



# Создание безопасного программного обеспечения. Как начать?

Алексей Смирнов  
Основатель CodeScoring

# План прост

- «коробочка»
- качество => безопасность
- безопасная разработка и вот это вот всё
- инструментарий безопасной разработки
- полезные материалы
- как начать?



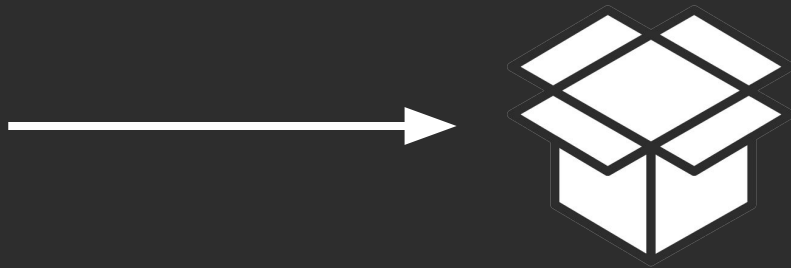
A decorative graphic consisting of numerous small, solid pink circles scattered across the left and center portions of a dark gray background. The dots are arranged in various patterns, including small clusters and single dots, creating a textured, abstract effect.

«Коробочка»

«Коробочка» = Продукт



# Наполнить продукт



# Поставить и сопровождать






Качество  $\Rightarrow$  Безопасность

# Качество ПО —

это степень, в которой ПО обладает  
требуемой комбинацией свойств

[1061-1998 IEEE Standard for Software Quality Metrics Methodology]





# Безопасная разработка приложений

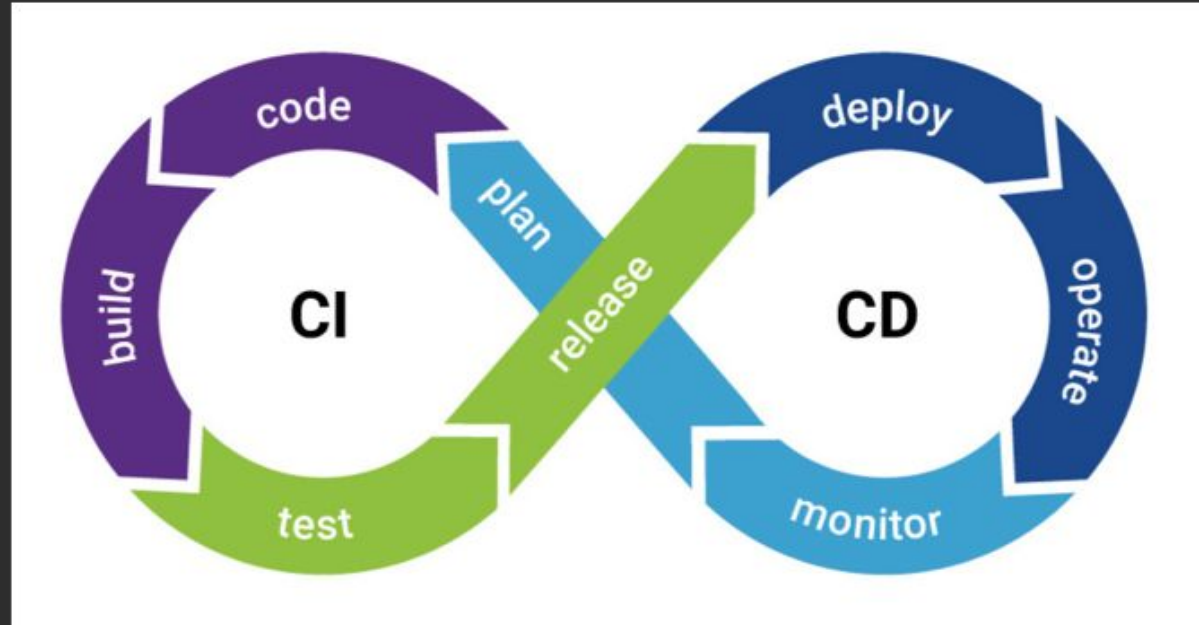
# DevSecOps —

процессы обеспечивающие  
безопасность на всех этапах создания  
и сопровождения ПО

