CODE SCORING

Эффективный композиционный анализ

Путь к успешному внедрению
и упражнения на статику

#АЛЕКСЕЙ СМИРНОВ

CodeScoring /компания Profiscope

С 2011 года занимаемся анализом исходных кодов и артефактов разработки:

- Разработка собственных анализаторов исходного кода
- Услуги автоматизированного аудита приложений
- Выпускаем решение композиционного анализа ПО CodeScoring

> 5 лет обучения аналитических моделей и разработки анализаторов

> 200 млн. проанализированных OSS-проектов и библиотек

> 30 экспертов работает над продуктом CodeScoring

Платформа композиционного анализа

В 2019 году создано решение композиционного анализа CodeScoring.

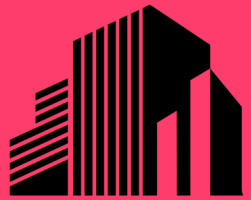
В 2021 выведено на рынок. Включает в себя функциональность:

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

Немного про решение:

- 20+ 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 (+интеграция с **Kaspersky**)
- 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 22-23 гг. реализован ряд крупных проектов федерального уровня и мы непрерывно получаем обратную связь от заказчиков для регулярного улучшения продукта.

Безопасность цепочки поставки программного обеспечения

Проверка проектов при приёмке заказного ПО

Анализ проектов в разрезе авторов

Наш вклад в Сообщество

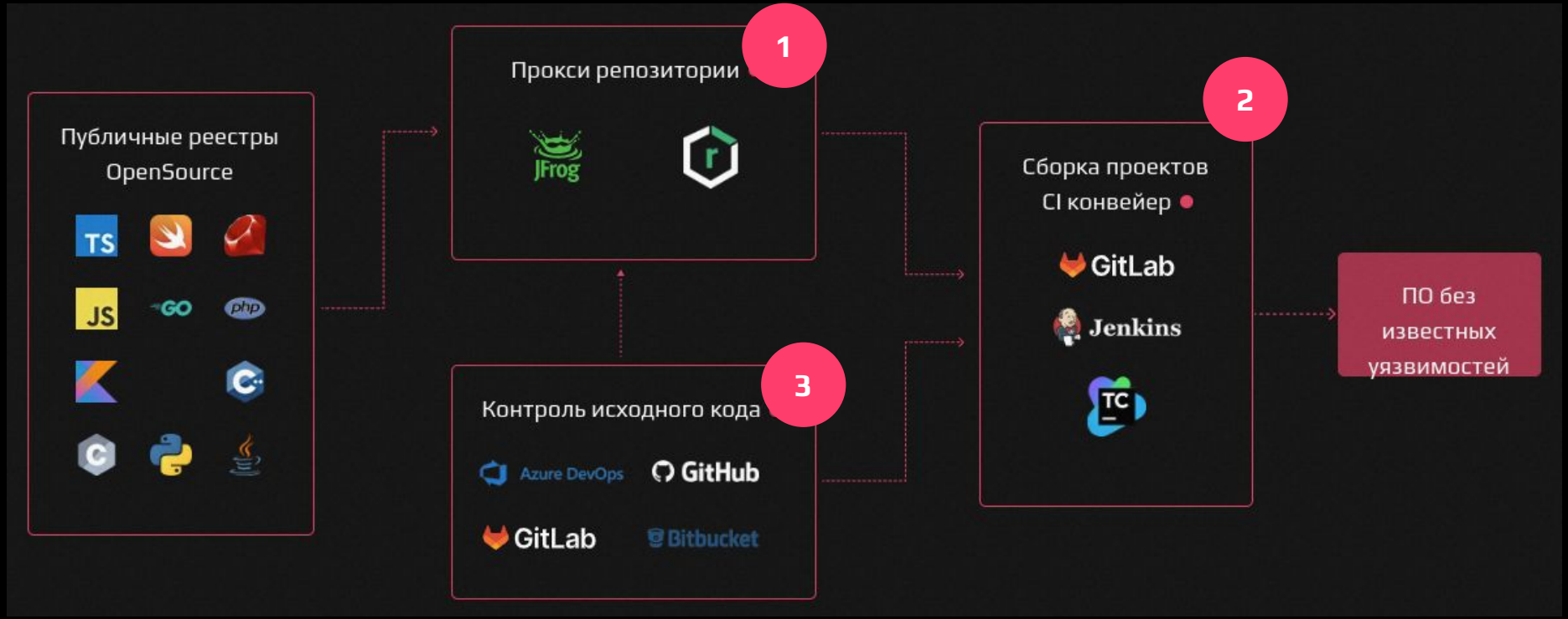
- С 2022 года Участник SDL-сообщества.
- Экспертиза команды отмечена ФСТЭК России и ИСП РАН, ведется сотрудничество в части применения основной компетенции — анализ безопасности и качества сторонних компонентов.
- Членство в ТК362, руководство РГ7 по разработке проекта ГОСТа «ЗИ. Композиционный анализ программного обеспечения».
- Совместно с Фобос-НТ организуем встречу SDL-сообщества в Петербурге на регулярной основе: t.me/SDL_Community_SPb

