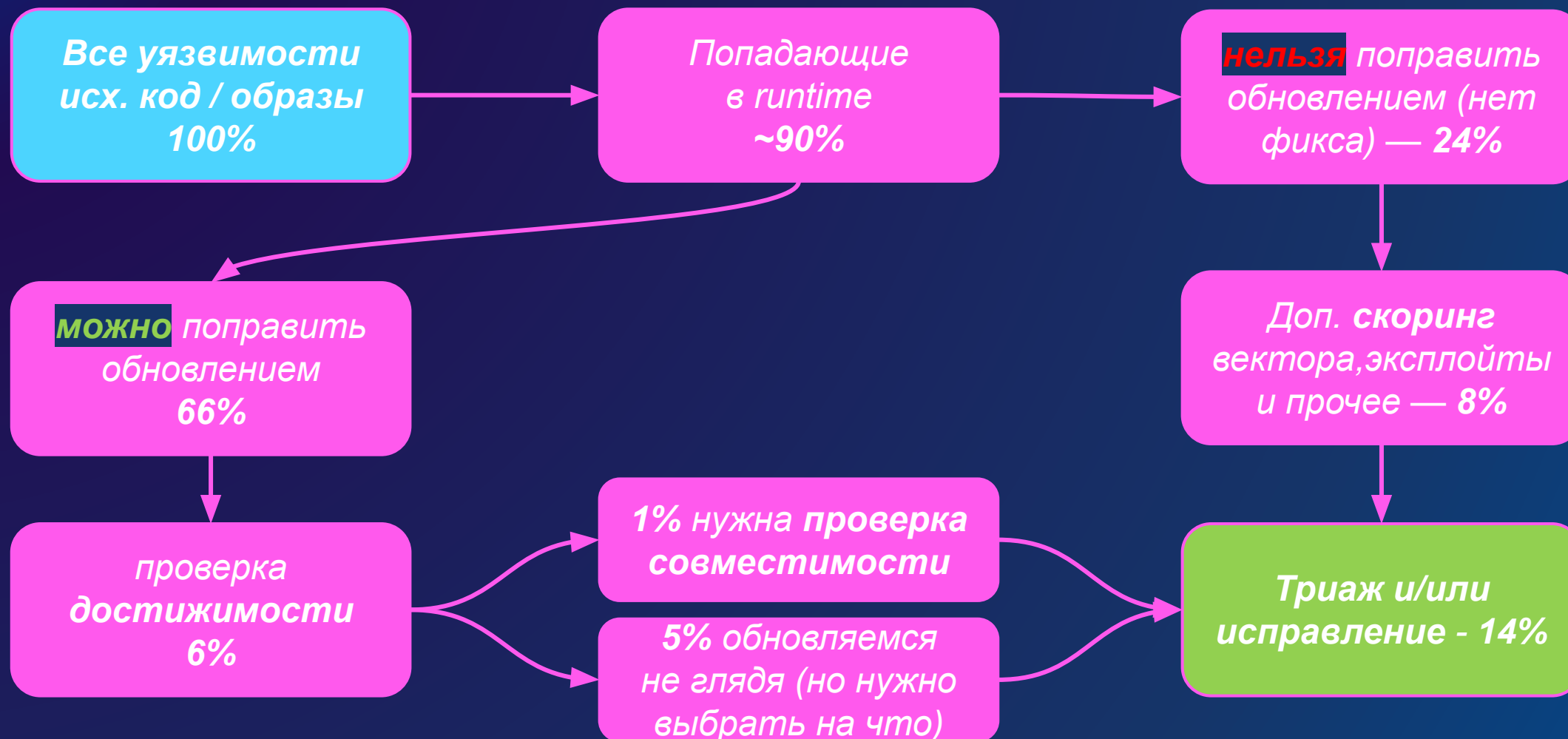
Крутилки воронки уязвимостей

/числа средние - по популярным технологиям и проектам разного размера



Крутилки воронки уязвимостей

/числа средние - по популярным технологиям и проектам разного размера



Можно *накрутить*
эффективность SCA
за счет *разных* ручек
в 5-10 раз

в теории в 100, но это в теории (и в маркетинге)



02



Разберем стадию

Что и где проверяется?



