CODE SCORING

Три этюда о защите цепочки поставки программного обеспечения

Алексей Смирнов
Основатель CodeScoring

Компания Profiscope

С 2011 года занимаемся анализом исходных кодов и артефактов разработки.

Последние 5 лет:

- **Разработка собственных анализаторов исходного кода**
- **Услуги автоматизированного аудита приложений: сопровождение сделок M&A и проведение IT Due Diligence в интересах частных и государственных компаний, инвестиционных холдингов**
- **Выпускаем решение композиционного анализа ПО CodeScoring**

> 5 лет обучения аналитических моделей и разработки анализаторов

> 200 млн. проанализированных OSS-проектов и библиотек

> 20 млн. обработанных **уникальных** профилей программистов в мире

Решение композиционного анализа CodeScoring

В 2019 году создано решение композиционного анализа CodeScoring.
В 2021 выведено на рынок. Включает в себя функциональность:

- **OSA** /Open Source Analysis
- **SCA** /Software Composition Analysis
- **TQI** /Teams & Quality Intelligence

Немного про решение:

- 15+ интегрированных баз уязвимостей
- собственная база знаний про мировой open source
- собственная база знаний про open source лицензии
- интеграция на всех этапах разработки ПО
- широкий набор политик отслеживания и блокирования рисков
- защита цепочки поставки от популярных атак
- вся ключевая функциональность собственной разработки

Уже используют CodeScoring

- Банки и Финансовые компании
- Нефтяные и Энергетические компании
- Телеком и Медиа
- Государственные учреждения (госпорталы)
- Разработчики ПО

По итогам 2022 г. реализован ряд крупных проектов федерального уровня и мы непрерывно получаем обратную связь от заказчиков для непрерывного улучшения продукта.

Безопасность цепочки поставки программного обеспечения

Проверка проектов при приёмке заказного ПО

Анализ проектов в разрезе авторов

Наш вклад в Сообщество

С 2022 года участник SDL-сообщества. Экспертиза команды отмечена ФСТЭК России и ИСП РАН, ведется сотрудничество в части применения основной компетенции — анализ безопасности и качества заимствованных компонентов.

Совместно с Фобос-НТ организовали встречу SDL-сообщества в СПб и планируем выводить её на регулярную основу.

С 2019 года ведем конференцию CodeMining в сообществе OpenDataScience

В 2021 сделали открытый курс по лицензированию Open Source. Поддерживаем образовательные учреждения

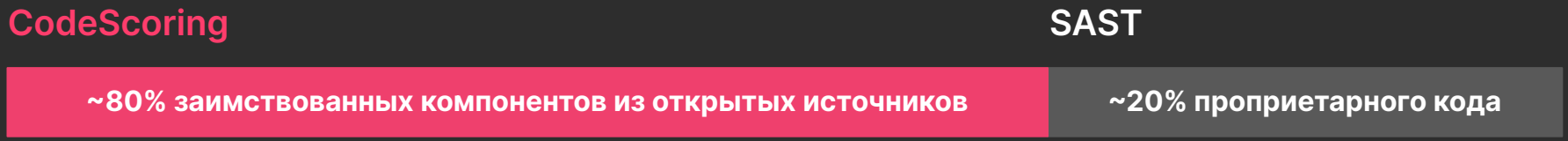
Спикеры конференций: Highload++, PHDays, DevOpsConf, Data Fest, ChaosConstructions, HackConf, PyCon и PiterPy и иных

Этюд #1. Незнание

/критический

Безопасная разработка с использованием заимствованных компонентов

Знаете ли вы, из чего состоит ваш продукт?



CodeScoring проверяет большую часть программного проекта
на **известные уязвимости и лицензионную чистоту**.