В предыдущих сериях...

Поговорили про печаль и радость практики защиты цепочки поставки.

Не забыли и про ожидания от нормативно-правовой базы.

Практика анализа сторонних компонентов



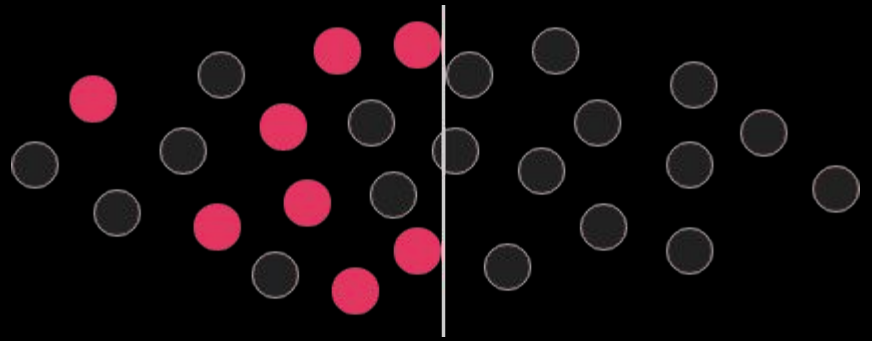
[OSA] Защита цепочки поставки

Open Source Analysis

Главная задача безопасности —
не пропустить в разработку ненужное:
вредонос и ПО с «особенностями».

Вторая задача — сохранять
зависимости из которых вы собираете
ваши продукты.

Проблема: 70% разработчиков
не уверены в том, что все их
зависимости проходят через прокси.



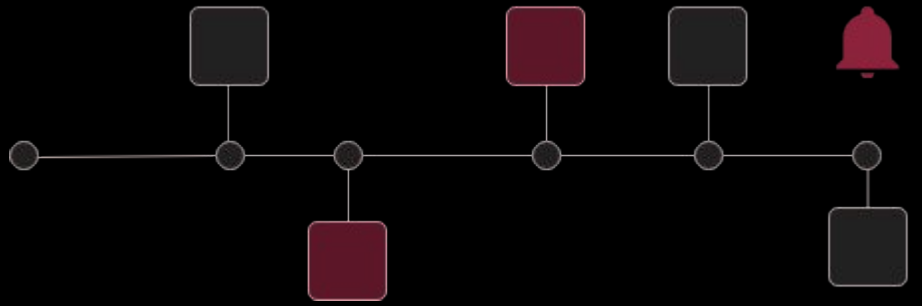
[SCA] Композиционный анализ

Software Composition Analysis

Главная задача — не пропустить в сборку ненужное и отслеживать ранее выпущенные релизы.

Ненужное — уязвимости, лицензионные и иные особенности компонентов.

