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

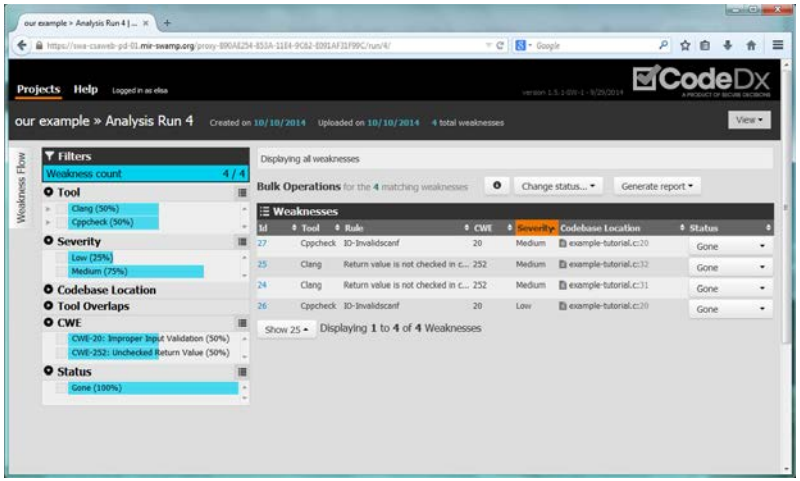
CTSC
CENTER FOR TRUSTWORTHY SCIENTIFIC CYBERINFRASTRUCTURE
The NSF Cybersecurity Center of Excellence

1. What You Need to Know about How Tools Work

2. The Tools And Their Use

Source Code Analysis Tools

```
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

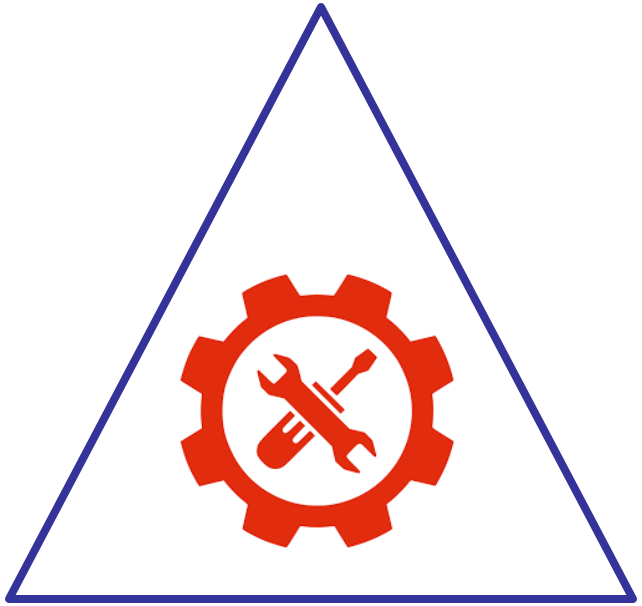
- Designed to analyze source code or binaries to help find security flaws.
- The source code may contain inadvertent or deliberate weaknesses that could lead to security vulnerabilities in the executable versions of the application program.
- Better to use them from the beginning of the software development 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

precision



speed

#checks

You can get 2 out of 3

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

```
cmd = “/bin/ls”;
execl (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

```
cmd=makecmd();
execl (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

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.

```
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++;
  }
}
```

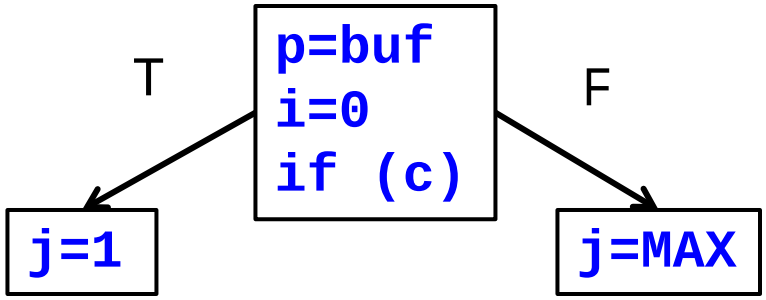
```
void foo() {
   char buf[MAX] = "example";
   int i, j, k;
   char a, b;
   char *p = buf;

   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++;
   }
}
```

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

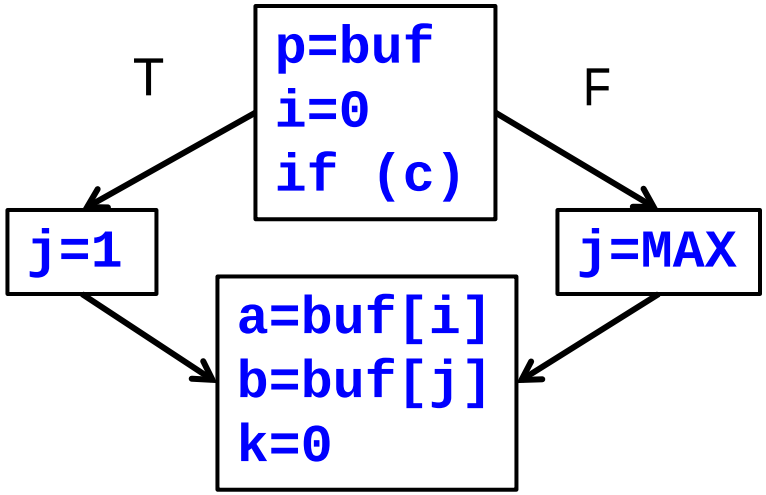
```
void foo() {
   char buf[MAX] = "example";
   int i, j, k;
   char a, b;
   char *p = buf;

   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++;
   }
```



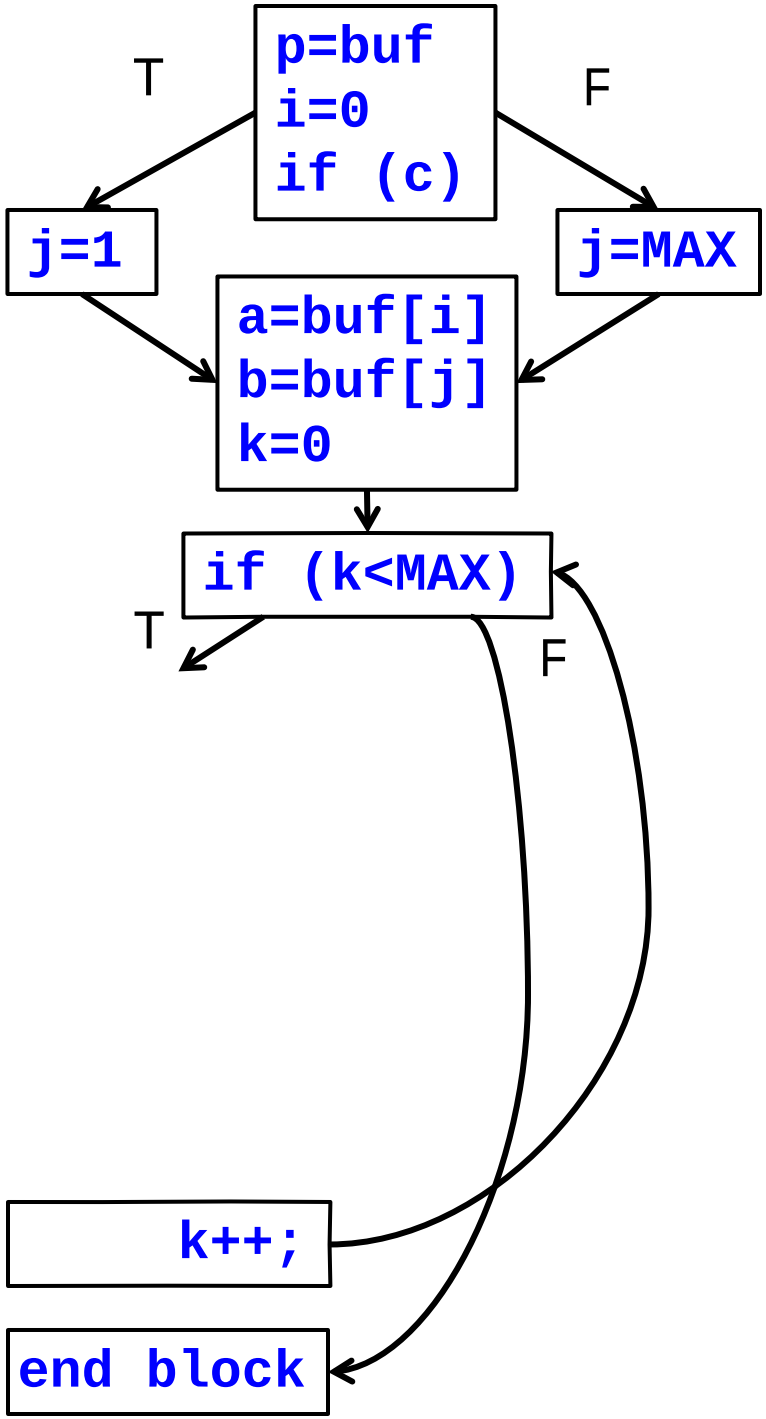
```
void foo() {
   char buf[MAX] = "example";
   int i, j, k;
   char a, b;
   char *p = buf;

   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++;
   }
```



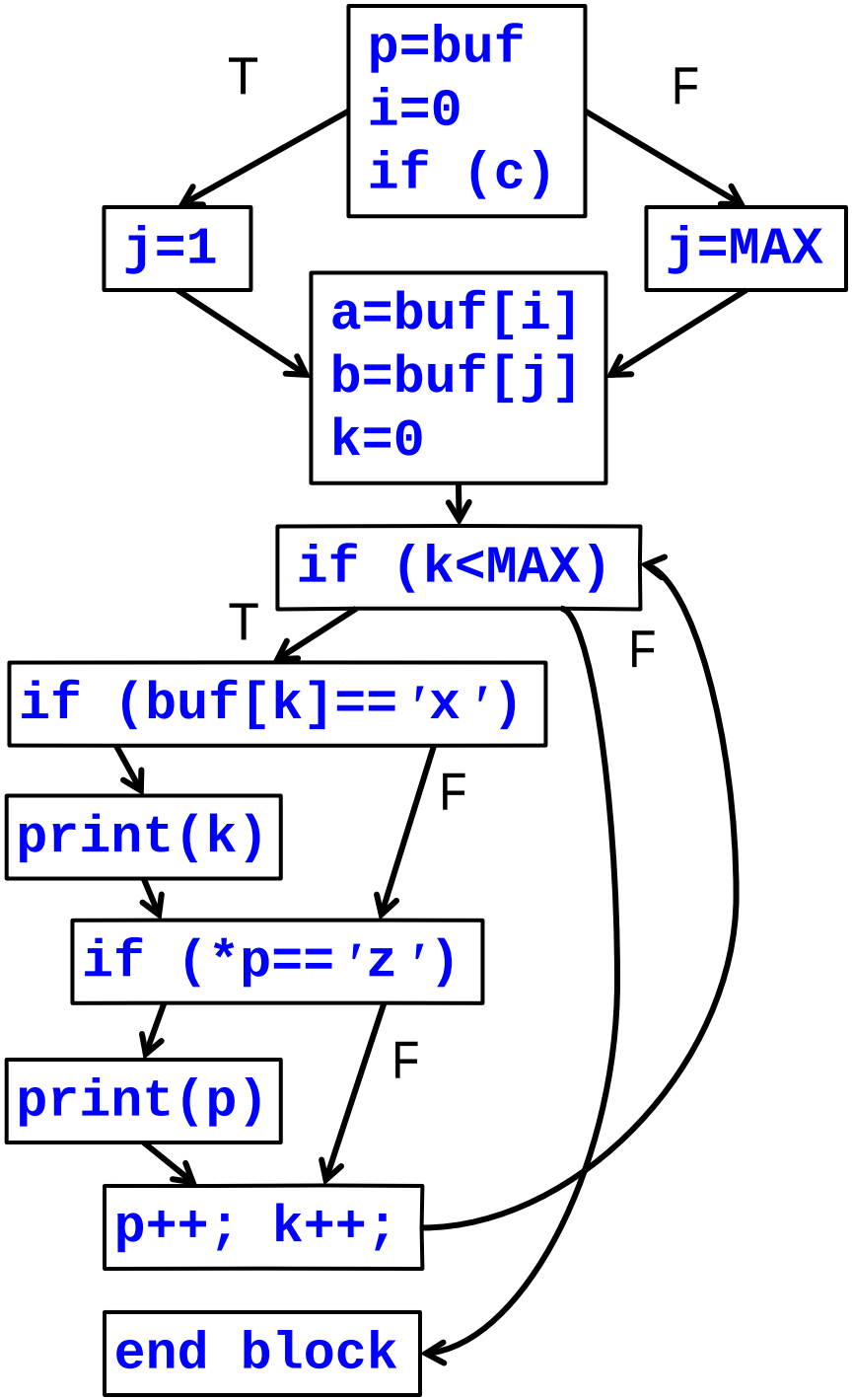
```
void foo() {
  char buf[MAX] = "example";
  int i, j, k;
  char a, b;
  char *p = buf;

  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?

