
Яндекс

От атаки к уязвимости через большие данные

Эльдар Заитов

whoami

- Яндекс
- “Smoked Chicken” / MSLC CTF team
- CTFtime.org

Яндекс

- Десятки сервисов и продуктов
- 60+ миллионов уникальных посетителей в месяц
- ~20 миллиардов запросов в сутки

«Охота за ошибками»

- Найди уязвимость и получи награду
- Свыше 1000 писем
- Выплачено больше 4 млн. рублей
- Участники – весь земной шар

https://company.yandex.ru/security/

Вопросы

- Известно ли нам обо всех уязвимостях, о которых известно внешним исследователям?
- Существуют ли внешние злоумышленники, эксплуатирующие уязвимости в сервисах Яндекса так, что мы об этом не знаем?
- Что еще мы можем сделать, чтобы улучшить безопасность наших сервисов?

Логи

- Все сервисы пишут логи
- Доступны в MapReduce / YT

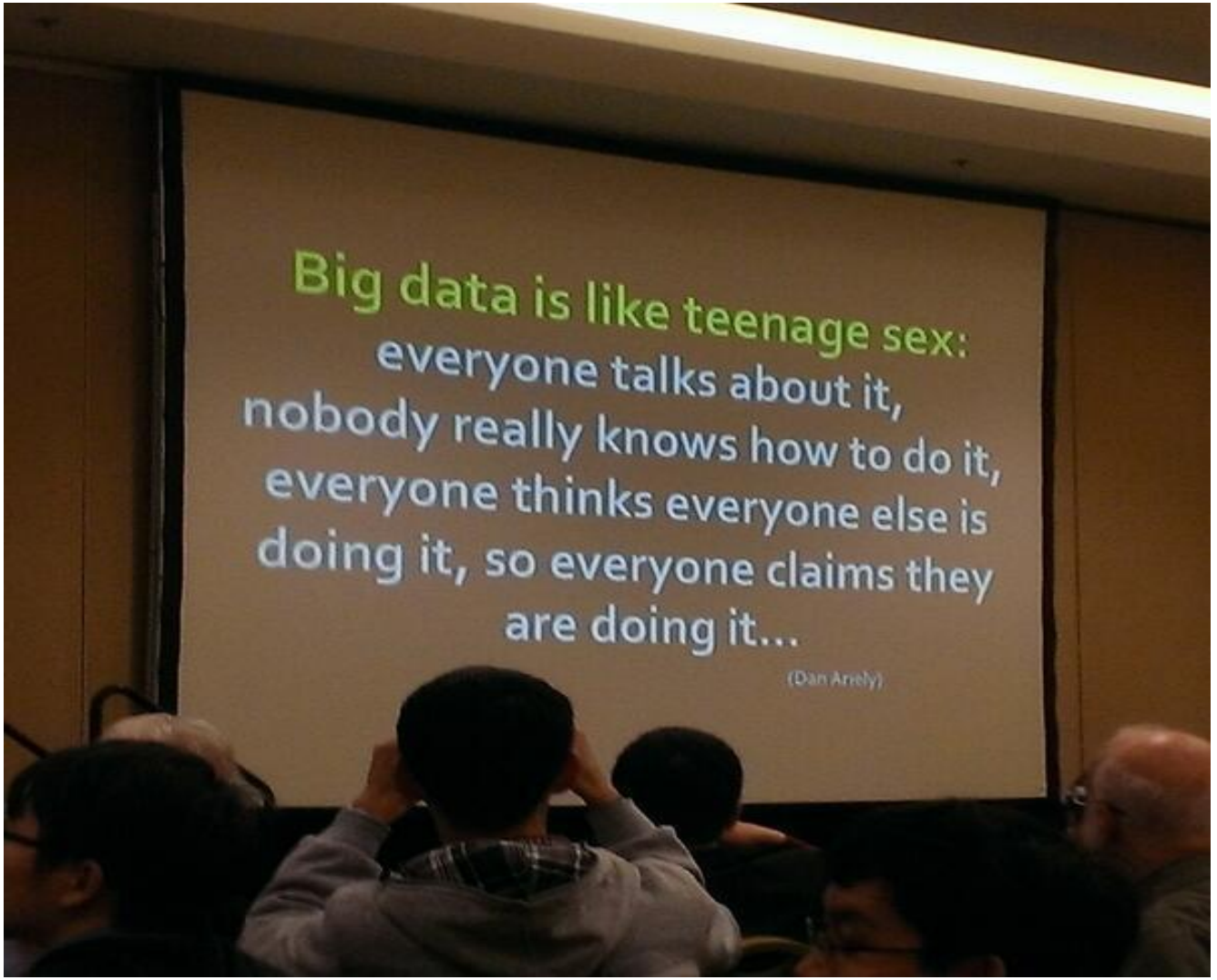
Что есть в логах

- Метод запроса
- Хост
- Timestamp
- URI и параметры для GET
- Cookies
- Источник (IP)
- Реферер

Что еще есть в логах

- Статус ответа
- Размер тела ответа
- Время ответа бэкэнда
- Всё что пожелает добавить администратор

