Статическое тестирование безопасности инструментами из open-source

Александра Сватикова

Эксперт по информационной безопасности, Одноклассники

alexandra.svatikova@corp.mail.ru

https://ok.ru/alexandra.svatikova

План

Часть 0. Введение

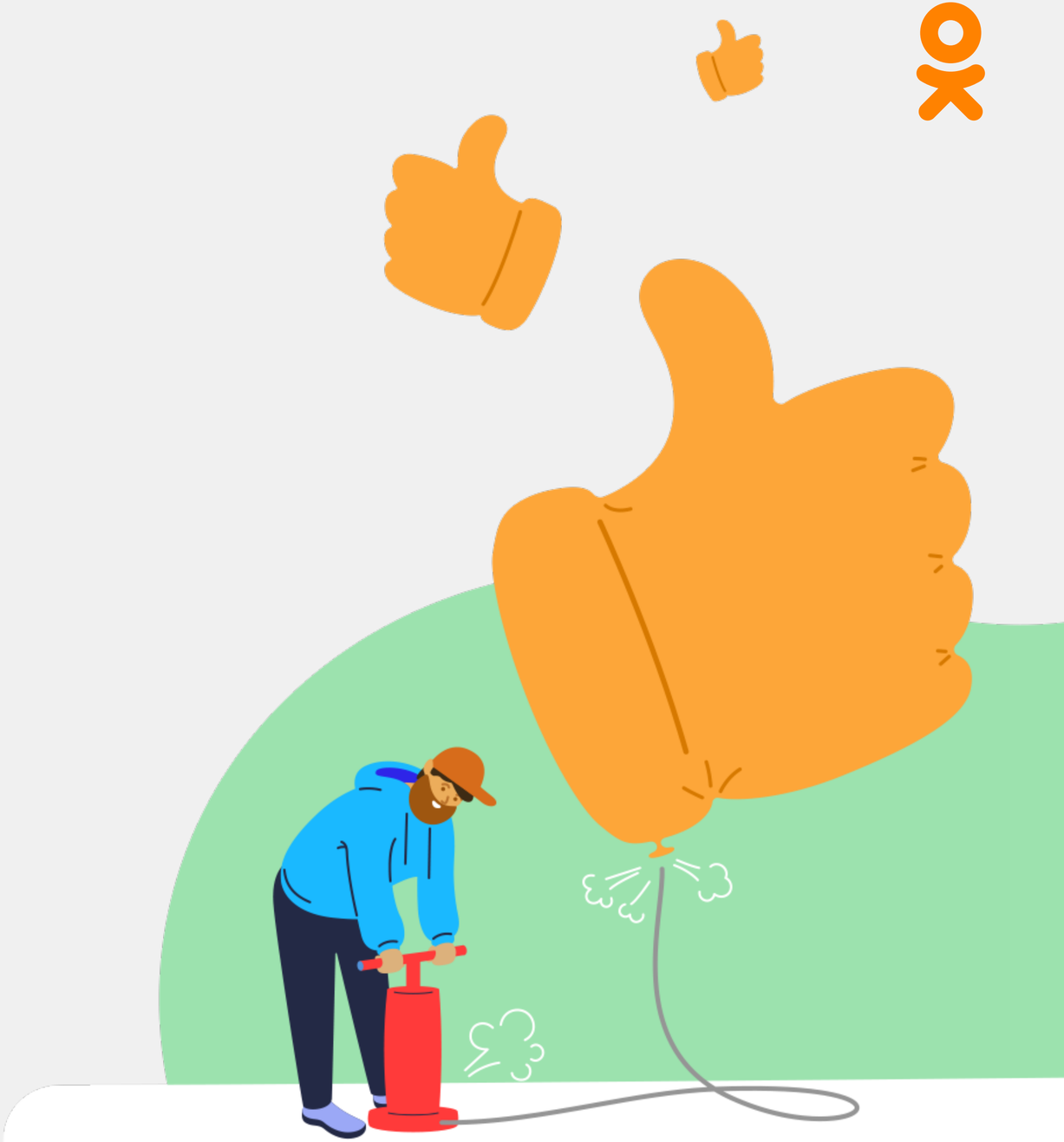
Часть 1. Мотивация

Часть 2. Теория

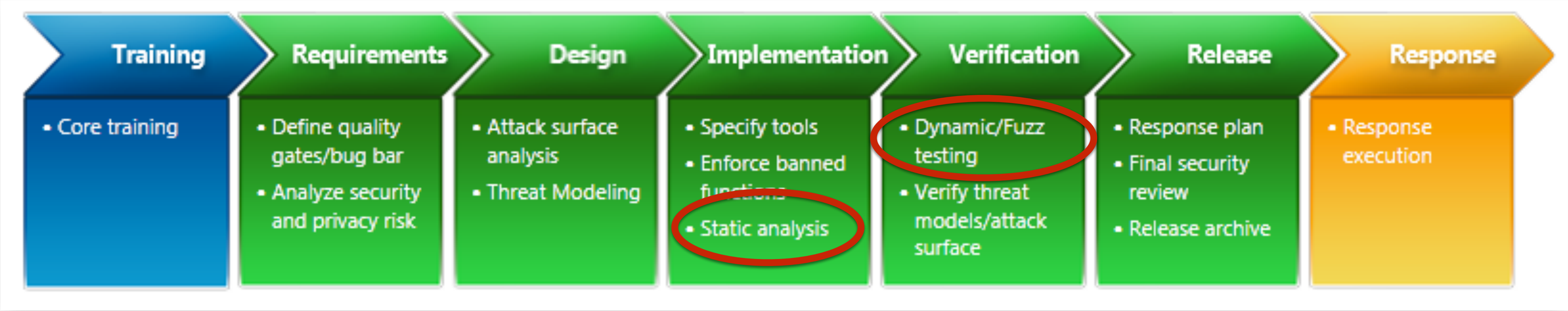
Часть 3. Практика

https://github.com/alexandra-s/heisenbug-demo

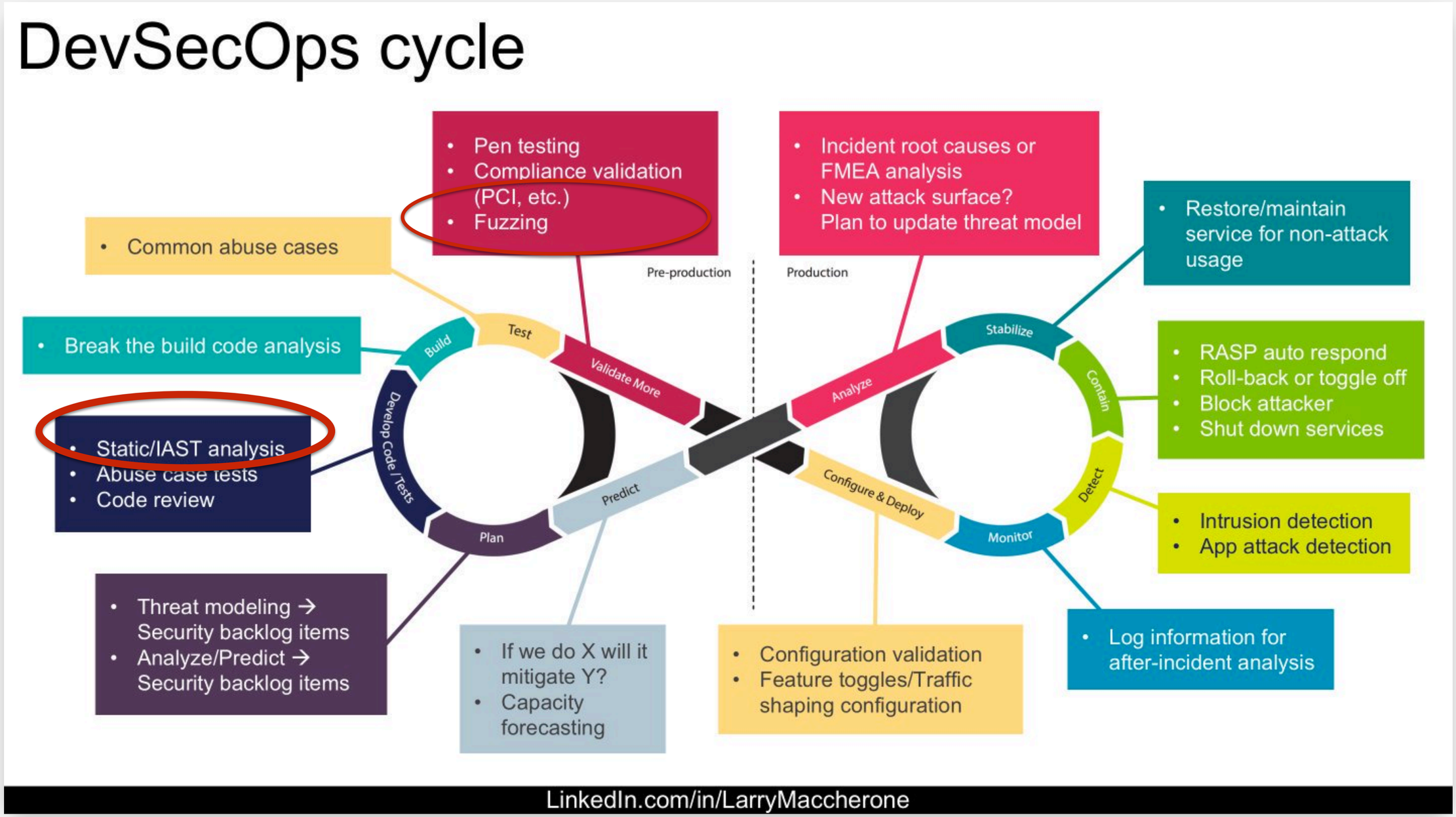
0 Введение



Автоматизация тестирования безопасности

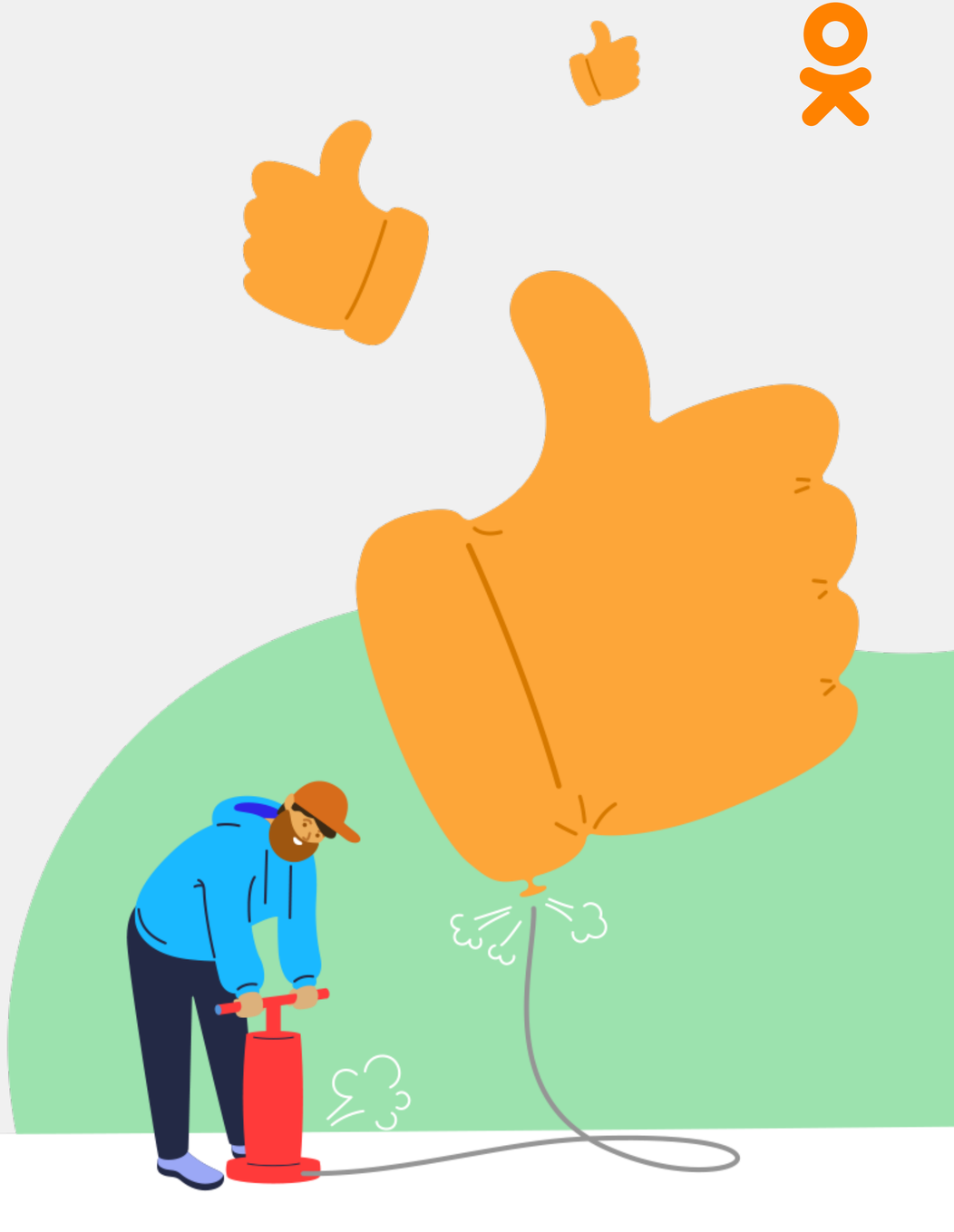