Следует проверять на разных этапах разработки и сборки ПО.



Рекомендуемая цепочка выполняемых проверок для **выпускаемых** продуктов

CODE SCORING

проверка на вредонос в прокси-репо	регулярное сканирование исходных кодов	сканирование исходных кодов в CI-конвейере	сканирование артефактов сборки (CI)	сканирование образов сборки (CI)
1	2	3	4	5
проверка **всех компонентов** попадающих в разработку на **наличие вредоносного кода** и особо опасные уязвимости, подозрительные факторы если есть — блокируем и в песочницу	проверка и **заблаговременное предупреждение** разработчиков об использовании небезопасных компонентов уведомляем ответственных: дефект-трекеры, почта	**предварительная** проверка исходных кодов поступающих на конвейер (валим) сборку, если нарушена политика исправление по рекомендациям	проверка того, **что вошло в сборку артефакта**, иногда в неё входит не только то, что написали разработчики валим сборку, если нарушена политика исправление по рекомендациям	проверка собранных образов на небезопасные компоненты. в них появляется знание о **системных пакетах** валим сборку, если нарушена политика исправление по рекомендациям
OSA	**фиксация SBoM**	**фиксация SBoM**	**фиксация SBoM**	**фиксация SBoM**
OSS firewalling	**design**	**pre-build**	**build**	**build**

Рекомендуемая цепочка выполняемых проверок для **выпущенных** продуктов

CODE SCORING

[рекуррентные проверки SBoM]

6

регулярная перепроверка компонентов ранее выпущенных версий продуктов

в них уязвимости не появляются,
а обнаруживаются!

нашли проблему — отзывная кампания и выпуск коррекции

перепроверка SBoM

operations

[вывод версии продукта из эксплуатации]

7

выпущенные продукты заканчивают свою жизнь, вместе с ними и SBoM-спецификация

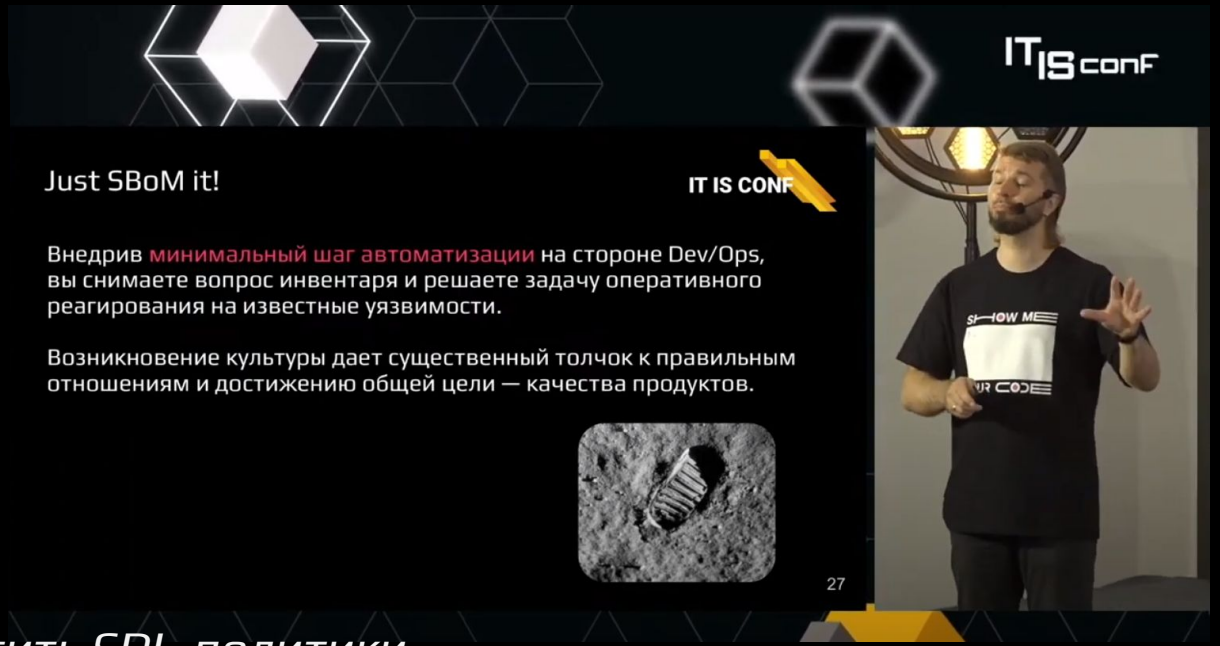
проводим ревью компонентного состава и, при необходимости, — **выводим компоненты из эксплуатации**