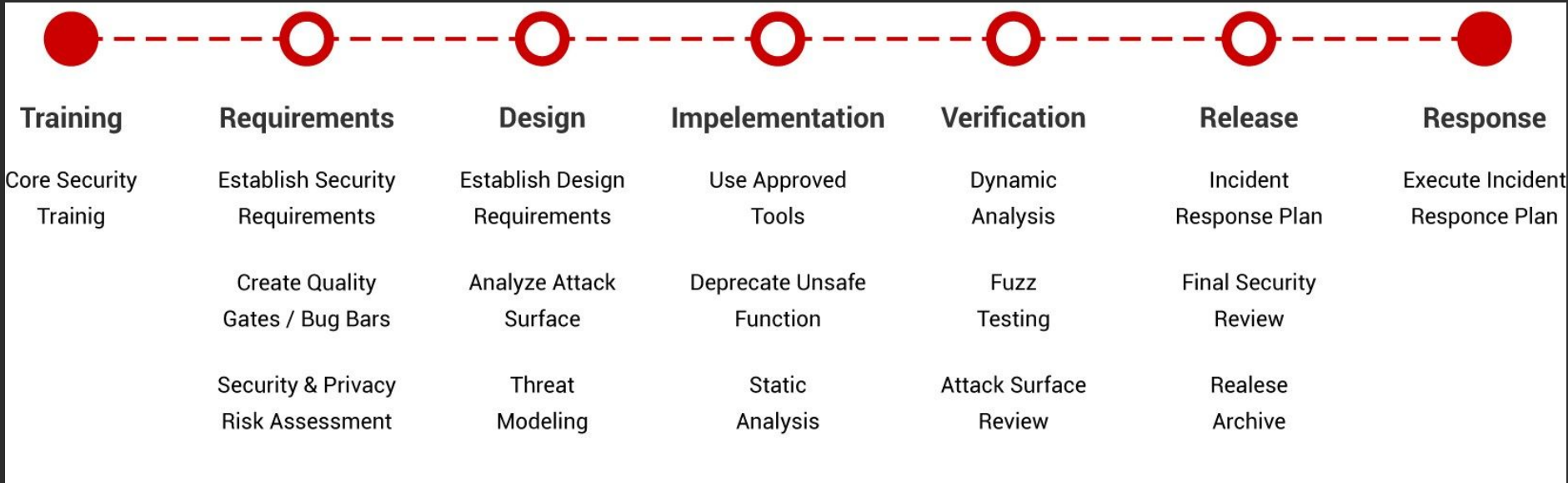
# DevOps

- Plan
- Code
- Build
- Test
- Release
- Deploy
- Operate
- Monitor

2008



# Microsoft Security Development Lifecycle



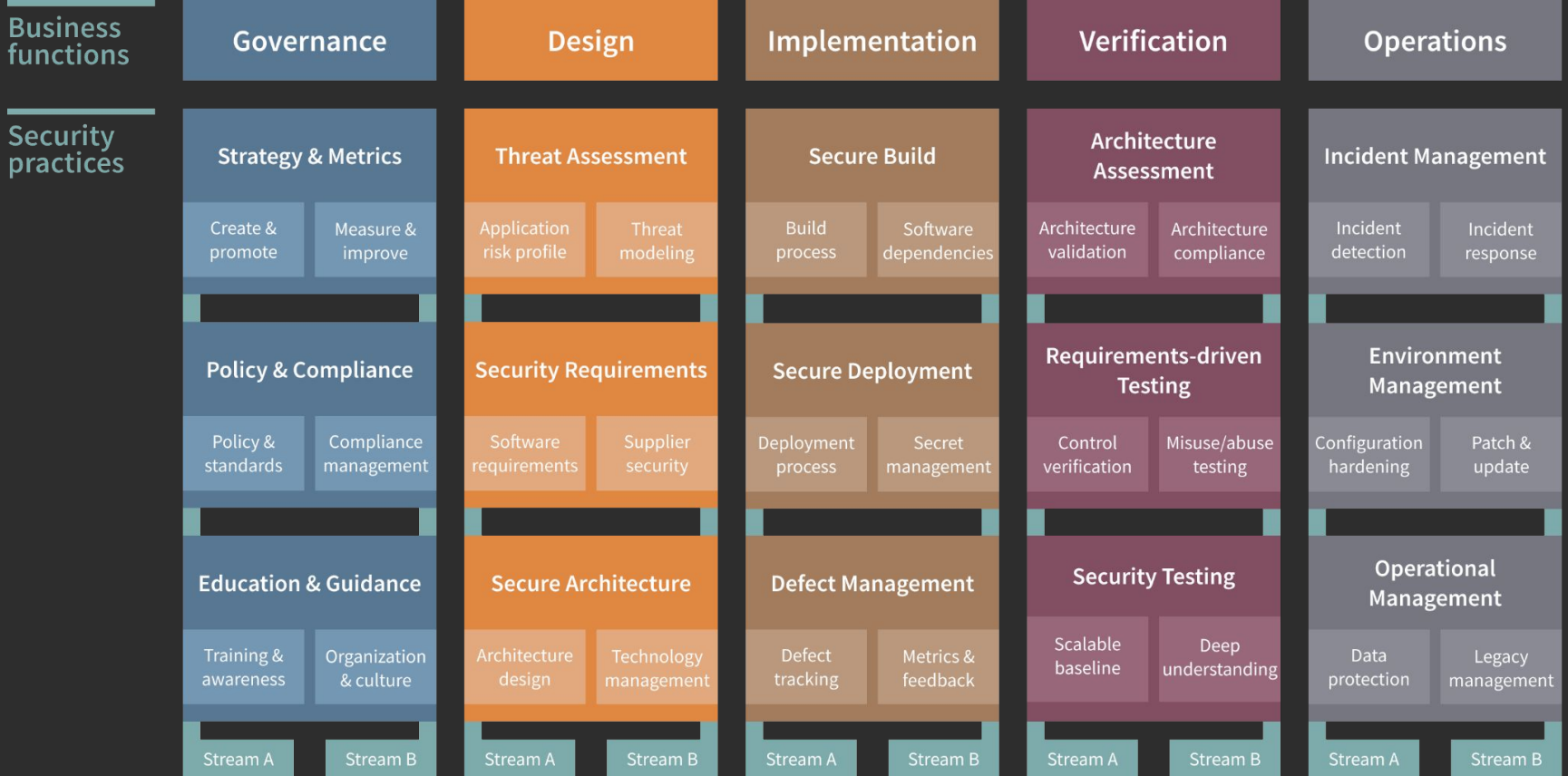
