Automated Assessment Tools

Barton P. Miller
Computer Sciences Department
University of Wisconsin
bart@cs.wisc.edu

Elisa Heymann
Computer Sciences Department
University of Wisconsin
&
Universitat Autònoma de Barcelona
elisa@cs.wisc.edu
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
You can get 2 out of 3

Source Code Analysis Tools

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);
exec1 (cmd, NULL);

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

Semantic analysis
Sometimes definitely no.
Sometimes definitely yes.
Different kind of tools

```c
cmd = makecmd();
execl (cmd, NULL);
```

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

Semantic analysis
Sometimes definitely no.
Sometimes definitely yes.
Sometimes undeetermined.
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

Control Flow Analysis
Detected 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.
void foo() {
    char buf[MAX] = "example";
    int i, j, k;
    char a, b;
    char *p = buf;

    i = 0;
    if (c)
        j = i;
    else
        j = MAX;
    a = buf[i];
    b = buf[j];
    k = 0;
    while (k < MAX) {
        if (buf[k] == 'x')
            print(k);
        if (*p == 'z')
            print(p);
        p++;
        k++;
    }
}
void foo() {
    char buf[MAX] = "example";
    int i, j, k;
    char a, b;
    char *p = buf;

    i = 0;
    if (c)
        j = i;
    else
        j = MAX;
    a = buf[i];
    b = buf[j];
    k = 0;
    while (k < MAX) {
        if (buf[k] == 'x')
            print(k);
        if (*p == 'z')
            print(p);
        p++;
        k++;
    }
}

void foo() {
    char buf[MAX] = "example";
    int i, j, k;
    char a, b;
    char *p = buf;

    i = 0;
    if (c)
        j = i;
    else
        j = MAX;
    a = buf[i];
    b = buf[j];
    k = 0;
    while (k < MAX) {
        if (buf[k] == 'x')
            print(k);
        if (*p == 'z')
            print(p);
        p++;
        k++;
    }
}
void foo() {
    char buf[MAX] = "example";
    int i, j, k;
    char a, b;
    char *p = buf;

    i = 0;
    if (c)
        j = i;
    else
        j = MAX;

    a = buf[i];
    b = buf[j];
    k = 0;

    while (k < MAX) {
        if (buf[k] == 'x')
            print(k);
        if (*p == 'z')
            print(p);
        p++;
        k++;
    }
}
void foo() {
    char buf[MAX] = "example";
    int i, j, k;
    char a, b;
    char *p = buf;

    i = 0;
    if (c)
        j = i;
    else
        j = MAX;
    a = buf[i];
    b = buf[j];
    k = 0;
    while (k < MAX) {
        if (buf[k] == 'x')
            print(k);
        if (*p == 'z')
            print(p);
        p++;
        k++;
    }
}
void foo() {
    char buf[MAX] = "example";
    int i, j, k;
    char a, b;
    char *p = buf;

    i = 0;
    if (c)
        j = i;
    else
        j = MAX;
    a = buf[i];
    b = buf[j];
    k = 0;
    while (k < MAX) {
        if (buf[k] == 'x')
            print(k);
        if (*p == 'z')
            print(p);
        p++; k++;
    }
}
Data Flow Analysis

**Goal:** Is this code safe?

**Subgoal:**

Do we ever violate the borders of **buff**?

- Simple dependence
- Flow insensitive
- Loops
- Pointers
- Aliasing
Data Flow Analysis

- Simple dependence

```c
p = buf
i = 0
if (c)
    j = 1

a = buf[i]
b = buf[j]
k = 0

if (buf[k] == 'x')
    if (k < MAX)
        print(k)
    end block

if (*p == 'z')
p++; k++;
end block
```
Data Flow Analysis

• Flow insensitive

```
p = buf
i = 0
if (c)
j = 1

k = 0
a = buf[i]
b = buf[j]

if (buf[k] == 'x')
if (k < MAX)
print(k)
p++; k++;
end block

if (*p == 'z')
print(p)
```

T F
T F
F
Data Flow Analysis

• Loops

```
p = buf
i = 0
if (c)
j = 1
a = buf[i]
b = buf[j]
k = 0
if (buf[k] == 'x')
if (k < MAX)
  print(k)
  if (*p == 'z')
    p++; k++;
end block
```
Data Flow Analysis