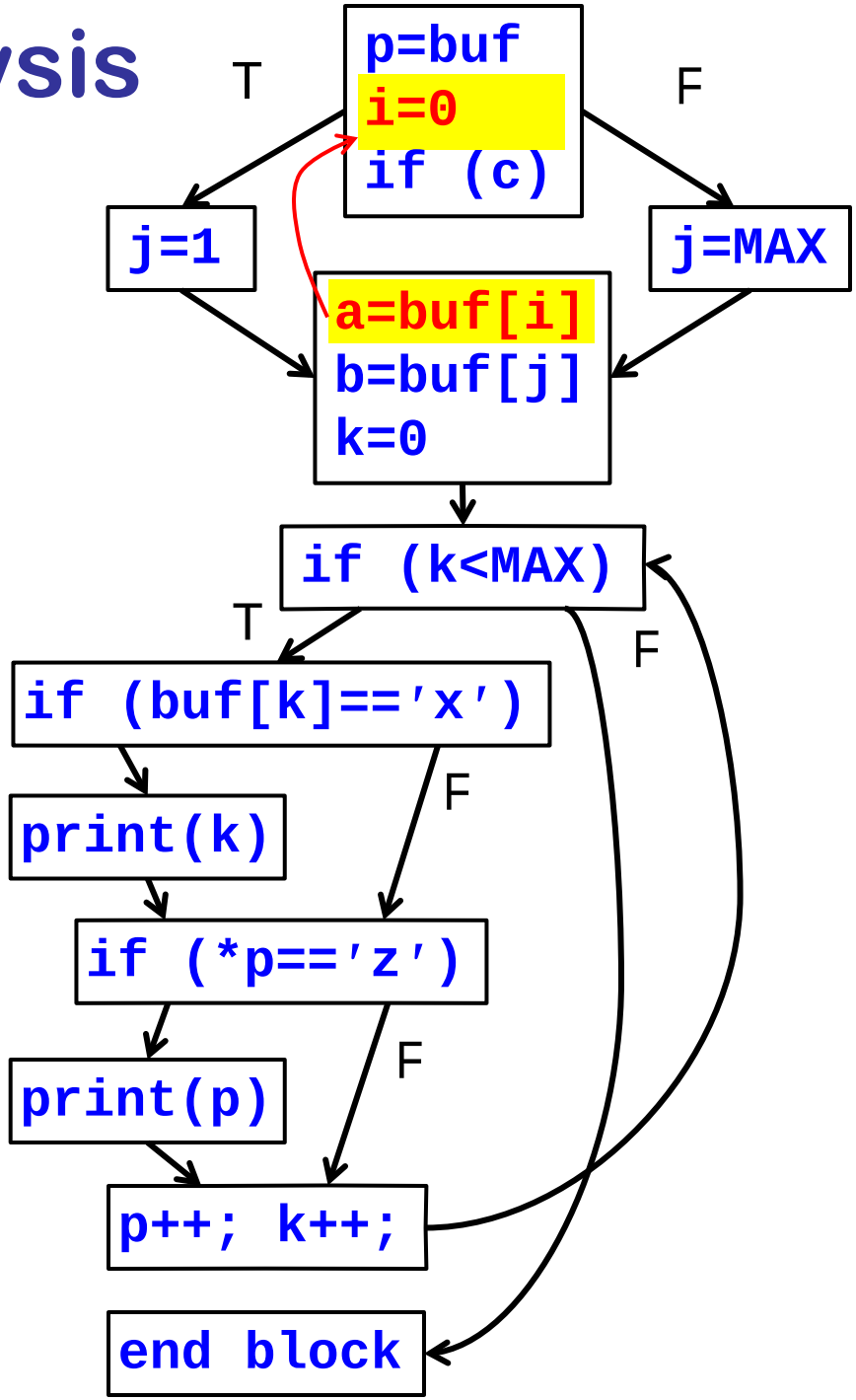
Subgoal:

Do we ever violate the borders of `buf`?