архивация SBoM

post-operations

Ключ к успеху — отслеживание качества безопасности в ранее выпущенных релизах и управление процессом вывода из эксплуатации.

Вроде всё просто — нужно построить SBoM



«Построить SBoM, вырастить SDL-политики, воспитать культуру безопасной разработки» @IT IS Conf, 23

Треугольник внедрения любой практики

Процессы

Любая организация может выстроить процессы РБПО.

Эти процессы должны учитывать:

- компетенции
- технический стек
- особенности организации (политики безопасности)

Технологии

Любая организация может построить SBoM.

Это поможет: процессам и людям.

Но людей мало.

Упражнения на статику

или как помочь людям

Композиционный анализ —
легковесная и полезная практика

В сравнении со статикой
«съедает» меньше ресурса
на процедуру триажа (разметки)

Но есть один нюанс

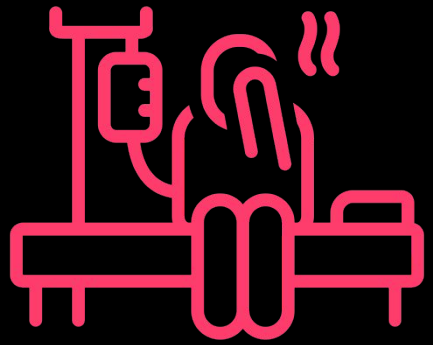
Открытые фиды обладают «особенностями»

Из описания уязвимости не всегда понятно, где именно проблема в коде компонента

Открытые фиды обладают «особенностями»

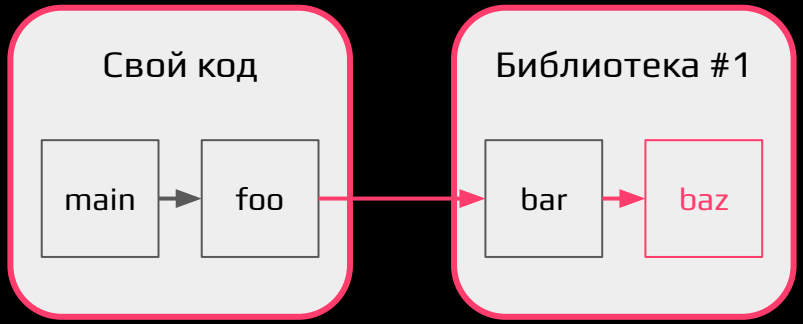
Из описания уязвимости не всегда понятно, где именно проблема в коде компонента