Автоматизация тестирования безопасности

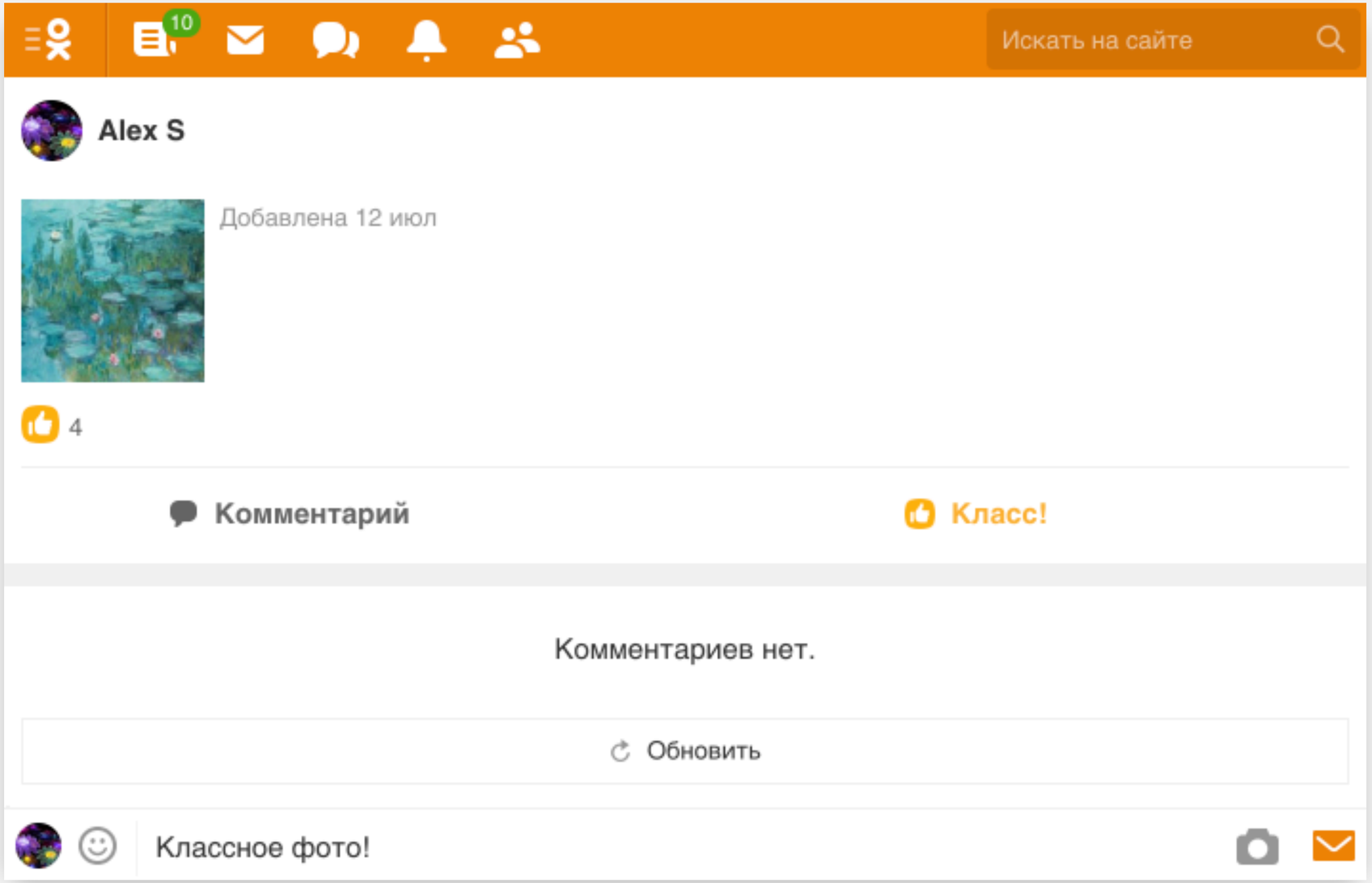


Автоматизация тестирования безопасности

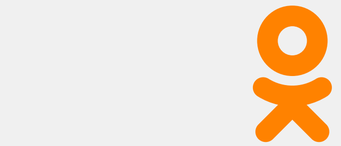
1 Мотивация



Проблема со сканерами уязвимостей

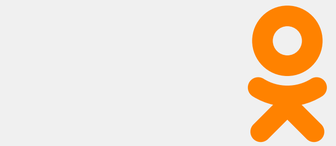


Проблема со сканерами уязвимостей



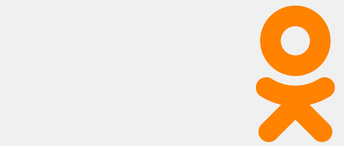
```
POST /dk?
st.cmd=userAlbumPhotoComments&st.edit=off&st.phoId=890573095885&st.page=
1&tkn=8431 HTTP/1.1
Host: m.ok.ru
...
Cookie: ... JSESSIONID=123456; BANNER_LANG=ru

fr.posted=set& ...
fr.emoCnt=0&ds.pcd=u576523818189&fr.msg=%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D
0%BD%D0%BE%D0%B5%20%D1%84%D0%BE%D1%82%D0%BE!&button_submit=button_submit
```

Проблема со сканерами уязвимостей



```
POST /dk?
st.cmd=userAlbumPhotoComments&st.edit=off&st.phoId=890573095885&st.page=
1&tkn=8431 HTTP/1.1
Host: m.ok.ru
...
Cookie: ... JSESSIONID=123456; BANNER_LANG=ru

fr.posted=set& ...
fr.emoCnt=0&ds.pcd=u576523818189&fr.msg=%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D
0%BD%D0%BE%D0%B5%20%D1%84%D0%BE%D1%82%D0%BE!&button_submit=button_submit
```