2008

# Сегодня много стандартов и фреймворков

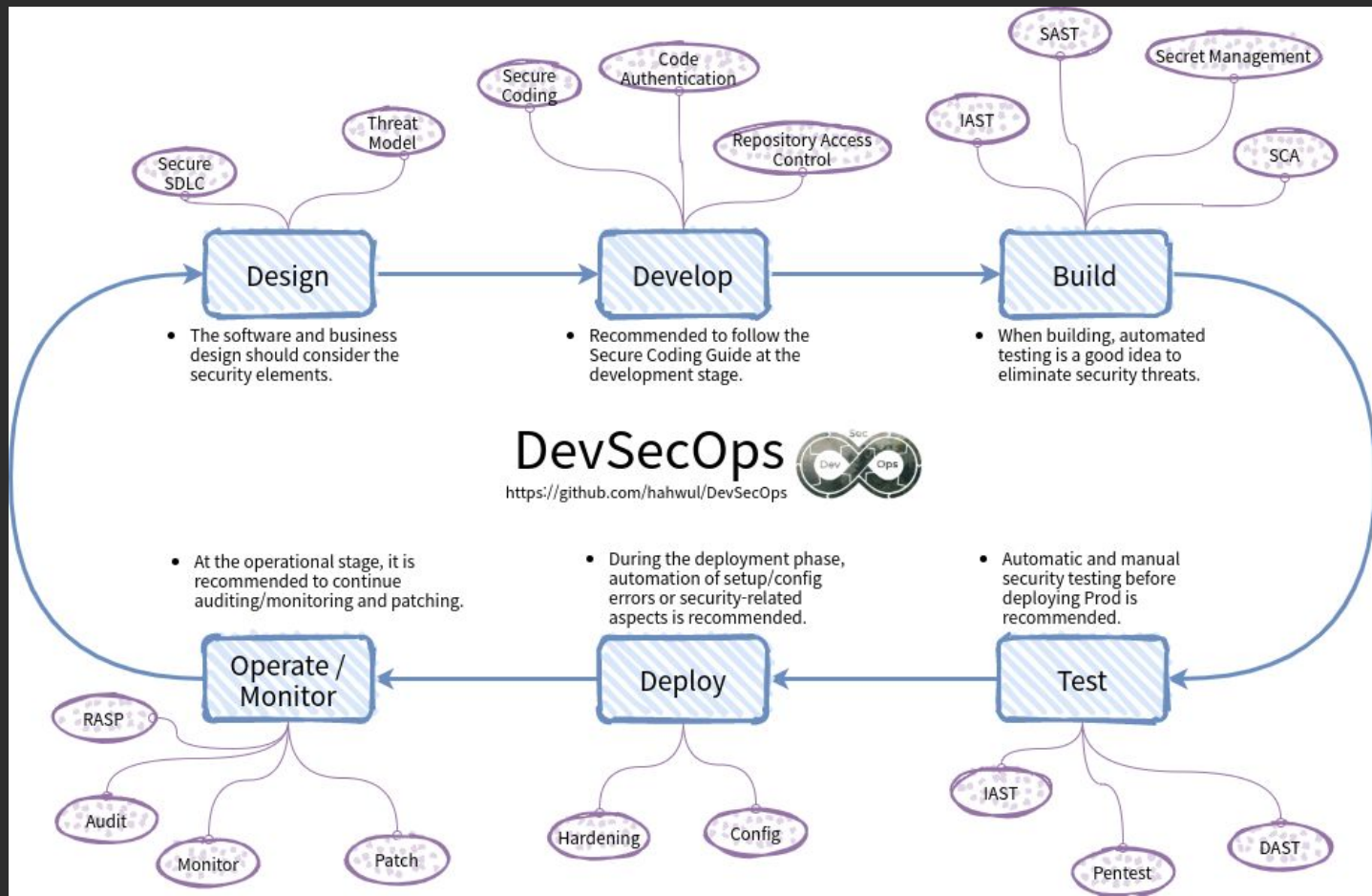
- OpenSAMM /opensamm.org
- OWASP SAMM /owasp.org/www-project-samm
- BSIMM /bsimm.com
- ГОСТ Р 56939-2016
- И т. д.




