1. What You Need to Know about How Tools Work

2. The Tools And Their Use
p = requesttable;
while (p != (struct table *)0)
{
    if (p->entrytype == PEER_MEET)
    {
        found = (!(strcmp (her, p->me)) &&
                   !(strcmp (me, p->her)));
    }
    else if (p->entrytype == PUTSERVR)
    {
        found = !(strcmp (her, p->me));
    }
    if (found)
        return (p);
    else
        p = p->next;
}
return ((struct table *) 0);
A Bit of History

Compiler warnings

Let the Compiler Help

- Turn on compiler warnings and fix problems
- Easy to do on new code
- Time consuming, but useful on old code
- Use lint, multiple compilers
- **-Wall** is not enough!
  
gcc: **-Wall, -W, -O2, -Werror, -Wshadow, -Wpointer-arith, -Wconversion, -Wcast-qual, -Wwrite-strings, -Wunreachable-code** and many more
  
  - Many useful warning including security related warnings such as format strings and integers
A Bit of History

• Lint (1979)
  – C program checker.
  – Detects suspicious constructs:
    • Variables being used before being set.
    • Division by zero.
    • Conditions that are constant.
    • Calculations whose result is likely to overflow.

• Current automated assessment tools are a sort of “super-Lint”.

Source Code Analysis Tools

• Designed to analyze source code or binaries to help find security flaws.

• The source code may contain inadvertent or deliberate weaknesses that could lead to security vulnerabilities in the executable versions of the application program.

• Better to use them from the beginning of the software development life cycle.
  – Though commonly applied to legacy code.
Source Code Analysis Tools

• Program that parses and then analyses the source code.
• Doesn’t know what the program is supposed to do.
• Looks for violations of good programming practices.
• Looks for specific programming errors.
• Works like a compiler
  – Instead of binaries, it produces an intermediate representation
Source Code Analysis Tools

You can get 2 out of 3

precision

speed

#checks

Courtesy of RedLizards
Source Code Analysis Tools

Different kind of tools:

- Syntax vs. semantics
- Interprocedural
  Whole program analysis
  Local vs. paths
  Data flow analysis
- Sound vs. approximate

Implications:

- Scalability
- Accuracy
Different kind of tools

```c
char* cmd = "/bin/ls";
exec1 (cmd, NULL);
```

Pattern (syntax) matching
- Will say “always dangerous”.

Semantic analysis
- Sometimes definitely no.
Different kind of tools

fgets(cmd, MAX, stdin);
execl (cmd, NULL);

Pattern (syntax) matching
  Will say “always dangerous”.
Semantic analysis
  Sometimes definitely no.
  Sometimes definitely yes.
Different kind of tools

```c
std::string cmd = makecmd();
exec1 (cmd, NULL);
```

**Pattern (syntax) matching**
Will say “always dangerous”.

**Semantic analysis**
Sometimes definitely **no**.
Sometimes definitely **yes**.
Sometimes **undetermined**.
Source Code Analysis Tools
How do they work

Identify the code to be analyzed.

– Scripts or build systems that build the executable.

The parser interprets the source code in the same way that a compiler does.
Source Code Analysis Tools
How do they work

Each invocation of the tool creates a model of the program:

– Abstract representations of the source
  • Control-flow graph
  • Call graph
  • Information about symbols (variables and type names)
Source Code Analysis Tools
How do they work

Symbolic execution on the model:

- Abstract values for variables.
- Explores paths.
- Based on abstract interpretation and model checking.
- The analysis is **path sensitive**.
  - The tool can tell the path for the flow to appear.
  - Points along that path where relevant transformations occur and conditions on the data values that must hold.
Source Code Analysis Tools
How do they work

The tool issue a set of warnings.

– List with priority levels.

The user goes through the warning list and labels each warning as:

– True positive.
– False Positive.
– Don’t care.
Source Code Analysis Tools
The Output

A tool grades weaknesses according things such as severity, potential for exploit, or certainty that they are vulnerabilities.