Big data


Big data is like teenage sex:
everyone talks about it,
nobody really knows how to do it,
everyone thinks everyone else is
doing it, so everyone claims they
are doing it...
(Dan Ariely)

MapReduce

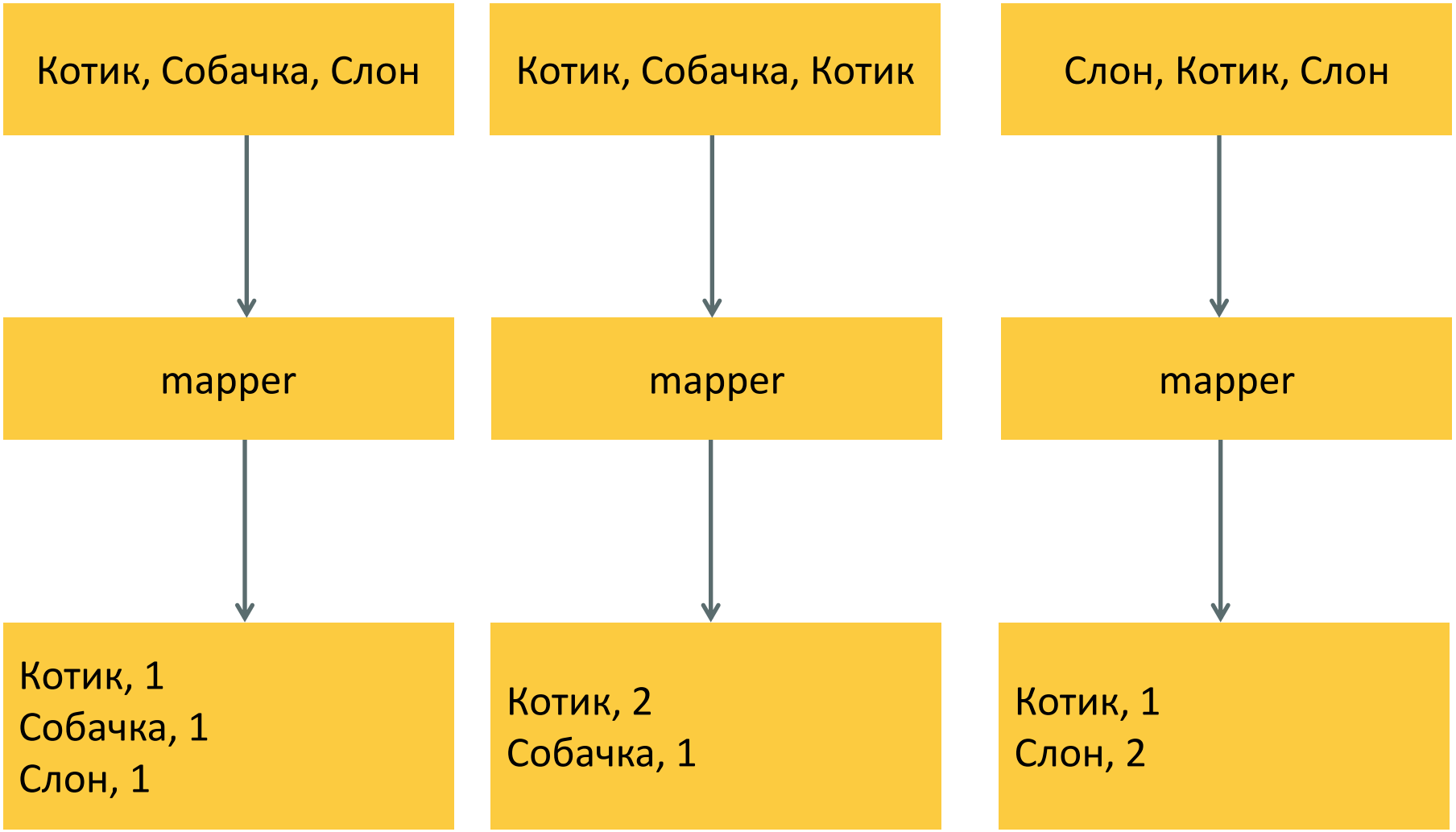
- Map

$$(k_1, v_1) \rightarrow [(k_2, v_2)]$$

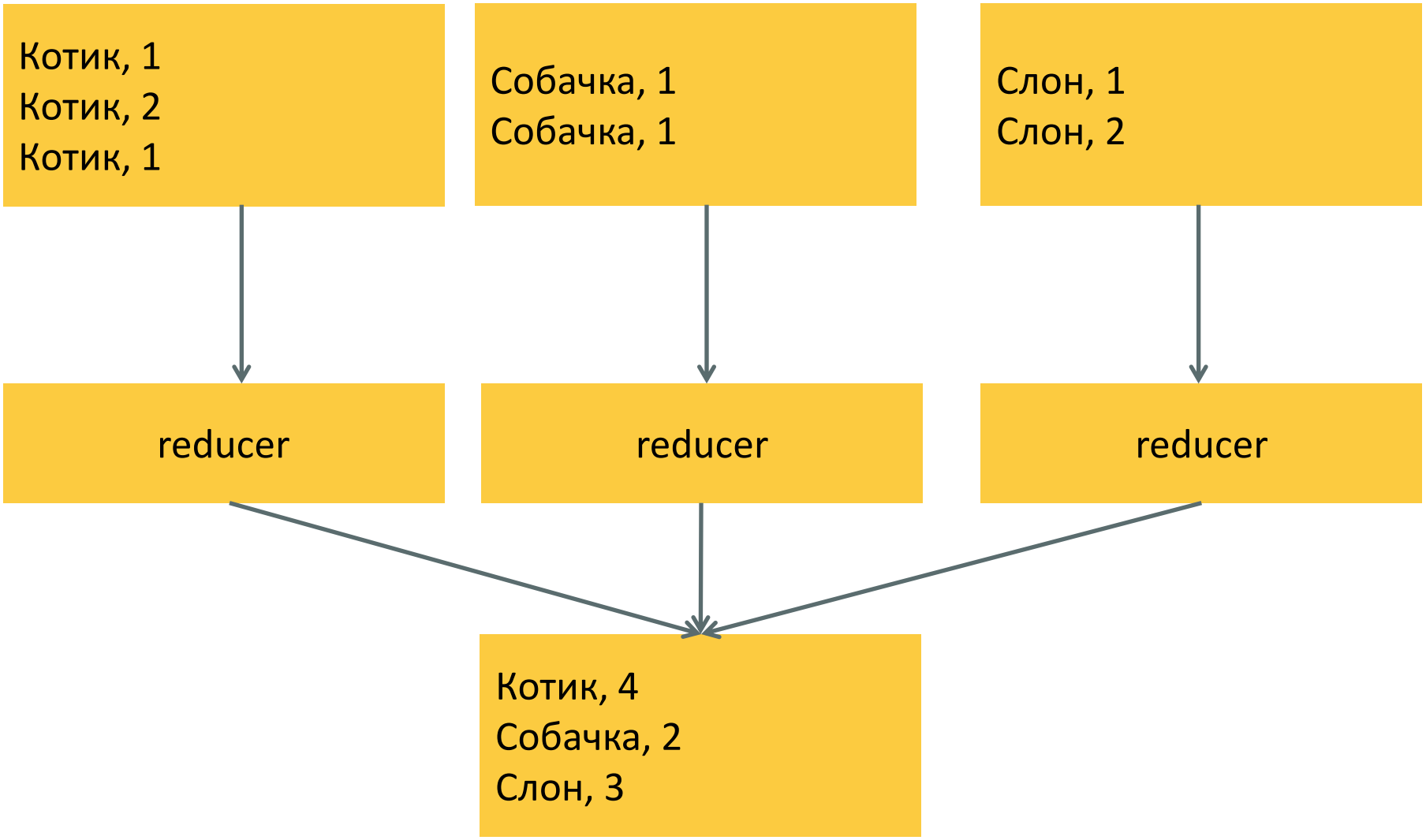
- Reduce

$$(k_2, [v_2]) \rightarrow [(k_3, v_3)]$$

Map


Котик, Собачка, Слон
Котик, Собачка, Котик
Слон, Котик, Слон
mapper
mapper
mapper
Котик, 1
Собачка, 1
Слон, 1
Котик, 2
Собачка, 1
Котик, 1
Слон, 2

Reduce


Котик, 1
Котик, 2
Котик, 1
Собачка, 1
Собачка, 1
Слон, 1
Слон, 2
reducer
reducer
reducer
Котик, 4
Собачка, 2
Слон, 3

Анализируем логи

- Находим небезопасные конструкции и приводим в порядок сервисы
- Ищем в логах успешные атаки

Небезопасные конструкции: JSONP

- В операции Map ищем все запросы с именами параметров “cb” и “callback”
- Кластеризуем по URI

Результат:

- ~500M JSONP вызовов в сутки
- ~200 JSONP ресурсов

Типы атак

- Injections
- XSS
- Любые, где Payload в Пути, Параметрах запроса или Cookies

SQL Injection

- Error-based или другие с прямой выдачей