Стадии *проверки* Open Source



Локальная среда

Pre-commit

OSA

SCA проверка в IDE
/ локальный агент

Конвейер (CI/CD)

Pre-build

Build

Post-build

OSA

Source
SCA

SCA on
build

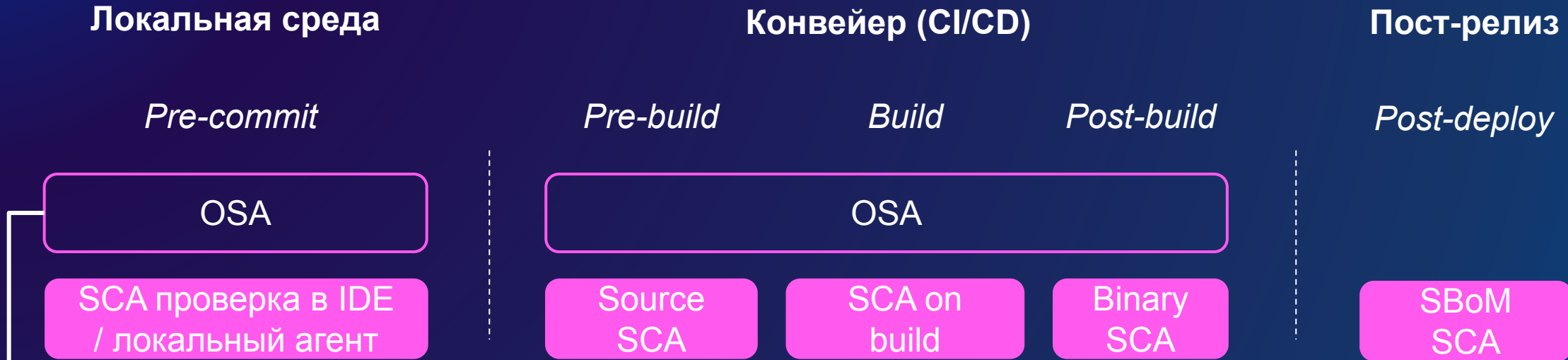
Binary
SCA

Пост-релиз

Post-deploy

SBoM
SCA

