Introduction

- Secure programming is the practice of writing programs that are resistant to attacks by malicious people or programs
  - Check for SQL Injection.
  - Check for Cookie Poisoning.
  - Check for XSS.
  - ...

- Security vulnerability or just vulnerability is a flaw that can be exploited to allow an attacker to cause unintended operations

- 86% of audited websites has at least 1 serious security vulnerability (WhiteHat - 2013)
Top 10 vulnerabilities

- Open Web Application Security Project (OWASP)

- OWASP Top 10 – 2013
  - From 8 datasets from 7 companies with over 500,000 vulnerabilities
  - 01 – (SQL/Command) Injection
  - 02 – Broken Authentication and Session Management
  - 03 – Cross-Site Scripting (XSS)
  - 04 – Insecure Direct Object References
  - 05 – Security Misconfiguration
  - 06 – Sensitive Data Exposure
  - 07 – Missing Function Level Access Control
  - 08 – Cross-Site Request Forgery (CSRF)
  - 09 – Using Known Vulnerable Components
  - 10 – Unvalidated Redirects and Forwards

Static analysis can help

- SSVChecker, FindBug, ASIDE, Lapse+, CodePro Analytics, CodeProfiler, JeSS and AppScan IBM

Key characteristics

- Late detection
- Pattern matching
How is usually done... Late detection

- SQL Injection
- Broken Authentication
- Cross-site scripting (XSS)
- Security flaw: Cross Object References
- Security Misconfiguration
- Privilege Escalation
- Bypass Function Level Access Control
- Cross Site Request Forgery
- Known Vulnerable Components
- Uncontrolled Redirects and Forwarding

Early detection can help

String login = request.getParameter("login");
String sql = "SELECT * FROM USER WHERE LOGIN = " + login + ""
Statement statement = conn.createStatement();
ResultSet resultSet = statement.executeQuery(sql);
request.setAttribute("login", login);
Early detection requirements

- High accurate detection technique
- Rate of false positives

- The usage of time and resources can not disturb the developer

Early detection using low accurate technique

```java
@override
@override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    request.getParameter(" ");
}
```
**Frequently used technique**

- **Pattern matching** is a technique for checking if a pattern matches a given sequence of tokens (letters, numbers, punctuation, and certain symbols)
- 20/30% of false positives - (Nadeem 2012)

**Pattern matching example**

```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) {
    try {
        PrintWriter printWriter = response.getWriter();
        printWriter.println("a");
        printWriter.println("b");
        String d = "d";
        printWriter.println((null != "") ? "c" : d);
        printWriter.println(request.getParameter("bad"));
    }
}
```

Luciano Sampaio

Pattern Matching technique
Data-flow analysis as an alternative

- Data-flow analysis (DFA) is a technique for gathering information about the possible set of values calculated at various points in a computer program.

- Originally created and commonly used for implementing optimizations on compilers.

- Currently, there is only one solution that uses DFA to detect security vulnerabilities (CodePro).

Data-flow analysis example

```java
@Override
protected void doGet(HttpServletRequest request,
                        HttpServletResponse response)
        throws ServletException, IOException
```
```java
    PrintWriter printWriter = response.getWriter();
    printWriter.print("a");
    printWriter.print("b");
```
```java
    String d = "d";
    printWriter.print((null != "" || d) ? "c" : d);
```
```java
    printWriter.println(Boolean.parseBoolean(request.getParameter("bad")));
    printWriter.println(request.getParameter("request"));
```
```java
private String getParameterString(HttpServletRequest request) {
    int i = 0;
    if (i > 10) { return null; }
    String bad = request.getParameter("bad");
    return bad;
    return "ok";
}
Key limitations of the state-of-the-art

- Late detection does not support secure programming but rather security analysis
- Frequently used vulnerability detection techniques present a high rate of false positives
  - False positives are even more critical in early detection
  - Pattern matching or primitive DFA

Research questions

- 01 - Can advanced DFA decrease the rate of false positives when compared to other techniques?
- 02 - Can the early detection approach help developers produce more secure code when compared to late detection?
Proposed solutions

- Propose to support a change from the default behavior of late detection to early detection
- Propose new heuristics using a technique named context-sensitive data flow analysis
  - Pattern matching
  - Context-insensitive data flow analysis
- Designed and implemented a prototype
- Performed 2 empirical studies

How to identify something as unsafe/safe?

- Entry-Point
- Sanitization-Point
- Exit-Point
```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    PrintWriter printWriter = response.getWriter();
    Animal animal1 = new Animal();
    String ok = "ok";
    animal1.setName(ok);
    Animal animal2 = new Animal();
    String bad = request.getParameter("bad");
    animal2.setName(bad);
    printWriter.print(animal1.getName());
    printWriter.print(animal2.getName());
}
```
Our proposed heuristics

Data Flow Analysis >>

Supported vulnerabilities

- Vulnerabilities that stem from program input and output not being properly validated are recognized as being the most common ones
- These vulnerabilities are not dependent on how they are implemented

- 01 - Command Injection
- 02 - Cookie Poisoning
- 03 - Cross-Site Scripting (XSS)
- 04 - HTTP Response Splitting
- 05 - LDAP Injection
- 06 - Log Forging
- 07 - Path Traversal
- 08 - Reflection Injection
- 09 - Security Misconfiguration
- 10 - SQL Injection
- 11 - XPath Injection
New type of problems - Infinite Loop