А когда стало понятно — непонятно «о чем это» для вашего приложения



Исправить директив — относительно легко

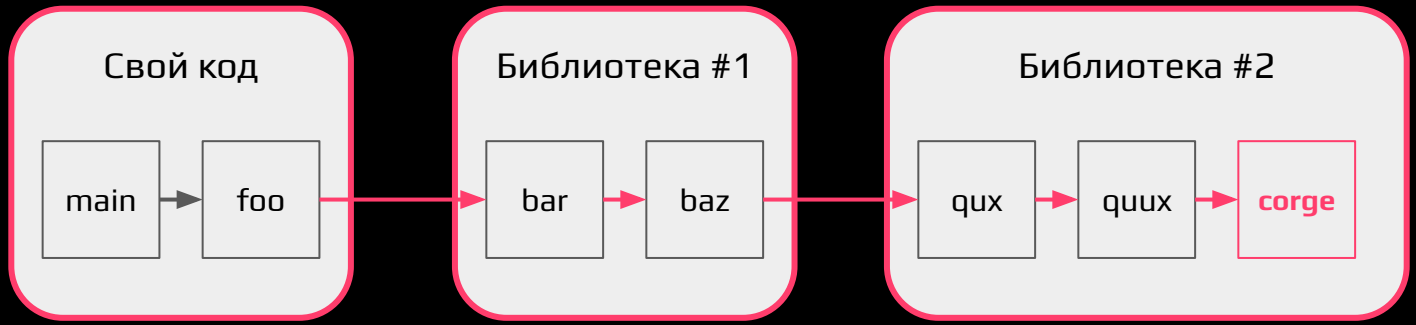
/обновить не вникая или в триаж



уязвимая функция
в директивной
зависимости

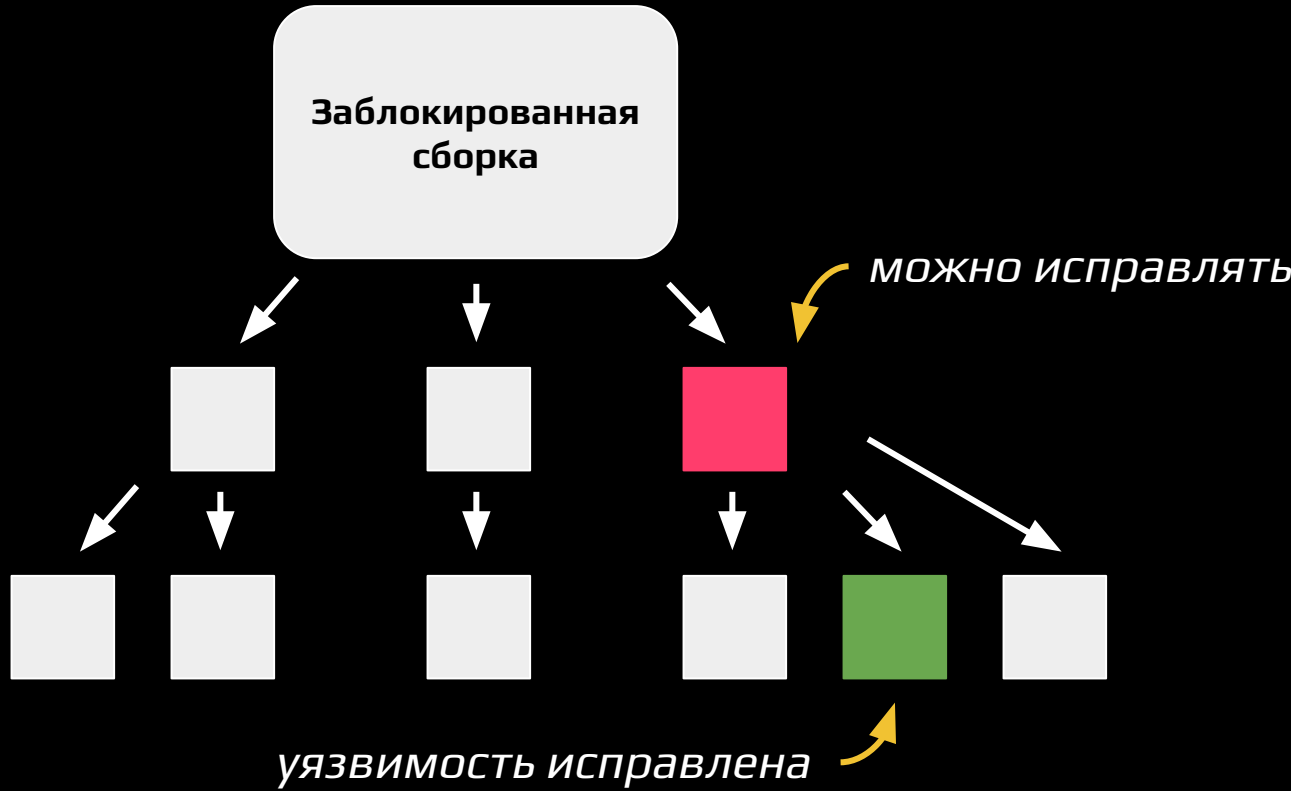
Проверить и исправить транзитив — сложно