- Blind boolean-based
- Blind timing-based
- Out-of-band

Методы детектирования

- Сигнатуры
- Интерпретация и анализ payload
- Машинное обучение

Опасные строки

- False positives
- Легко обходится
- Можно использовать как грубый фильтр для более «тяжелых» проверок

Regular expressions

- False positives
- Очень сложные «тяжелые» конструкции
- Всё еще можно обойти
- Жизнеспособно для некоторых типов атак

Kant ▾ System Dashboard Operations Navigation More ▾

GET	371	HTTP/1.1	-	/_c/sGFxka9TLQOk8ocvih6OumGB4nE.gif	200
GET	205073	HTTP/1.1	http://market.yandex.ru /guru.xml?CMD=-RR=0,0,0,0-PF=2140131887~LT~sel~289.917633-PF=2142357018~EQ~sel~x1897526220-PF=2142357017~TR~des~exclude-PF=2142357015~EQ~sel~x477105268-PF=2142357005~EQ~sel~x1729117497-VIS=78-CAT_ID=1004703-EXC=1-PG=10&hid=91529	/model.xml?modelid=2190832& hid=91529	200
GET	355	HTTP/1.1	-	/_c/moIgee5owP8JFaPDvP57rjYE_UY.png	200
GET	264	HTTP/1.1	-	/_c/Z3jABDSRezQ_Z675TetexLKEBD4.png	200
GET	259	HTTP/1.1	-	/_c/Z3jABDSRezQ_Z675TetexLKEBD4.png	200
GET	258	HTTP/1.1	-	/_c/moIgee5owP8JFaPDvP57rjYE_UY.png	200
GET	373	HTTP/1.1	-	/_c/sGFxka9TLQOk8ocvih6OumGB4nE.gif	200