```java
@override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    infiniteLoop(request, response);
}

private void infiniteLoop(HttpServletRequest request, HttpServletResponse response)
    throws IOException {
    PrintWriter printWriter = response.getWriter();
    String a = request.getParameter("a");
    printWriter.print(a);
    infiniteLoop(request, response);
    String b = request.getParameter("b");
    printWriter.print(b);
}
```

Current limitations of our implementation

- Containers
- InnerClasses

Evaluation

- Rate of false positives
  - Exploratory study - Benchmark on 5 open-source projects and 1 custom-made project

- Early detection effectiveness
  - Controlled experiment - Participants were asked to create a code using our tool

Study 1: Accuracy Benchmarking

<table>
<thead>
<tr>
<th></th>
<th>Blueblog</th>
<th>Personalblog</th>
<th>WebGoat</th>
<th>Roller</th>
<th>Pebble</th>
<th>NCO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1.0</td>
<td>1.2.6</td>
<td>5.4</td>
<td>0.9.9</td>
<td>2.6.4</td>
<td>1.0</td>
</tr>
<tr>
<td>Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>of packages</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>of classes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>of methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lines of Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>of Vulnerabilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Analyzed projects

<table>
<thead>
<tr>
<th>Pattern Matching</th>
<th>Data Flow Analysis - CI</th>
<th>Data Flow Analysis - CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lapse+</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ASIDE</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CodePro</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ESVD</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Selected solutions

Luciano Sampaio
## Analyzed vulnerabilities

<table>
<thead>
<tr>
<th>Nr</th>
<th>Vulnerability</th>
<th>Pattern Matching</th>
<th>DFA - CI</th>
<th>DFA - CS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ASIDE</td>
<td>Lapse+</td>
<td>CodePro</td>
</tr>
<tr>
<td>1</td>
<td>Command Injection</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Cookie Poisoning</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Cross-Site Scripting (XSS)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>HTTP Response Splitting</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>LDAP Injection</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Log Forging</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Path Traversal</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>Reflection Injection</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>Security Misconfiguration</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>SQL Injection</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>XPath Injection</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>4</td>
<td>9</td>
<td>11</td>
</tr>
</tbody>
</table>

## Summary

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
<th>% False Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIDE</td>
<td>0,48</td>
<td>0,39</td>
<td>0,43</td>
<td>51,78%</td>
</tr>
<tr>
<td>CodePro</td>
<td>0,62</td>
<td>0,07</td>
<td>0,13</td>
<td>37,62%</td>
</tr>
<tr>
<td>Lapse+</td>
<td>0,55</td>
<td>0,36</td>
<td>0,43</td>
<td>44,73%</td>
</tr>
<tr>
<td>ESVD</td>
<td>0,88</td>
<td>0,66</td>
<td>0,75</td>
<td>11,70%</td>
</tr>
</tbody>
</table>
Study 1 >>

**Summary of False Positives**

<table>
<thead>
<tr>
<th></th>
<th>Blueblog</th>
<th>Personalblog</th>
<th>WebGoat</th>
<th>Roller</th>
<th>Pebble</th>
<th>NCO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIDE</td>
<td>74%</td>
<td>13%</td>
<td>51%</td>
<td>70%</td>
<td>50%</td>
<td>29%</td>
</tr>
<tr>
<td>Lapse+</td>
<td>20%</td>
<td>25%</td>
<td>50%</td>
<td>22%</td>
<td>29%</td>
<td>64%</td>
</tr>
<tr>
<td>CodePro</td>
<td>59%</td>
<td>17%</td>
<td>32%</td>
<td>59%</td>
<td>54%</td>
<td>61%</td>
</tr>
<tr>
<td>ESVD</td>
<td>0%</td>
<td>3%</td>
<td>21%</td>
<td>1%</td>
<td>5%</td>
<td>61%</td>
</tr>
</tbody>
</table>

Results of false positives per analyzed project

---

**WebGoat - False Positive**