• Loops

```c
p=buf
i=0
if (c)
    j=1
    a=buf[i]
b=buf[j]
k=0
    if (buf[k]=='x')
        if (k<MAX)
            print(k)
        else
            if (*p=='z')
                p++; k++;
            end block
```

Data Flow Analysis

• Pointers

\[
p = \text{buf} \\
i = 0 \\
\text{if (c)} \\
j = 1 \\
a = \text{buf}[i] \\
b = \text{buf}[j] \\
k = 0 \\
\text{if (buf[k] == 'x')} \\
\text{if (k < MAX)} \\
\text{print(k)} \\
\text{if (*p == 'z')} \\
p++; k++; \\
\text{end block}
\]
Data Flow Analysis

• Pointers

```plaintext
p = buf
i = 0
if (c)
j = 1
a = buf[i]
b = buf[j]
k = 0
if (buf[k] == 'x')
if (k < MAX)
    print(k)
end block
if (*p == 'z')
p++; k++;
end block
```
Data Flow Analysis

- Aliasing

Are these the same?
Semantic Analysis

But it’s harder than it looks:

– Pointers to functions
– Virtual functions
– Interprocedural analysis
– Context sensitivity

These make program analysis slow, imprecise, 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/
Common Weakness Scoring System (CWSS)

• It “provides a mechanism for prioritizing software weaknesses in a consistent, flexible, open manner”.

• Based on three metric groups:
  – Base finding metric group.
  – Attack surface metric group.
  – Environmental metric group.
Background on Automated Assessment Tools

Elisa Heymann  
Elisa.Heymann@uab.es

Barton P. Miller  
bart@cs.wisc.edu

http://www.cs.wisc.edu/mist/
Questions?

http://www.cs.wisc.edu/mist
1. What You Need to Know about How Tools Work

2. The Tools And Their Use
Roadmap

• Motivation
• Source code examples
• Tools for C/C++ applied to the source code
• Tools for Java applied to the source code
• The SWAMP
What and Why

• Learn about different automated tools for vulnerability assessment.
• Start with small programs with weaknesses.
• Apply different tools to the programs.
• Understand the output, and strong and weak points of using specific tools.
CWE 78: OS Command Injection

- void
  CWE78_OS_Command_Injection__char_console_execl_41_bad()
  {
    char * data; char dataBuffer[100] = "";
    data = dataBuffer;
    /* Read input from the console */
    size_t dataLen = strlen(data);
    /* If there is room in data, read into it from the cons */
    if (100-dataLen > 1) {
      /* POTENTIAL FLAW: Read data from the console */
      
      /* POTENTIAL FLAW: Read data from the console */
    }
  }
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.

(CWEXX_Long_Detailed_File_Name_Containing_The_Code_yy.cpp)
OS Command Injection (CWE 78)

User supplied data is used to create a string that will be interpreted by a command shell.

– **Attack Point**: Input read from the console.
– **Impact Point**: Executing command with `system()`.
– **Mitigation**: Don’t execute user provided input; instead use a fixed string.

CWE78_OS_Command_Injection__char_console_execl_41.c
(Highly modified to compensate for errors.)
Tools for C/C++

• Goanna (RedLizards)
• Coverity analyze
Goanna (RedLizards)
Goanna

• Commercial tool by Red Lizard Software available at redlizards.com
• The Goanna suite of static analysis tools pinpoints defects and vulnerabilities in C/C++ programs.
  – Access violations
  – Memory leaks
  – Array and string overruns
  – Division by zero
  – Unspecified, non-portable, and/or dangerous constructs
  – Security vulnerabilities
Goanna

1. Download Goanna Central
2. Activate the license and install the software
   ./install-goanna
3. Include in PATH the location of goanna/bin.
4. Initialize goanna for the project
   goanna-init
3. Enable the security package:
   goanna-package --enable-pkg security
4. Goanna Dashboard is the web interface to
   navigate and interact with analysis results.
   $goreporter start-server &
Goanna

Three-step process:

– Run a full build of your program using the Goanna build integration utility to capture settings of the build.
  
  `$goanna-trace make`

– Use this information from full build to run analysis.
  
  `$goanna-analyze`

– Produce an analysis report
  
  `$goanna-report`

– Read and interact with the analysis results.
  
  • After `goreporter` is running, load the provided URL in a web browser.
Goanna. OS Command Injection

$ goanna-trace make
$ goanna-analyze
$ goanna-report

• 0 false positive.
• 0 false negative.
• 1 true positive: It detects the command injection.
Goanna. OS Command Injection
Goanna. OS Command Injection
Coverity Analyze
Coverity

• Commercial tool. Available at http://www.coverity.com/

• Starting Point: Accurate Compilation.