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

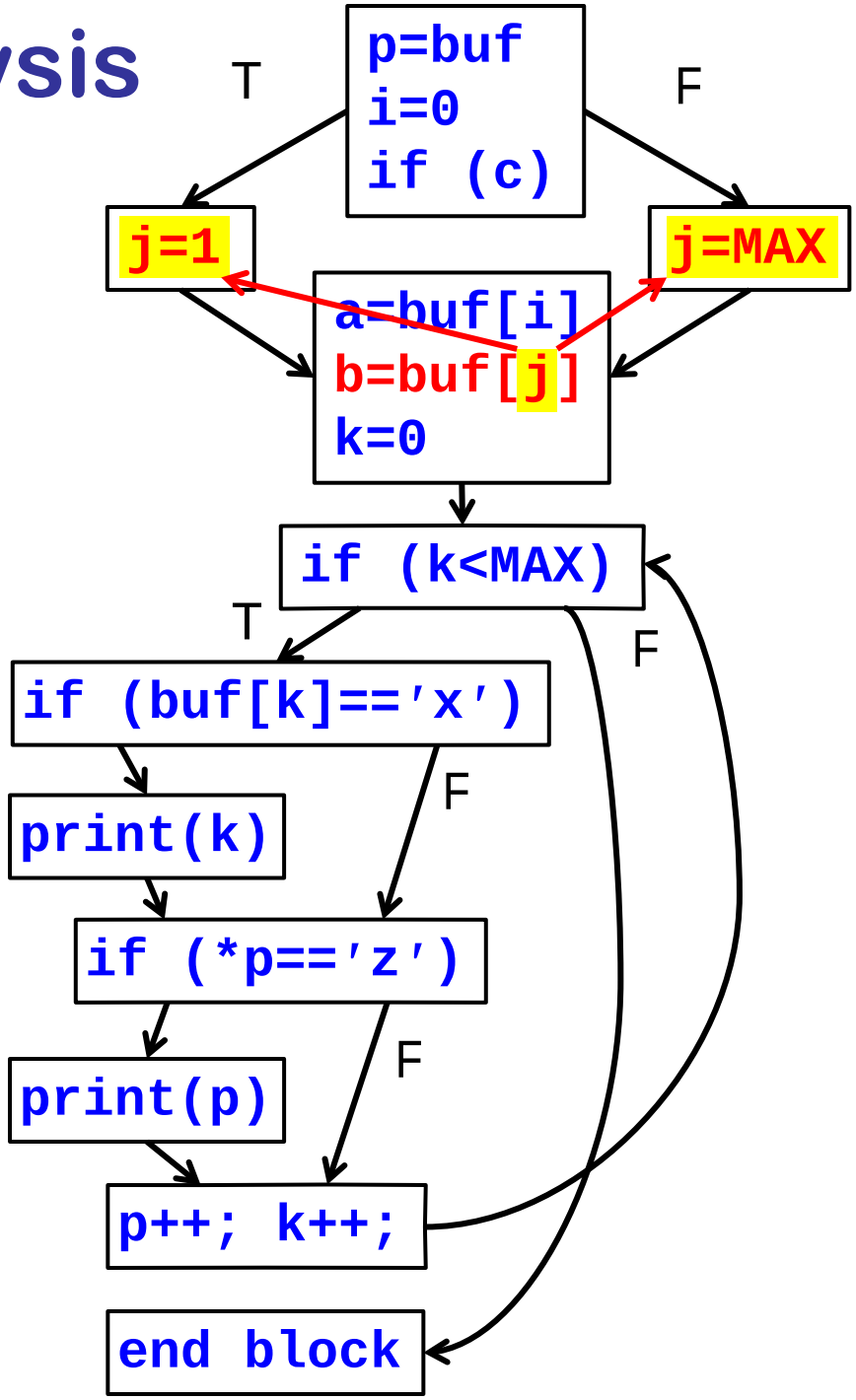
Data Flow Analysis

- **Simple dependence**



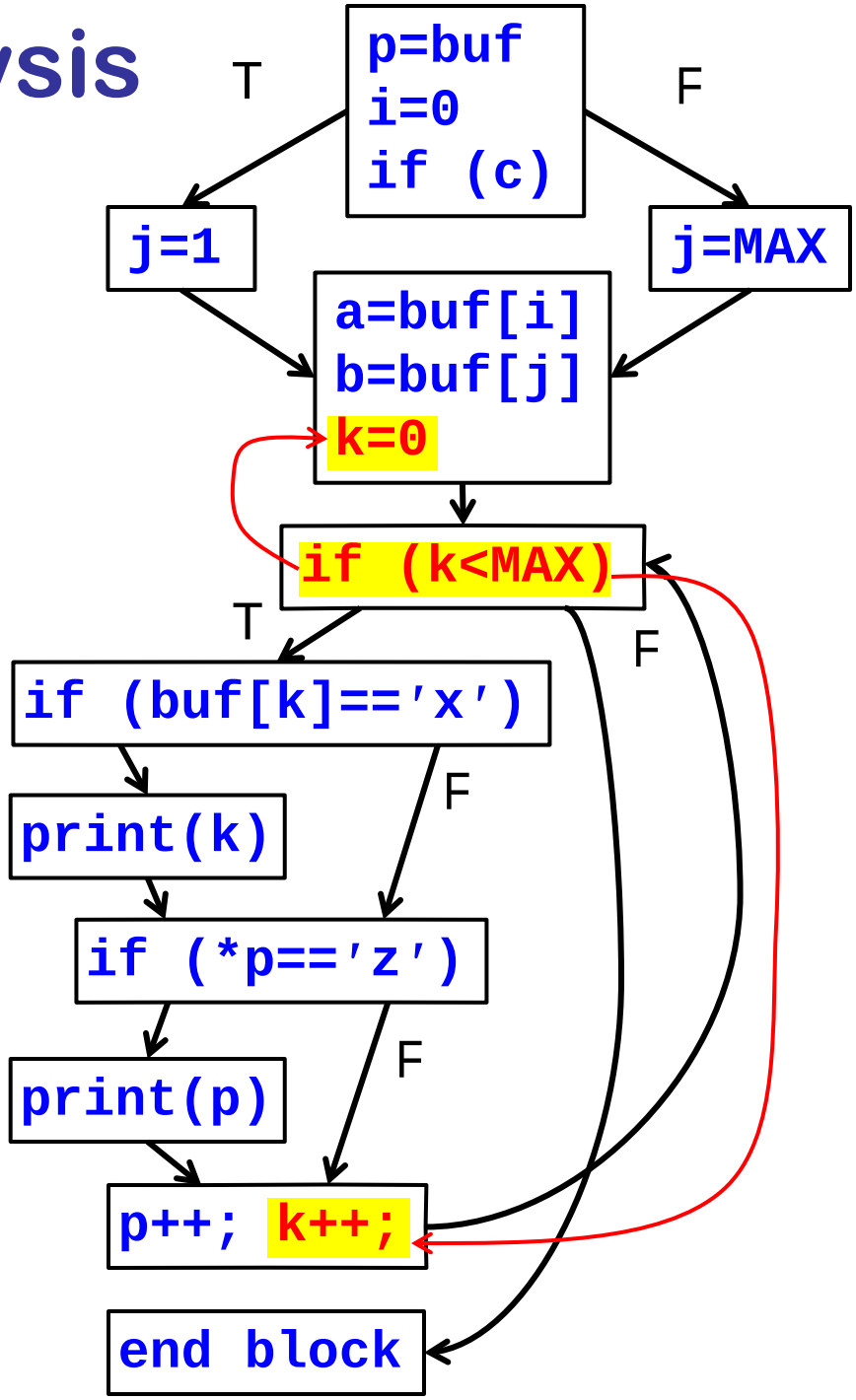
Data Flow Analysis

- Flow insensitive



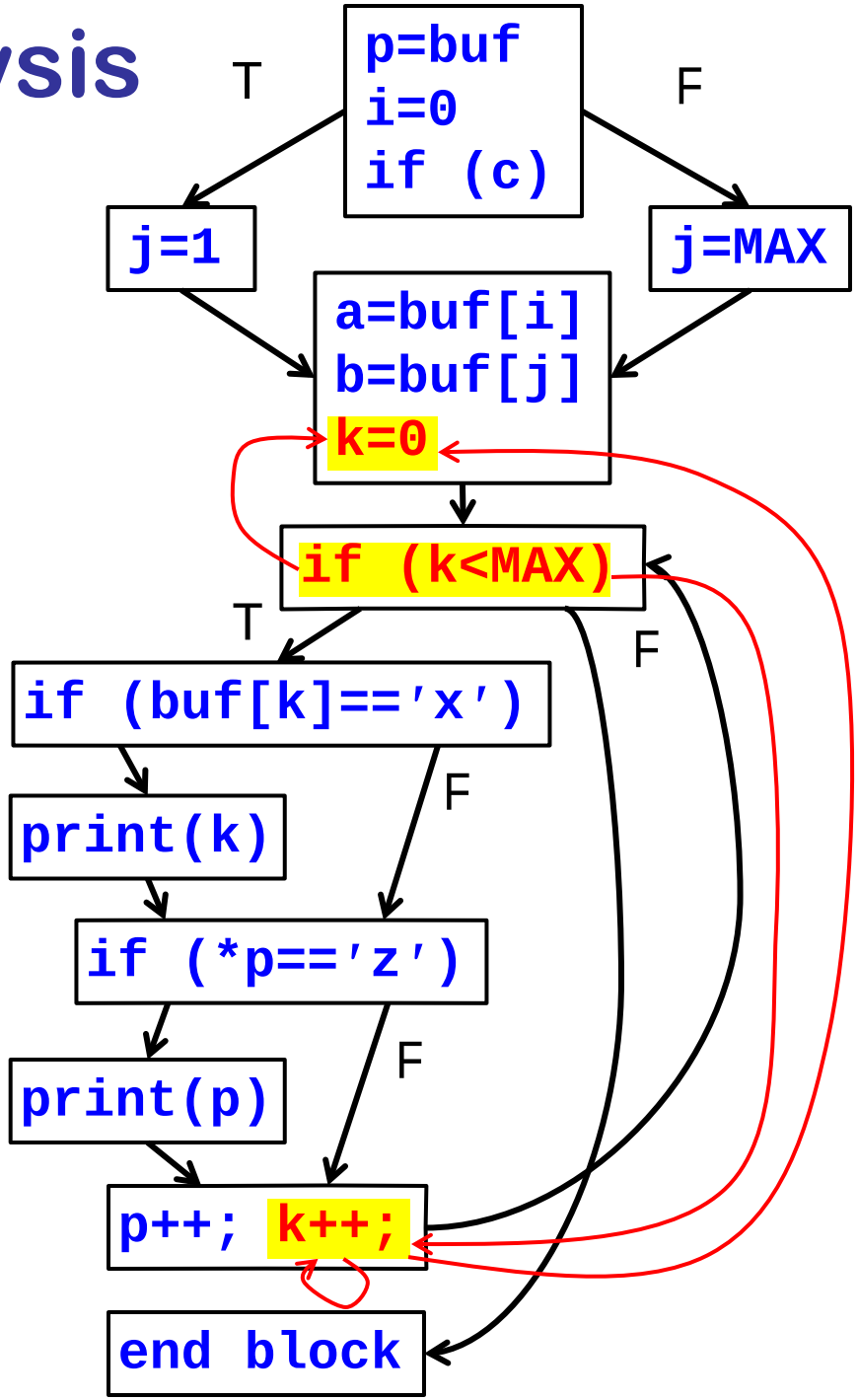
Data Flow Analysis

- Loops



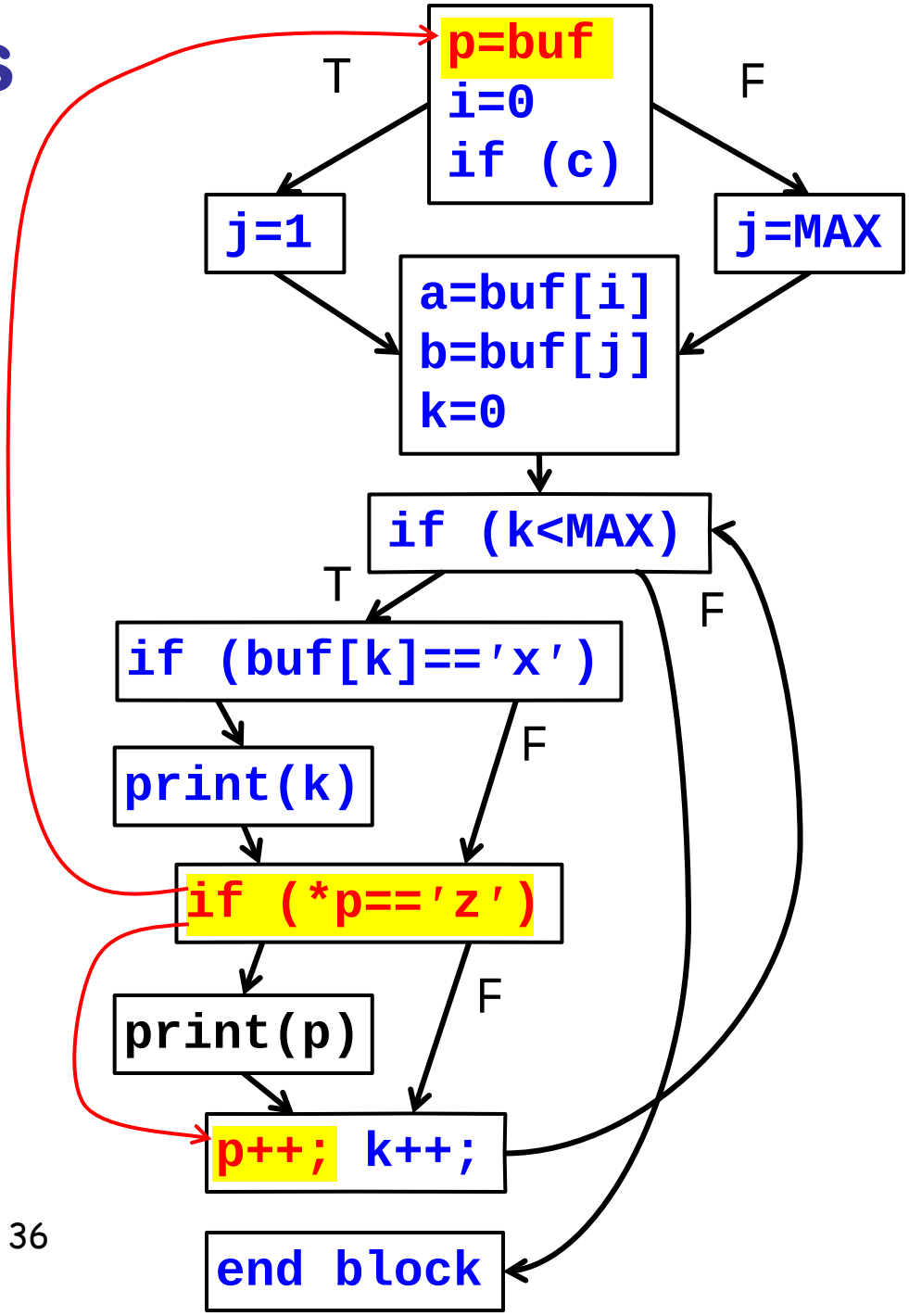
Data Flow Analysis

- Loops



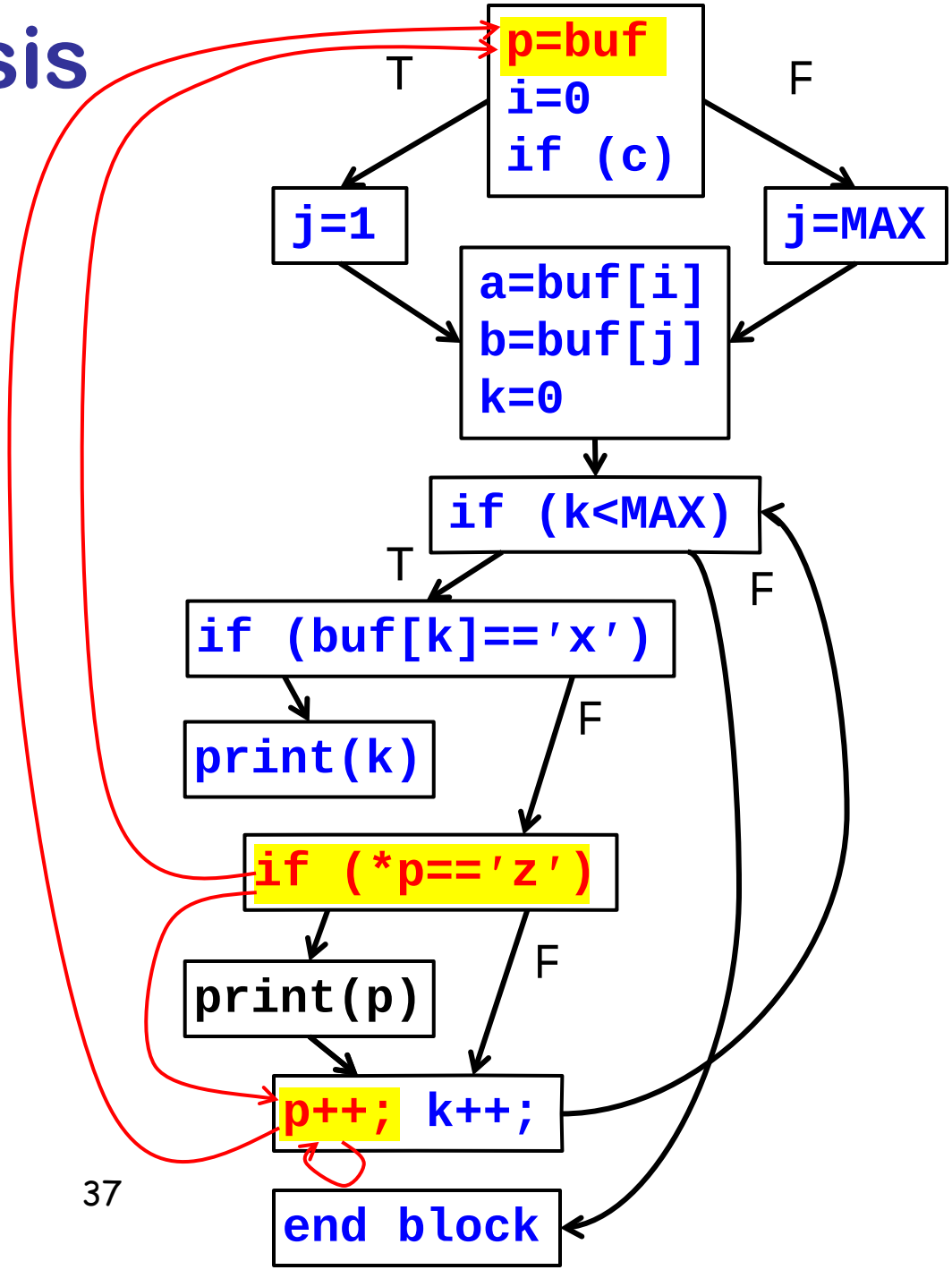
Data Flow Analysis

- **Pointers**



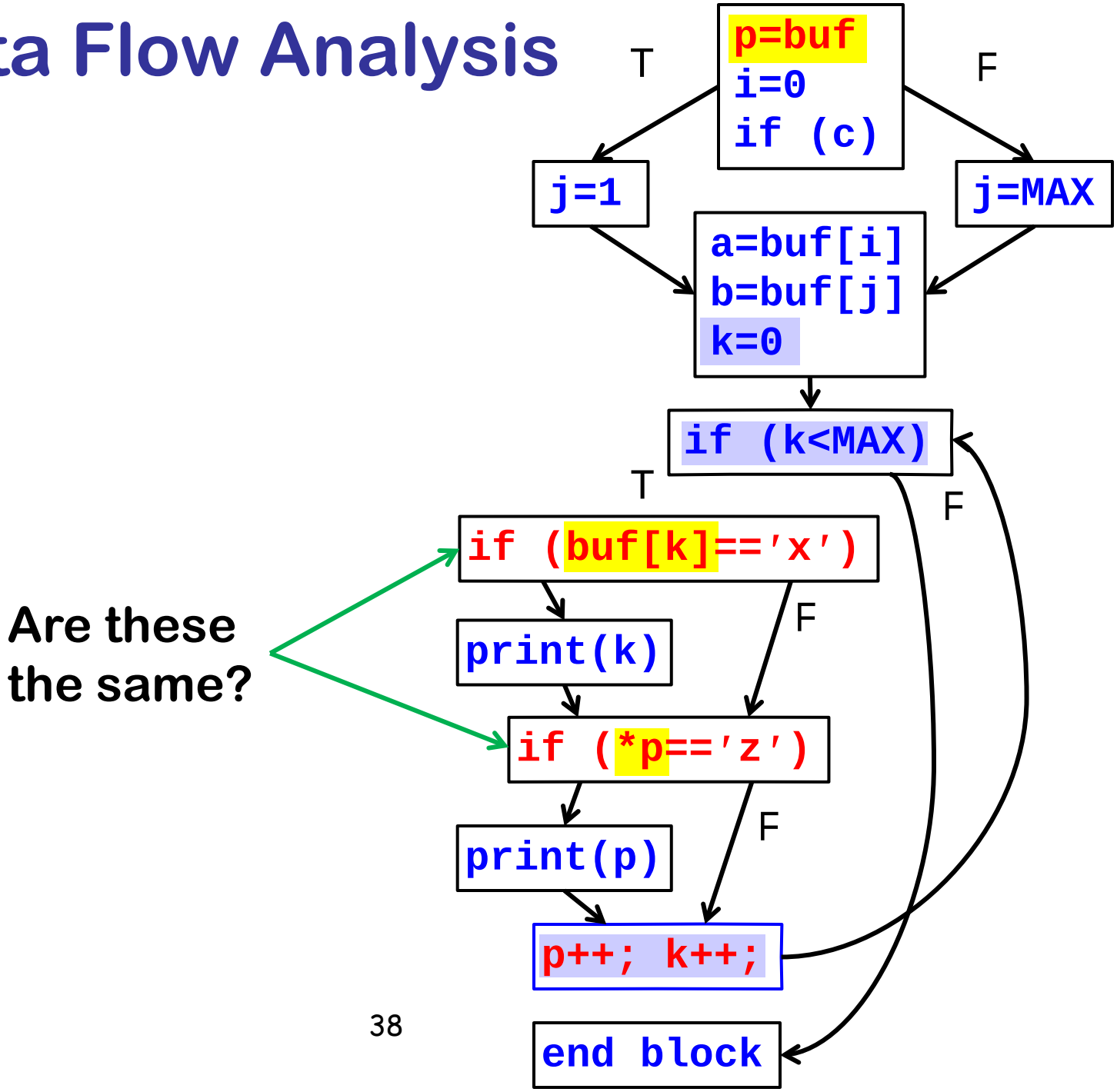
Data Flow Analysis

- **Pointers**



Data Flow Analysis

- Aliasing

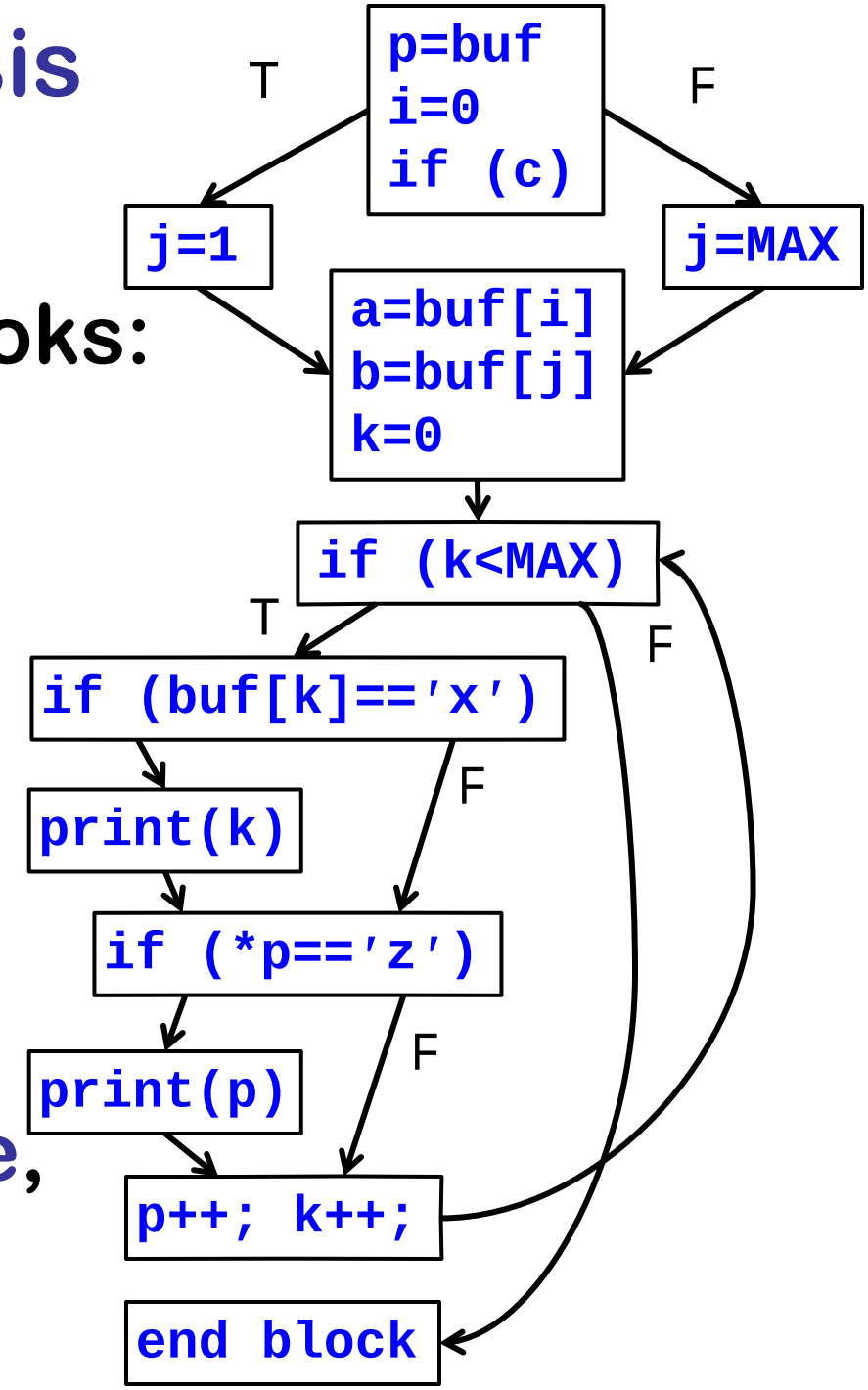


Semantic Analysis

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.

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

```
1.  void CWE78_OS_Command_Injection__char_console_execl_41_bad(){
2.      char *data; char dataBuffer[100] = "";
3.      data = dataBuffer;
4.      /* Read input from the console */
5.      size_t dataLen = strlen(data);
6.      /* If there is room in data, read into it from the cons */
7.      if (100-dataLen > 1)  {
8.      /* POTENTIAL FLAW: Read data from the console */
9.        if (fgets(data+dataLen,(int)(100-dataLen),stdin)!= NULL)
10.       {
11.           /* Remove the carriage return from the string */
12.           dataLen = strlen(data);
13.           if (dataLen > 0 && data[dataLen-1] == '\n')
14.                data[dataLen-1] = '\0';
15.           else {
16.                printf("fgets() failed\n");
17.                data[dataLen] = '\0';
18.           }
19.           /* POTENTIAL FLAW: Execute command without
                 validating */
20.           system (data);
21.       }
22. }
```

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_Containg_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