[0] OSA проверки

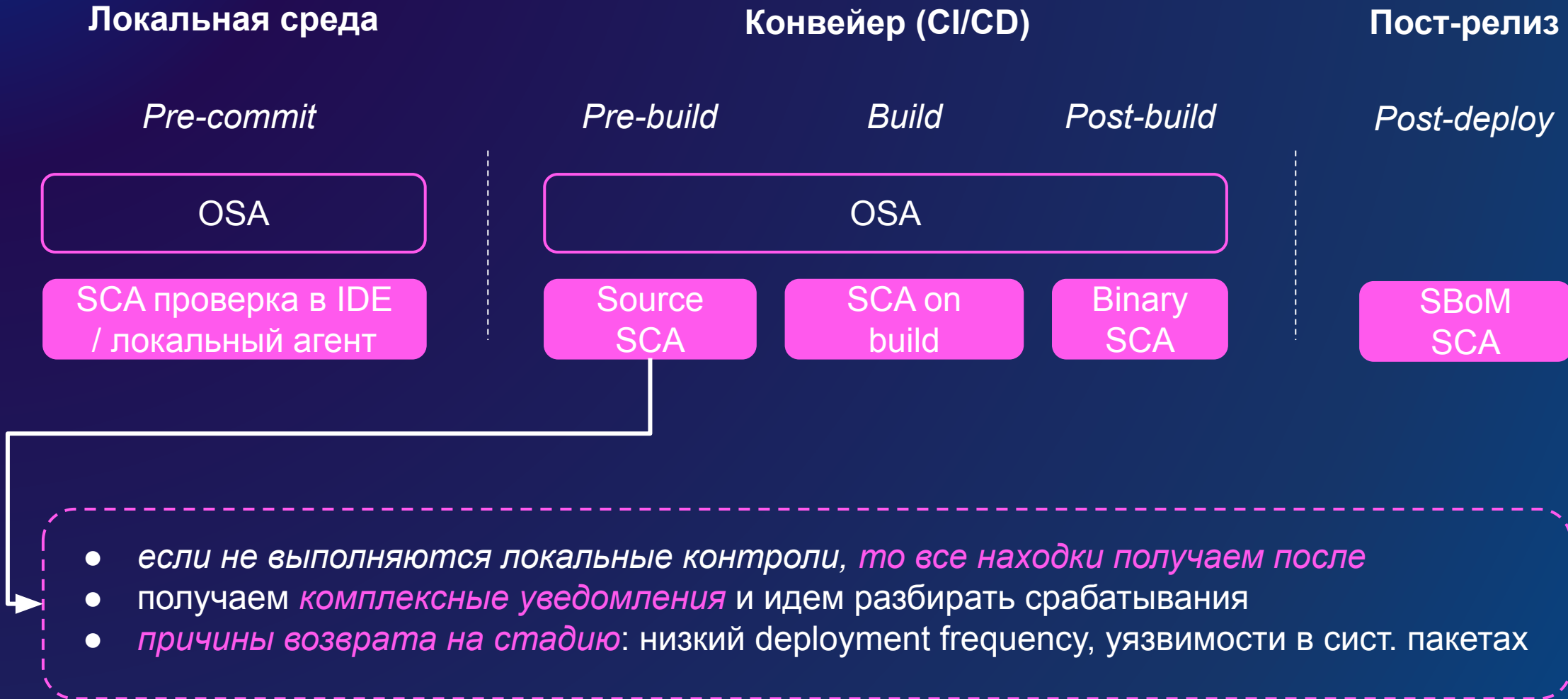


- можно *не брать* особо уязвимые и вредоносные компоненты *к себе в периметр* (ура?)
- *на всех стадиях* есть возможность сразу *понять, что пошло не так*
- главное *не пережестить с политиками*, а то получим «бунт на корабле»