Problems:
  – False positives.
  – False negatives.
Source Code Analysis Tools
The Output

Ultimately people must analyze the tool’s report and the code then decide:

– Which reported items are not true weaknesses.
– Which items are acceptable risks and will not be mitigated.
– Which items to mitigate, and how to mitigate them.
Source Code Analysis Tool Limitations

No single tool can find every possible weaknesses:

– A weakness may result in a vulnerability in one environment but not in another.
– No algorithm can correctly decide in every case whether or not a piece of code has a property, such as a weakness.
– Practical analysis algorithms have limits because of performance, approximations, and intellectual investment.
– And new exploits are invented and new vulnerabilities discovered all the time!
Source Code Analysis Tools
What can they find

• Stylistic programming rules.
• Type discrepancies.
• Null-pointer dereferences.
• Buffer overflows.
• Race conditions.
• Resource leaks.
• SQL Injection.
Source Code Analysis Tools
What is difficult to find

• Authentication problems.
  – Ex: Use of non-robust passwords.

• Access control issues.
  – Ex: ACL that does not implement the principle of least privilege.

• Insecure use of cryptography.
  – Ex: Use of a weak key.
Source Code Analysis Tools
What is not possible to find

• Incorrect design.
• Code that incorrectly implements the design.
• Configuration issues, since they are not represented in the code.
• Complex weaknesses involving multiple software components.
Code Analysis Basics

Control flow analysis
- Analyze code structure and build a graph representation.
- Basics blocks and branch/call edges.
- Pointers are difficult.

Data flow analysis
- Usage, calculation, and setting of variables.
- Extract symbolic expressions.
- Arrays are annoying.
- Pointers are difficult.
Control Flow Analysis

Detects control flow dependencies among different instructions.

Control Flow Graph (CFG)

- Abstract representation of the source code.
- Each node represents a basic block.
- Call or jump targets start a basic block.
- Jumps end a basic block.
- Directed edges represent the control flow.
int Find(char *pat, char *buf,
        unsigned int plen,
        unsigned int blen) {

    int i, j;
    char *p;

    i = 0;

    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }

    return -1;
}
int Find(char *pat, char *buf,
unsigned int plen,
unsigned int blen) {

int i, j;
char *p;

i = 0;

while (i <= (blen - plen)) {
    p = &buf[i];
    j = 0;
    while (j < plen) {
        if (*p != pat[j]) break;
        p++;
        j++;
    }
    if (j >= plen) return i;
    i++;
}

return -1;
}
```c
int Find(char *pat, char *buf,
        unsigned int plen,
        unsigned int blen) {

    int i, j;
    char *p;

    i = 0;
    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }

    return -1;
}
```
int Find(char *pat, char *buf,
    unsigned int plen,
    unsigned int blen) {

    int i, j;
    char *p;

    i = 0;

    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }

    return -1;
}
int Find(char *pat, char *buf, unsigned int plen, unsigned int blen) {

    int i, j;
    char *p;

    i = 0;

    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }

    return -1;
}
int Find(char *pat, char *buf,
unsigned int plen,
unsigned int blen) {

    int i, j;
    char *p;

    i = 0;

    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }

    return -1;
}
int Find(char *pat, char *buf,
unsigned int plen,
unsigned int blen) {

int i, j;
char *p;

i = 0;
while (i <= (blen - plen)) {
p = &buf[i];
j = 0;
while (j < plen) {
  if (*p != pat[j]) break;
p++;
j++;
}
if (j >= plen) return i;
i++;
}
return -1;
}
int Find(char *pat, char *buf, unsigned int plen, unsigned int blen) {

    int i, j;
    char *p;

    i = 0;

    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }

    return -1;
}
int Find(char *pat, char *buf,
    unsigned int plen,
    unsigned int blen) {

    int i, j;
    char *p;

    i = 0;

    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
    }

    if (j >= plen) return i;

    return -1;
}
int Find(char *pat, char *buf, unsigned int plen, unsigned int blen) {

    int i, j;
    char *p;

    i = 0;
    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }
    return -1;
}
int Find(char *pat, char *buf, unsigned int plen, unsigned int blen) {

    int i, j;
    char *p;

    i = 0;

    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }

    return -1;
}
int Find(char *pat, char *buf, 
unsigned int plen, 
unsigned int blen) {

    int i, j;
    char *p;

    i = 0;
    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }

    return -1;
}
int Find(char *pat, char *buf,
unsigned int plen,
unsigned int blen) {

    int i, j;
    char *p;

    i = 0;
    while (i <= (blen - plen)) {
        p = &buf[i];
        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;
            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }
    return -1;
}
Data Flow Analysis

**Goal:** Is this code safe?

**Subgoal:**
Do we violate the borders of `buf` and `pat`?

- Simple dependences
- Flow insensitivity
- Loop carried dependences
- Pointers
- Aliasing
Data Flow Analysis