/обновить скорее не получится или дорого, триажить непросто



уязвимая функция
в транзитивной
зависимости

Добиться исправления ещё сложнее



Варианты исправления транзитива

- Ничего не делать с транзитивами

Варианты исправления транзитива

- ~~Ничего не делать с транзитивами~~
- Верификация (подтверждение и отнесение к поверхности атаки)
- Ждать обновления головной компоненты (долго или никогда)
- Произвести патчинг цепочки самостоятельно - жить с этим, ждать
- Закрыться наложенным средством безопасности приложений (н-р WAF)

Варианты исправления транзитива

- ~~Ничего не делать с транзитивами~~
- Верификация (подтверждение и отнесение к поверхности атаки)
- ~~Ждать обновления головной компоненты (долго или никогда)~~
- Произвести патчинг цепочки самостоятельно - жить с этим, ждать
- Принять компенсирующие меры

85%

уязвимостей в **транзитивных** зависимостях
(и они **могут** находиться на поверхности атаки)

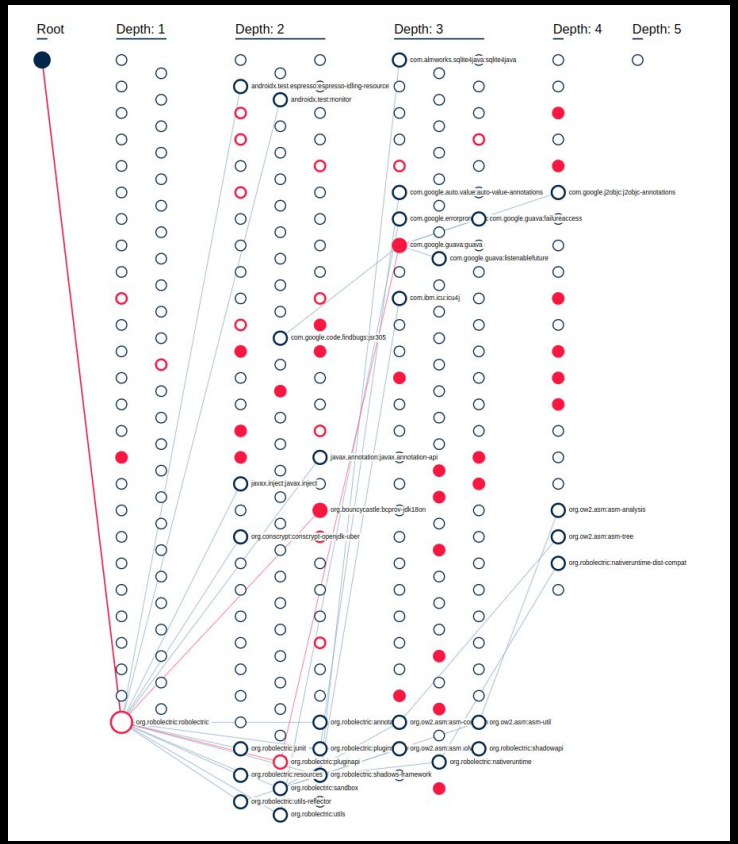
Компонентов много, проверять много

~200

среднее количество зависимостей
в «серверных» проектах
(Java, Python, C#, ...)