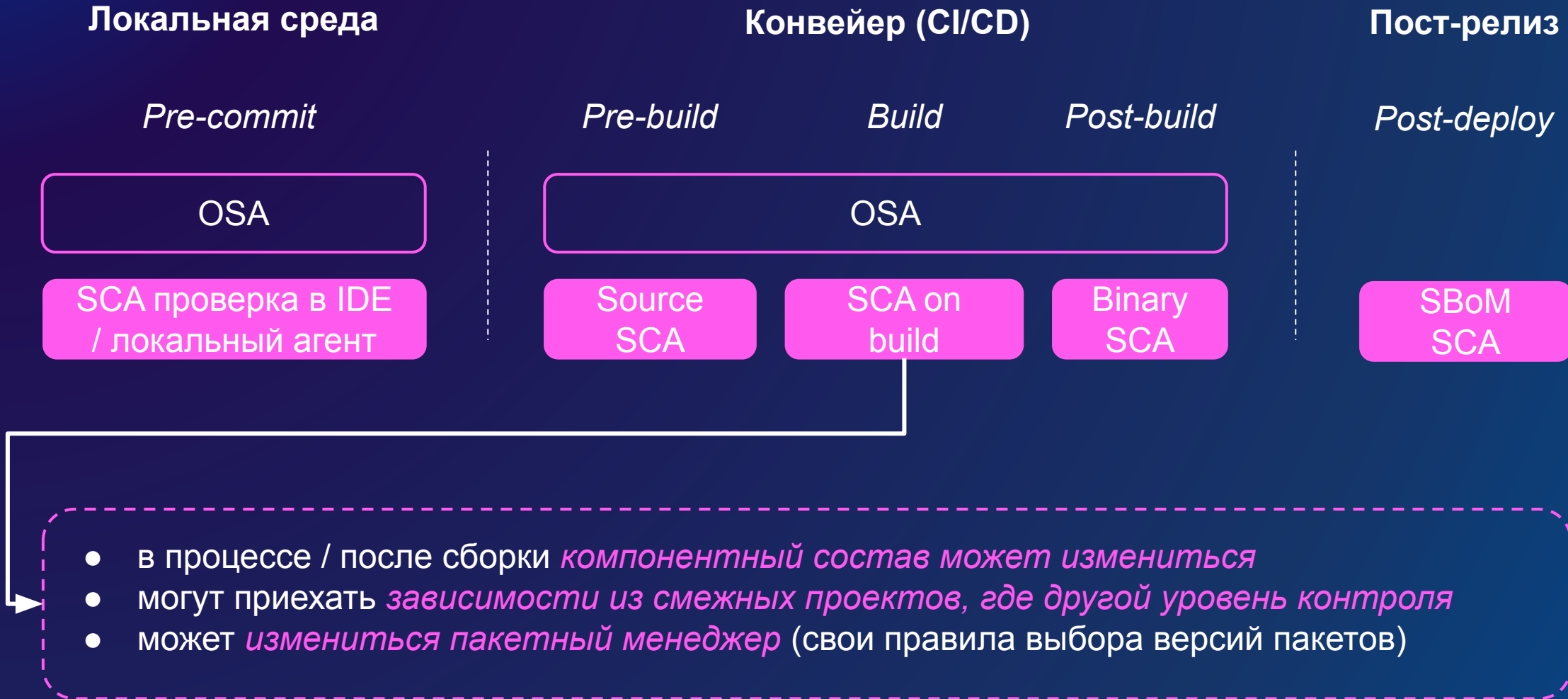
[1] SCA проверка в IDE



[2] Source SCA



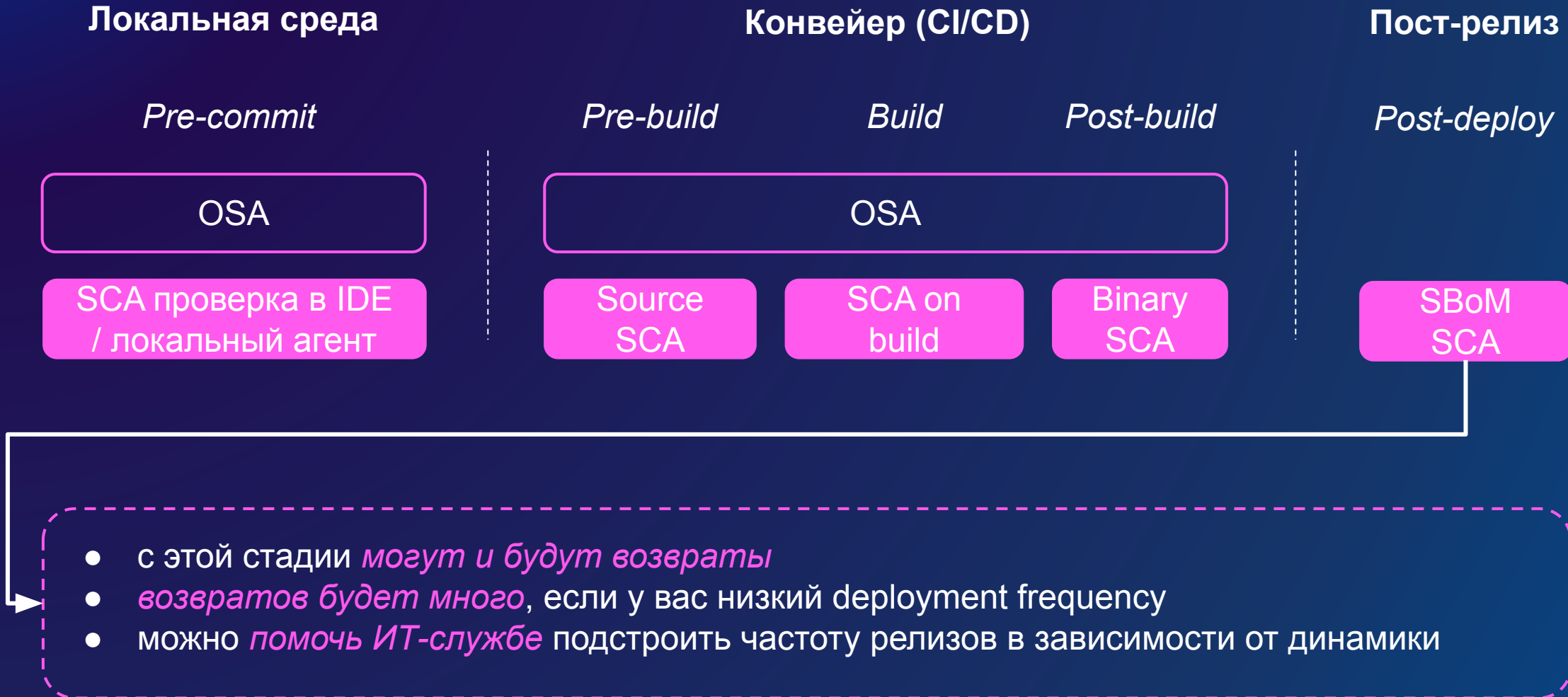
[3] SCA *on build*



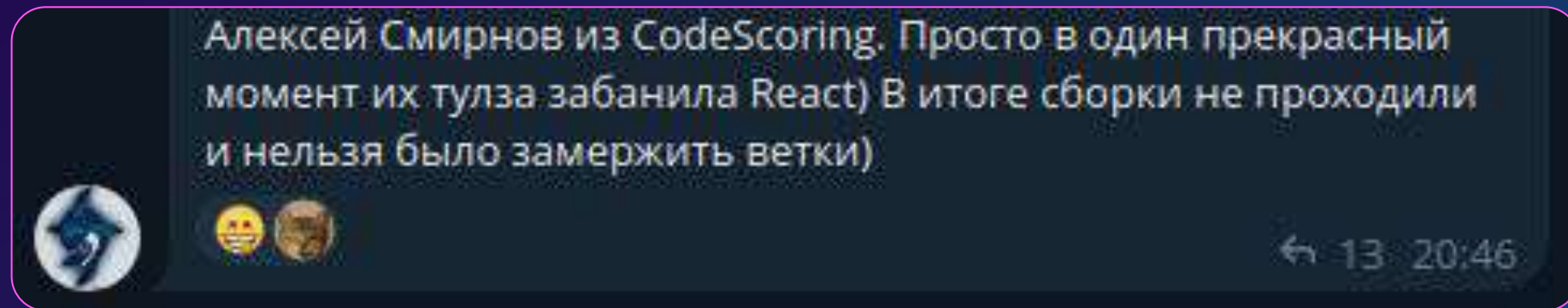
[4] Binary SCA



[5] *SBoM* SCA



Предварительные *наблюдения*



*Обсуждения после CyberWave [@рохек](#).
Важно уметь корректно настроить политики *у себя*.*

03



Полезные числа

Для понимания, где мы можем «встрять»



Сколько сторонних уязвимостей в проектах?

Экосистема	Зависимостей	Директивных	Транзитивных	Уязвимостей, когда нет Sec	Уязвимостей, когда Sec есть
JavaScript	488	42	446	50	10
Java	90	54	36	60	6
C#	58	23	35	4	1
Python	39	16	23	17	2