• Depth and Accuracy of Analysis
  – Interprocedural Dataflow Analysis.
  – False Path Pruning.
  – Design Pattern Intelligence.
  – Enterprise Framework Analyzer.
  – White Box Fuzzer.

• Scalable.
Coverity

1. Download the license and the software: https://coverity.secure.force.com/apex/LicenseManagement2

2. Run the installation script: cov-analysis-linux64-7.6.0.sh

3. Include in **PATH** the location of ~elisa/cov-analysis-linux64-7.6.0/bin

4. Command line and graphic interface.
Coverity

Steps:

– Generate a configuration for the compiler:
  cov-configure --gcc

– Build the intermediate representation of the source code:
  cov-build --dir <intermediate-dir> make

– cov-analyze --dir <intermediate-dir>

– Check the checkers included by cov-analize:
  cov-analyze --list-checkers

– Read and interact with the analysis results.

– Graphic mode: cov-wizard
Coverity. OS Command Injection

$ cov-build --dir cov-comm-injection make
$ cov-analyze --dir cov-comm-injection --security

• 1 defect found.
• 1 true positive: It detects the command injection.
Coverity. OS Command Injection
Coverity. OS Command Injection
Coverity. OS Command Injection
Coverity. OS Command Injection

Analysis options

Options...

Use 1 worker processes (instead of the maximum allowed)

Analysis results summary:

Analysis on 1/12/15 found 1 defect occurrences.

Run Analysis
Covérité. Injection d'injection de commande OS.

Covérité Console

> /afs/cs.wisc.edu/u/e/l/elisa/cov-analysis-linux64-7.6.0/bin/cov-analyze --dir /afs/cs/

Covérité Static Analysis version 7.6.0 on Linux 2.6.32-431.3.1.el6.x86_64 x86_64
Internal version numbers: 9b77a50df0 p-harmony-push-21098.563

Using 4 workers as limited by CPU(s)
Looking for translation units
[0--------25----------50----------75--------100]
******************************************************************************
[STATUS] Loading topological sort from disk (6 functions)
[0--------25----------50----------75--------100]
******************************************************************************
[STATUS] Computing node costs
[0--------25----------50----------75--------100]
******************************************************************************
[STATUS] Starting analysis run
[0--------25----------50----------75--------100]
******************************************************************************
[STATUS] Calculating 46 cross-reference bundles...
[0--------25----------50----------75--------100]
******************************************************************************
Analysis summary report:

Files analyzed : 1
Total LoC input to cov-analyze : 13485
Functions analyzed : 6
Paths analyzed : 25
Time taken by analysis : 00:00:01
Defect occurrences found : 1 TAIANTED_STRING

Analysis finished successfully.
Coverity. OS Command Injection
Coverity. OS Command Injection
Java
CWE 601: Open Redirect

public void doGet(HttpServletRequest request,
• HttpServletResponse response)
• throws ServletException,
IOException {
• response.setContentType("text/html");
• PrintWriter returnHTML =
response.getWriter();
•
returnHTML.println("<html><head><title>");
• PrintWriter returnHTML =
response.getWriter();
•
returnHTML.println(“Open Redirect”);
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
Tools for Java

- FindBugs
- Parasoft Jtest
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.
• 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**
  - `$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 Redirect

At OpenRedirect.java:[line 50]
In method OpenRedirect.doGet(HttpServletRequest, HttpServletResponse)

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:
2. The user is redirected to a fake login page that looks like a site they trust. (http://evil.vwebsite.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.
Parasoft Jtest
Jtest

- Automates a broad range of practices proven to improve development team productivity and software quality.
- Standalone Linux 9.5 version used.
  - gui mode and command line mode.
- Installation process: Slow download & easy installation.
Jtest

1. Include `/u/e/l/elisa/Jtest/9.5` in path.
2. Include the license.
3. Learn the command line instructions and also use the graphical interface.
Jtest

1. Command line interface: $jtestcli <options>

2. Graphic Interface: jtest&

3. Create a project and copy the .java files to the project/src directory.

4. Different tests available. We chose Security->CWE Top 25.
Create the OpenRedir project.
Include servlet-api.jar in the OpenRedir project.

cp OpenRedirect.java
~elisa/parasoft/workspace1/OpenRedir/src

• 4 issues detected:
  – getCookies() returns tainted data.
  – cookieSources[0].getValue() should be validated.
  – 2 Open Redirect detected.