Интерпретация payload

- Малое количество false positives
- Всё еще можно обойти
- Требует понимания payload
- «Дорого»

Libinjection

- Библиотека на C с биндингами под основные скриптовые языки. Представлена Nick Galbreath (Etsy) на Blackhat 2012
- Интерпретирует payload как SQL конструкцию
- Быстрая (~100k sps)
- Позволяет получить т.н. “fingerprint” атаки длиной 5 байт

Libinjection

```
SELECT 1 /*!00000AND 2>1*/
```

```
[('k', 'SELECT'),       // keyword
 ('1', '1'),            // number
 ('o', 'AND'),          // operator
 ('1', '2'),            // number
 ('o', '>'),            // operator
 ('1', '1')]            // number
```

Подводные камни

- False positives

```
/startup?&deviceid=7ddc91a7934edf7a1fbc4d3491c497ae&device_type=phone&protocol_version=[redacted]&locale=en-(null)&
rsion=5.1&screen_width=320&app_platform=iphone&screen_height=480&screen_dpi=326&query_h[redacted]%2C1&
acturer=Apple&scalefactor=2.00&app_version=526&app_version_name=5.26&uuid=0935a0917a112f1ae9c45cd23eecb465&
h_promotion_timestamp=0
```

```
- startup.mobile.yandex.net: [
    - {
          example: "",
        - params: [
              "locale"
          ],
          type: "rx",
          uri: "/(analytics|maps|browser|metro)/startup"
      }
  ]
```

- База известных FP ~200 записей

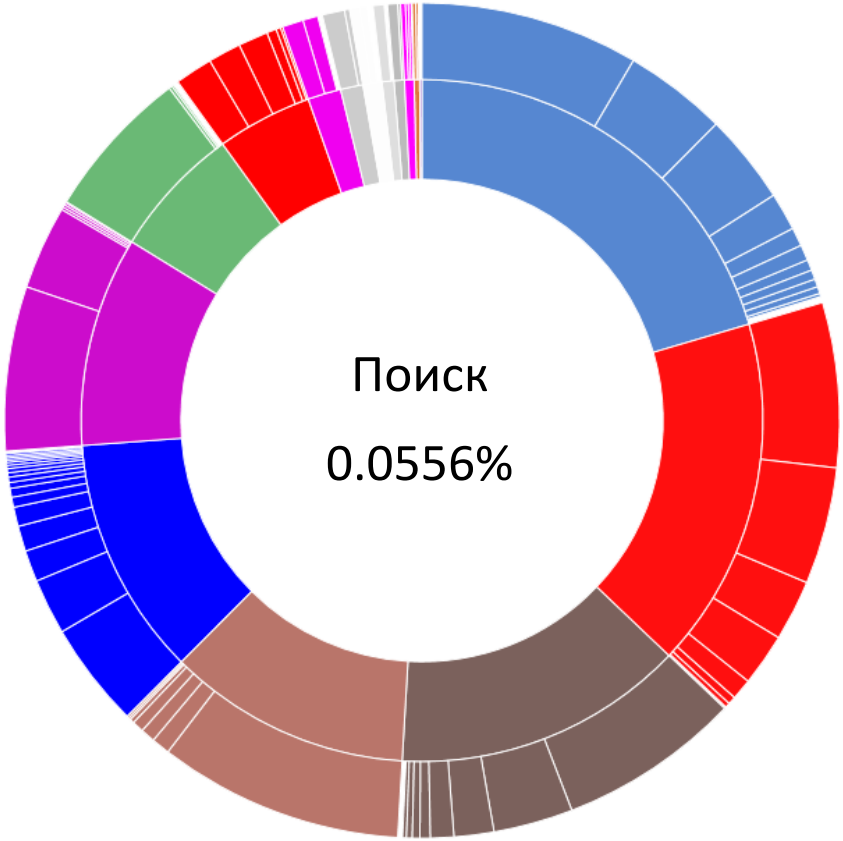
Сканирование != уязвимость

- Как отличить успешную атаку от неуспешной?
- Каковы критерии успешности атаки?