- Simple dependences
  - Back edges
  - Same node edges
- Loop carried dependences
- Need to understand the values for $i$ to know that references to $buf[i]$ are safe.
- Same for $j$ and $pat[j]$. 

```c
entry(pat, buf, plen, blen)

i = 0

if (i <= (blen - plen))
    p = &buf[i]; j = 0

if (j < plen)
    if (*p != pat[j])
        p++; j++

if (j >= plen)
    p = &buf[i]; j = 0

i++

return i

return -1
```
Data Flow Analysis

• Pointers
  • Similar to the data flow analysis on the previous slide.
  • Goal is to answer the question: where does \( p \) point? Are the references safe?
  • On what variables is \( p \)'s value based?
  • Of course, to calculate \( p \)'s value, we also have to know \( i \)'s value.
Data Flow Analysis

- Pointers
  - Similar to the data flow analysis on the previous slide.
  - Goal is to answer the question: where does $p$ point? Are the references safe?
  - On what variables is $p$’s value based?
  - Of course, to calculate $p$’s value, we also have to know $i$’s value.
Data Flow Analysis

- **Aliases**
  - Note that there are two completely different ways to name the same memory locations.
  - Understand these aliases can be important to understanding how memory is being referenced.

```c
entry(pat,buf,plen,blen)

i=0
if i <= (blen-plen)
p=&buf[i]; j=0
if (j < plen)
  if (*p != pat[j])
    p++; j++
i++
if (j >= plen)
p=&buf[i]; j=0
return i
```

```
if (j >= plen)
i++
return -1
```
int Find(char *pat, char *buf,
unsigned int plen,
unsigned int blen) {

    int i, j;
    char *p;

    i = 0;

    while (i <= (blen - plen)) {
        p = &buf[i];

        j = 0;
        while (j < plen) {
            if (*p != pat[j]) break;

            p++;
            j++;
        }
        if (j >= plen) return i;
        i++;
    }

    return -1;
}
Semantic Analysis

And this was a pretty simple example. It had no
- Pointers to functions
- Virtual functions
- Interprocedural analysis
- Context sensitivity

These make program analysis slower, less precise, or both.
Source Code Analysis Tools. What is expensive to find

It’s difficult for a tool to explore all the paths.

– Loops handled considering a small fixed number of iterations.
– Most tools ignore concurrency.
– Many tools ignore recursive calls.
– Many tools struggle with calls made through function pointers.
Common Weakness Enumeration (CWE)

• “CWE provides a unified, measurable set of software weaknesses”.
• “Allows a more effective use of software security tools”.
• 719 weaknesses in 244 categories.
• Id, description, consequences, examples, relationship, taxonomy mapping.

http://cwe.mitre.org/
1. What You Need to Know about How Tools Work

2. The Tools And Their Use
Roadmap

• Motivation
• Source code example
• Tools for Java applied to the source code
• The SWAMP
How to Describe a Weakness

Descriptive name of weakness (CWE XX)

An intuitive summary of the weakness.

– **Attack point:** How does the attacker affect the program.

– **Impact point:** Where in the program does the bad thing actually happen.

– **Mitigation:** A version of the program that does not contain the weakness.
CWE 601: Open Redirect

```java
class CWE601OpenRedirect {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter returnHTML = response.getWriter();
        returnHTML.println("<html><head><title>");
        returnHTML.println("Open Redirect");
        returnHTML.println("</title></head><body>");

        String data;
        data = "; // initialize data in case there are no cookies.
        // Read data from cookies.
        Cookie cookieSources[] = request.getCookies();
        if (cookieSources != null)
            // POTENTIAL FLAW: Read data from the first cookie value.
            data = cookieSources[0].getValue();
        if (data != null) {
            URI uri;
            uri = new URI(data);
            // POTENTIAL FLAW: redirect is sent verbatim.
            response.sendRedirect(data);
            return;
        }
    }
}
```
Open Redirect (CWE 601)

Web app redirects user to malicious site chosen by an attacker.

- **Attack Point:** Reading data from the first cookie using `getCookies()`.
- **Impact Point:** `SendRedirect()` uses user supplied data.
- **GoodSource:** Use a hard-coded string.