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
```

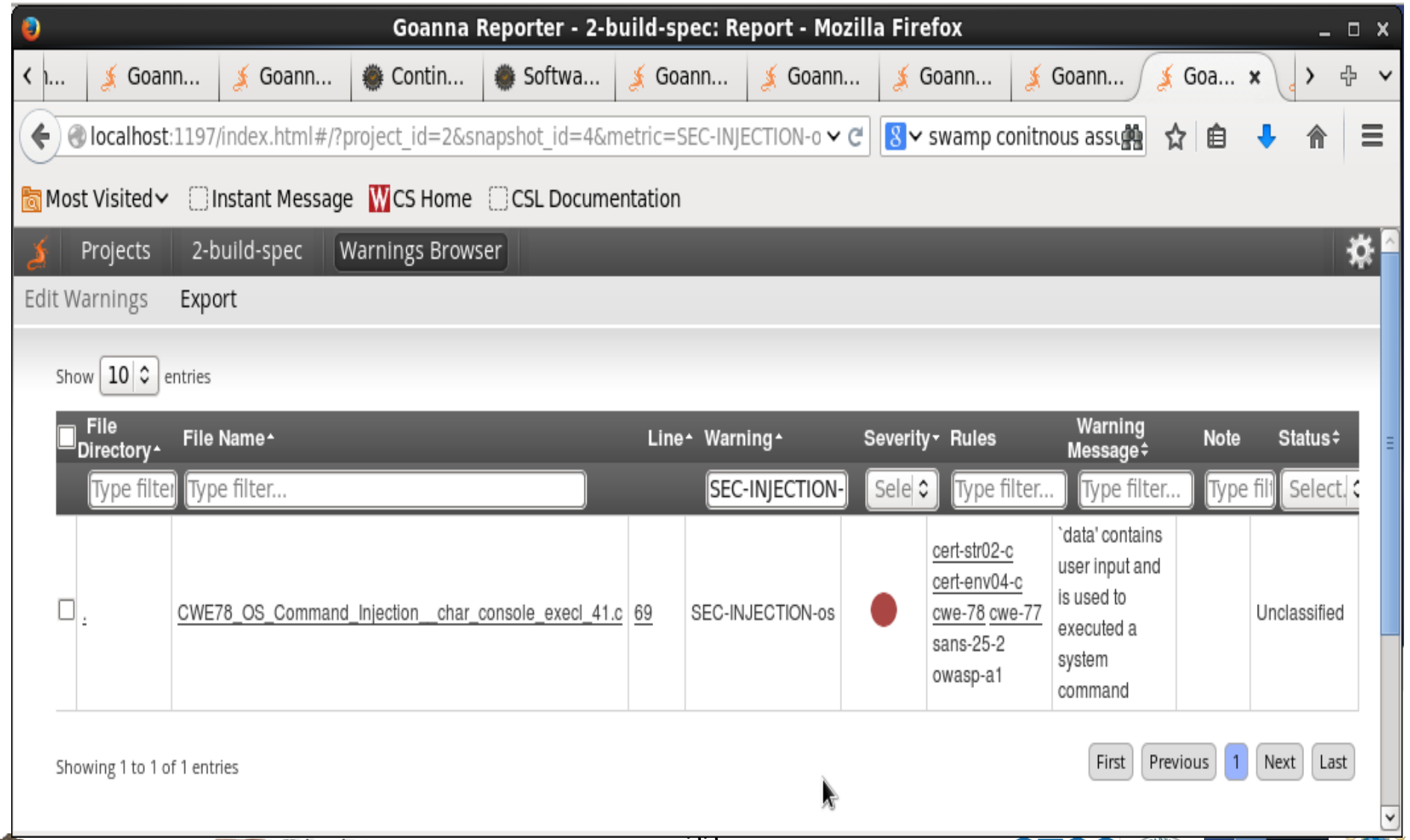
- Read and interact with the analysis results.
  - After goreporter is running, load the provided URL in a web browser.

Goanna. OS Command Injection

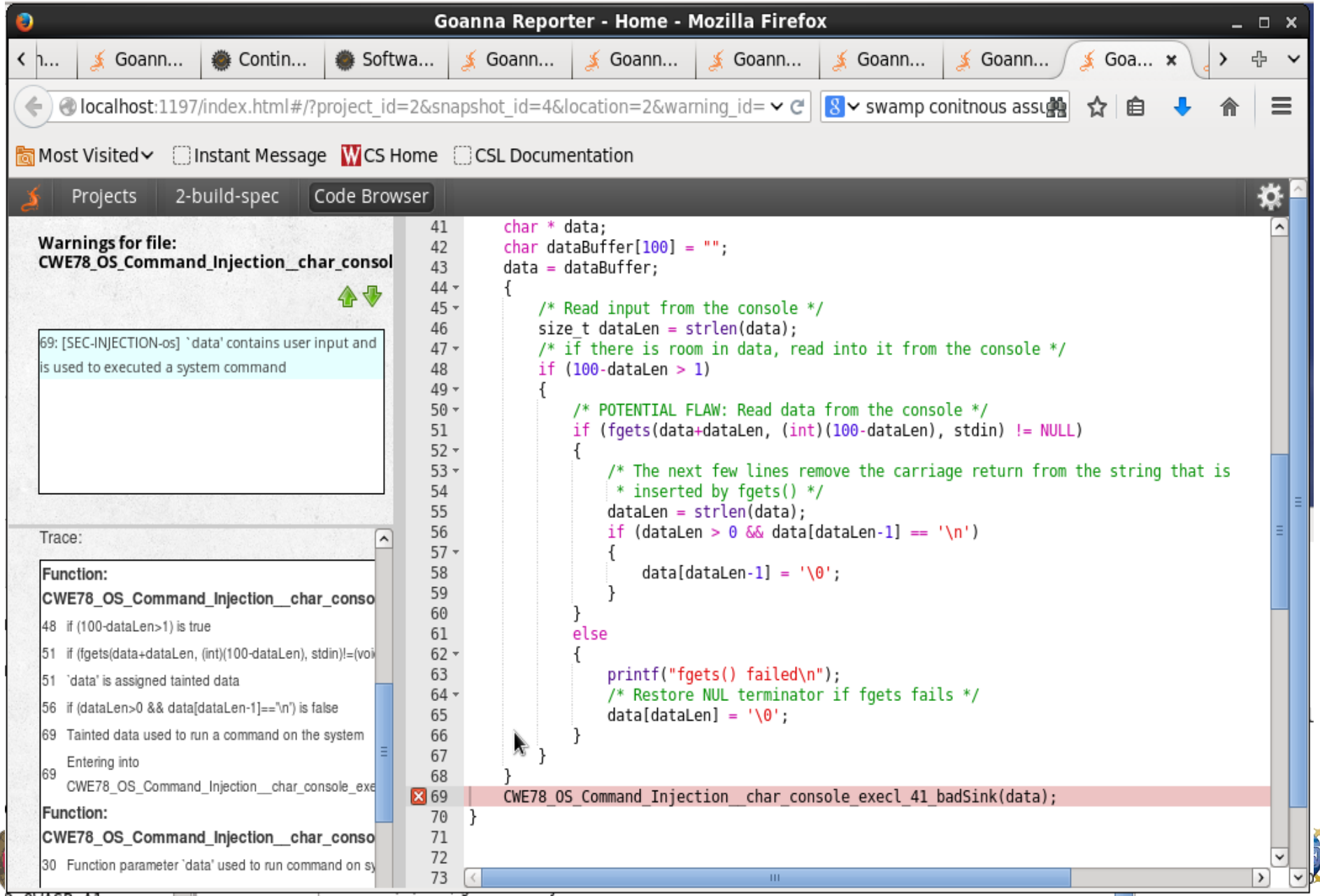
```
$ goanna-trace make
$ goanna-analyze
$ goanna-report
```

- 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:

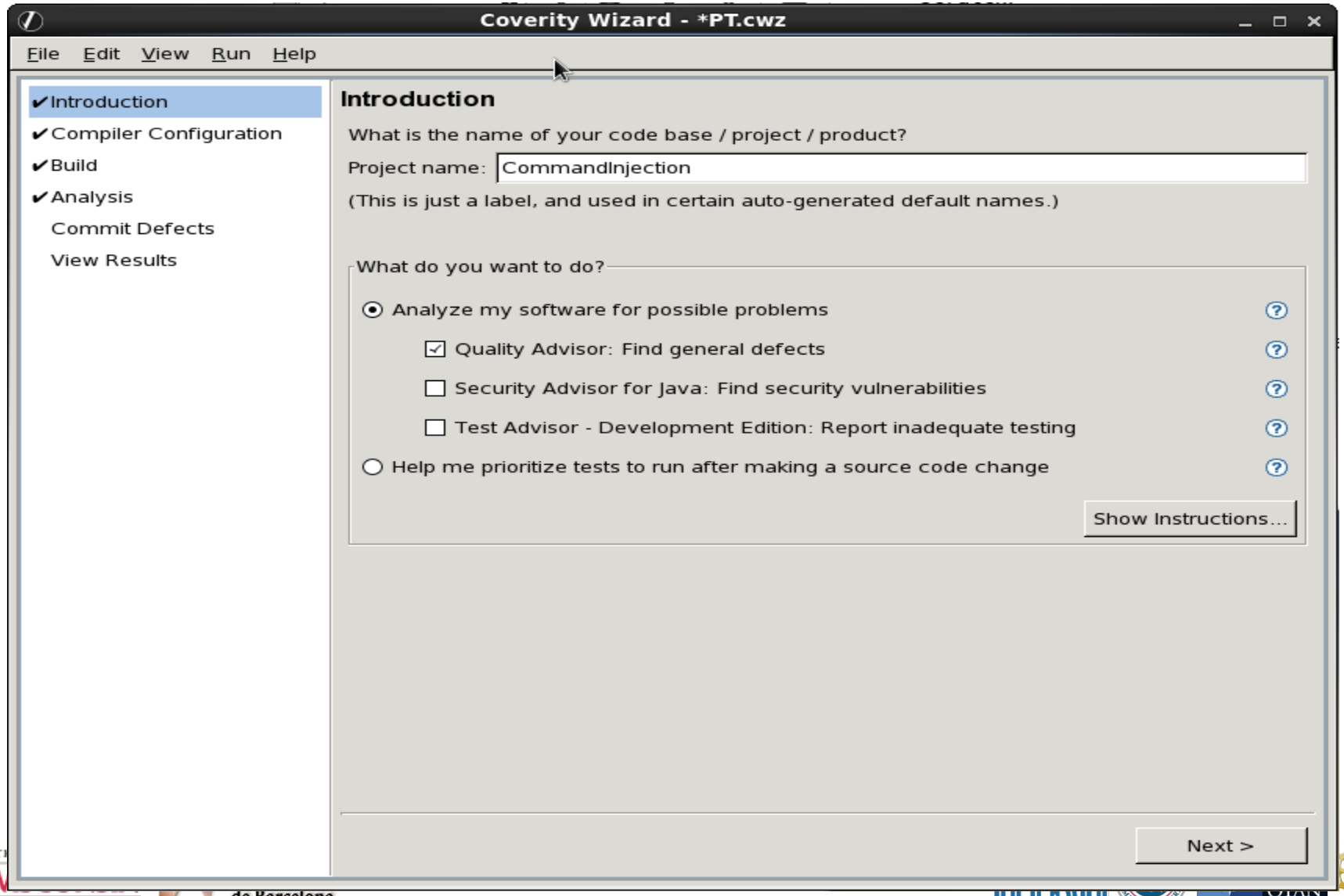
- Generate a configuration for the compiler:
  `cov-configure --gcc`
- Build the intermediate representation of the source code:
  `cov-build --dir <intermediate-dir> make`
- `cov-analyze --dir <intermediate-dir>`
- Check the checkers included by `cov-analize`:
  `cov-analyze --list-checkers`
- Read and interact with the analysis results.