~1500

среднее количество зависимостей
в JavaScript/TypeScript проектах



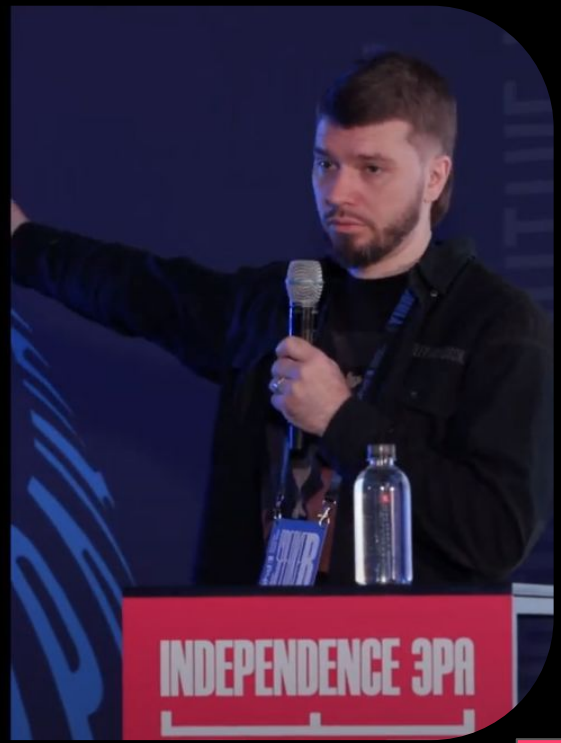
«Я же говорил» /22 год @PHD11

Взаимоустремления

SCA ⇦⇨ **SAST**

Добавление проактивных функций SAST

Применение SAST опыта в контексте SCA



И ещё говорил /22 год @PHD11

SCA должен эволюционировать

- проблемы словарей зависимостей должны исправляться систематически (но мы и сами справимся)
- SCA должен быть более прозорливым в части SAST
- учитывать проблемы пакетных индексов и карму авторов пакетов

Как мы повышаем точность SCA

База дедуплицированных уязвимостей

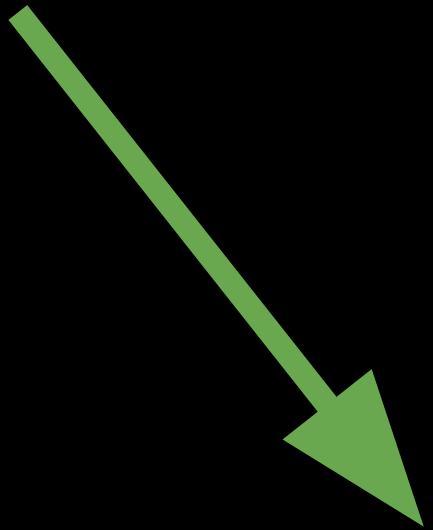
Более 20 фидов, которые обогащаются информацией о: коммитах, авторах, эксплоитах, разных оценках CVSS и пр.

Разметка и описание известных уязвимостей

Ручная разметка и создание понятных описаний уязвимостей группой профессиональных аналитиков, в сотрудничестве с НТЦ Фобос-НТ и МГТУ им. Баумана.

Проверка наличия трассы до уязвимых методов

Интеграция возможностей статического анализа в композиционный с применением технологий ИСП РАН.