CWE601_Open_Redirect__Servlet_getCookies_Servlet_01.java

It’s a Servlet
FindBugs
FindBugs

• Open source tool available at findbugs.sourceforge.net/downloads.html
• Uses static analysis to look for bugs in Java code.
• Need to be used with the FindSecurityBugs plugin:
  – Available from find-sec-bugs.github.io
  – Version 1.6.0 (March 2017)
• Installation: Easy and fast.
FindBugs

1. Define `FINDBUGS_HOME` in the environment.

2. Install the Find Security Bugs plugin.

3. Learn the command line instructions and also use the graphical interface.

4. Command line interface:

   ```
   $FINDBUGS_HOME/bin/findbugs -textui -javahome $JAVA_HOME RelativePathTraversal.java
   ```

5. Graphic Interface: `java -jar $FINDBUGS_HOME/lib/findbugs.jar -gui`
FindBugs. Open Redirect

$FINDBUGS_HOME/bin/findbugs -textui -auxclasspath ./servlet-api.jar OpenRedirect.class

• 1 irrelevant warning.
• 1 true positive: It detects the Open Redirect vulnerability.
FindBugs. Open Redirect

Unvalidated Redirects occur when an application redirects a user to a destination URL specified by a user supplied parameter that is not validated. Such vulnerabilities can be used to facilitate phishing attacks.

Scenario
1. A user is tricked into visiting the malicious URL: http://website.com/loginRedirect=http://evil.website.com/fake/login
2. The user is redirected to a fake login page that looks like a site they trust: (http://evil.website.com/fake/login)
3. The user enters his credentials.
4. The evil site steals the user’s credentials and redirects him to the original website.

This attack is plausible because most users don’t double check the URL after the redirection. Also, redirection to an authentication page is very common.

Vulnerable Code

```java
if (cookieSources != null) {
    // POTENTIAL FLAW: Read data from the first cookie value */
    data = cookieSources[0].getValue();
}
if (data != null) {
    /* This prevents \n (and other chars) and should prevent incidentals such
     * as HTTP Response Splitting and HTTP Header Injection.
     */
    try {
        url = new URI(data);
    }
    catch (URISyntaxException exceptURISyntax) {
        response.getWriter().write("Invalid redirect URL");
        return;
    }
    /* POTENTIAL FLAW: redirect is sent unencoded so it could contain substitution items
     * IMPORTANT: Comment the 2 following lines to see the good case working!
     */
    response.sendRedirect(data);
    return;
}
```
Roadmap

• What is the SWAMP?

• Using the SWAMP
  – Register
  – Create a project
  – Upload your software package
  – Run your assessment
  – View the results

Getting Started with the SWAMP

• **Software Assurance Market Place.**

• **Objective:** Automate and simplify the use of (multiple) tools.

• A national, no-cost resource for software assurance (SwA) technologies used across research institutions, non-governmental organizations, and civilian agencies and their communities as both a research platform and a core component of the software development life cycle.
Core SWAMP Functionality

Upload Package Source Code and Build Description

Download SCARF Results

View Results

SWAMP (Build & SCA Testing)

Tools

Platforms

SWAMP Result Viewers

Code Dx
ThreadFix
Native Viewer

Supports Task Send Application to SWAMP for build and assess Send results
# SWAMP Tools and Platforms

## Tools

<table>
<thead>
<tr>
<th>C/C++</th>
<th>Python</th>
<th>Ruby</th>
<th>JavaScript</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cppcheck</td>
<td>Bandit</td>
<td>Brakeman</td>
<td>ESLint</td>
<td>Debian</td>
</tr>
<tr>
<td>Clang Static Analyzer</td>
<td>Flake8</td>
<td>Dawnscanner</td>
<td>Flow</td>
<td>Ubuntu</td>
</tr>
<tr>
<td>Gcc Warnings</td>
<td>Pylint</td>
<td>Reek</td>
<td>JSHint</td>
<td>CentOS</td>
</tr>
<tr>
<td>Parasoft C/C++Test</td>
<td></td>
<td>Rubocop</td>
<td>Retire.js</td>
<td>Scientific</td>
</tr>
<tr>
<td>GrammaTech CodeSonar</td>
<td></td>
<td>Ruby-lint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synopsys Coverity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FindBugs with FindSecurityBugs and fb-contrib plug-ins</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Prone</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checkstyle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWASP Dependency-Check</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parasoft Jtest</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android Lint</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RevealDroid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Platforms

<table>
<thead>
<tr>
<th>Debian</th>
<th>Ubuntu</th>
<th>CentOS</th>
<th>Scientific</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Commercial Tool (requires license for SiB)
Register to use the SWAMP
What can I do in the SWAMP?
Create a Project

Projects are used to share assessment results with other SWAMP users. You can invite other users to join a project and then all members of the project can add assessments to that project and view assessment results belonging to that project.