- Graphic mode: `cov-wizard`

Coverity. OS Command Injection

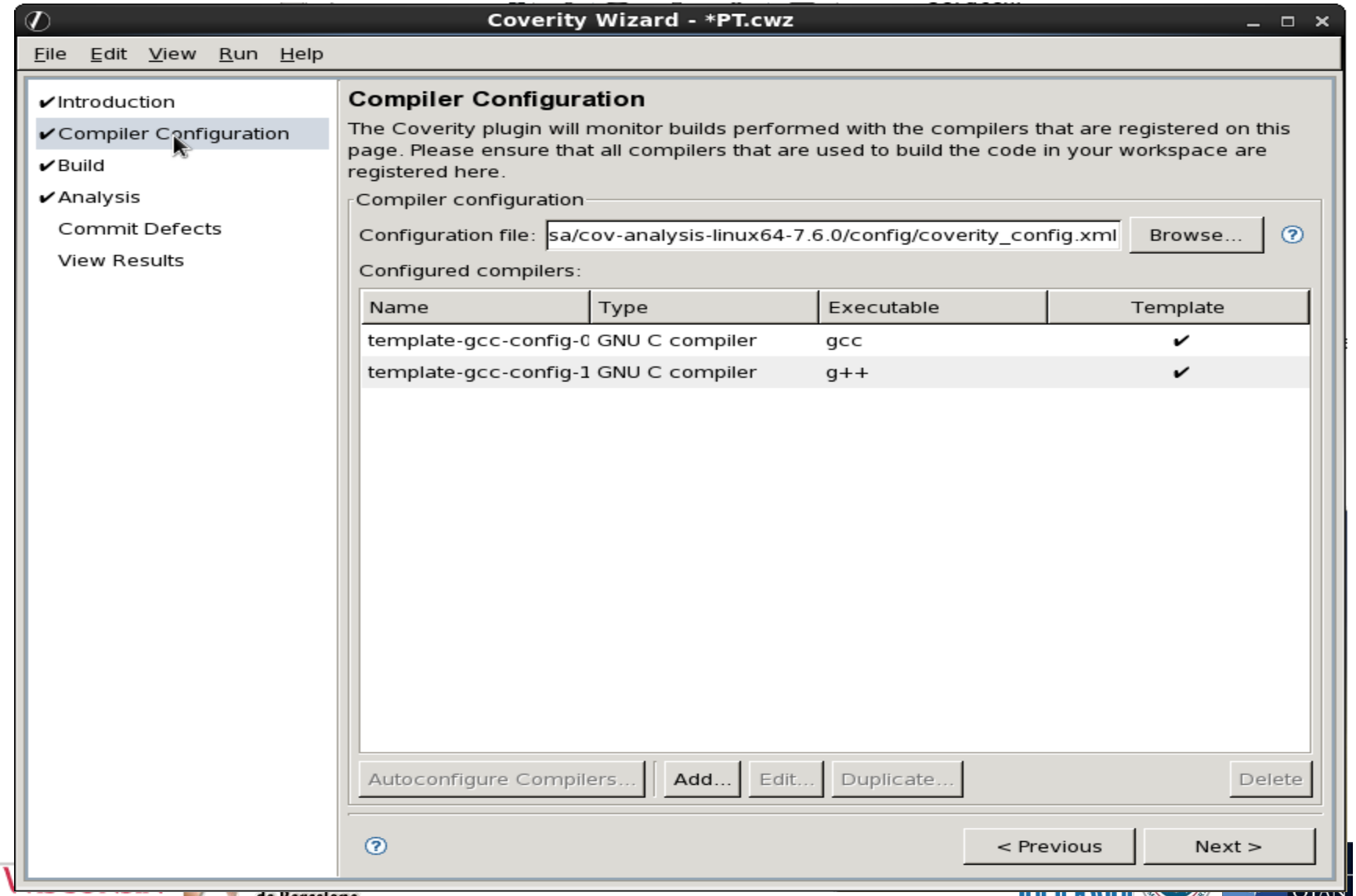
```
$ cov-build --dir cov-comm-injection make
$ cov-analyze --dir cov-comm-injection
              --security
```

- 1 defect found.
- 1 true positive: It detects the command injection.

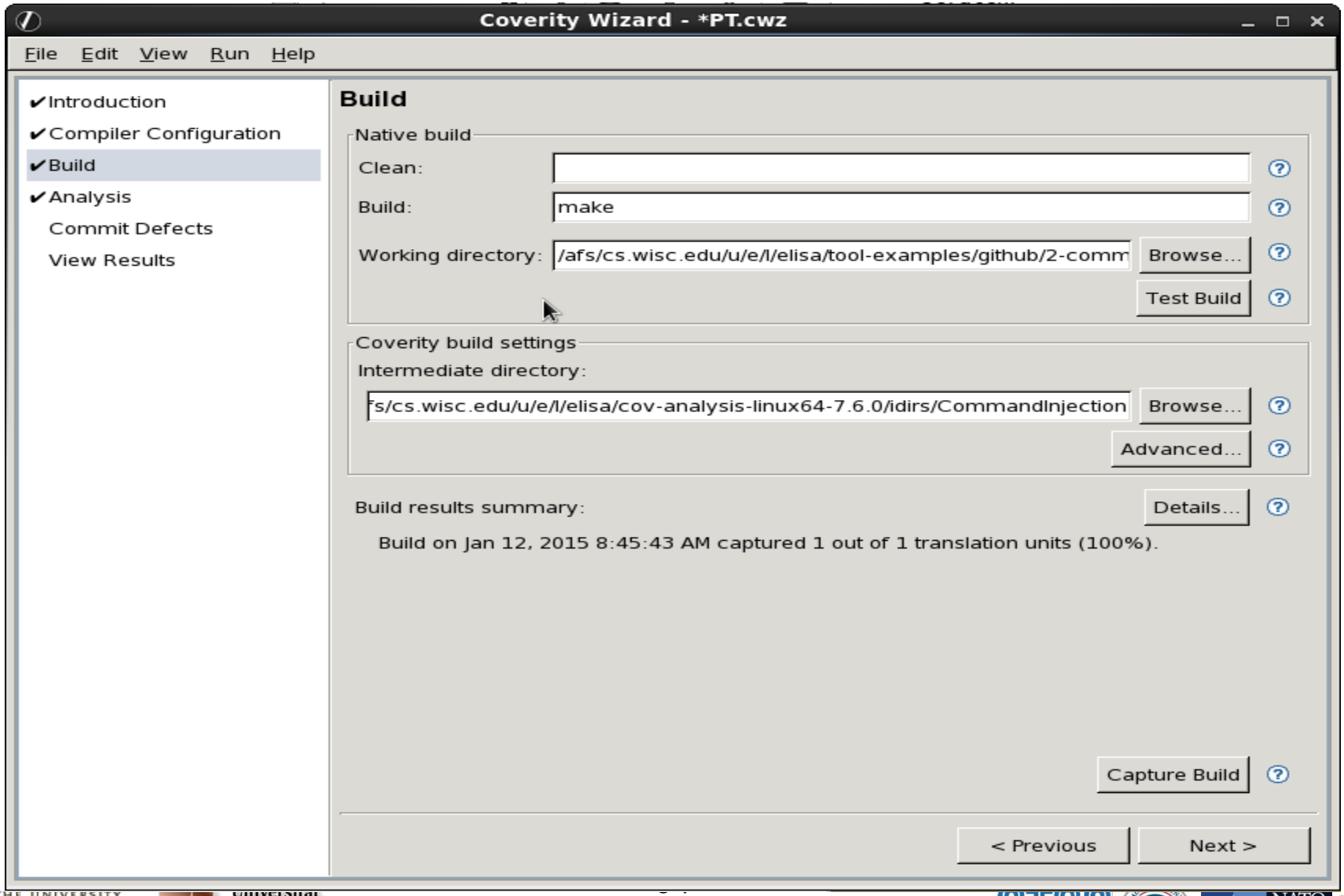
Coverity. OS Command Injection



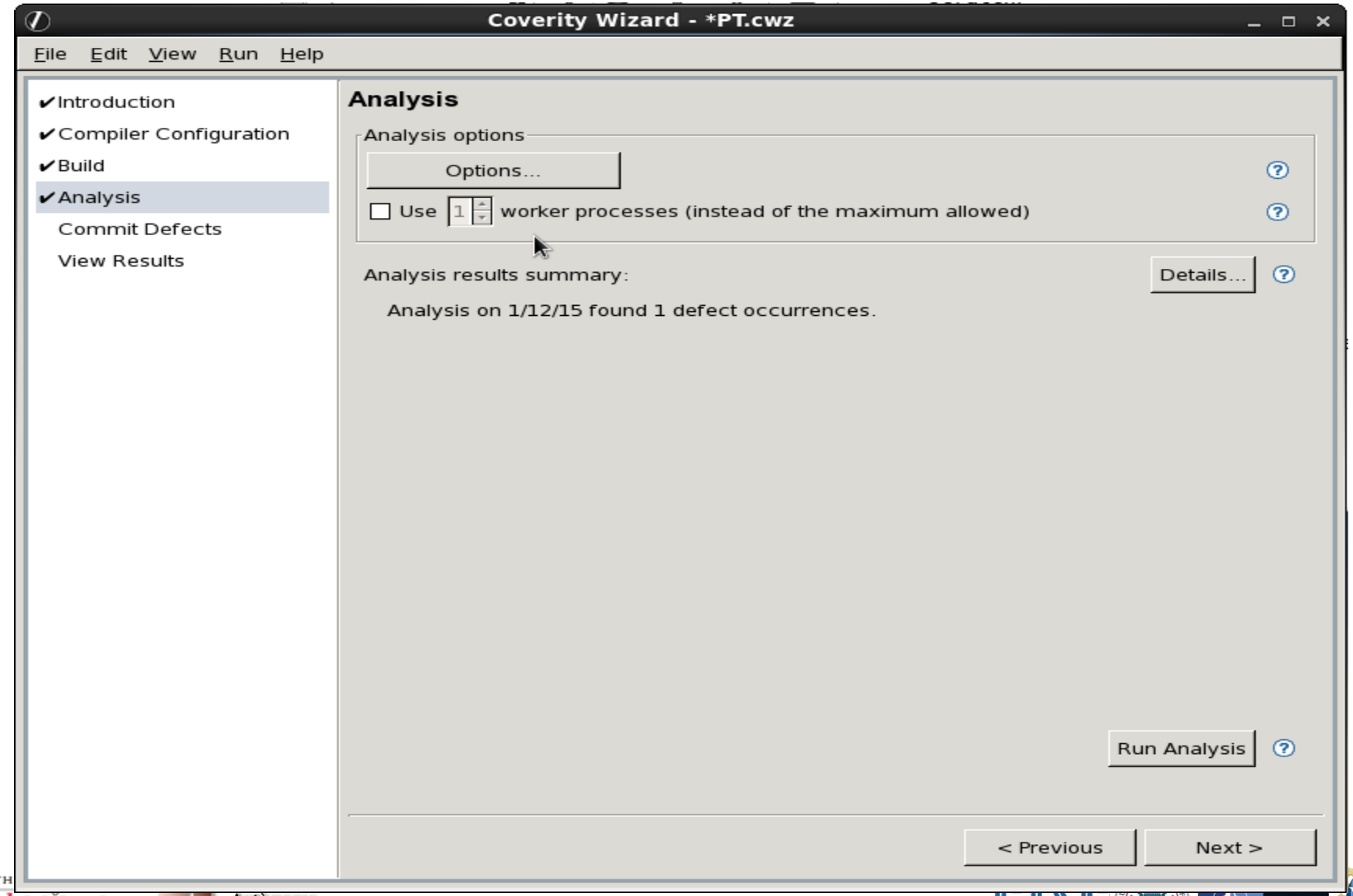
Coverity. OS Command Injection



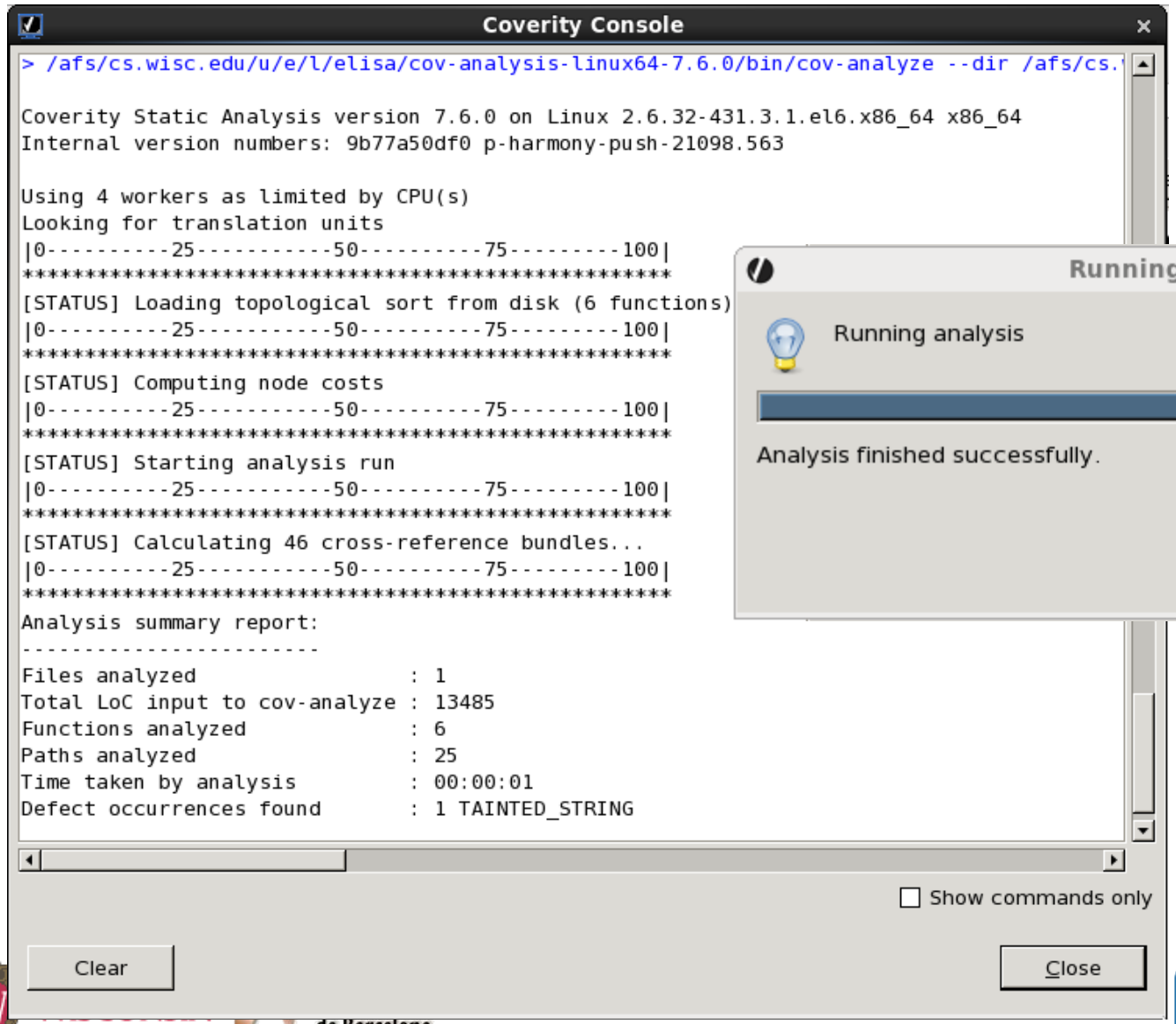
Coverity. OS Command Injection



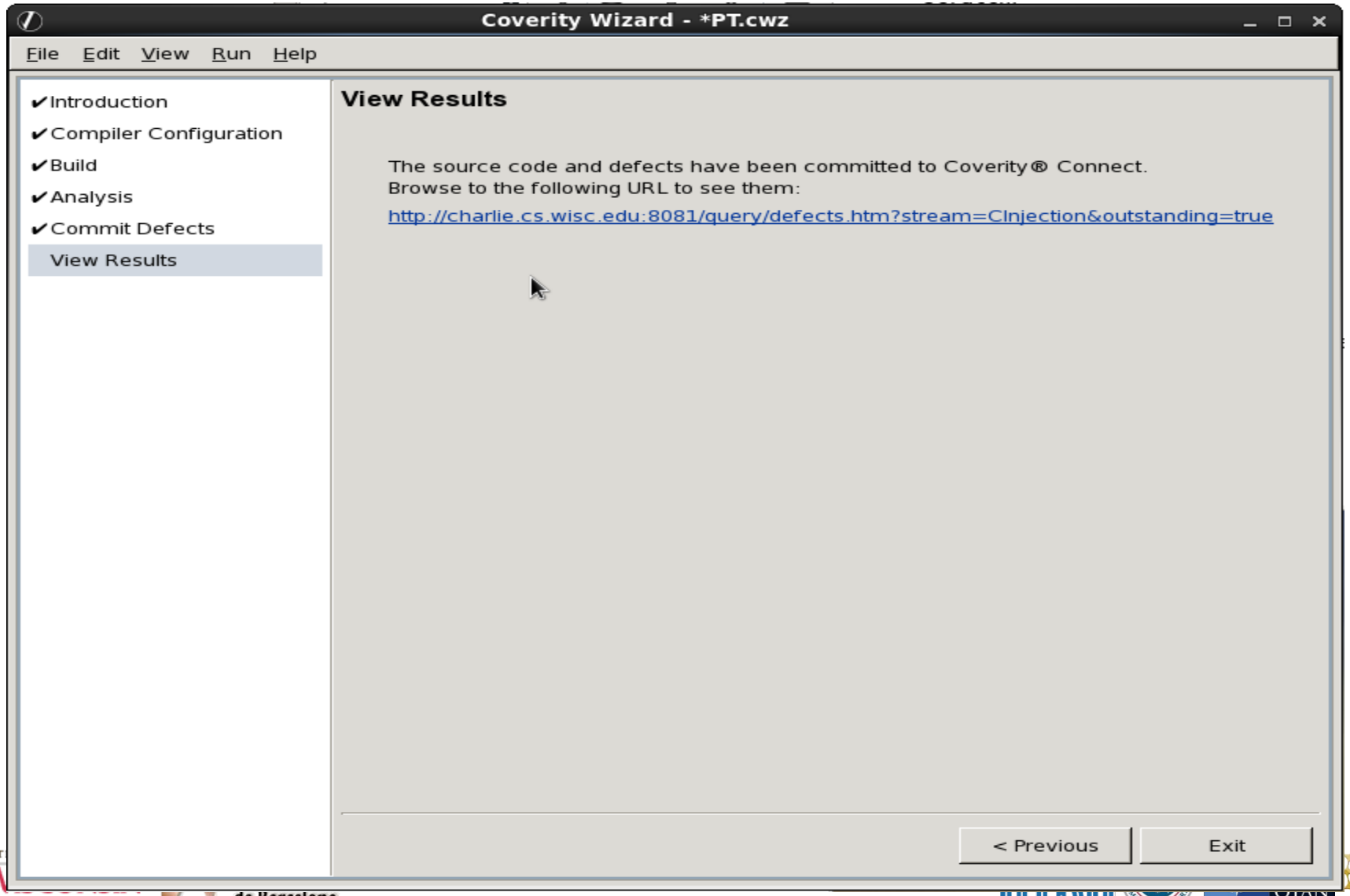
Coverity. OS Command Injection



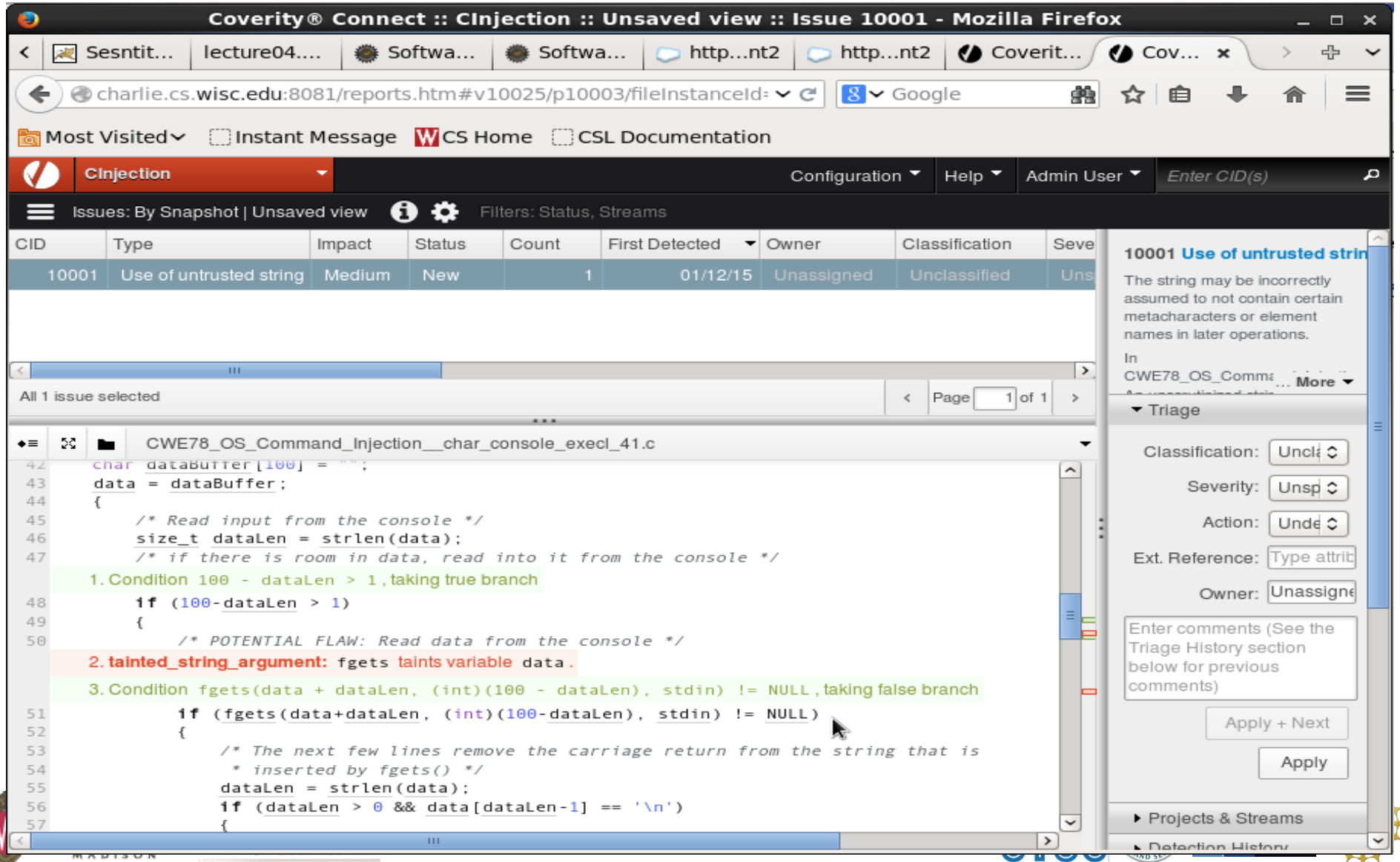
Coverity. OS Command Injection



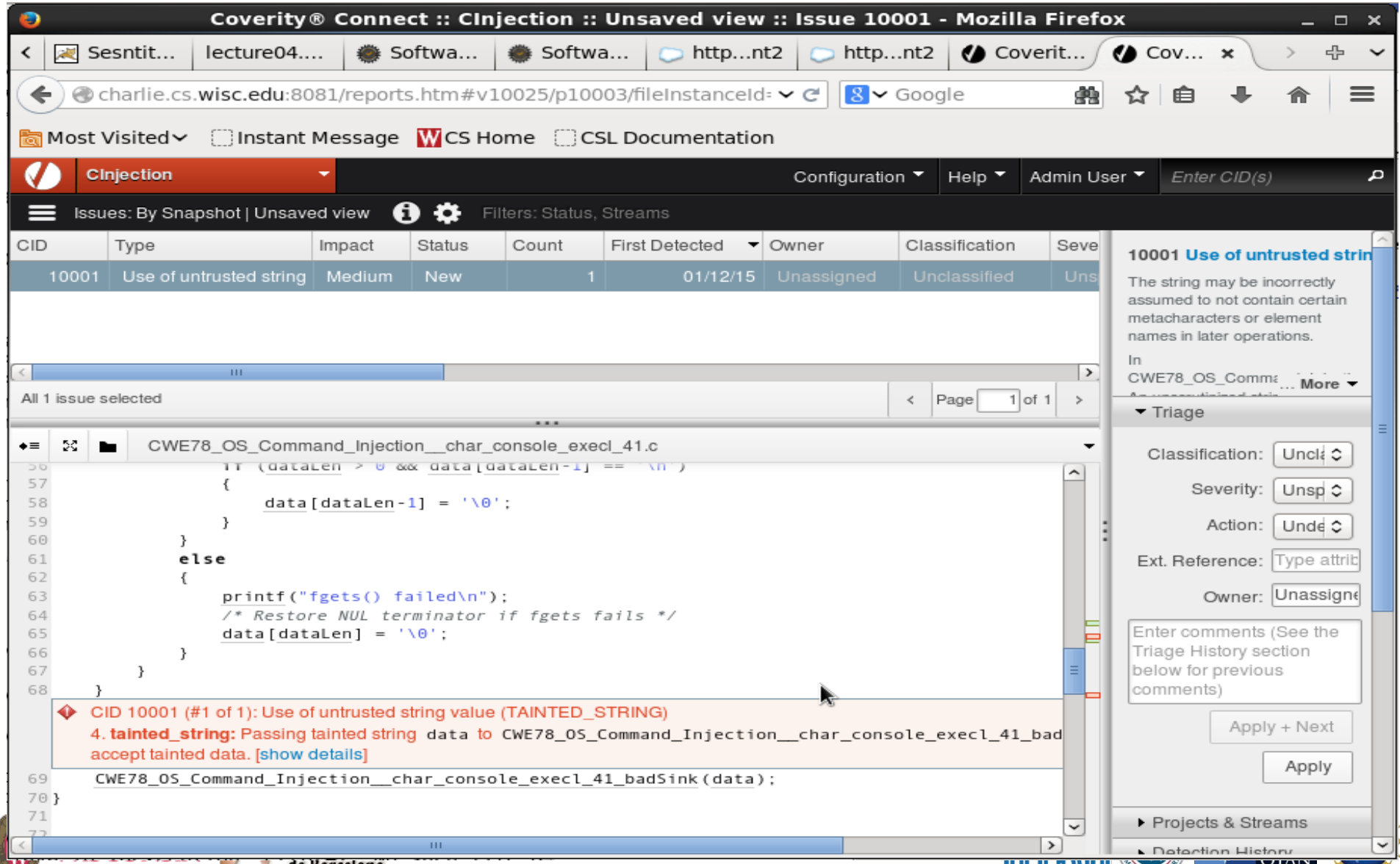
Coverity. OS Command Injection



Coverity. OS Command Injection



Coverity. OS Command Injection