Гипотеза

- Сканирование с результатом, приводящим к ошибке 500 на стороне сервера может быть одним из возможных критериев успешной атаки

Проверка



- ~5 миллионов ответов с кодом 500 в сутки
- ~100 ресурсов

Time-based blind

- Критерий «успешности» атаки – возможность влиять на время ответа сервера в зависимости от результатов переданного запроса

- При аномально большом размахе вариации времен ответов сервера считаем атаку успешной

Подводные камни

- «Кластеризация» ресурсов

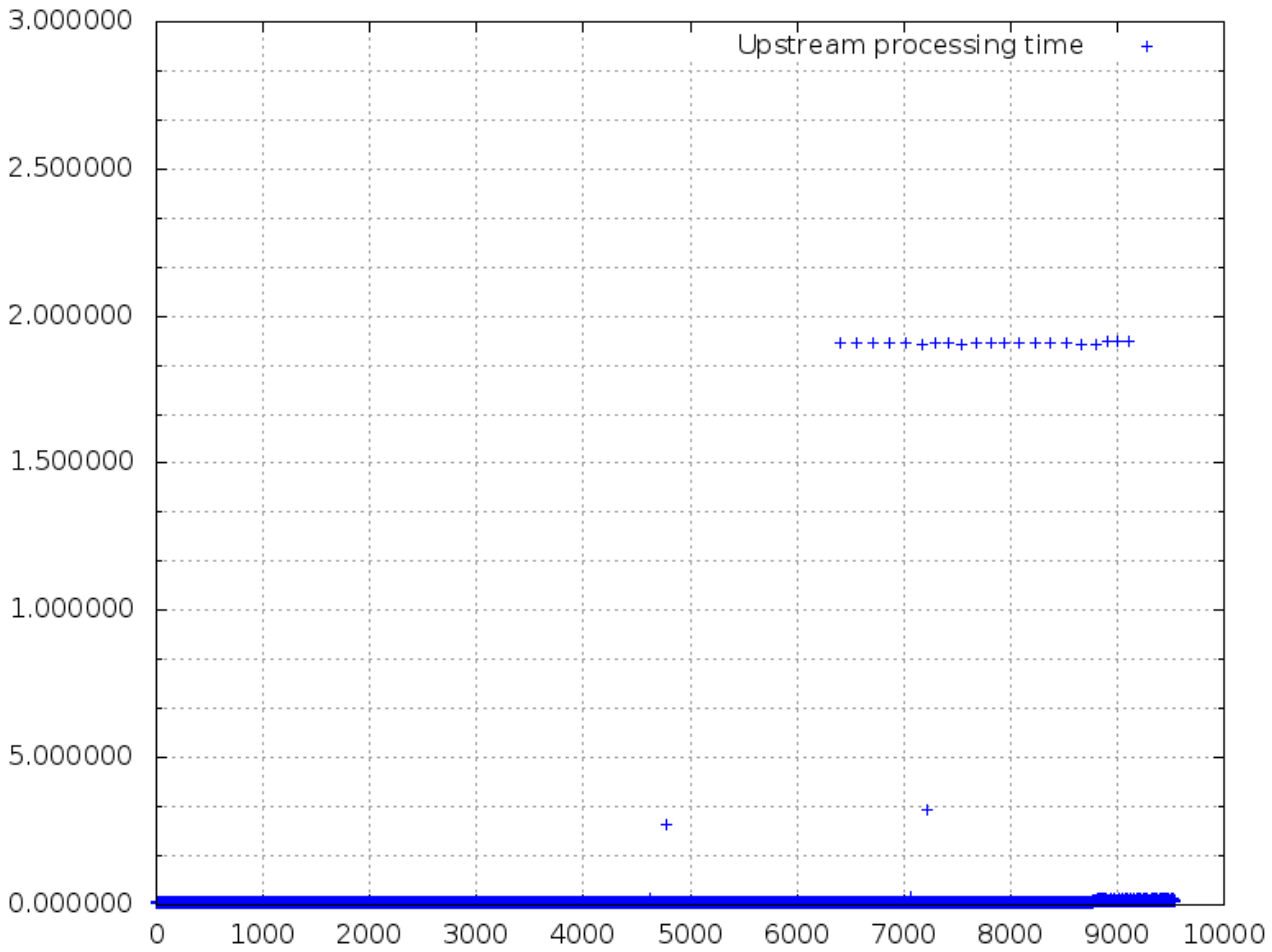
/user/alice/product/1234/

/user/bob/product/1235/

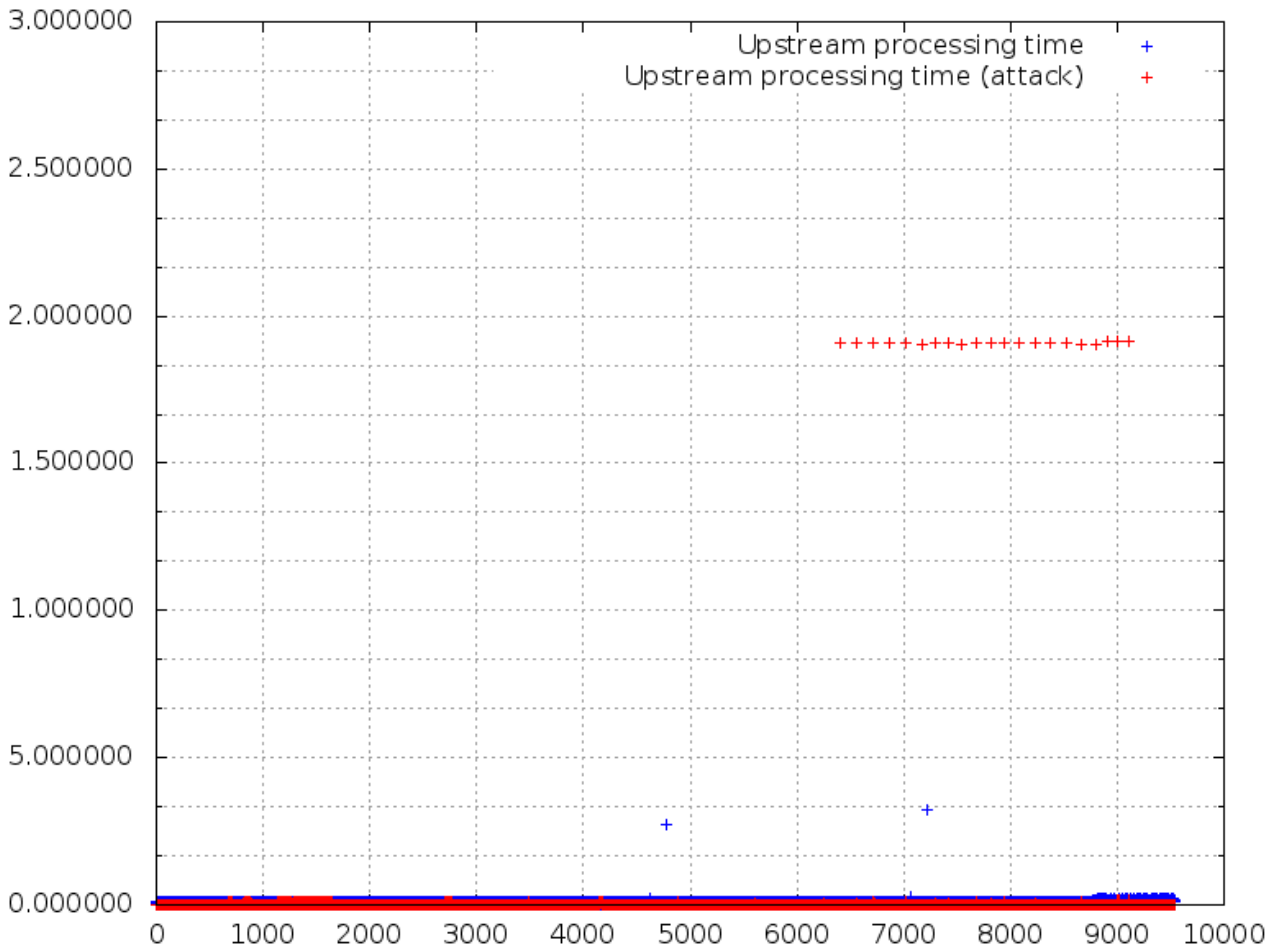
/user/*/product/*/

- Пример: Яндекс.Маркет кластеризуется в ~300 ресурсов