# OWASP SAMM v2 / Software Assurance Maturity Model



# Безопасный цикл разработки ПО





# Инструментарий создания безопасной «коробочки»

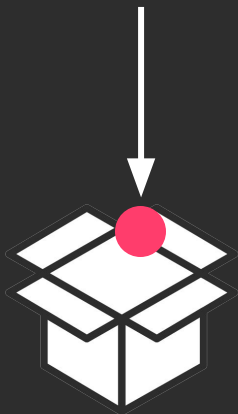


# Инструменты проверки

- Secrets — проверка секретов
- SAST — статическая проверка на безопасность
- OSA/SCA — проверка сторонних компонентов
- DAST — динамическая проверка на безопасность
- IAST — интерактивное тестирование безопасности



# Положить то, что можно положить



Порой, среди исходников можно найти:

- ключи
- токены
- логины-пароли
- адреса
- и т.п.

Важно использовать средства контроля, например **gitleaks** или аналоги.

А секреты хранить отдельно, например, в **HashiCorp Vault**.

# Компоновка коробочки

## OSA/SCA

~80% заимствованных компонентов из открытых источников

Известны

Идентифицируемы

Исследуемы

## SAST

~20% собственного кода

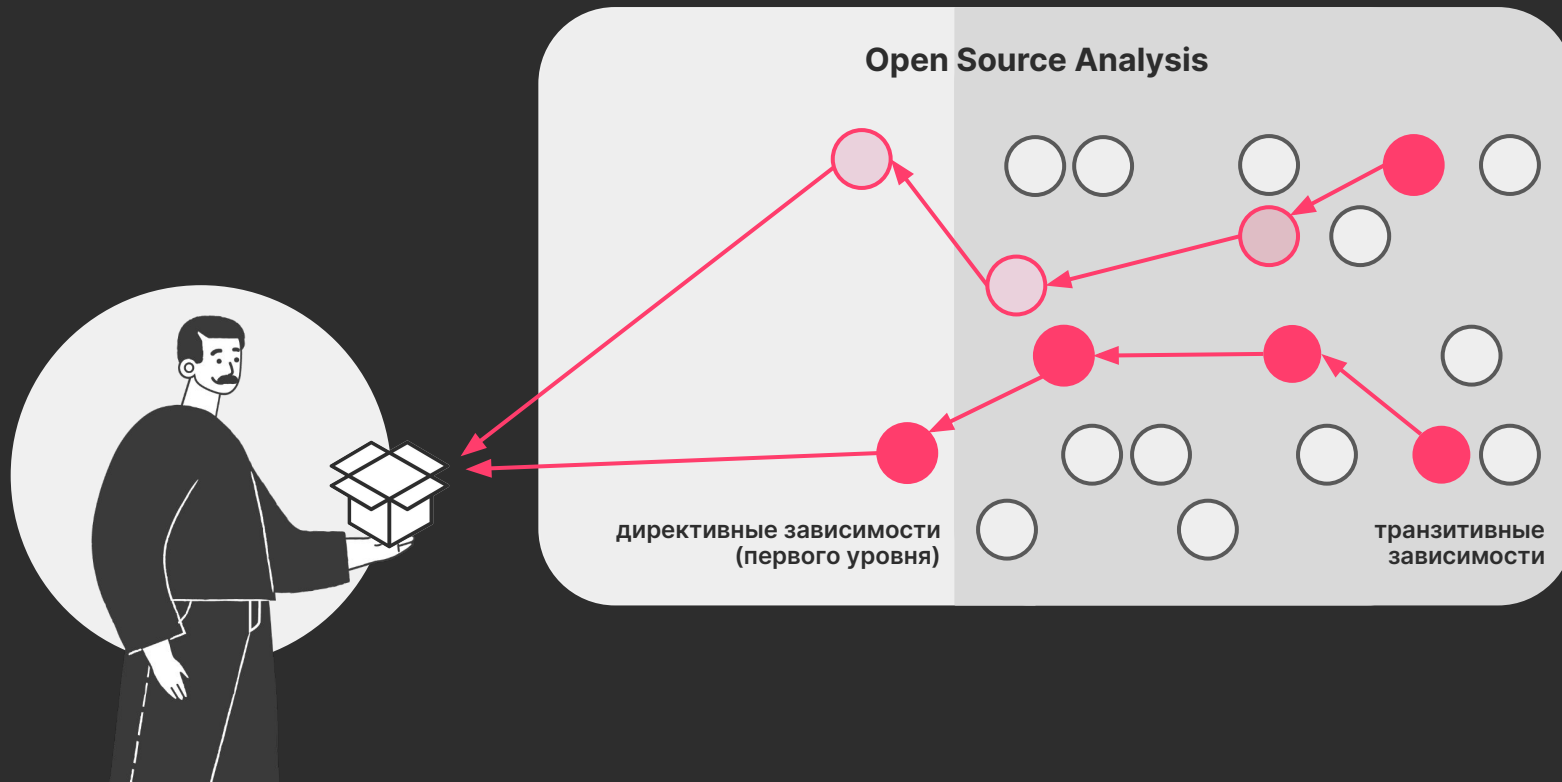