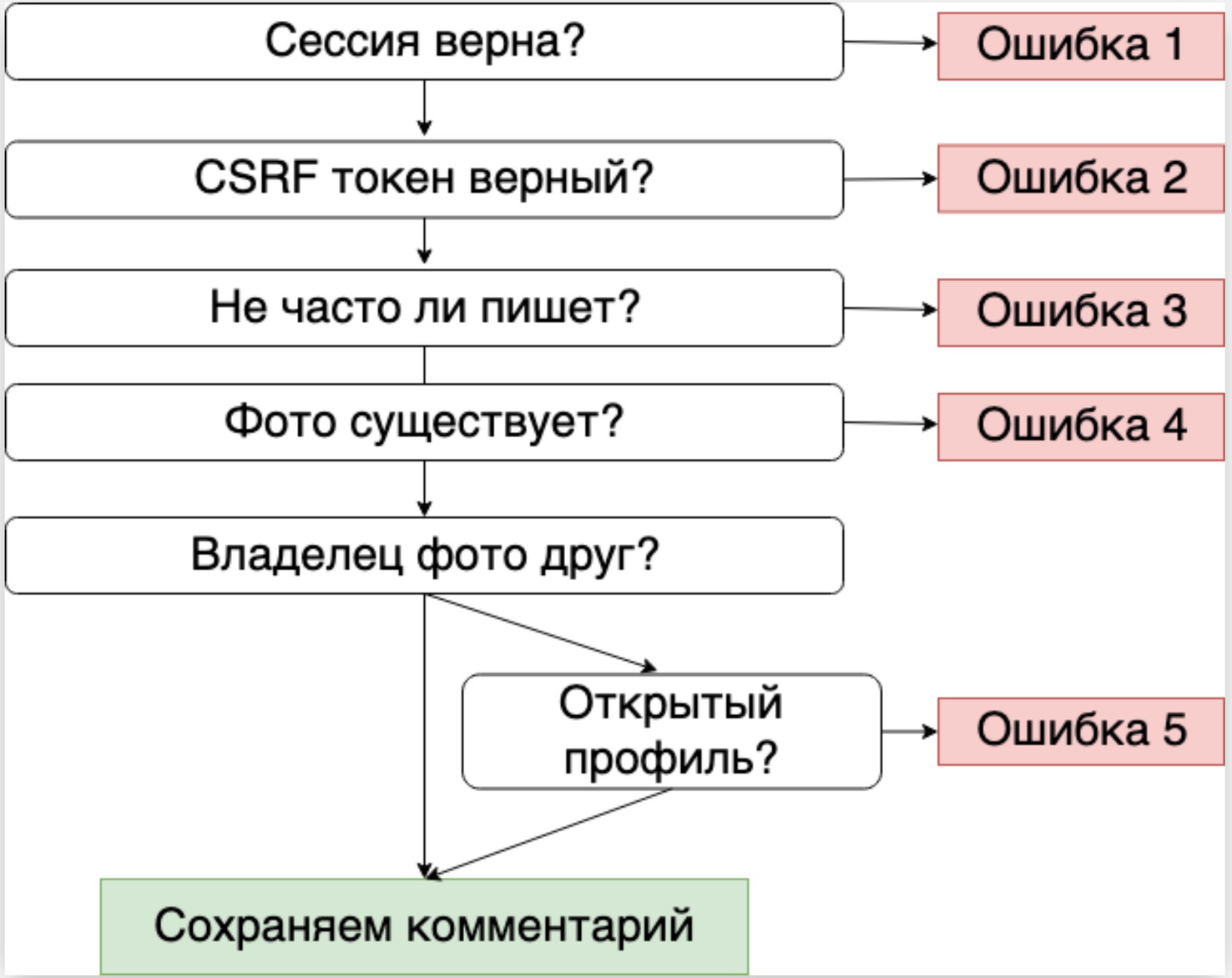
Проблема со сканерами уязвимостей



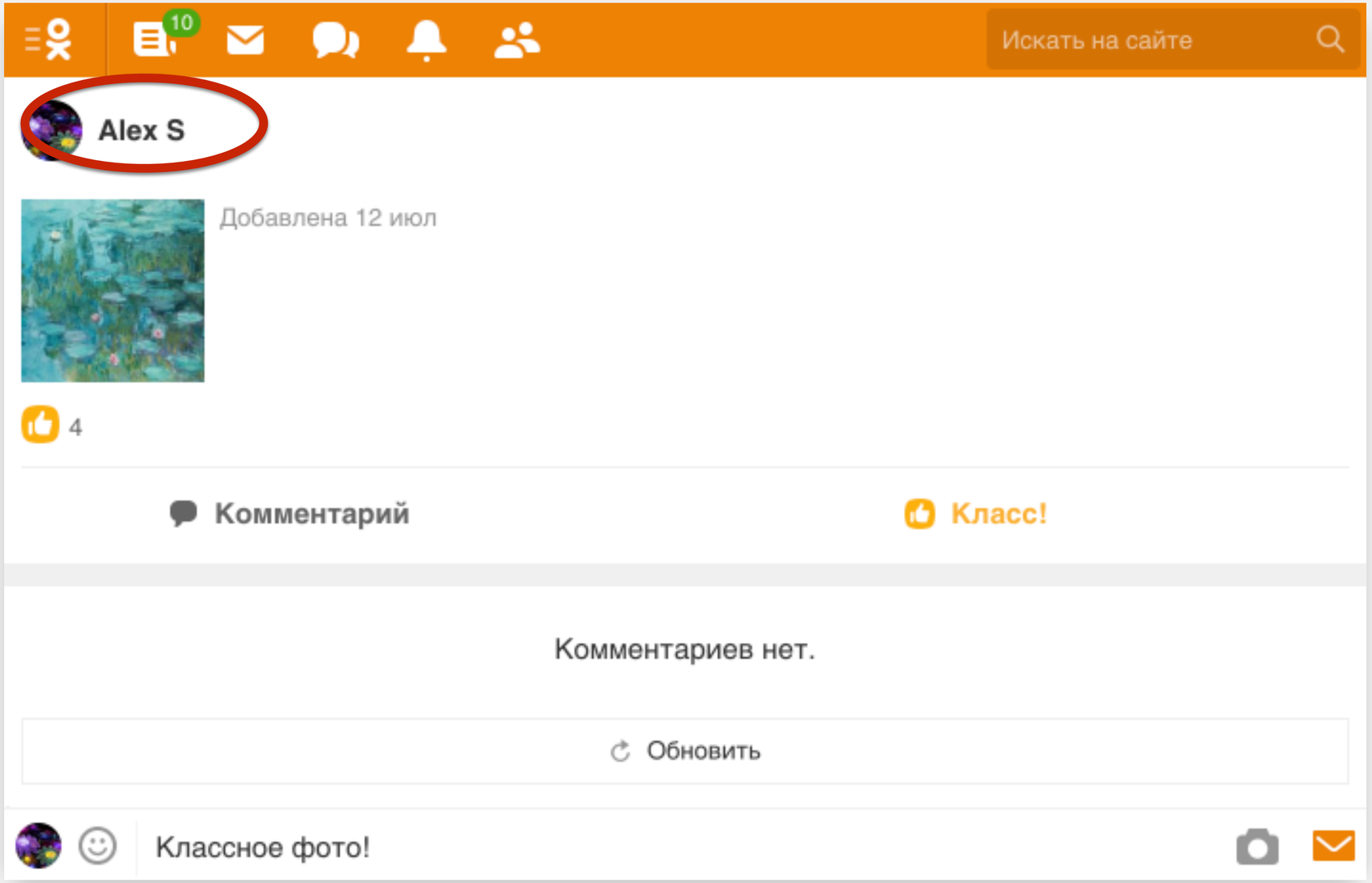
```
\";alert(String.fromCharCode<script>alert('xss')</script>
//--></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
'';!--"<XSS>=&{()}
<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>
<IMG SRC="javascript:alert('XSS');">
<IMG SRC=javascript:alert('XSS')>
<IMG SRC=javascrscriptipt:alert('XSS')>
<IMG SRC=JaVaScRiPt:alert('XSS')>
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">
<IMG SRC=" &#14;  javascript:alert('XSS');">
<SCRIPT/XSS SRC="http://ha.ckers.org/xss.js"></SCRIPT>
<SCRIPT/SRC="http://ha.ckers.org/xss.js"></SCRIPT>
<<SCRIPT>alert("XSS");//<</SCRIPT>
<SCRIPT>a=/XSS/alert(a.source)</SCRIPT>
\";alert('XSS');//
</TITLE><SCRIPT>alert("XSS");</SCRIPT>
<TABLE><TD BACKGROUND="javascript:alert('XSS')">
<DIV STYLE="background-image: url(javascript:alert('XSS'))">
<DIV STYLE="background-image:\0075\0072\006C\0028'\006a\...
<DIV STYLE="width: expression(alert('XSS'));">
```

```
'
a' or 1=1--
"a"" or 1=1--"
 or a = a
a' or 'a' = 'a
1 or 1=1
a' waitfor delay '0:0:10'--
1 waitfor delay '0:0:10'--
6f006e00 exec(@q)
declare @s varchar (200) select @s = 0x73656c65637420404076657273696f6e
exec(@s)
a'
?
' or 1=1
x' AND userid IS NULL; --
x' AND email IS NULL; --
anything' OR 'x'='x
x' AND 1=(SELECT COUNT(*) FROM tabname); --
x' AND members.email IS NULL; --
x' OR full_name LIKE '%Bob%
23 OR 1=1
```