Projects I Own

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>Date Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tutorial Java</td>
<td>Tutorial Java</td>
<td>11/13/2014 15:59</td>
</tr>
<tr>
<td>Tools tutorial</td>
<td>Tools tutorial</td>
<td>10/09/2014 16:33</td>
</tr>
</tbody>
</table>

Projects I Joined

No projects.
Create a Project

Add New Project

Please enter the details of your new project below.

Full name *: MyProject
Short name *: Pro
Description *:
This is an example.

*Fields are required

Save Project
Packages

Packages are collections of files containing code to be assessed along with information about how to build the software package, if necessary. Packages may be written in a variety of programming languages and may have multiple versions.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
<th>Type</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer overflow</td>
<td></td>
<td>C/C++</td>
<td>1.0</td>
</tr>
<tr>
<td>command injection</td>
<td></td>
<td>C/C++</td>
<td>1.0</td>
</tr>
<tr>
<td>hard coded password</td>
<td></td>
<td>C/C++</td>
<td>1.0</td>
</tr>
<tr>
<td>info exposure</td>
<td></td>
<td>C/C++</td>
<td>1.0</td>
</tr>
</tbody>
</table>

[Add New Package]
Upload your Software Package
Upload your Software Package
Upload your Software Package
Upload your Software Package
Upload your Software Package
Run your Assessments
Run your Assessments

Run New Assessment

To create a new assessment, please specify the following information:

- **Package**
  - Select a package to assess:
    - OpenRedirect2017
  - Select a version:
    - Latest

- **Tool**
  - Select a tool to use:
    - Findbugs
  - Select a version:
    - Latest

[Save and Run]
My Assessments

results of a single assessment run or you may view the output of several runs of a package using different tools in order to compare the results.

<table>
<thead>
<tr>
<th>Package</th>
<th>Tool</th>
<th>Platform</th>
<th>Date</th>
<th>Status</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenRedirect2017 1.0</td>
<td>Parasoft Jtest 9.6.0</td>
<td>Ubuntu Linux 16.04 LTS 64-bit Xenial Xerus</td>
<td>05/02/2017 22:00</td>
<td>finished</td>
<td>1</td>
</tr>
<tr>
<td>OpenRedirect2017 1.0</td>
<td>Findbugs 3.0.1</td>
<td>Ubuntu Linux 16.04 LTS 64-bit Xenial Xerus</td>
<td>05/02/2017 21:08</td>
<td>finished</td>
<td>3</td>
</tr>
</tbody>
</table>
View your Results. FindBugs - Native

findbugs v3.0.1 Report

Summary

<table>
<thead>
<tr>
<th>Total</th>
<th>Priority 1</th>
<th>Priority 2</th>
<th>Priority 3</th>
<th>Priority 4</th>
<th>Priority 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Priority Group</th>
<th>File</th>
<th>Line</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STYLE OpenRedirect.java</td>
<td>41</td>
<td>Dead store to uri in OpenRedirect.doGet(HttpServletRequest, HttpServletResponse)</td>
</tr>
<tr>
<td>2</td>
<td>STYLE OpenRedirect.java</td>
<td>59</td>
<td>Redundant nullcheck of data, which is known to be non-null in OpenRedirect.doGet(HttpServletRequest, HttpServletResponse)</td>
</tr>
<tr>
<td>1</td>
<td>SECURITY OpenRedirect.java</td>
<td>50</td>
<td>The following redirection could be used by an attacker to redirect users to a fishing website.</td>
</tr>
</tbody>
</table>
View your Results. FindBugs - CodeDX
View your Results. FindBugs - CodeDX

OpenRedirect2017 > Analysis Run 2 > Weakness 34  Non-validated redirect detected by FindBugs [UNVALIDATED_REDIRECT]
First seen on 5/2/2017  1 weakness in this file  1 similar weakness in this analysis run  Medium severity
CWE 601 - URL Redirection to Untrusted Site (Open Redirect) [CWEVis | MITRE]

Status
New

Activity Stream
Post  Clear

Status set to New during Analysis Run 2 by admin
77 minutes ago

Description
The following redirection could be used by an attacker to redirect users to a phishing website.

Unvalidated requests are a vulnerability that facilitate phishing attacks.