Неизвестен

Нужно сканировать

Сложно исследовать

# OSA+SCA / Open Source & Software Composition Analysis

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



# Немного статистики про Open Source

> 200 млн.

проектов с открытым исходным кодом, а также: 5 млн. готовых пакетов; 75 млн. их версий.

~70 млн.

разработчиков хотя бы раз поучаствовали в разработке, из них регулярно участвует ~5 млн.

x2

увеличилась скачиваемость открытых компонентов (с 20 на 21). В 22 году — более 4 трлн. скачиваний.

x13

Выросло количество известных атак на цепочки поставки: Dependency Confusion, Typosquatting, Namesquatting, Brandjacking, Malicious Code Injection и другие

> 1k

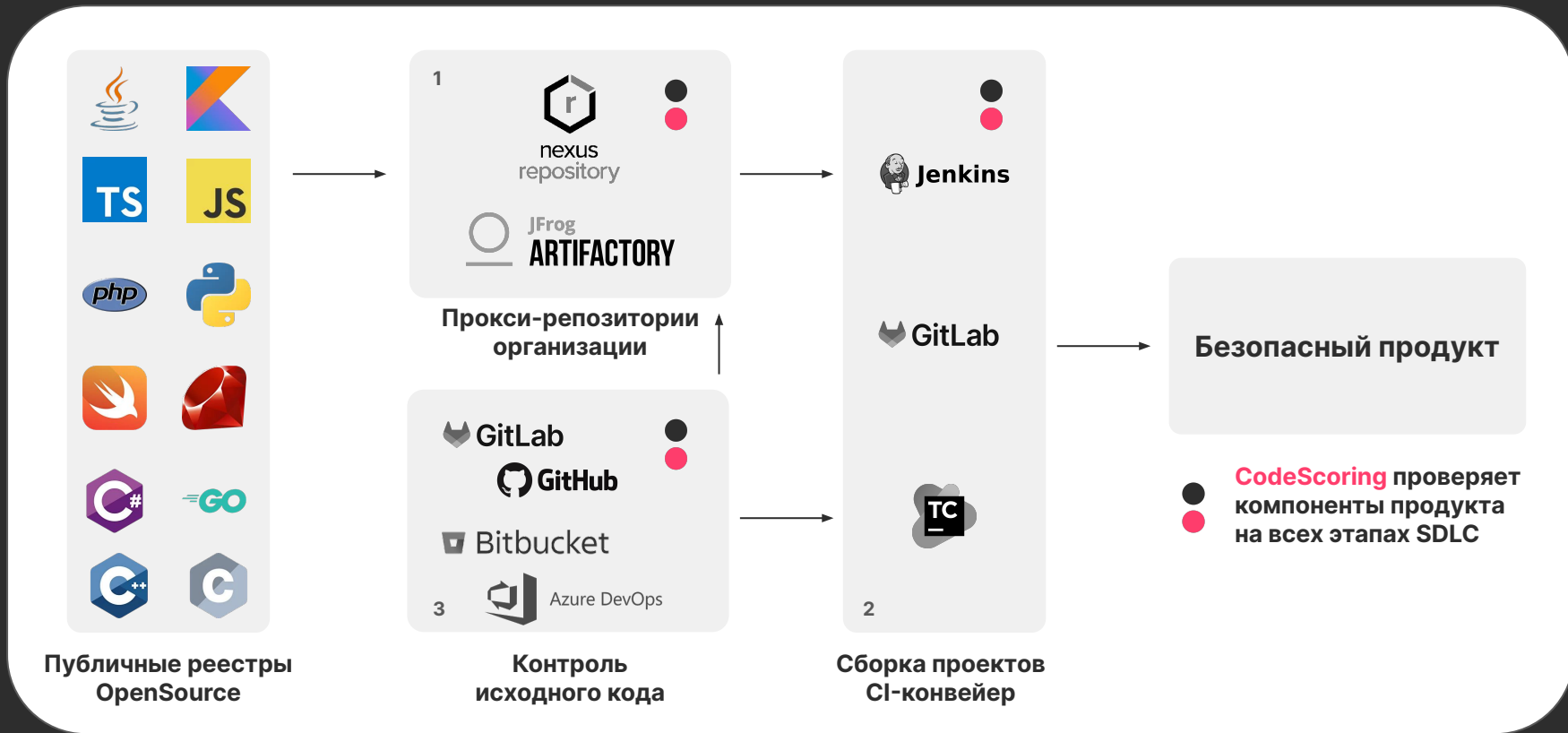
Вредоносных пакетов выявлено экспертами по безопасности за последний год: кража параметров окружения, backdoors, шифровальщики и т. п.

protestware<sup>new</sup>

Саботаж в OpenSource.