• It detects the Open Redirect for both the good and bad cases.
Jtest. Open Redirect

```java
data = ""; /* initialize data in case there are no cookies */
/* Read data from cookies */
Cookie cookiesSources[] = request.getCookies();
if (cookiesSources != null) {
    /* POTENTIAL FLAW: Read data from the first cookie value */
    data = cookiesSources[0].getValue();
}
if (data != null) {
    /* This prevents \r\n (and other chars) and should prevent incidentals such
    * as HTTP Response Splitting and HTTP Header Injection.
    */
    URI uri;
    try {
        uri = new URI(data);
    }
}
```

0 errors, 2 warnings, 0 others

- SECURITY.IBA.VPPD: 'getCookies()' is a tainted data-returning method and should be encapsulated by a validation
- SECURITY.IBA.VPPD: 'getValue()' is a dangerous data-returning method and should be encapsulated by a validation
JTest. Open Redirect

```java
String 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) {
    /* This prevents \r\n (and other chars) and should prevent incidentals such as HTTP Response Splitting and HTTP Header Injection. */
    URI uri;
    try {
        uri = new URI(data);
    }
}
```
Jtest. Open Redirect

```java
/* This prevents \n (and other chars) and should prevent incidentals such as HTTP Response Splitting and HTTP Header Injection. */

URI uri;
try {
    uri = new URI(data);
} catch (URISyntaxException exceptURISyntax) {
    response.getWriter().write("Invalid redirect URL");
    return;
}

/* POTENTIAL FLAW: redirect is sent verbatim; escape the string to prevent a possible XSS. */
// 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
    • Java
    • C/C++

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.
Register to use the SWAMP
What can I do in the SWAMP?

- **Projects**: Create and manage projects to share assessment results with other SWAMP users.
Create a Project

Add New Project

Please enter the details of your new project below.

Full name *: Tutorial Java

Short name *: Tutorial Java

Description *: Tutorial Java

Save Project

*Fields are required
My Projects

Projects

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>Name</th>
<th>Description</th>
<th>Date Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tutorial Java</td>
<td>Tutorial Java</td>
<td>2014-11-13 14:59</td>
</tr>
<tr>
<td>Tools tutorial</td>
<td>Tools tutorial</td>
<td>2014-10-09 15:33</td>
</tr>
</tbody>
</table>

Projects I Joined

No projects.
Upload your Software Package

Add New Package

Name: J Open Redir
Description: Code containing an Open Redir weakness.
External URL:
File: 10-open-redirection.tar
Version: 1.0

Package Info
My software Packages

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.

Filters:

- any project
- any type
- any date
- all items

+ Add New Package

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Date Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>J Open Redir</td>
<td>Java Source Code</td>
<td>2015-01-07 18:26</td>
<td></td>
</tr>
<tr>
<td>Java Path Traversal</td>
<td>Java Path Traversal</td>
<td>2014-11-13 21:20</td>
<td></td>
</tr>
<tr>
<td>our example</td>
<td>C/C++</td>
<td>2014-10-09 20:51</td>
<td></td>
</tr>
</tbody>
</table>
Run your Assessments

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

**Package**
Select a package to assess:
- J Open Redir

Select a version:
- Latest

**Tool**
Select a tool to perform the assessment:
- Parasoft Jtest

Select a version:
- Latest

[Save and Run]
# My Assessments

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Status</th>
<th>Version</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>integer overflow</td>
<td>latest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>J Open Redirect</td>
<td>latest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Java Command Injectio</td>
<td>latest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Java Command Injectio</td>
<td>latest</td>
<td></td>
<td>RHEL6.4 64-bit</td>
</tr>
<tr>
<td>20</td>
<td>Java Command Injectio</td>
<td>latest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Java Path Traversal</td>
<td>latest</td>