Scenario
- A user is requested to visit the malicious url:
- The user is redirected to a fake login: (evil.vwebsite.com/fake/login)
- The user enters his credentials.
- He is redirected to the original website.

This attack is plausible because most users don’t double check the URL after redirection. In this example, redirection after authentication is common.

Counter measures
- White list URLs (if possible).
- Validate that the beginning of the URL is part of white list.
View your Results. FindBugs - CodeDX
View your Results. FindBugs - CodeDX

OpenRedirect2017 > Analysis Run 2 > Weakness 34: Non-validated redirect detected by FindBugs [UNVALIDATED_REDIRECT]
First seen on 5/2/2017, 1 weakness in this file, 1 similar weakness in this analysis run, Medium severity.
CWE 601 - URL Redirection to Untrusted Site ('Open Redirect') [CWEVis | MITRE]

The weakness occurs in 10-b-openredirect.zip/10-b-OpenRedirect
/OpenRedirect.java on line 50

```java
25  data = ""; /* initialize data in case there are no cookies */
26  /* Read data from cookies */
27  Cookie cookieSources[] = request.getCookies();
28  if (cookieSources != null) {
29    /* POTENTIAL FLAW: Read data from the first cookie value */
30    data = cookieSources[0].getValue();
31  }
32
33  if (data != null)
34    {
35    /* This prevents \n (and other chars) and should prevent incidentals such
     * as HTTP Response Splitting and HTTP Header Injection.
     */
36    URI uri;
37    try
38      {
39      uri = new URI(data);
40    }
41    catch (URISyntaxException exceptURISyntax)
42      {
43      response.getWriter().write("Invalid redirect URL");
44      return;
45    }
46  /* POTENTIAL FLAW: redirect is sent verbatim; escape the string to prevent ancillary issues like XSS, Response splitting etc */
47  /* CRITICAL: Comment the following lines to see the good case working! */
48  response.sendRedirect(data);
49  }
```

Status
New

Activity Stream
Write comments with Markdown

Status set to New during Analysis Run 2 by admin
21 minutes ago
### ps-jtest v9.6.0 Report

#### Summary

<table>
<thead>
<tr>
<th>Group</th>
<th>File</th>
<th>Line</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURITY.IBA</td>
<td>OpenRedirect.java</td>
<td>27</td>
<td>'getCookies()' is a tainted data-returning method and should be encapsulated by a validation</td>
</tr>
<tr>
<td>SECURITY.IBA</td>
<td>OpenRedirect.java</td>
<td>30</td>
<td>'getValue()' is a dangerous data-returning method and should be encapsulated by a validation</td>
</tr>
<tr>
<td>BD.SECURITY</td>
<td>OpenRedirect.java</td>
<td>41</td>
<td>Injection of data received from servlet request (&quot;data&quot;) to method accepting network resource properties</td>
</tr>
<tr>
<td>BD.SECURITY</td>
<td>OpenRedirect.java</td>
<td>50</td>
<td>Injection of data received from servlet request (&quot;data&quot;) to http response</td>
</tr>
<tr>
<td>BD.PB</td>
<td>OpenRedirect.java</td>
<td>59</td>
<td>Condition &quot;data != null&quot; always evaluates to true</td>
</tr>
</tbody>
</table>
View your Results. JTest - CodeDX
View your Results. JTest - CodeDX

OpenRedirect2017 > Analysis Run 3 > Weakness 38

Protect against HTTP response splitting detected by JTest [BD_SECURITY.TDRESP]
First seen on 5/2/2017  5 weaknesses in this file  1 similar weakness in this analysis run  High severity
CWE 79 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') [CWEVis | MITRE 7.2]

Status
New

Activity Stream
Write comments with Markdown
Status set to New during Analysis Run 3 by admin
4 minutes ago

Description
Injection of data received from servlet request ("data") to http response

This rule detects cases of probable HTTP response splitting vulnerabilities. This rule triggers when tainted data is passed to the following methods:
java.servlet.http.HttpServletResponse
* void sendRedirect(...) methods
* void addCookie(...) methods
* void addIntHeader(...) methods
* void addDateHeader(...) methods
* void setHeader(...) methods

View your Results. JTest - CodeDX
Questions?

Elisa Heymann
Elisa.Heymann@uab.es
https://continuousassurance.org

Barton P. Miller
bart@cs.wisc.edu
http://www.cs.wisc.edu/mist/