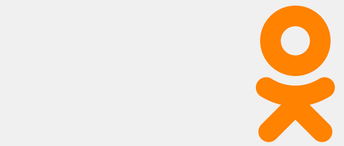
Проблема со сканерами уязвимостей



Проблема со сканерами уязвимостей

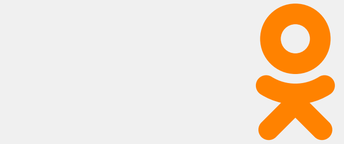


Проблема со сканерами уязвимостей



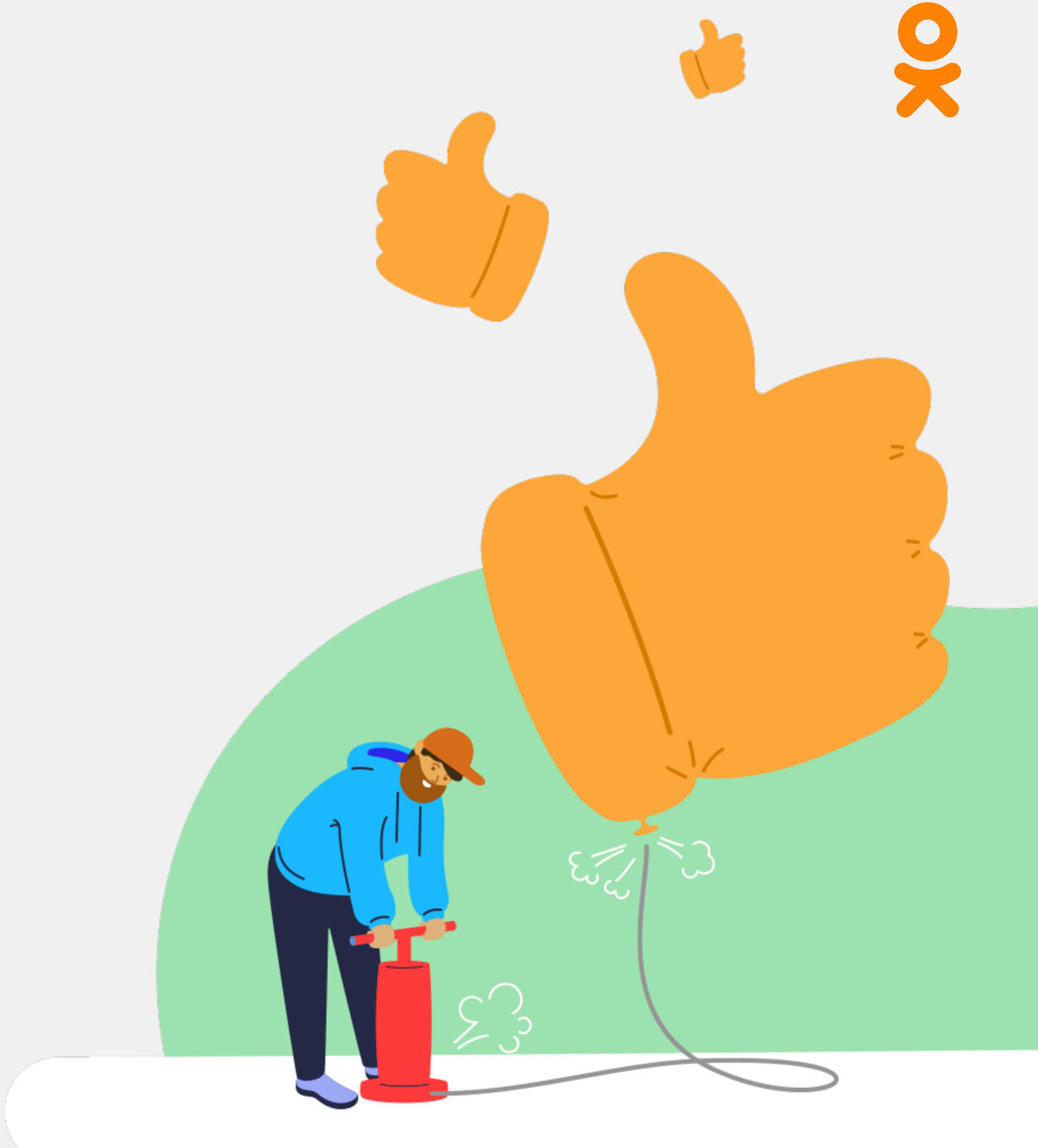
- Комбинаторный взрыв количества запросов
- Не запоминает состояние
- Нужно дообучать после апдейтов
- Не знает про бизнес логику

Что ждем от статического анализа



- Дешевое масштабирование
- Нет забот с окружением и тестовыми данными
- Исправили один баг => найдем другие такие же
- Не только веб

2 Теория



SAST

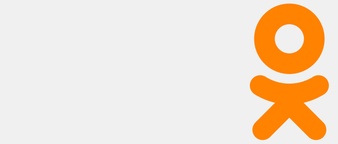
Static Application Security Testing

Поиск паттернов

Нарушения API,
опасные вызовы,
секреты в коде

Анализ потока выполнения / потока данных

Абстрактное синтаксическое дерево (AST)

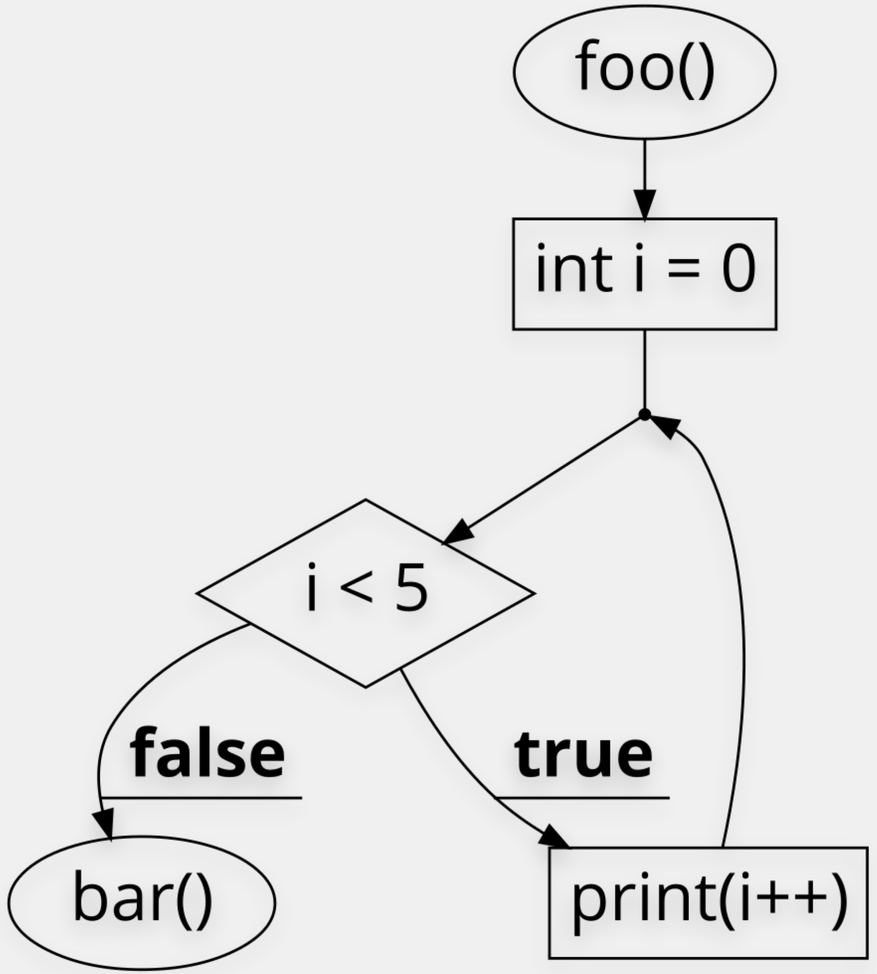


```
interface Foo {
 void bar(@NotNull String x);
}
```

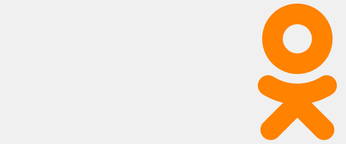
```
...
    - type:
      isInterface: "true"
      name:
        identifier: "Foo"
      members:
        - member:
          type:
          name:
            identifier: "bar"
          parameters:
            - parameter:
              isVarArgs: "false"
              name:
                identifier: "x"
              type:
                name:
                  identifier: "String"
              annotations:
                - annotation:
                  name:
                    identifier: "NotNull"
...
```

Граф потока управления

```
foo();

int i = 0;
while (i < 5) {
    print(i++);
}

bar();
```



Граф потока данных / Taint analysis



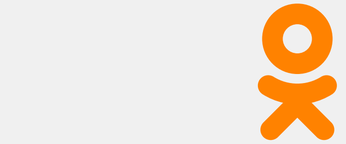
```
String foo = request.getParameter("foo");

response.getWriter().write(foo);
```

Taint analysis

TAINTED

```
String foo = request.getParameter("foo");

response.getWriter().write(foo);
```

Taint analysis



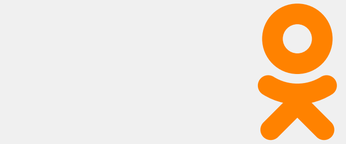
TAINTED

```
String foo = request.getParameter("foo");
```