Этюд #1. Незнание

1. Зависимости не фиксируются

Этюд #1. Незнание

1. Зависимости не фиксируются
2. Фиксируются, включением в код 😂

Этюд #1. Незнание

1. Зависимости не фиксируются
2. Фиксируются, включением в код 😂
3. Фиксируются, но в *.txt-файле (иногда не все)

Этюд #1. Незнание

1. Зависимости не фиксируются
2. Фиксируются, включением в код 😂
3. Фиксируются, но в *.txt-файле (иногда не все)
4. Фиксируются, без разделения окружений сборки

Этюд #1. Незнание

1. Зависимости не фиксируются
2. Фиксируются, включением в код 😂
3. Фиксируются, но в *.txt-файле (иногда не все)
4. Фиксируются, без разделения окружений сборки
5. Фиксируются, но без версий

Этюд #1. Незнание

1. Зависимости не фиксируются
2. Фиксируются, включением в код 😂
3. Фиксируются, но в *.txt-файле (иногда не все)
4. Фиксируются, без разделения окружений сборки
5. Фиксируются, но без версий
6. Фиксируются, но без фиксированной версии

Этюд #1. Незнание

1. Зависимости не фиксируются
2. Фиксируются, включением в код 😂
3. Фиксируются, но в *.txt-файле (иногда не все)
4. Фиксируются, без разделения окружений сборки
5. Фиксируются, но без версий
6. Фиксируются, но без фиксированной версии
7. Фиксируются, но без транзитивности

Этюд #1. Незнание

1. Зависимости не фиксируются
2. Фиксируются, включением в код 😂
3. Фиксируются, но в *.txt-файле (иногда не все)
4. Фиксируются, без разделения окружений сборки
5. Фиксируются, но без версий
6. Фиксируются, но без фиксированной версии
7. Фиксируются, но без транзитивности
8. Фиксируются, но без хэш-сумм

Этюд #1. Незнание

1. Зависимости не фиксируются
2. Фиксируются, включением в код 😂
3. Фиксируются, но в *.txt-файле (иногда не все)
4. Фиксируются, без разделения окружений сборки
5. Фиксируются, но без версий
6. Фиксируются, но без фиксированной версии
7. Фиксируются, но без транзитивности
8. Фиксируются, но без хэш-сумм
9. Фиксируются, но не соотносятся с релизами продуктов

Этюд #1. Незнание

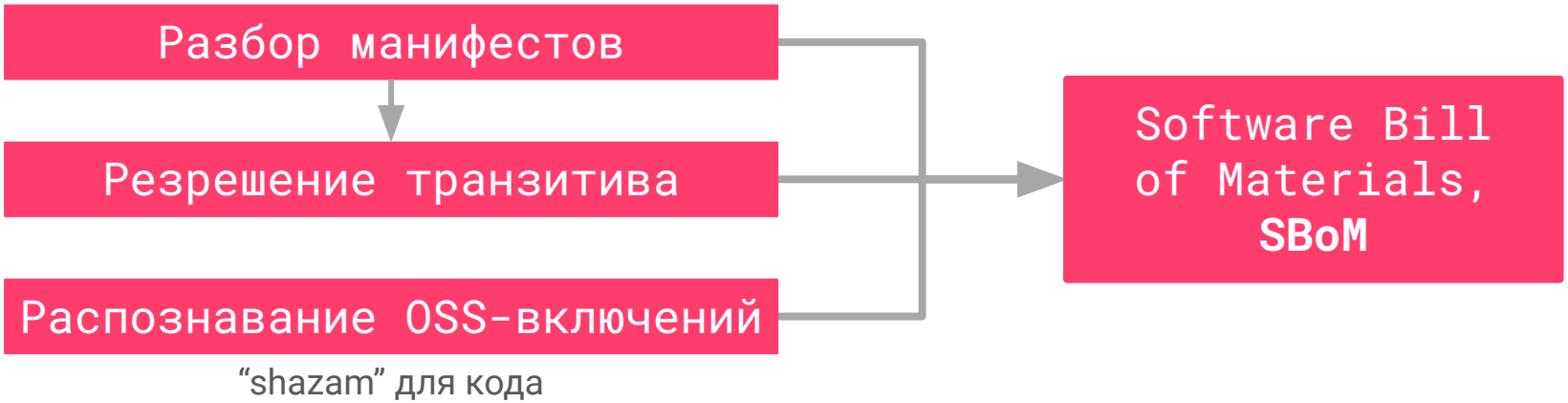
1. Зависимости не фиксируются
2. Фиксируются, включением в код 😂
3. Фиксируются, но в *.txt-файле (иногда не все)
4. Фиксируются, без разделения окружений сборки
5. Фиксируются, но без версий
6. Фиксируются, но без фиксированной версии
7. Фиксируются, но без транзитивности
8. Фиксируются, но без хэш-сумм
9. Фиксируются, но не соотносятся с релизами продуктов
10. Фиксируются, но не сохраняются (не кэшируются)