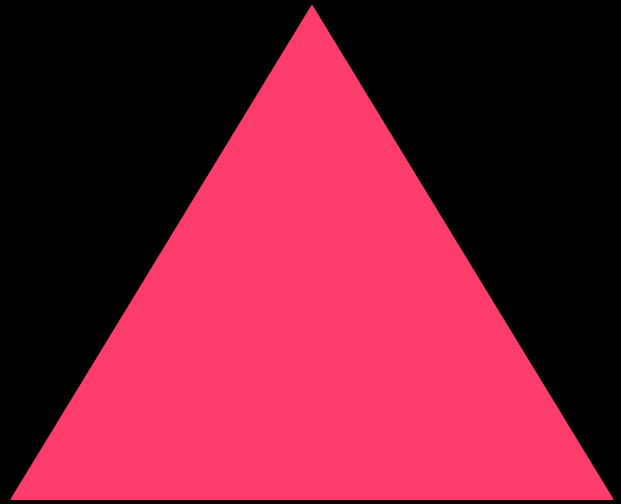
>90%

Снижение трудозатрат на разбор размеченных уязвимостей

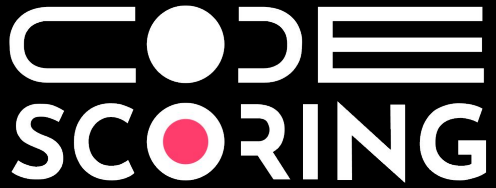
Подтверждено практикой CodeScoring и публичными[1] исследованиями

[1] On the Effect of Transitivity and Granularity on Vulnerability Propagation in the Maven Ecosystem https://arxiv.org/pdf/2301.07972.pdf

Больше треугольников!



Развитие сотрудничества во имя РБПО



CODE SCORING

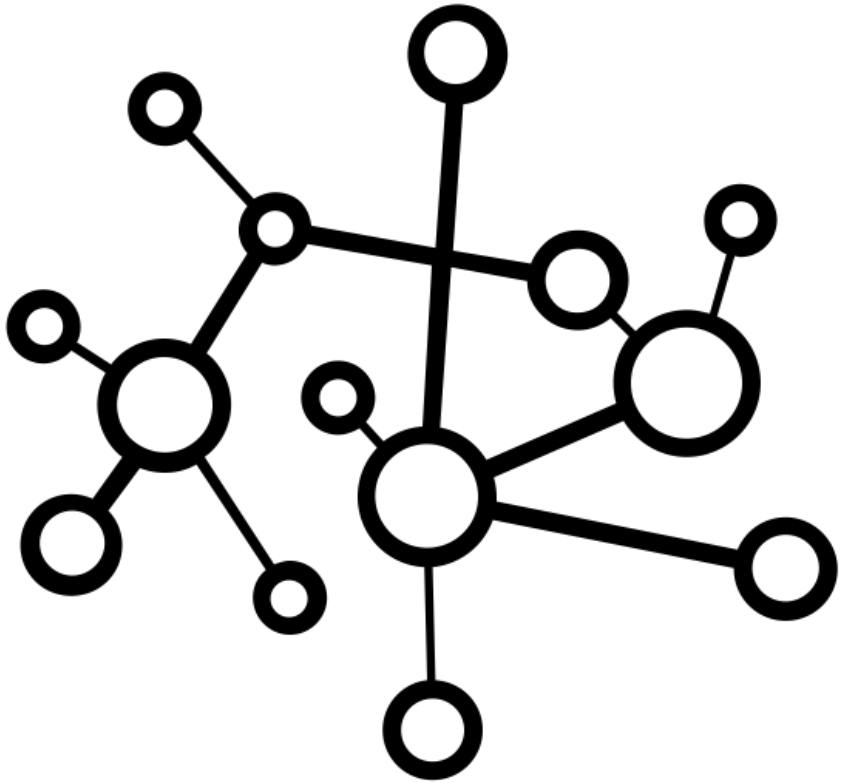
Чем ещё порадовать

Мегаграф мирового OpenSource

Как уязвимость повлияла на экосистему, как быстро она была исправлена в сообществе?

Ответы на эти вопросы будут в 24 году.

Где ждать? Канал CodeScoring



Open Source report

Большой отчет про Open Source, скоро.

Где ждать? Канал CodeScoring

Лицензии

Не будем забывать, что лицензии тоже часть композиционного анализа, но об этом в следующих сериях, пока можно посмотреть:

«Занимательные лицензии» @DUMP’23

Спасибо за внимание!

Алексей Смирнов,
основатель CodeScoring,
решения композиционного анализа (SCA)

alexey@codescoring.ru

@alsmirn — докладчик

@codescoring — новости продукта

О — Образование:

youtube.com/@codescoring