SINK

```
response.getWriter().write(foo);
```

Taint analysis



TAINTED

```
String foo = request.getParameter(“foo");
```

SAFE

```
foo = HtmlUtils.htmlEscape(foo);
```

~~SINK~~

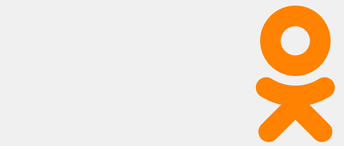
```
response.getWriter().write(foo);
```

Ограничения: Reflection

```
Method method =
    foo.getClass().getDeclaredMethod("bar");

method.setAccessible(true);

method.invoke(foo);
```

~~TAINT ANALYSIS~~

Ограничения: генерация кода на лету



```
<div class="user-info">
    <h1 th:text="${user.name}"></h1>
</div>
```

~~TAINT ANALYSIS~~

SAST vs OWASP Top 10

A1:2017-Injection	
A2:2017-Broken Authentication	
A3:2017-Sensitive Data Exposure	
A4:2017-XML External Entities (XXE)	
A5:2017-Broken Access Control	
A6:2017-Security Misconfiguration	
A7:2017-Cross-Site Scripting (XSS)	
A8:2017-Insecure Deserialization	
A9:2017-Using Components with Known Vulnerabilities	
A10:2017-Insufficient Logging&Monitoring	

SAST vs OWASP Top 10

A1:2017-Injection	
A2:2017-Broken Authentication	
A3:2017-Sensitive Data Exposure	
A4:2017-XML External Entities (XXE)	
A5:2017-Broken Access Control	
A6:2017-Security Misconfiguration	
A7:2017-Cross-Site Scripting (XSS)	
A8:2017-Insecure Deserialization	
A9:2017-Using Components with Known Vulnerabilities	+
A10:2017-Insufficient Logging&Monitoring	

SAST vs OWASP Top 10

A1:2017-Injection	+
A2:2017-Broken Authentication	
A3:2017-Sensitive Data Exposure	
A4:2017-XML External Entities (XXE)	+
A5:2017-Broken Access Control	
A6:2017-Security Misconfiguration	
A7:2017-Cross-Site Scripting (XSS)	+
A8:2017-Insecure Deserialization	+
A9:2017-Using Components with Known Vulnerabilities	+
A10:2017-Insufficient Logging&Monitoring	

SAST vs OWASP Top 10

A1:2017-Injection	+
A2:2017-Broken Authentication	
A3:2017-Sensitive Data Exposure	+
A4:2017-XML External Entities (XXE)	+
A5:2017-Broken Access Control	
A6:2017-Security Misconfiguration	
A7:2017-Cross-Site Scripting (XSS)	+
A8:2017-Insecure Deserialization	+
A9:2017-Using Components with Known Vulnerabilities	+
A10:2017-Insufficient Logging&Monitoring	

SAST vs OWASP Top 10

A1:2017-Injection	+
A2:2017-Broken Authentication	
A3:2017-Sensitive Data Exposure	+
A4:2017-XML External Entities (XXE)	+
A5:2017-Broken Access Control	
A6:2017-Security Misconfiguration	-
A7:2017-Cross-Site Scripting (XSS)	+
A8:2017-Insecure Deserialization	+
A9:2017-Using Components with Known Vulnerabilities	+
A10:2017-Insufficient Logging&Monitoring	

SAST vs OWASP Top 10

A1:2017-Injection	+
A2:2017-Broken Authentication	-
A3:2017-Sensitive Data Exposure	+
A4:2017-XML External Entities (XXE)	+
A5:2017-Broken Access Control	
A6:2017-Security Misconfiguration	-
A7:2017-Cross-Site Scripting (XSS)	+
A8:2017-Insecure Deserialization	+
A9:2017-Using Components with Known Vulnerabilities	+
A10:2017-Insufficient Logging&Monitoring	

SAST vs OWASP Top 10

A1:2017-Injection	+
A2:2017-Broken Authentication	-
A3:2017-Sensitive Data Exposure	+
A4:2017-XML External Entities (XXE)	+
A5:2017-Broken Access Control	+ / -
A6:2017-Security Misconfiguration	-
A7:2017-Cross-Site Scripting (XSS)	+
A8:2017-Insecure Deserialization	+
A9:2017-Using Components with Known Vulnerabilities	+
A10:2017-Insufficient Logging&Monitoring	

Требования к инструменту

- Java, js, ts, Android
- Taint analysis

Требования к инструменту

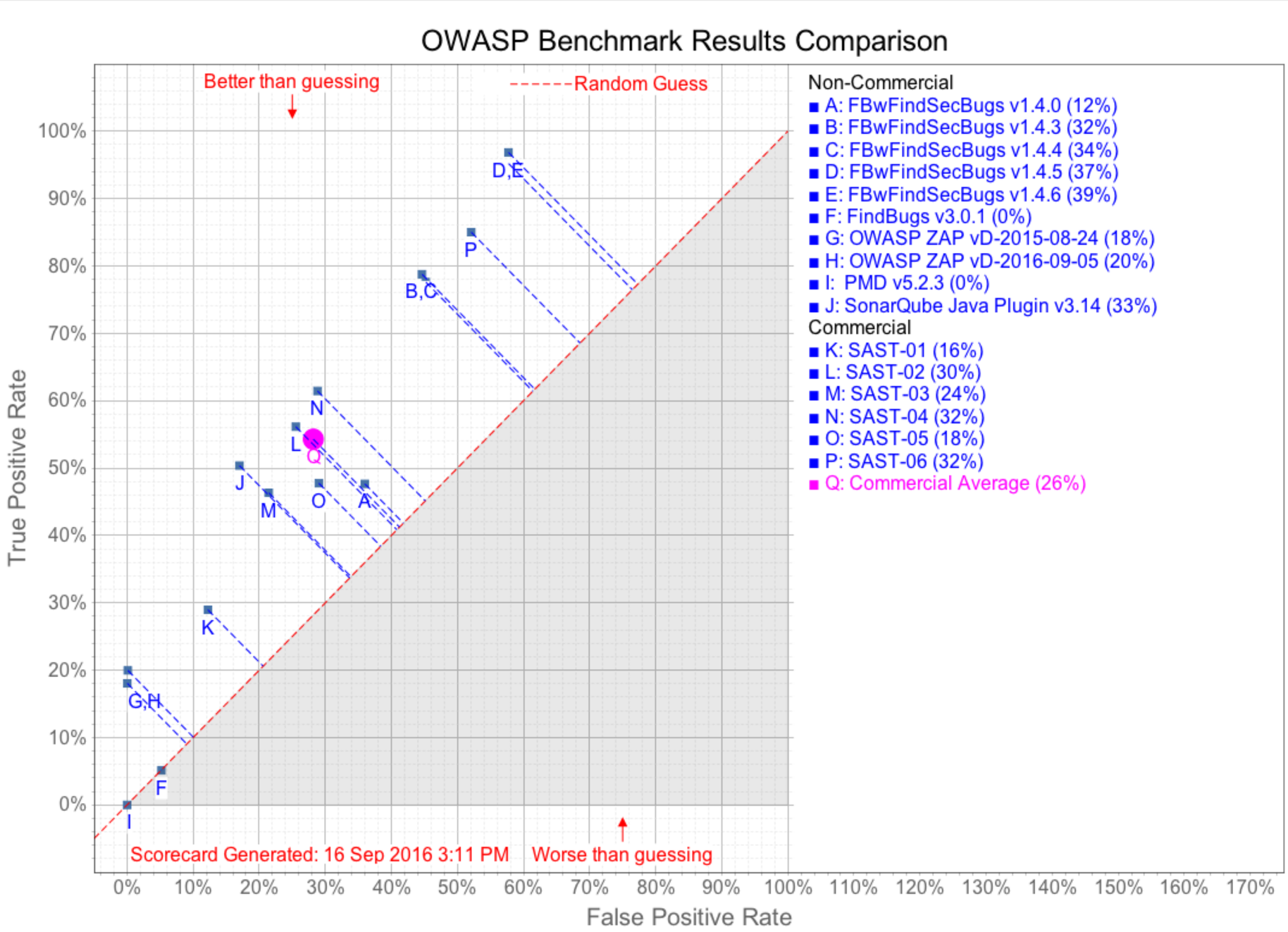
- Java, js, ts, Android
- Taint analysis
- Кастомизация правил