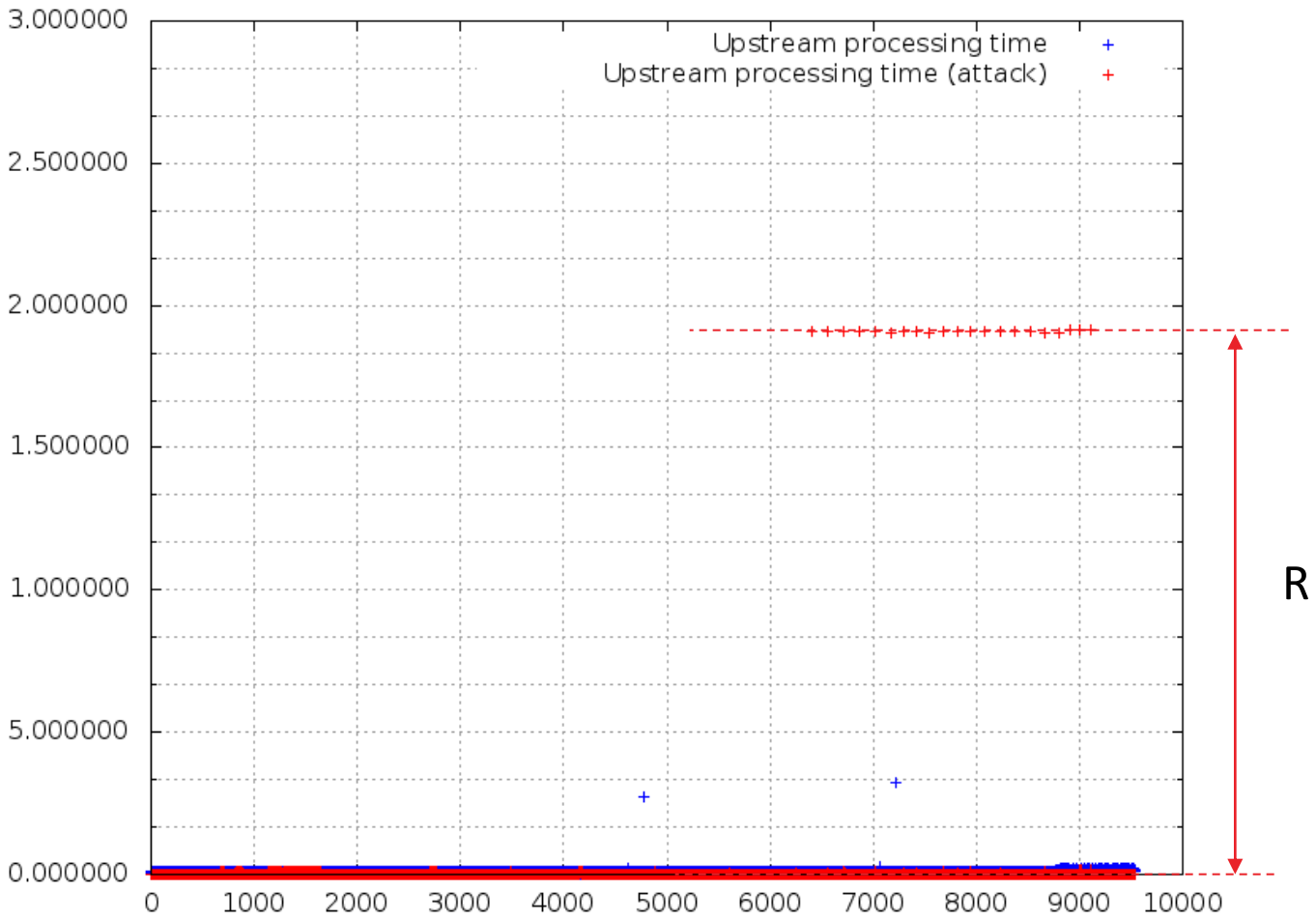
Time-based blind



Time-based blind



Time-based blind



Подводные камни

- Аномальные значения времен ответов сервера

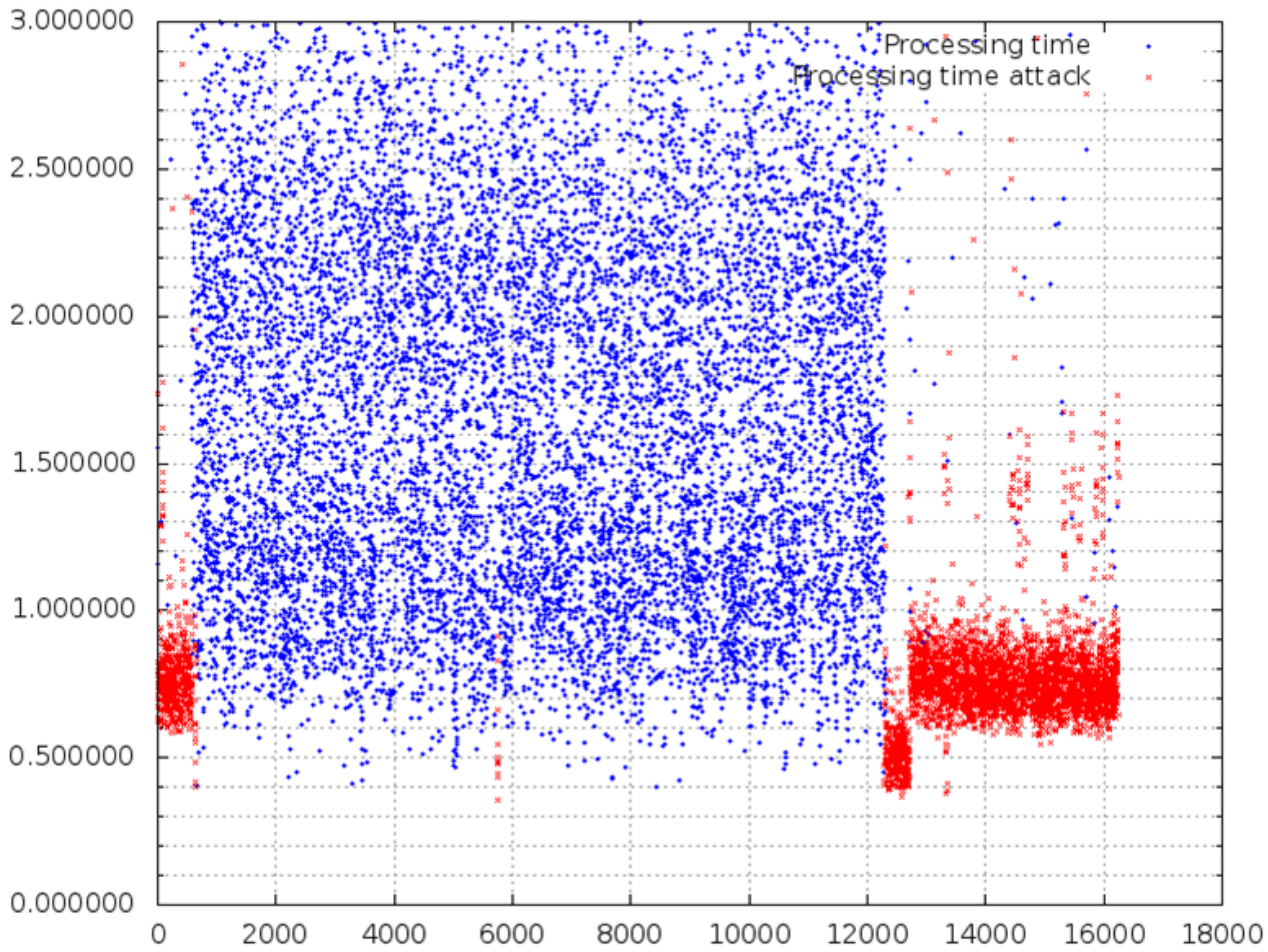
Значение не может превышать стандартное отклонение времени ответа больше чем в 3 раза для легитимных запросов