Этюд #1. Незнание

1. Зависимости не фиксируются
2. Фиксируются, включением в код 😂
3. Фиксируются, но в *.txt-файле (иногда не все)
4. Фиксируются, без разделения окружений сборки
5. Фиксируются, но без версий
6. Фиксируются, но без фиксированной версии
7. Фиксируются, но без транзитивности
8. Фиксируются, но без хэш-сумм
9. Фиксируются, но не соотносятся с релизами продуктов
10. Фиксируются, но не сохраняются (не кэшируются)

Правильная инвентаризация ПО

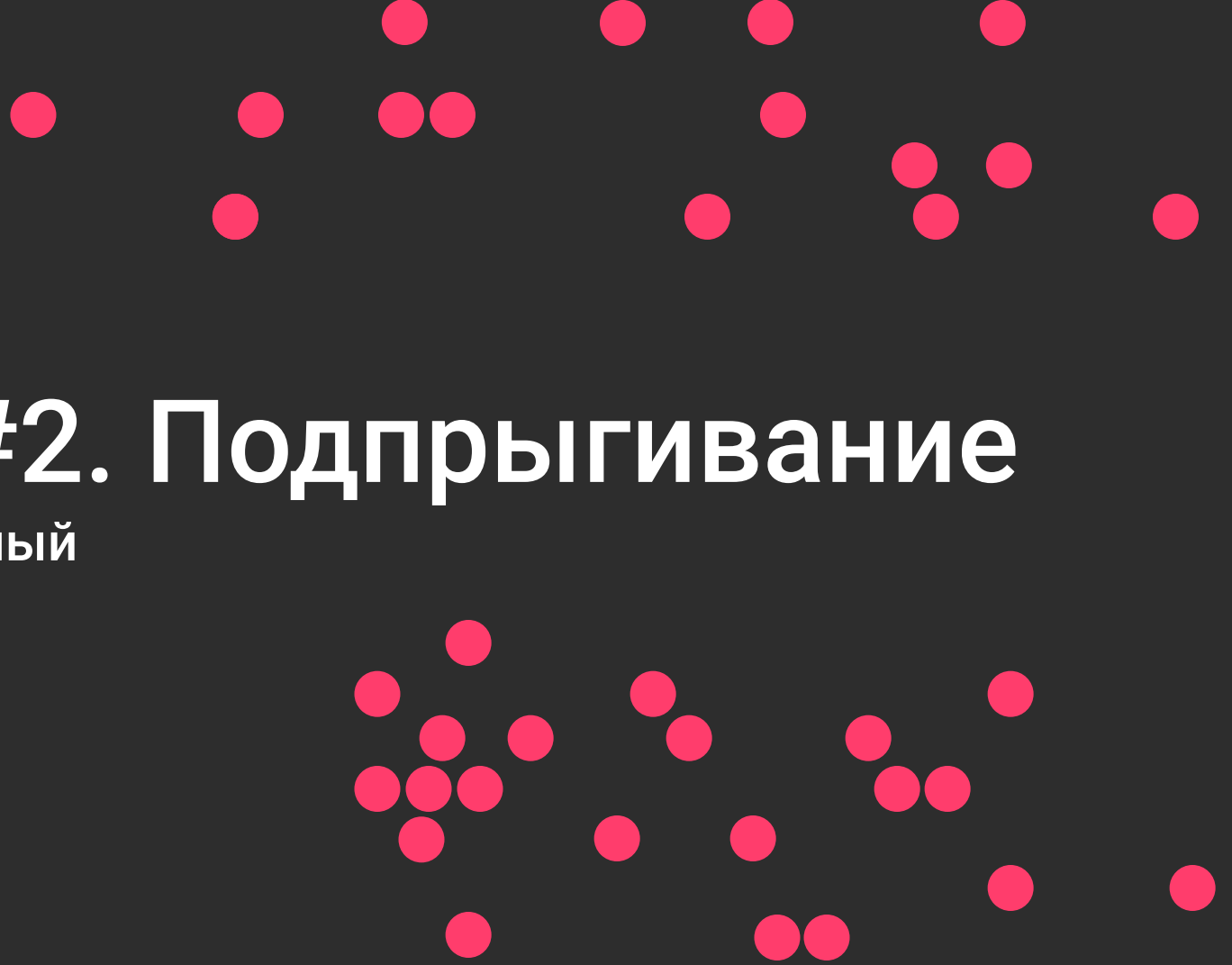
Найти манифесты, разобрать их, идентифицировать компоненты.
Нужно не забыть про «включения» Open Source в ваш код.