```java
int nextId = getNextId(s);
String query = "INSERT INTO employee VALUES (" + nextId + ",?,?,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,??,?,?...,?

// System.out.println("Query: " + query);

try {
    PreparedStatement ps = WebSession.getConnection().prepareStatement(query);
    ps.setString(1, employee.getFirstName().toLowerCase());
    ps.setString(2, employee.getLastName());
    ps.setString(3, employee.getSex());
    ps.setString(4, employee.getUsername());
    ps.setString(5, employee.getPhoneNumber());
    ps.setString(6, employee.getAddress());
    ps.setString(7, employee.getAddress());
    ps.setInt(8, employee.getManager());
    ps.setString(9, employee.getBirthDate());
    ps.setString(10, employee.getBirthday());
    ps.setString(11, employee.getContact());
    ps.setString(12, employee.getDisciplinaryActionDate());
    ps.setString(13, employee.getDisciplinaryActionNotes());
    ps.setString(14, employee.getPersonalDescription());
    ps.execute();
}
```

---

Luciano Sampaio
The applications are ordered by size
The applications are ordered by size

We achieved 11.70% of rate of false positives
- The best pattern-matching result was 44.73%

There is a trade-off, better results mean more time and memory usage
- This can be problem for large projects when using DFA-CS

RQ1 - Can DFA-CS decrease the rate of false positives when compared to other techniques? YES!
Study 2: Late vs. Early Detection – A Quasi-Experiment

- 2 groups of participants (students and professionals), divided in 2 groups (Early Detection and Late Detection)
  - Both using ESVD

- Asked them to develop some functionalities of a small system
  - Initial project and basic jsp pages already created
  - Login and logout
  - Add, Update, Delete and List comments

- Recorded their screen, audio and Eclipse’s interactions
  - ScreenFlow
  - Rabbit-eclipse

Participants

<table>
<thead>
<tr>
<th></th>
<th>Early</th>
<th>Late</th>
<th>Total</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>34</td>
</tr>
<tr>
<td>Professional</td>
<td>7</td>
<td>7</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

Graph:
- Quantity: 15, 1.2, 7.25
- Average of Years: 6.67

Luciano Sampaio
Study 2 >>

Participants - Final numbers

<table>
<thead>
<tr>
<th></th>
<th>Early</th>
<th>Late</th>
<th>Total</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>34</td>
</tr>
<tr>
<td>Professional</td>
<td>7</td>
<td>7</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Early</th>
<th>Late</th>
<th>Total</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>Professional</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Study 2 >>

Programming timing and completed tasks

<table>
<thead>
<tr>
<th></th>
<th>Programming time</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Early</td>
<td>Late</td>
<td></td>
</tr>
<tr>
<td>Professional</td>
<td>9:32:40</td>
<td>4:31:07</td>
<td>14:03:47</td>
</tr>
<tr>
<td>Student</td>
<td>1:44:18</td>
<td>2:46:15</td>
<td>4:30:33</td>
</tr>
<tr>
<td>Sum</td>
<td>11:16:58</td>
<td>7:17:22</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>18:34:20</td>
</tr>
</tbody>
</table>
During the experiment, 57 security vulnerabilities were added

- Early detection group added 35 vulnerabilities and removed 12 (or 34,2%) vulnerabilities
- Late detection group added 22 vulnerabilities and only removed 2 (or 9,09%)

RQ2 - Can the early detection approach help developers produce more secure code when compared to late detection? **YES!**
### Conclusion

- Based on our two studies:
  - Data flow analysis with context-sensitivity reduced the rate of false positives when compared to other techniques
  - Early detection combined with DFA-CS helped developers produce more secure code

---

### Conclusions

- The heuristic strategies capable of finding 11 security vulnerabilities that stem from input and output not being properly sanitized
- Proposal and implementation of the algorithm of data flow analysis with context sensitivity to find security vulnerabilities
- The complete list with known security vulnerabilities (ground truth) for each of the analyzed open-source projects
Future work

- Increase the number of supported vulnerabilities
  - We currently support 11 types

- Add a ranking system for the found vulnerabilities
  - Asked by several participants

- Allow developers to add, edit or remove methods from the lists of entry-points, exit-points and sanitization-points
An **entry-point**, also referred as source, is a point in the source code where external and untrusted input enters the application.

We have 81 entry-points registered.