- Качество логов

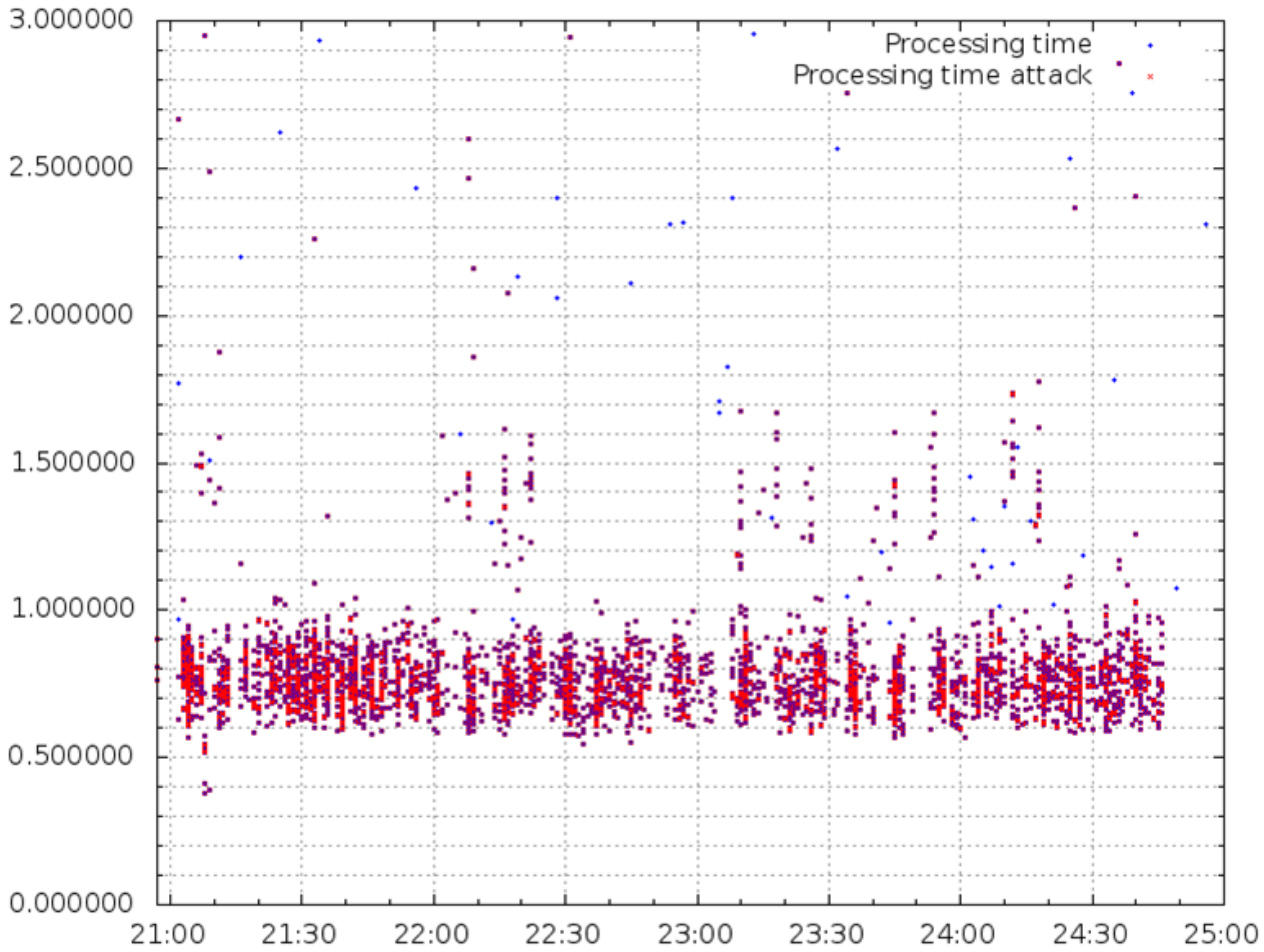
Time-based blind

- Размечаем запросы
- Кластеризуем ресурсы атакуемого проекта
- Считаем среднее время ответа бэкэнда в заданный день и час для атакуемых ресурсов
- Сравниваем среднее время ответа бэкэнда для атакующих и нормальных запросов

Condition-based blind



Condition-based blind



Выводы

- Простые статистические методы ограниченно работают на наших наборах данных
- Возможно, правильнее использовать методики активной проверки наличия уязвимостей

Спасибо за внимание!

Вопросы?

ezaitov@yandex-team.ru

@kyprizel