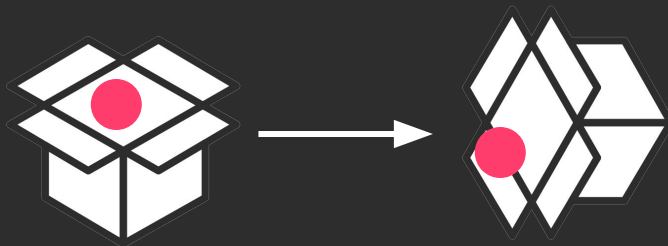
**Пакеты:** es5-ext, node-ipc, colors, faker, event-source-polyfill, styled-components и т. п.

# OSA/SCA — как это работает?



Среди OpenSource-решений: DependencyTrack, Trivy, etc.

# Заглянуть в коробочку: SAST



Тестирование продукта на наличие ошибок и уязвимостей в исходном коде с применением статического анализа.

Подходы бывают самые разные:

- поиск шаблонов (+ML)
- анализ абстрактных синтаксических деревьев (AST)
- межпроцедурный анализ (контексточувствительный и чувствительный к путям на основе символьного выполнения)
- анализ помеченных данных (пользователь может задать источники и приёмники помеченных данных, в том числе помеченные аргументы функций и поля структур)



# Статика (SAST) / Static Application Security Testing

Подходы:

- сканирование сниппетов (хорошо для IDE и начального уровня), как правило, это OpenSource-решения.
- сканирование приложения целиком (CI), как правило, это коммерческие решения.

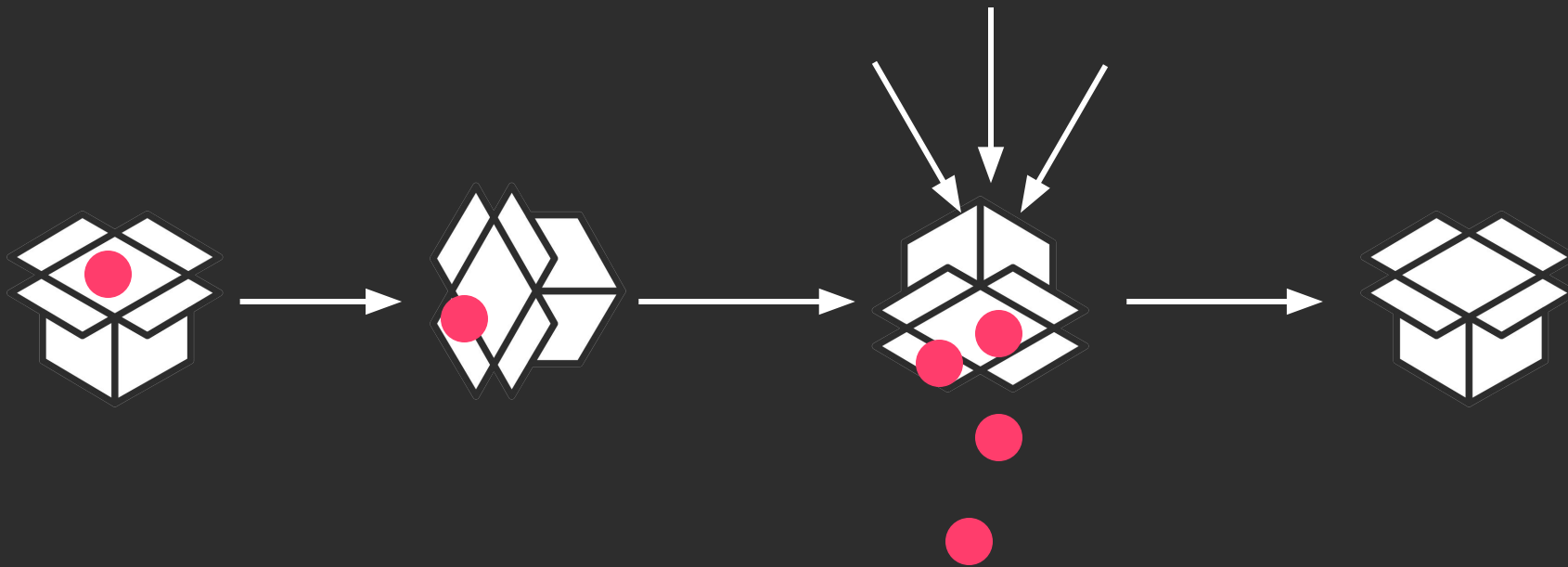
Большой перечень можно найти здесь



В России решения представляют:



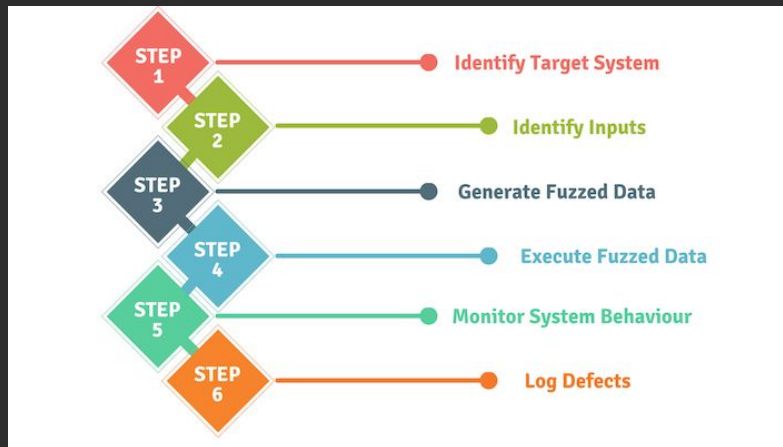
# Встряхнуть коробочку: DAST + IAST



# Динамика (DAST) / Dynamic Application Security Testing

Тестирование продукта на наличие ошибок и уязвимостей в исходном коде с применением динамического анализа.

Главная суть в передаче приложению на вход неправильных, неожиданных или случайных данных.



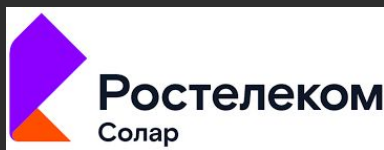
**Главный подход: сканирование черного ящика.**

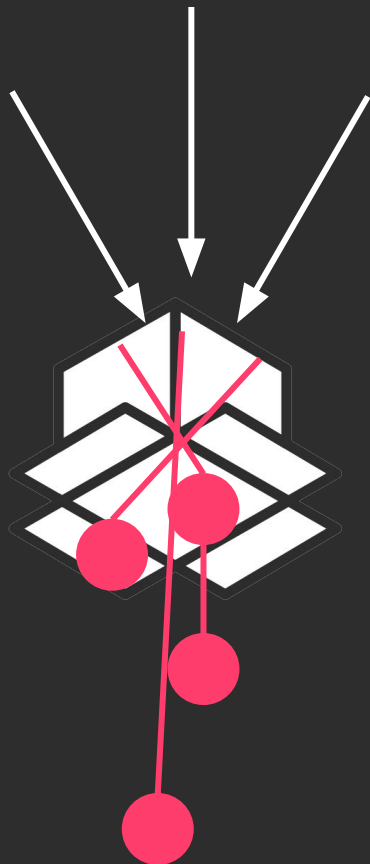
Важнейшим аспектом применения динамического анализа, является **определение поверхности атаки.**

Большой перечень можно найти здесь



В России решения представляют:





Объединение подходов статики и динамики.

Определение поверхности атаки, отслеживание помеченных данных, глубокое понимание: что, где и как сработало.

Продуктов немного.