Требования к инструменту

- Java, js, ts, Android
- Taint analysis
- Кастомизация правил
- Совместная работа команды

Требования к инструменту

- Java, js, ts, Android
- Taint analysis
- Кастомизация правил
- Совместная работа команды
- Инкрементальный анализ



https://github.com/OWASP/Benchmark

Требования к инструменту

- Java, js, ts, Android
- Taint analysis
- Кастомизация правил
- Совместная работа команды
- Инкрементальный анализ

Find Security Bugs

https://find-sec-bugs.github.io/

Требования к инструменту

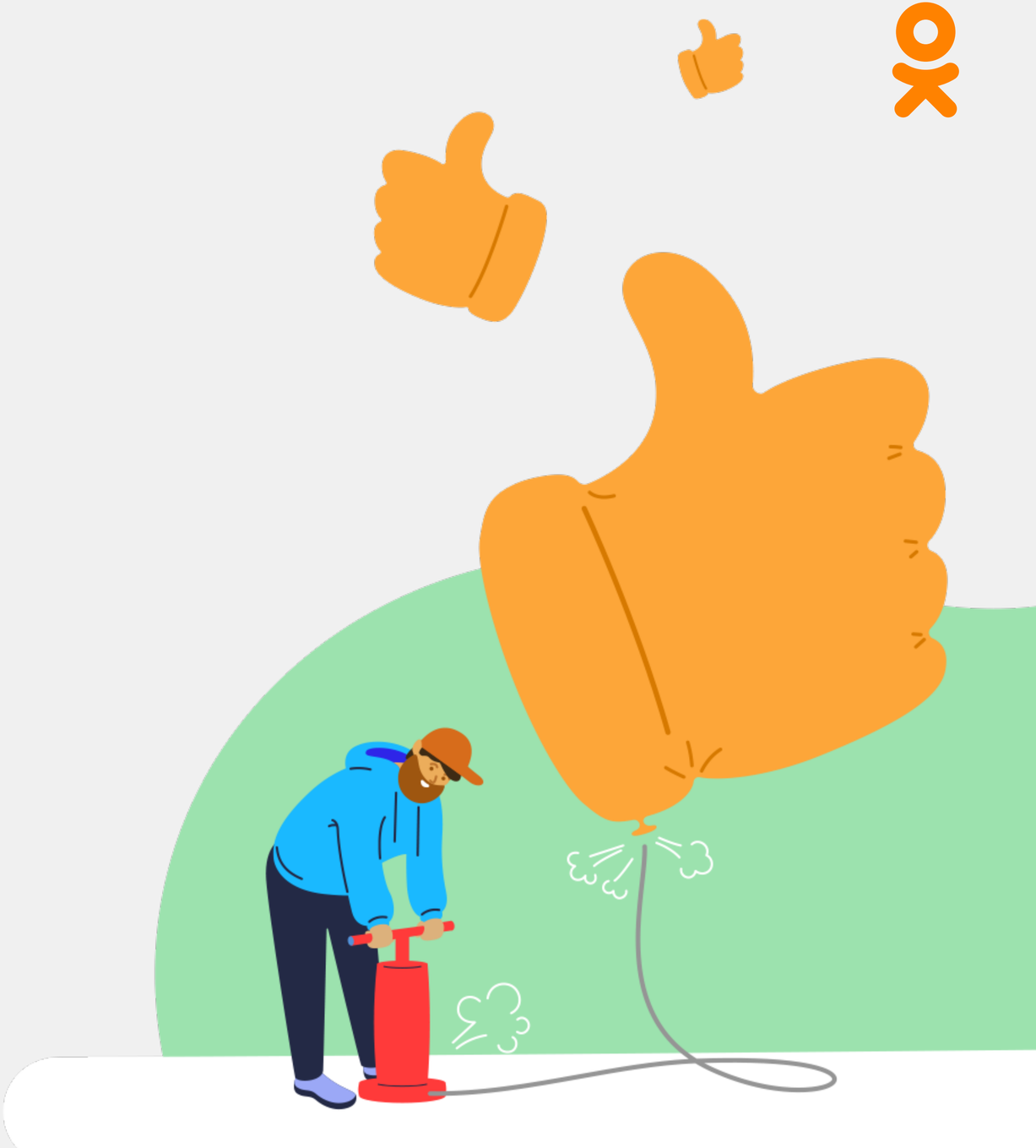
- Java, js, ts, Android
- Taint analysis
- Кастомизация правил
- Совместная работа команды
- Инкрементальный анализ

SonarJava
AndroidLint

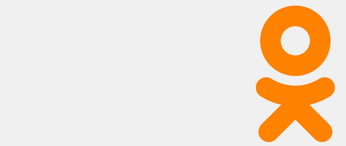
SonarQube

https://www.sonarqube.org/

3 Практика

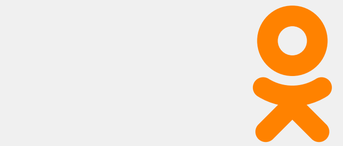


Пример #1



```
@ExceptionHandler({IllegalArgumentException.class})
public void oops(HttpServletRequest request, HttpServletResponse response) {
    String originalURL = request.getRequestURL() + "?" +
            URLDecoder.decode(request.getQueryString());

    …

    PrintWriter writer = response.getWriter();
    writer.write("<h1>Error procesing page " + originalURL + "</h1>");
    writer.flush();

    …
}
```

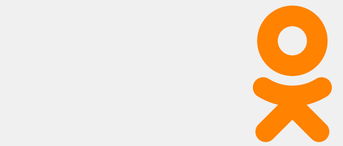
XSS #1



```
@ExceptionHandler({IllegalArgumentException.class})
public void oops(HttpServletRequest request, HttpServletResponse response) {
    String originalURL = request.getRequestURL() + "?" +
            URLDecoder.decode(request.getQueryString());

    …

    PrintWriter writer = response.getWriter();
    writer.write("<h1>Error procesing page " + originalURL + "</h1>");
    writer.flush();

    …
}
```

TAINT

XSS #1



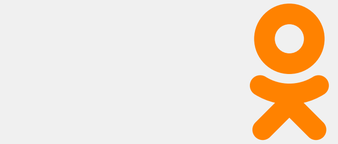
```
@ExceptionHandler({IllegalArgumentException.class})
public void oops(HttpServletRequest request, HttpServletResponse response) {
    String originalURL = request.getRequestURL() + "?" +
            URLDecoder.decode(request.getQueryString());

    …

    PrintWriter writer = response.getWriter();
    writer.write("<h1>Error procesing page " + originalURL + "</h1>");
    writer.flush();

    …
}
```

SINK

XSS #1

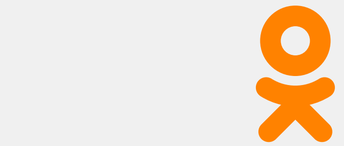

UserController.java in ru.ok.heisenbugdemo.controller
View in browser
87
88 return "/photo";
89 }
90
91 @ExceptionHandler({IllegalArgumentException.class})
92 public void oops(HttpServletRequest request, HttpServletResponse response) {
93 String originalURL = request.getRequestURL() + "?" +
94 URLDecoder.decode(request.getQueryString());
95
Find
Next
Previous
Potential XSS in Servlet
A potential XSS was found. It could be used to execute unwanted JavaScript in a client's browser. (See references)
Vulnerable Code:
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOExceptio
String input1 = req.getParameter("input1");
[...]
resp.getWriter().write(input1);