```
57                {
58                    data[dataLen-1] = '\0';
59                }
60            }
61            else
62            {
63                printf("fgets() failed\n");
64                /* Restore NUL terminator if fgets fails */
65                data[dataLen] = '\0';
66            }
67        }
68    }
69    CWE78_OS_Command_Injection__char_console_execl_41_badSink(data);
70}
```

Java

CWE 601: Open Redirect

```
   public void doGet(HttpServletRequest request,
1.                           HttpServletResponse response)
2.                          throws ServletException, IOException {
3.     response.setContentType("text/html");
4.     PrintWriter returnHTML = response.getWriter();
5.     returnHTML.println("<html><head><title>");
6.     returnHTML.println("Open Redirect");
7.     returnHTML.println("</title></head><body>");
8.
9.     String data;
10.    data = ""; // initialize data in case there are no cookies.
11.    // Read data from cookies.
12.    Cookie cookieSources[] = request.getCookies();
13.    if (cookieSources != null)
14.    // POTENTIAL FLAW: Read data from the first cookie value.
15.          data = cookieSources[0].getValue();
16.    if (data != null) {
17.          URI uri;
18.          uri = new URI(data);
19.          // POTENTIAL FLAW: redirect is sent verbatim.
20.          response.sendRedirect(data);
21.          return;
22.     }
23.
```

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

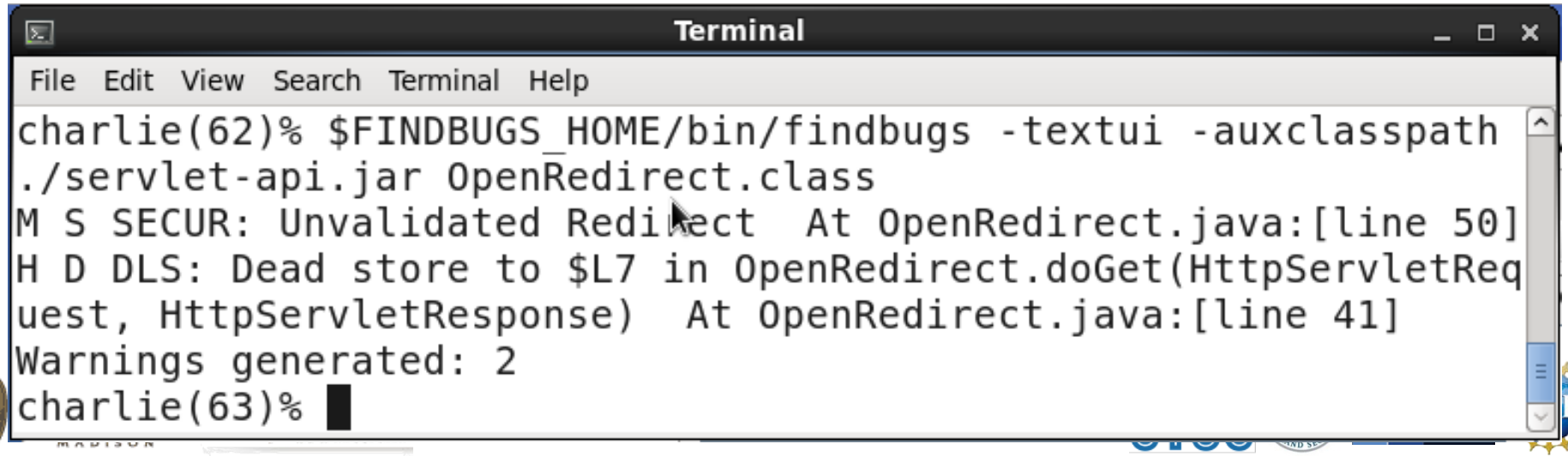
1. Define FINDBUGS_HOME in the environment.
2. Install the Find Security Bugs plugin.
3. Learn the command line instructions and also use the graphical interface.