Хранить и проверять все используемые компоненты в прокси-репозитории.

Этюд #2. Подпрыгивание

/иллюстративный



Этюд #2. Подпрыгивание

«Когда опубликовали log4shell, мы склонировали весь наш Gitlab и grep’ом искали log4j в исходниках» — (c) из полей.

Плюсы: это весело.

Минусы:

- ???

Этюд #2. Подпрыгивание

«Когда опубликовали log4shell, мы склонировали весь наш Gitlab и grep’ом искали log4j в исходниках» — (c) из полей.

Плюсы: это весело.

Минусы:

- это только последние версии кода!

Этюд #2. Подпрыгивание

«Когда опубликовали log4shell, мы склонировали весь наш Gitlab и grep’ом искали log4j в исходниках» — (c) из полей.

Плюсы: это весело.

Минусы:

- это только последние версии кода!
- проблема может быть в транзитивных

Этюд #2. Подпрыгивание

«Когда опубликовали log4shell, мы склонировали весь наш Gitlab и grep’ом искали log4j в исходниках» — (c) из полей.

Плюсы: это весело.

Минусы:

- это только последние версии кода!
- проблема может быть в транзитивных
- кто проверит старые сборки?

Этюд #2. Подпрыгивание

«Когда опубликовали log4shell, мы склонировали весь наш Gitlab и grep’ом искали log4j в исходниках» — (c) из полей.

Плюсы: это весело.

Минусы:

- это только последние версии кода!
- проблема может быть в транзитивных
- кто проверит старые сборки?
- неумение процессов

Этюд #2. Подпрыгивание

«Когда опубликовали log4shell, мы склонировали весь наш Gitlab и grep’ом искали log4j в исходниках» — (c) из полей.