XSS #1



```
This use of java/io/PrintWriter.write(Ljava/lang/String;)V could be vulnerable to XSS
At UserController.java:[line 101]
In method ru.ok.heisenbugdemo.controller.UserController.oops(HttpServletRequest, H
Sink method java/io/PrintWriter.write(Ljava/lang/String;)V
Sink parameter 0
Unknown source javax/servlet/http/HttpServletRequest.getRequestURL()Ljava/lang/S
Unknown source java/net/URLDecoder.decode(Ljava/lang/String;)Ljava/lang/String;
Unknown source javax/servlet/http/HttpServletRequest.getQueryString()Ljava/lang/St
At UserController.java:[line 93]
At UserController.java:[line 94]
```

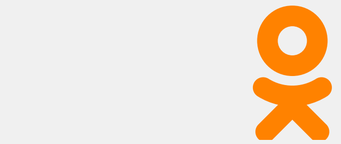
Пример #2



```
@GetMapping("/photo")
public String photo(@RequestParam("id") long id, Model m) {
  Photo photo = photoRepository.findOne(id);
  …
  m.addAttribute("photo", photo)
  …
  return "/photo";
}
```

```
…
<div th:each="comment : ${photo.comments}">
  <p th:utext="“${comment.text}”"></p>
</div>
…
```

XSS #2 — вариант 1



```
@GetMapping("/photo")
public String photo(@RequestParam("id") long id, Model m) {
  Photo photo = photoRepository.findOne(id);
  …
  m.addAttribute("photo", photo)
  …
  return "/photo";
}
```

TAINT

```
…
<div th:each="comment : ${photo.comments}">
  <p th:utext="“${comment.text}”"></p>
</div>
…
```

SINK

XSS #2 — вариант 2

```
@GetMapping("/photo")
public String photo(@RequestParam("id") long id, Model m) {
  Photo photo = photoRepository.findOne(id);
  …
  m.addAttribute("photo", photo)
  …
  return "/photo";
}
```

TAINT

SINK

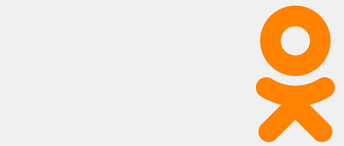
```
…
<div th:each="comment : ${photo.comments}">
  <p th:utext="“${comment.text}”"></p>
</div>
…
```

XSS #2 — вариант 2 — детектор

```
-Dfindsecbugs.taint.customconfigfile=/my/config/custom_sink.txt
```

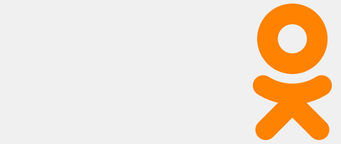
```
org/springframework/ui/Model.addAttribute(Ljava/lang
  String;Ljava/lang/Object;)Lorg/springframework/ui/Model;:1
```

Пример #3



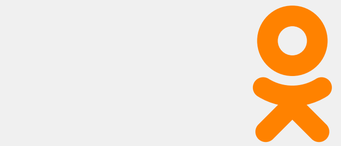
```
@GetMapping(“/photo")
public String photo(@RequestParam("id") long id, Model model) {
    Photo photo = photoRepository.findOne(id);

    . . .

    model.addAttribute("photo", photo);

    return "/photo";
}
```

Insecure Direct Object Reference (IDOR)



```
@GetMapping(“/photo")
public String photo(@RequestParam("id") long id, Model model)
{
        Photo photo = photoRepository.findOne(id);
        . . .

/*      User currentUser = getCurrentUser();

        if (!canAccess(currentUser, author)) {
            return "/error/403";
        }
*/
        model.addAttribute("user", author);
        model.addAttribute("photo", photo);

        return "/photo";
}
```

IDOR — детектор



```
import edu.umd.cs.findbugs.Detector;

public class IdorDetector implements Detector {

    @Override
    public void visitClassContext(ClassContext classContext) {

    }

    @Override
    public void report() {

    }
}
```

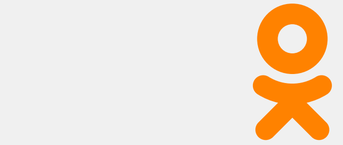
IDOR — детектор

```
    public void visitClassContext(ClassContext classContext) {

    List<Method> endpoints = findEndpoints(classContext);


    for (Method m : endpoints) {

        checkCanAccessCalled(classContext, m);

    }
}
```

IDOR — детектор

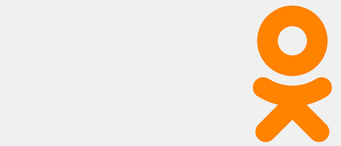
```
    REQUEST_MAPPING_ANNOTATION_TYPES = Arrays.asList(
            "Lorg/springframework/web/bind/annotation/GetMapping;",
            "Lorg/springframework/web/bind/annotation/PostMapping;",
              . . .
private List<Method> findEndpoints(JavaClass javaClass) {
    . . .
    for (Method m : javaClass.getMethods()) {

        for (AnnotationEntry ae : m.getAnnotationEntries()) {

            if (REQUEST_MAPPING_ANNOTATION_TYPES
                    .contains(ae.getAnnotationType())) {
                endpoints.add(m);
            }
        }
    }
    . . .
```

IDOR — детектор



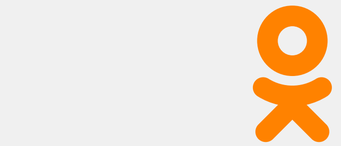
```
private void checkCanAccessCalled(ClassContext classContext, Method m) {
    . . .
    CFG cfg = classContext.getCFG(m);

    for (Iterator<Location> i = cfg.locationIterator(); i.hasNext(); ) {

        Instruction inst = i.next().getHandle().getInstruction();

        if (inst instanceof INVOKESPECIAL) {
            . . .
            if (CAN_ACCESS_METHOD_NAME.equals(invoke.getMethodName(cpg)) &&

                                    className.equals(invoke.getClassName(cpg))) {
                found = true;
            }
        }
    }
    . . .
```

IDOR — детектор



```
. . .

if (!found) {

    bugReporter.reportBug(
        new BugInstance(this, "IDOR", Priorities.NORMAL_PRIORITY)
            .addClass(javaClass)
            .addMethod(javaClass, method));
}

. . .
```

IDOR — детектор — что дальше?



```
public abstract class BaseController {

    protected boolean canAccess() {
        return false;
    }

}

public class Controller extends BaseController {

    @GetMapping("foo")
    public String foo() {
        . . .
        canAccess()
        . . .
}
```

IDOR — детектор — что дальше?

```
public class Controller  {

    private boolean canAccess() {
        . . .
    }

    private boolean canAccessEx() {
        canAccess()
        . . .
    }

    @GetMapping("foo")
    public String foo() {
        . . .
        canAccessEx()
        . . .
    }
}
```

Выводы

- Статический анализ позволит дешево найти типовые уязвимости

- В реальном мире стандартных детекторов недостаточно

Полезные ссылки

1. https://www.owasp.org/index.php/
   Category:OWASP_Application_Security_Verification_Standard_Project

2. https://www.owasp.org/index.php/Static_Code_Analysis

3. https://find-sec-bugs.github.io/

4. https://github.com/find-sec-bugs/find-sec-bugs/wiki/Writing-a-detector

5. https://www.ysofters.com/2015/08/31/taint-analysis-added-to-findbugs/

6. https://cwe.mitre.org/


одноклассники