```xml
<entrypoint id="01">
  <qualifiedname>javax.servlet.ServletRequest</qualifiedname>
  <methodname>getAttribute</methodname>
  <parameters type="java.lang.String" />
</entrypoint>
<entrypoint id="02">
  <qualifiedname>javax.servlet.ServletRequest</qualifiedname>
  <methodname>getAttributeNames</methodname>
</entrypoint>
<entrypoint id="03">
  <qualifiedname>javax.servlet.ServletRequest</qualifiedname>
  <methodname>getCharacterEncoding</methodname>
</entrypoint>
```

A **sanitization-point**, also referred as sanitizer, is a point in the source code where a method or class receives an untrusted input and returns it as a trusted output.

We have 52 sanitization-points registered.

```xml
<sanitizer id="01">
  <qualifiedname>org.owasp.encoder.Encode</qualifiedname>
  <methodname>forHtml</methodname>
  <parameters type="java.lang.String" />
</sanitizer>
<sanitizer id="02">
  <qualifiedname>org.owasp.encoder.Encode</qualifiedname>
  <methodname>forHtmlContent</methodname>
  <parameters type="java.lang.String" />
</sanitizer>
<sanitizer id="03">
  <qualifiedname>org.owasp.encoder.Encode</qualifiedname>
  <methodname>forHtmlAttribute</methodname>
  <parameters type="java.lang.String" />
</sanitizer>
```
Accepted Rules

- 0 - Anything
- 1 - Sanitized
- 2 - Null
- 4 - Literal
- 8 - Concatenation (used for SQL Injection)
- ...

Pattern Matching vs Data Flow Analysis

Pattern Matching

Data Flow Analysis
Data Flow Analysis >>

Early Detection

```
@Override
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
    request.getParameter("id");
}
```

```
@Override
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
    String bad = request.getParameter("bad");
    response.getWriter().println(bad);
}
```

```
@Override
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
    String bad = request.getParameter("bad");
    String safe = ESAPI.encoder().encodeForHTML(bad);
    response.getWriter().println(safe);
    response.getWriter().println(ESAPI.encoder().encodeForHTML(bad));
}
```

Study 2 >>

Searching for help

<table>
<thead>
<tr>
<th>Link</th>
<th>Nr Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>52</td>
</tr>
</tbody>
</table>

You can learn from HelloWold, but should never use its source code.
ESVD - Early Security Vulnerability Detector - 0.3.9;

Download at: (FREE)
https://marketplace.eclipse.org/content/early-security-vulnerability-detector-esvd/

A project containing several security vulnerabilities:
http://www.inf.puc-rio.br/~lsampaio/plugin/early_vulnerability-detector/latest/WebDemo.zip

How to use ESVD: (Portuguese)
https://www.youtube.com/watch?v=pN38gMWvHQ

More info at: http://thecodemaster.net/
The plug-in >> Preferences Page

Vulnerability Detector

We would like to thank the following people and organisations for their direct and/or indirect contributions to the Security Analyzer Plug-in.

Authors

Luciano Sampaio

Contributors

Alessandro Garcia
Bruno Caffe
Danilo Albuquerque
Diego Albuquerque
Diego Cedrim
Eiji Adachi

Everton Tavares
João Neves
Leonardo Sousa
Manuel Ferreira
Marcio Carner
Wilian Otsumi

http://thecodemaster.net

The plug-in >> Preferences Page

Settings

The main settings of the plug-in.

Run Mode

- Run on Save
- Run Manually

Output options

- Security view

Luciano Sampaio
The plug-in >>
Provide possible solutions

```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    PrintWriter printWriter = response.getWriter();

    // Some code...
    String a = request.getParameter("a");
    printWriter.println(a);
    String b = request.getParameter("b");
    printWriter.println(b);
    printWriter.println(a + b);
}
```

Luciano Sampaio
The plug-in

Early Security Vulnerability Detector - ESVD 0.3.9

Marketplace Client Install

Luciano Sampaio