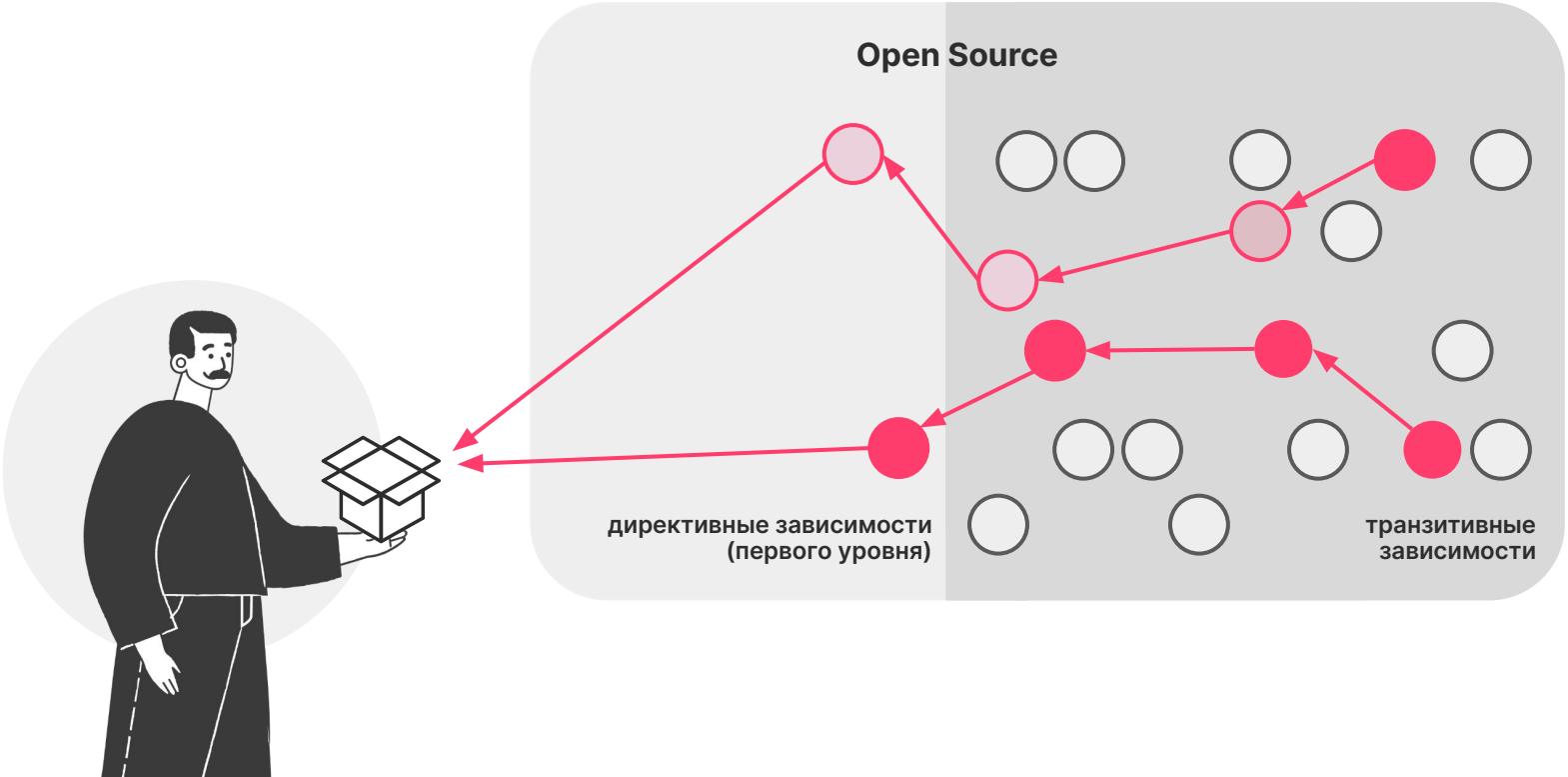
Плюсы: это весело.

Минусы:

- это только последние версии кода!
- проблема может быть в транзитивных
- кто проверит старые сборки?
- неумение процессов
- неумение инструментария