* для читаемости приведена статистика по наиболее популярным языкам

** из доклада «Проблемы отцов и детей: Аналитика и триаж транзитивных зависимостей», PHD, 2024

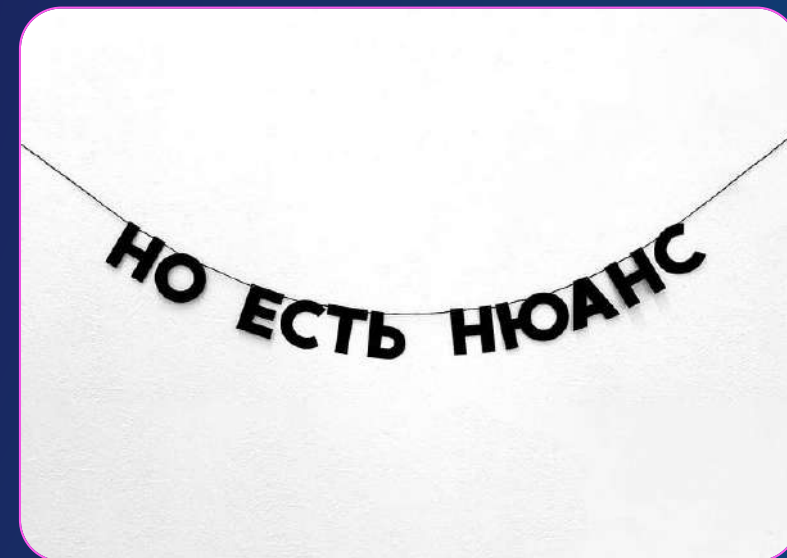
23

МИН

Столько в среднем обходится анализ открытой уязвимости:
понимание её проблемы и описание человеческим языком.
Но помним, что есть ещё стадия митигации.

* из доклада «Проблемы отцов и детей: Аналитика и триаж транзитивных зависимостей», PHD, 2024

до 86%
уязвимостей
в транзитиве



*если делаете достижимость, следите,
чтобы проверялись и транзитивы*

~100

уязвимостей в год

накапливает ПО, если не выходят обновления безопасности



*На 20-30% / год
средний проект
обрастает
зависимостями*



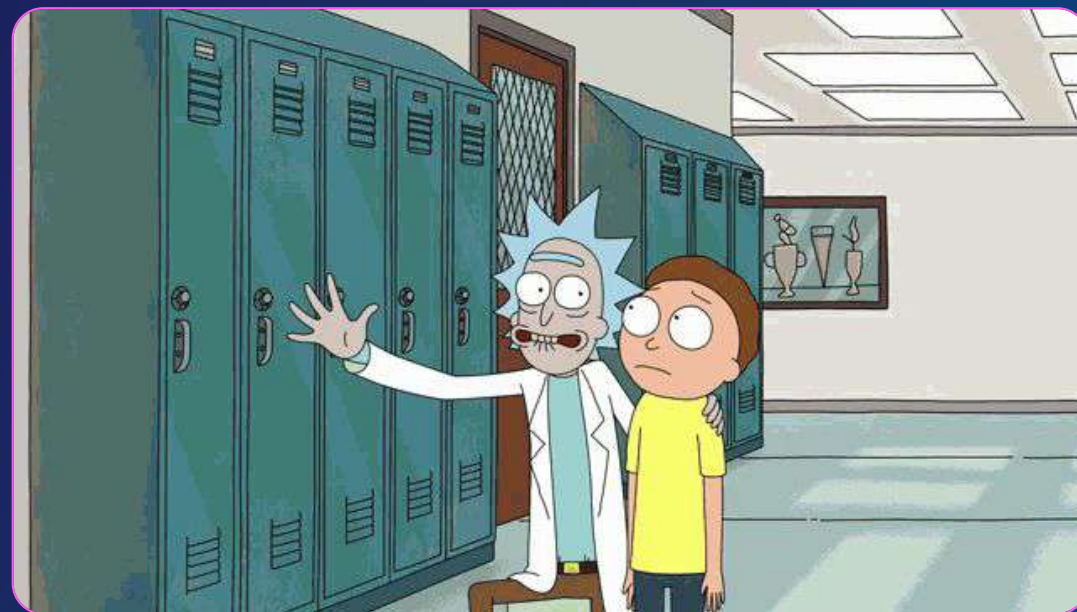
*Возможно, все зависимости полезные,
но это не точно*