<td></td>
<td>RHEL6.4 64-bit</td>
</tr>
<tr>
<td>22</td>
<td>Java Path Traversal</td>
<td>latest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Java Path Traversal</td>
<td>latest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>NIST Juliet CWE023_01</td>
<td>latest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>NIST Juliet CWE023_01</td>
<td>latest</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Click on the circled row to view details.
<table>
<thead>
<tr>
<th>Date / Time</th>
<th>Package</th>
<th>Tool</th>
<th>Platform</th>
<th>Status</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-03-12 17:44</td>
<td>J Open Redir 1.0</td>
<td>Parasoft Jtest 9.5.13</td>
<td>Red Hat Enterprise Linux 64-bit RHEL6.4 64-bit</td>
<td>Finished</td>
<td></td>
</tr>
<tr>
<td>2014-12-15 21:55</td>
<td>Java Path Traversal 1.0</td>
<td>Parasoft Jtest 9.5.13</td>
<td>Red Hat Enterprise Linux 64-bit RHEL6.4 64-bit</td>
<td>Finished</td>
<td></td>
</tr>
<tr>
<td>2014-12-15 21:55</td>
<td>Java Command Injection 1.0</td>
<td>Parasoft Jtest 9.5.13</td>
<td>Red Hat Enterprise Linux 64-bit RHEL6.4 64-bit</td>
<td>Finished</td>
<td></td>
</tr>
<tr>
<td>2014-12-15 19:52</td>
<td>hard coded password 1.0</td>
<td>Parasoft C/C++test 9.5.4.103</td>
<td>Red Hat Enterprise Linux 64-bit RHEL6.4 64-bit</td>
<td>Finished</td>
<td></td>
</tr>
<tr>
<td>2014-12-15 19:52</td>
<td>info exposure 1.0</td>
<td>Parasoft C/C++test 9.5.4.103</td>
<td>Red Hat Enterprise Linux 64-bit RHEL6.4 64-bit</td>
<td>Finished</td>
<td></td>
</tr>
<tr>
<td>2014-12-15 19:52</td>
<td>our example 1.0</td>
<td>Parasoft C/C++test 9.5.4.103</td>
<td>Red Hat Enterprise Linux 64-bit RHEL6.4 64-bit</td>
<td>Finished</td>
<td></td>
</tr>
<tr>
<td>2014-12-09 21:18</td>
<td>Java Command Injection 1.0</td>
<td>error-prone 1.1.1</td>
<td>Red Hat Enterprise Linux 64-bit RHEL6.4 64-bit</td>
<td>Finished</td>
<td></td>
</tr>
<tr>
<td>2014-12-09 21:18</td>
<td>Java Path Traversal 1.0</td>
<td>error-prone 1.1.1</td>
<td>Red Hat Enterprise Linux 64-bit RHEL6.4 64-bit</td>
<td>Finished</td>
<td></td>
</tr>
</tbody>
</table>
My Results for Java: Jtest – J Open Redir - Native

ps-jtest v9.5 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>/TempProject/10-b-OpenRedirect/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>/TempProject/10-b-OpenRedirect/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></td>
<td>/TempProject/10-b-OpenRedirect/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></td>
<td>/TempProject/10-b-OpenRedirect/OpenRedirect.java</td>
<td>50</td>
<td>Injection of data received from servlet request ('&quot;;data&quot;;') to http response</td>
</tr>
<tr>
<td></td>
<td>/TempProject/10-b-OpenRedirect/OpenRedirect.java</td>
<td>59</td>
<td>Condition '&quot;;data&quot; != null; always evaluates to true</td>
</tr>
</tbody>
</table>
My Results for Java: Jtest – J Open Redir - CodeDx
My Results for Java: Jtest – J Open Redir - CodeDx

J Open Redir > Analysis Run 11 > Weakness 48

BD.SECURITY.TDNET detected by ps-jtest with Unspecified severity
First seen on 3/12/2015 5 weaknesses in this file 1 similar weakness in this analysis run
No Common Weakness Enumeration information available

Status
Unresolved

Activity Stream
admin changed status to Unresolved
20 minutes ago
admin changed status to New
about an hour ago

Description
Injection of data received from servlet request ("data") to method accepting network resource properties

Source Code
The weakness occurs in 10-b-OpenRedirect/OpenRedirect.java on line 41

```java
1 import javax.servlet.*;
2 import javax.servlet.http.*;
3 import java.io.*;
4 import java.net.URI;
5 import java.net.URISyntaxException;
6 public class OpenRedirect extends HttpServlet {
7   public void doGet(HttpServletRequest request,
```
My Results for Java: Jtest – J Open Redir - CodeDx
CWE 601: Open Redirect

```java
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");
```

```
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
Questions?

Elisa Heymann
Elisa.Heymann@uab.es

Barton P. Miller
bart@cs.wisc.edu

http://www.cs.wisc.edu/mist/