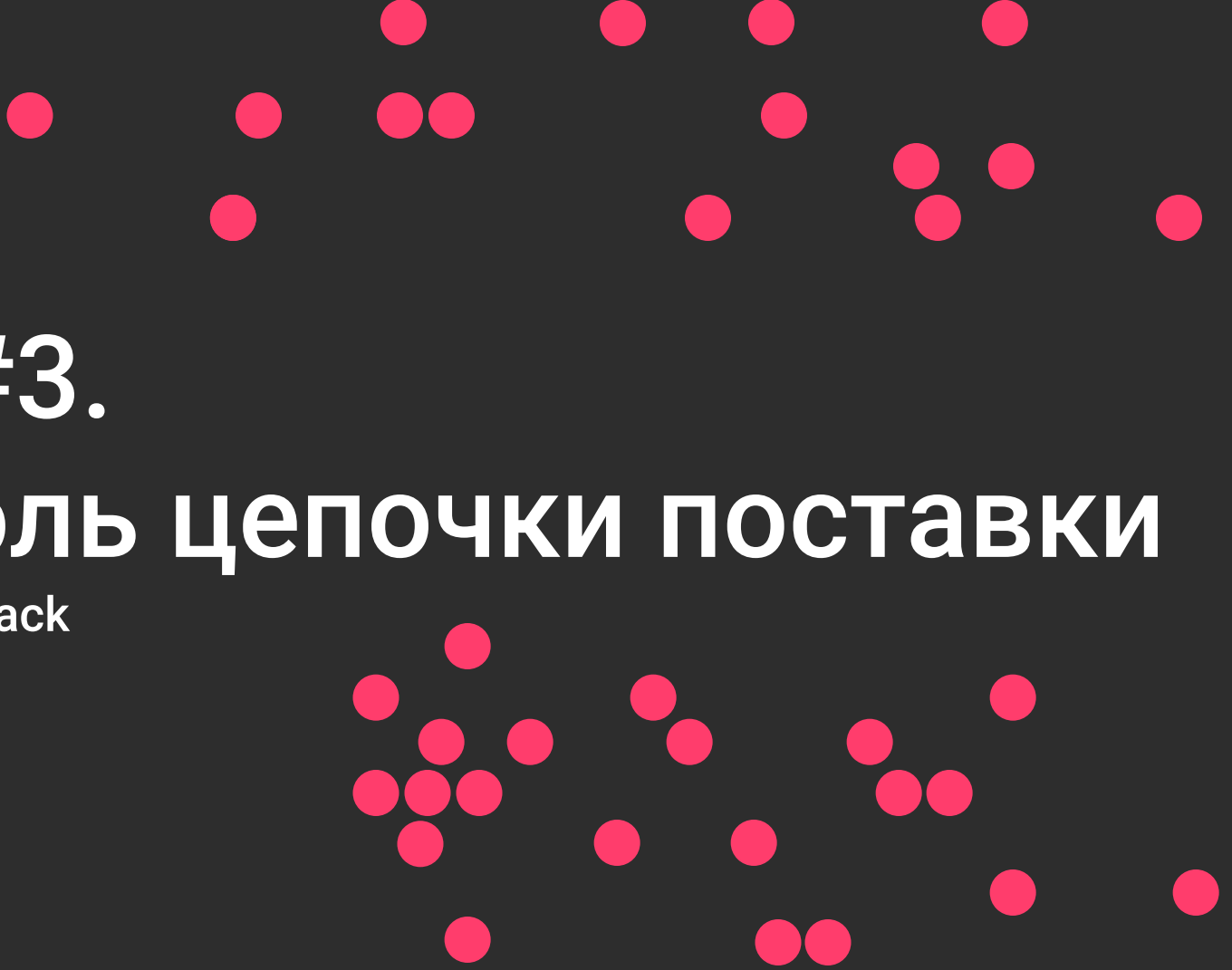
Проблема глубже чем кажется

Проверка заимствованных компонентов и понимание состава программного продукта — ключевой аспект безопасной разработки.



Этюд #3.
Контроль цепочки поставки

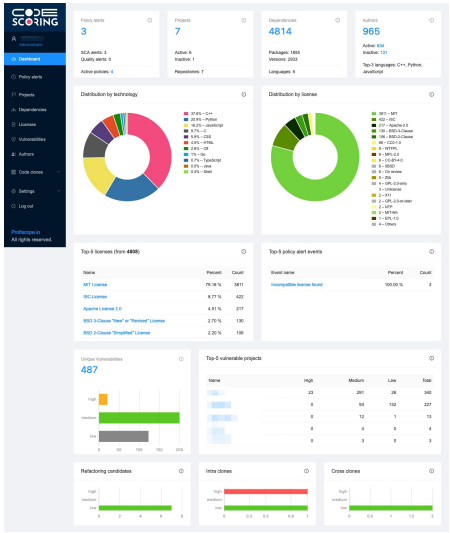
/take the power back



Этюд #3. Контроль цепочки поставки

Взять под контроль цепочку поставки:

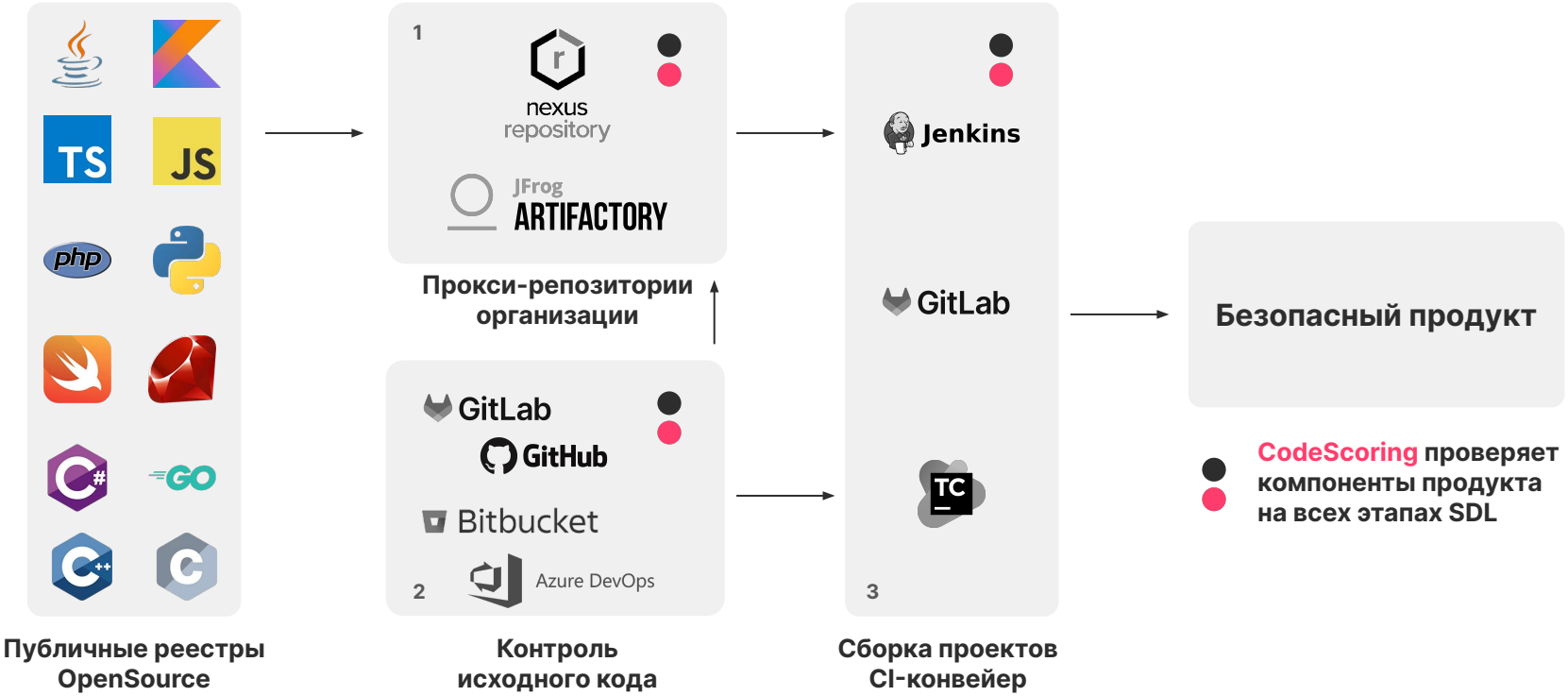
1. прокси-репозиторий
2. контроль пакетов на сборке
3. непрерывный мониторинг кода



Этюд #3. Контроль на всех этапах SDL

«Не надо кошмарить разработчиков» (с)

Важно применять **разные** политики на **разных** этапах разработки.



Подводя итоги

Всё хорошо и плохо

Open Source это целый мир со своими культурными ценностями, которые нужно уважать, но не стоит забывать о безопасности.

Хорошо

Повышение скорости разработки (TTM)

Широкая доступность открытых компонентов

Свобода изучения, использования и распространения (согласно лицензии)

Возможность самостоятельной модификации программ

Плохо

Состав продуктов часто не зафиксирован

Контроль точечный, пока не потребуют

Риски атаки на цепочку поставки (вредонос, опасные уязвимости на поверхности атаки)

Отсутствие процессов контроля стороннего ПО

Ожидание нормативно-правовой базы

Регулятор ведет работу (https://t.me/sdl_community/2218)
в части учета сторонних компонентов.

Важно довести работу в части систематизации процессов работы с заимствованными и привлекаемыми компонентами:

1. Должна официально появиться машиночитаемая нотация реестра компонентных связей (SBoM).

2. Должно быть организовано систематическое донесение его пользы до сообщества разработки в виде рекомендаций и методик оценки с учетом определения Поверхности атаки и других факторов.

Сейчас эта информация и инструменты есть только под пальцами практикующих AppSec’ов в реальном секторе.

Спасибо за внимание!

https://codescoring.ru/

Запись в росреестре №13008 от 05.03.2022