26% фиксов НЕ чистые обновления безопасности

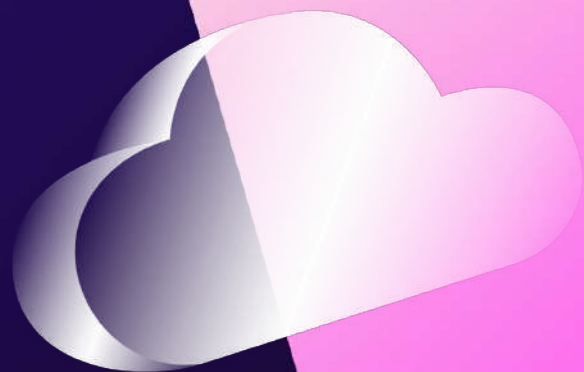


Исправление приходит с *новой функциональностью* или фикс *ненастоящий / слабый*.
Можно поймать проблему с *совместимостью*.

*~1-10 мин.
на выбор подходящего
компонента, если
сработал фаервол /
локальная проверка*



04



Модель

Какая может быть модель, как вы думаете?



Частота развертываний — главная модель

- Контролируйте ранние стадии разработки, это позволяет получать меньше возвратов (база)
- Если релизитесь редко, то AppSec может доходчиво объяснить в т. ч. DevOps-практиками
- Но учитывайте *пост-релизные возвраты*

	Elite	High	Medium	Low
Частота развертываний Сколько в среднем проходит времени между окончанием разработки кода и его развертыванием на продуктивном окружении?	От раза в день до раза в неделю	От раза в 2 недели до раза (3 вхождения) в месяц	От раза в месяц до раза (3 вхождения) в 3 месяцев	От раза в 3 месяца до раза (3 вхождения) в 6 месяцев
Срок поставки Как часто происходит развертывание новой версии приложения на продуктивное окружение?	Меньше дня	От дня до недели	От 1 до 3 месяцев	От 3 до 6 месяцев
Время восстановления Сколько в среднем времени занимает восстановление приложения на продуктивном окружении после инцидента, деградации сервиса или обнаружения ошибки, влияющей на пользователей приложения?	Меньше дня*	Меньше дня*	От дня до недели	От 1 до 3 месяцев
Неуспешные изменения Какой процент развертываний на продуктивном окружении приводит к деградации приложения или инцидентам и требует устранения последствий?	6-15%**	6-15%**	6-15%**	16-30%

Исследование состояния DevOps в России 2024,
<https://devopsrussia.ru/wp-content/themes/devopsrussia/assets/docs/StateOfDevOpsRussia2024.pdf>

Насколько часто *стреляет пост-релиз?*



Если всё было чисто на релизе, то можно ожидать примерно такую картинку:

~25 уязвимостей *в квартал*

~2.5 уязвимостей, которые *окажутся достижимыми*

=> Следует релизиться *чаще, чем раз в квартал*

Отслеживание *нескольких в релизов* в полях *добавляет работы* — $O(n^2)$.

С системными пакетами немного другая история, но об этом в следующих сериях.

05



Выводы



Найдите процессы вашей разработки и влияйте

- ❖ Можно накрутить SCA-опыт
эффективность
- ❖ *Считайте* свои трудозатраты
- ❖ Сдвинуться влево — *следует*

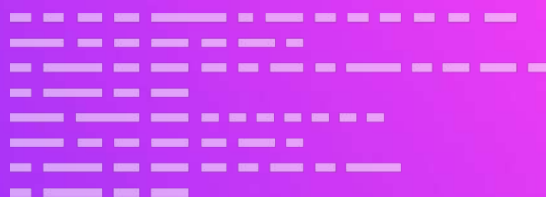


Спасибо!



Оценка доклада

Алексей Смирнов
alexey@codescoring.ru



Следите за нами