Основная экспертиза  
здесь →



ИСП РАН

# Полезные сообщества

- обмен опытом и лучшими практиками между энтузиастами одного дела



**Статика**  
[t.me/sdl\\_static](https://t.me/sdl_static)

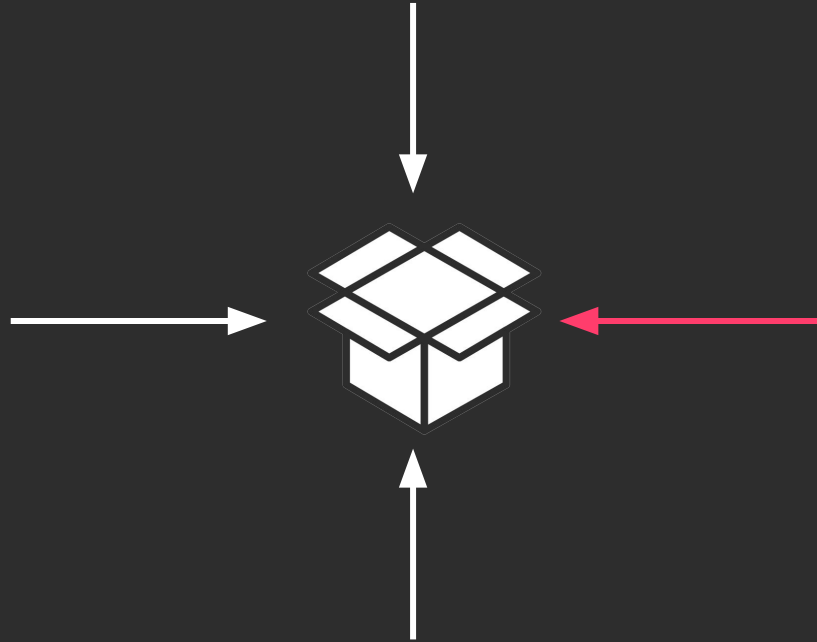


**Динамика**  
[t.me/sdl\\_dynamic](https://t.me/sdl_dynamic)



**CodeMining**  
[t.me/codemining](https://t.me/codemining)

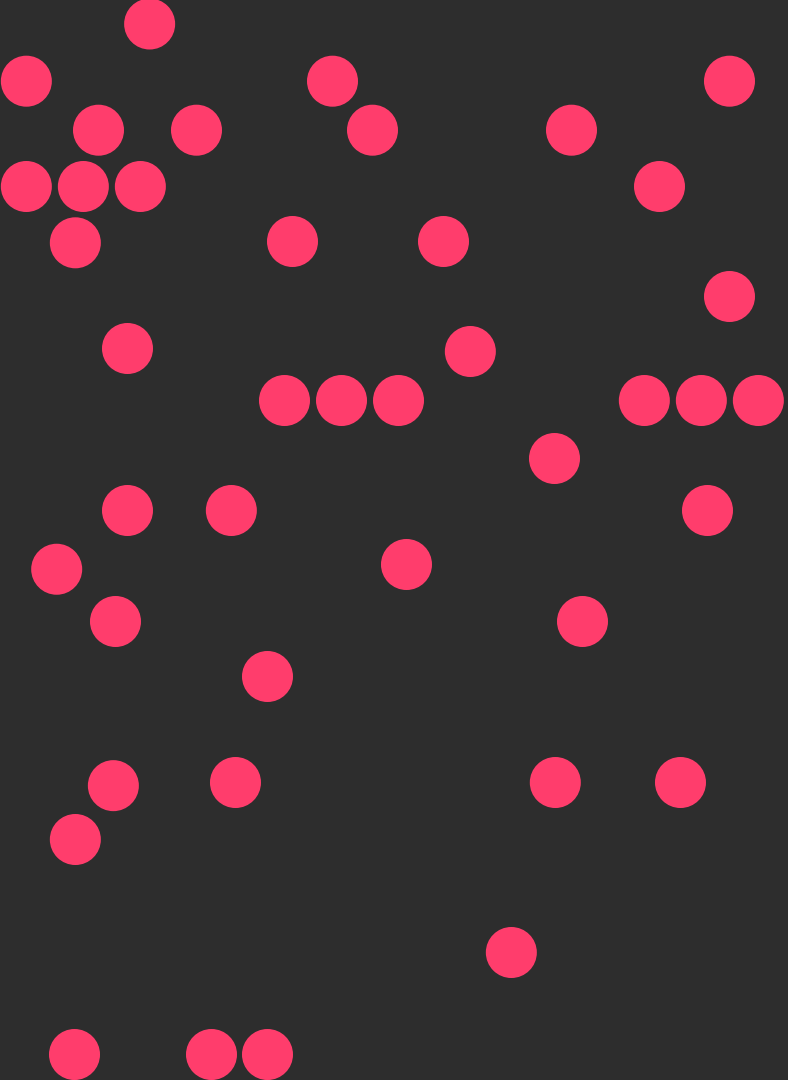
# Жизнь коробочки и её окружения



Немного другая история, но тоже очень важная.

При обеспечении безопасной  
разработки важны  
люди, технологии и процессы

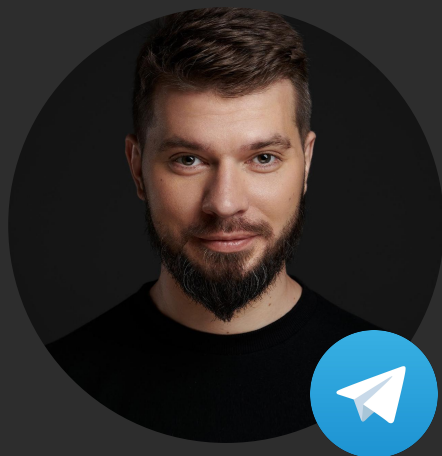




# ВЫВОД

Начинание —  
то, что было начато

# Спасибо за внимание!



Алексей Смирнов,  
основатель [CodeScoring](#),  
решения композиционного  
анализа (SCA)

[alexey@codescoring.ru](mailto:alexey@codescoring.ru)

[@alsmirn](#) — докладчик

[@codescoring](#) — новости продукта

[@codemining](#) — анализ кода

**O — Образование:**

[youtube.com/@codescoring](https://youtube.com/@codescoring)