4. Command line interface:

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

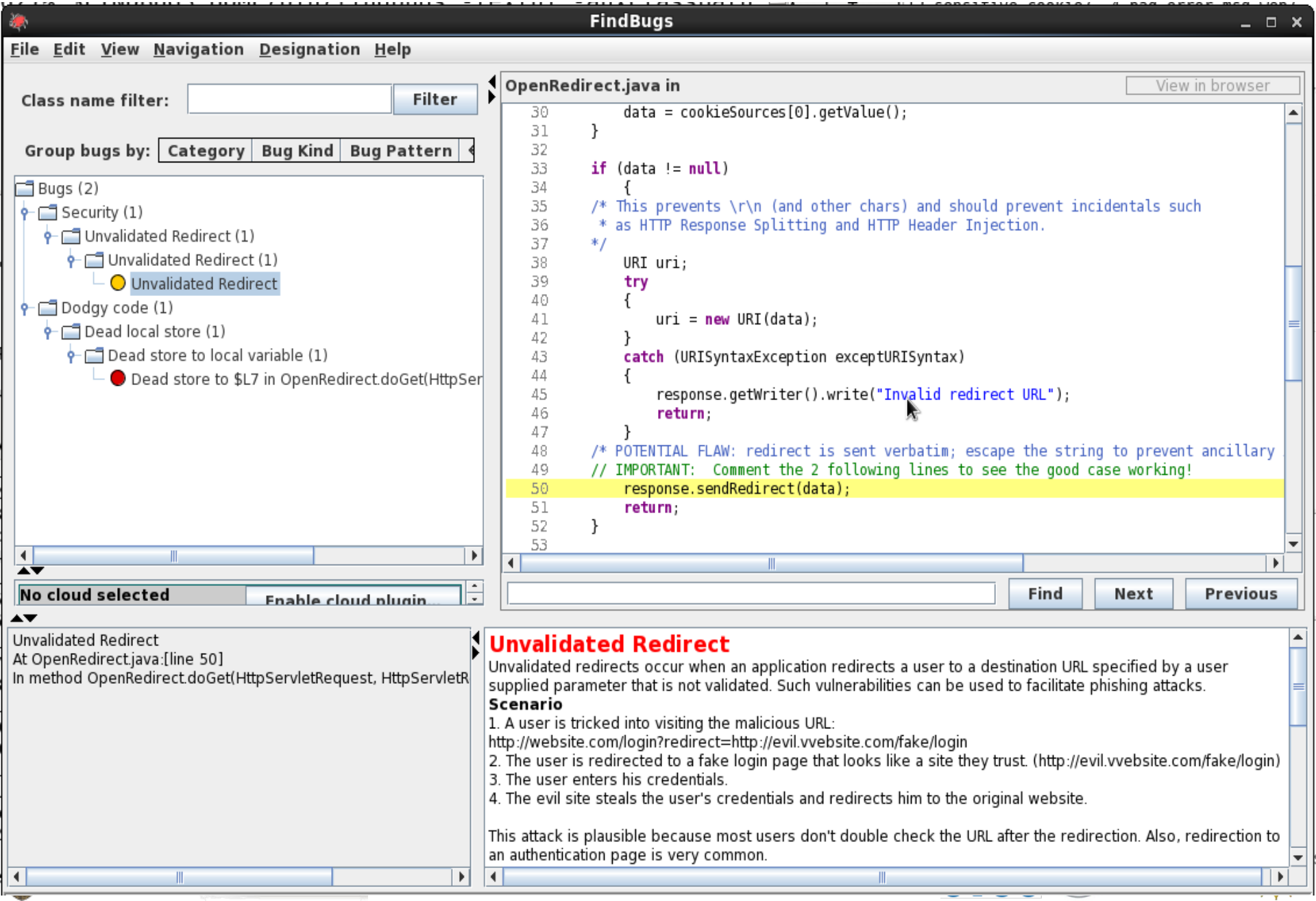
5. Graphic Interface: `java -jar $FINDBUGS_HOME/lib/findbugs.jar -gui`

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



Parasoft Jtest

Jtest

PARASOFT

- Commercial tool available at http://www.parasoft.com/product/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`.

Jtest. Open Redirect

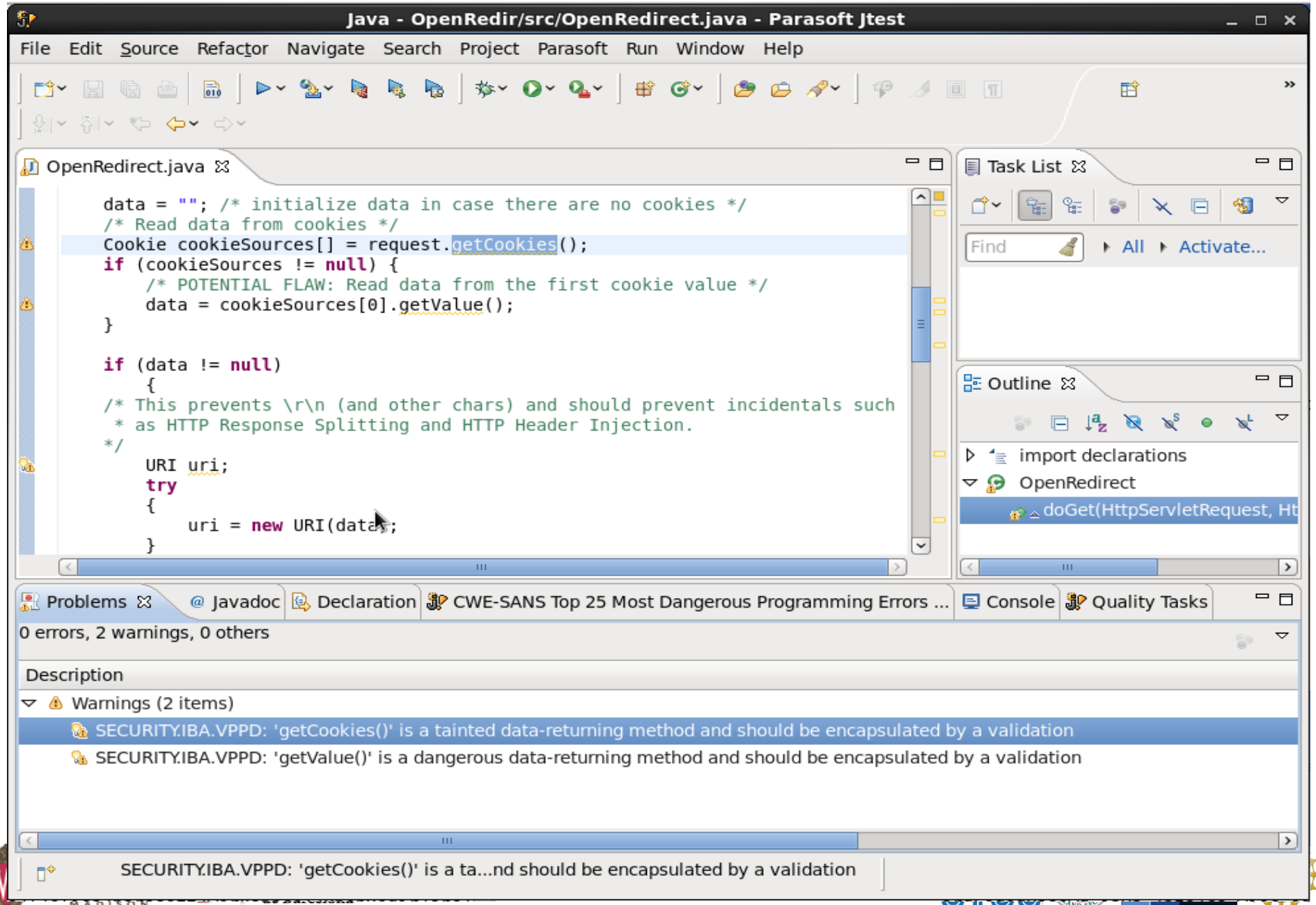
Create the OpenRedir project.

Include servlet-api.jar in the OpenRedir project.

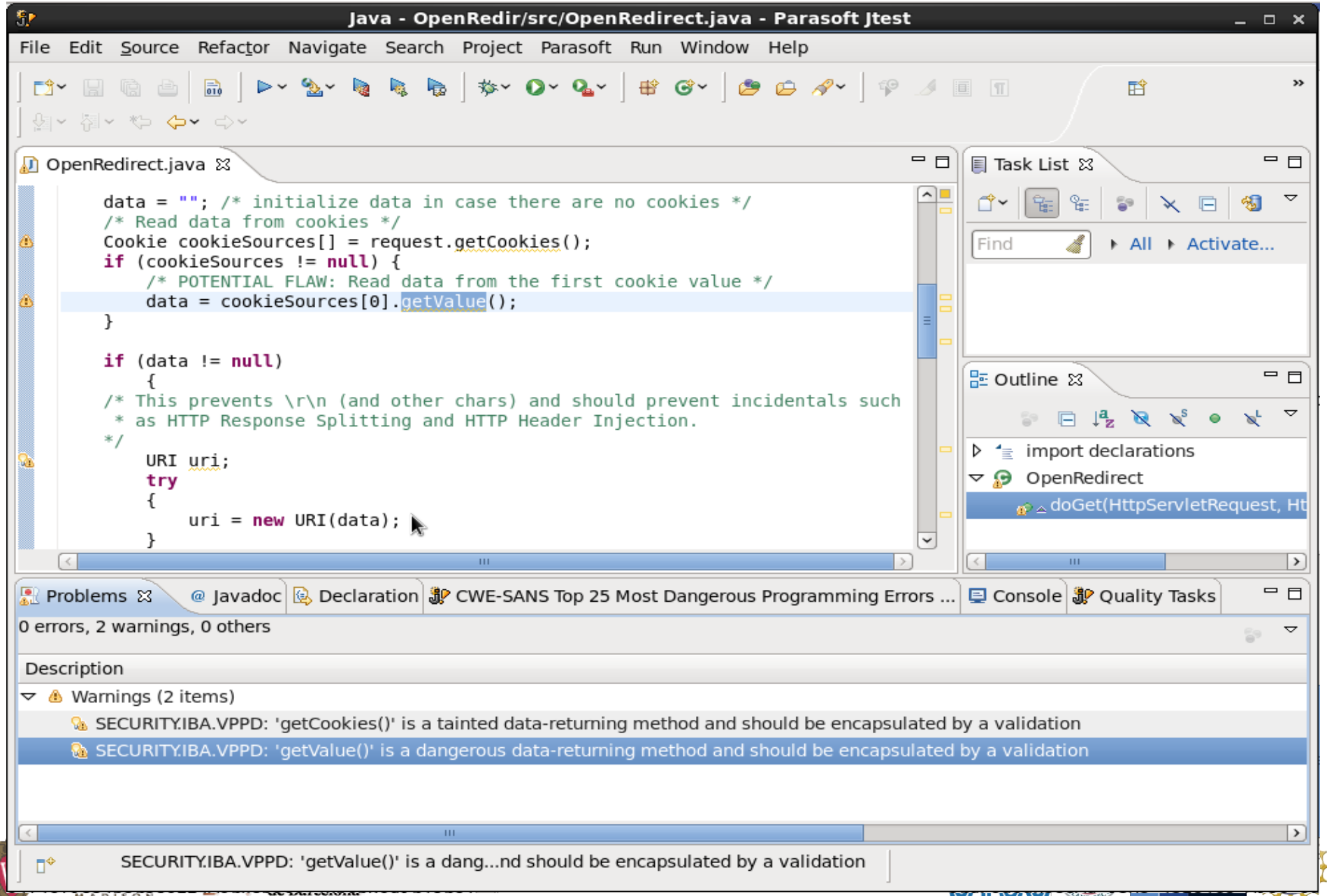
```
cp OpenRedirect.java
~elisa/parasoft/workspace1/OpenRedir/src
```

- **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.**

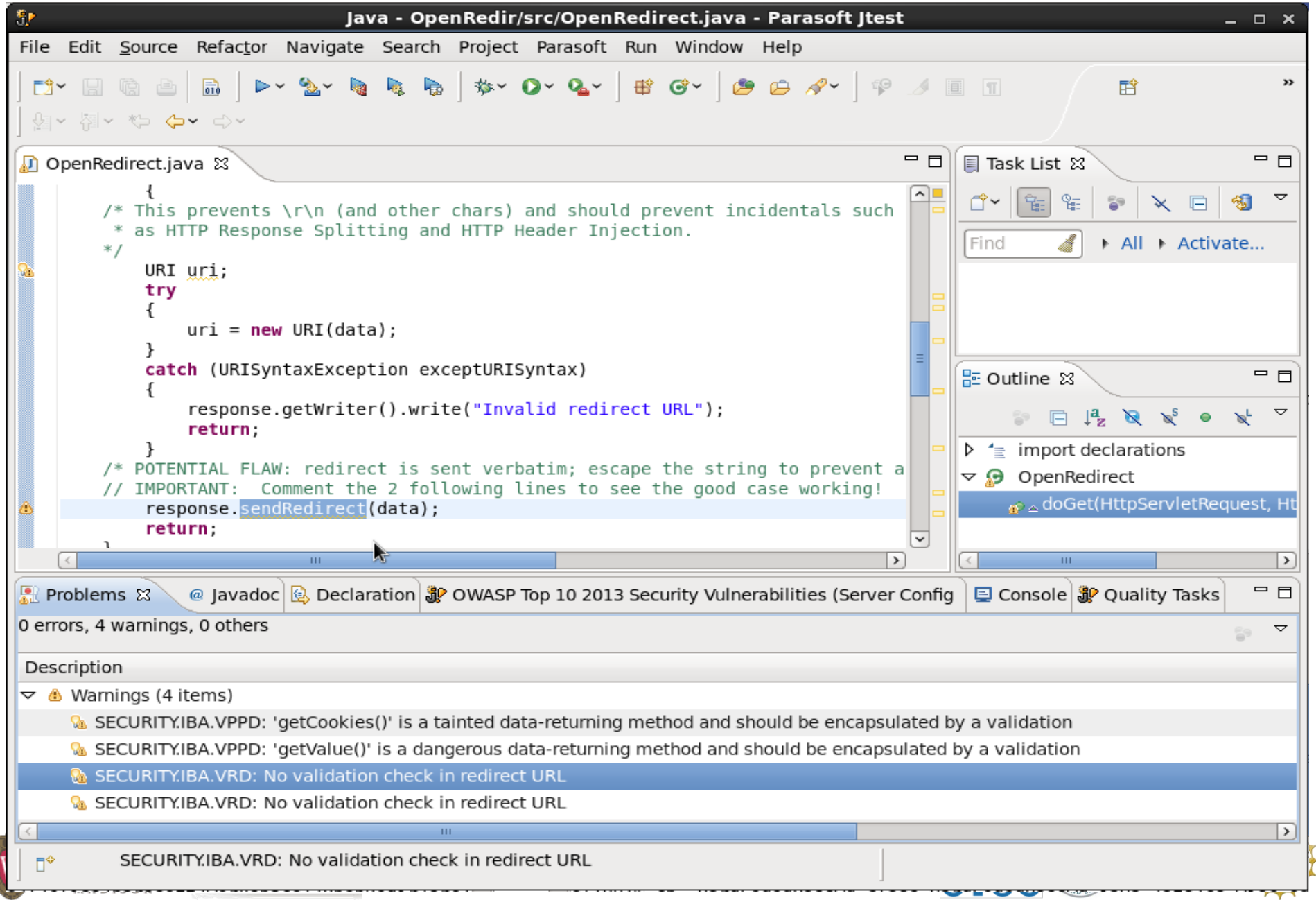
Jtest. Open Redirect



Jtest. Open Redirect



Jtest. Open Redirect



Roadmap

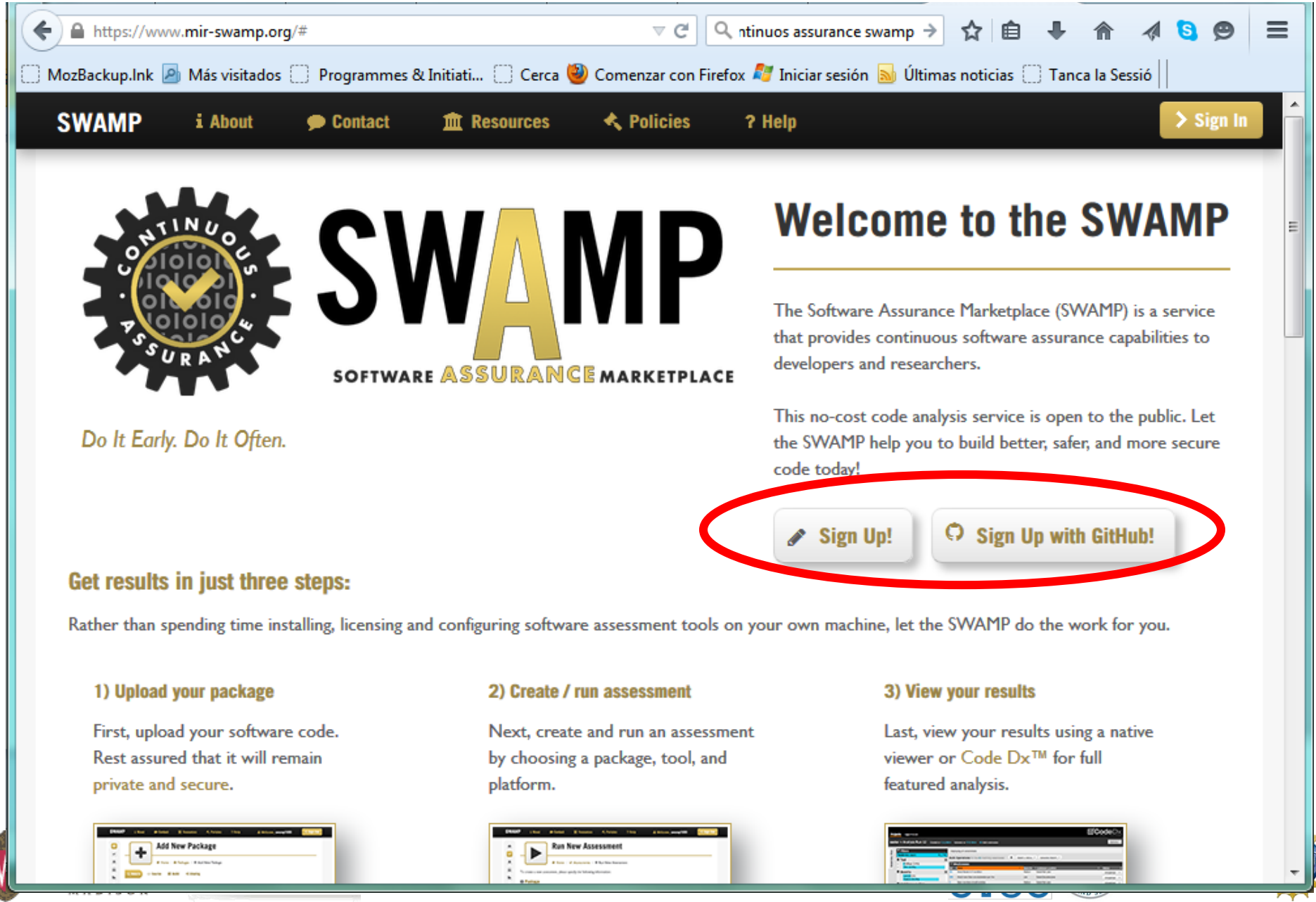
- What is the SWAMP?
- Using the SWAMP
  - Register
  - Create a project
  - Upload your software package
  - Run your assessment
  - View the results
    - Java
    - C/C++

https://continuousassurance.org/swamp/SWAMP-User-Manual.pdf

Getting Started with the SWAMP

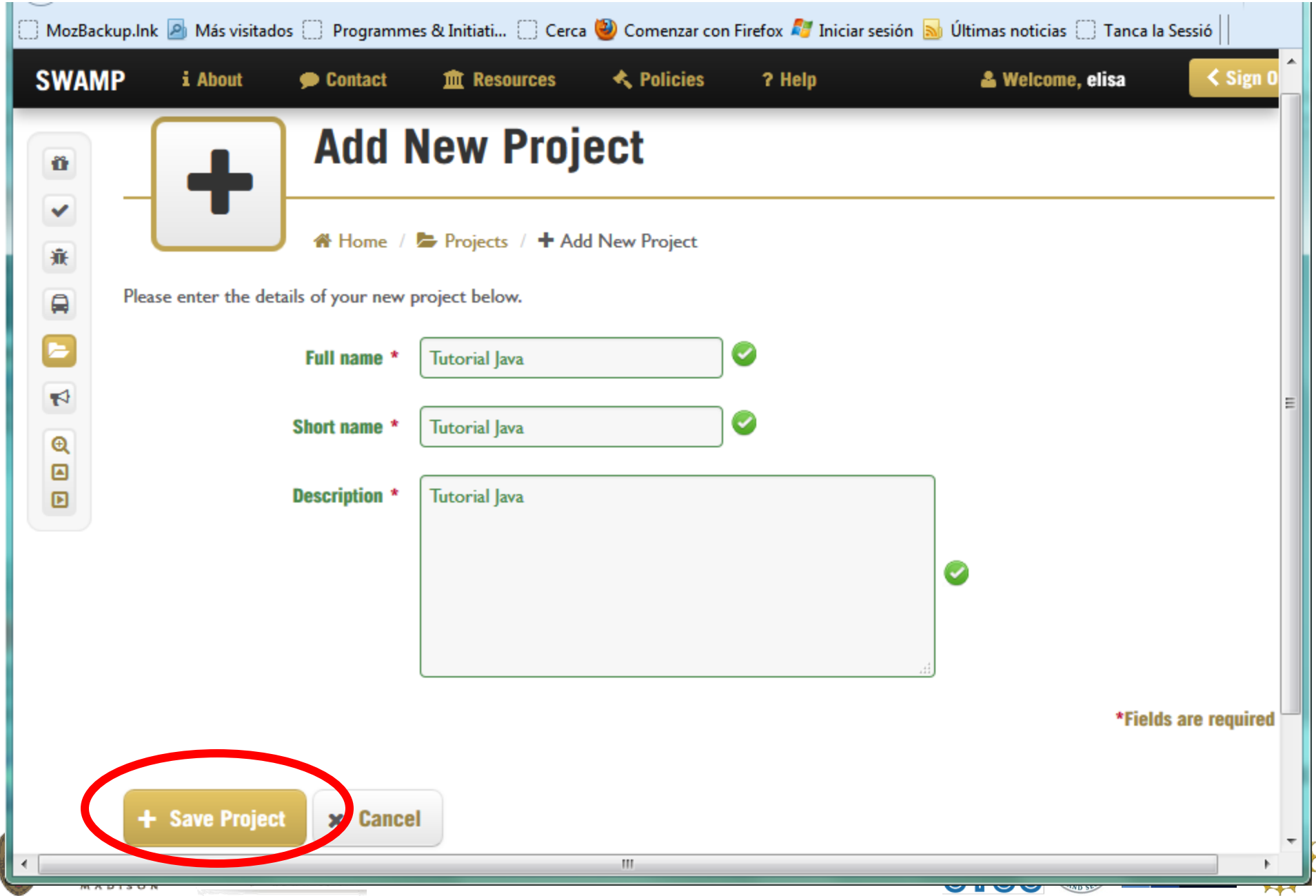
- Software Assurance Market Place.
- Objective: Automate and simplify the use of (multiple) tools.
- A national, no-cost resource for software assurance (SwA) technologies used across research institutions, non-governmental organizations, and civilian agencies and their communities as both a research platform and a core component of the software development life cycle.

Register to use the SWAMP

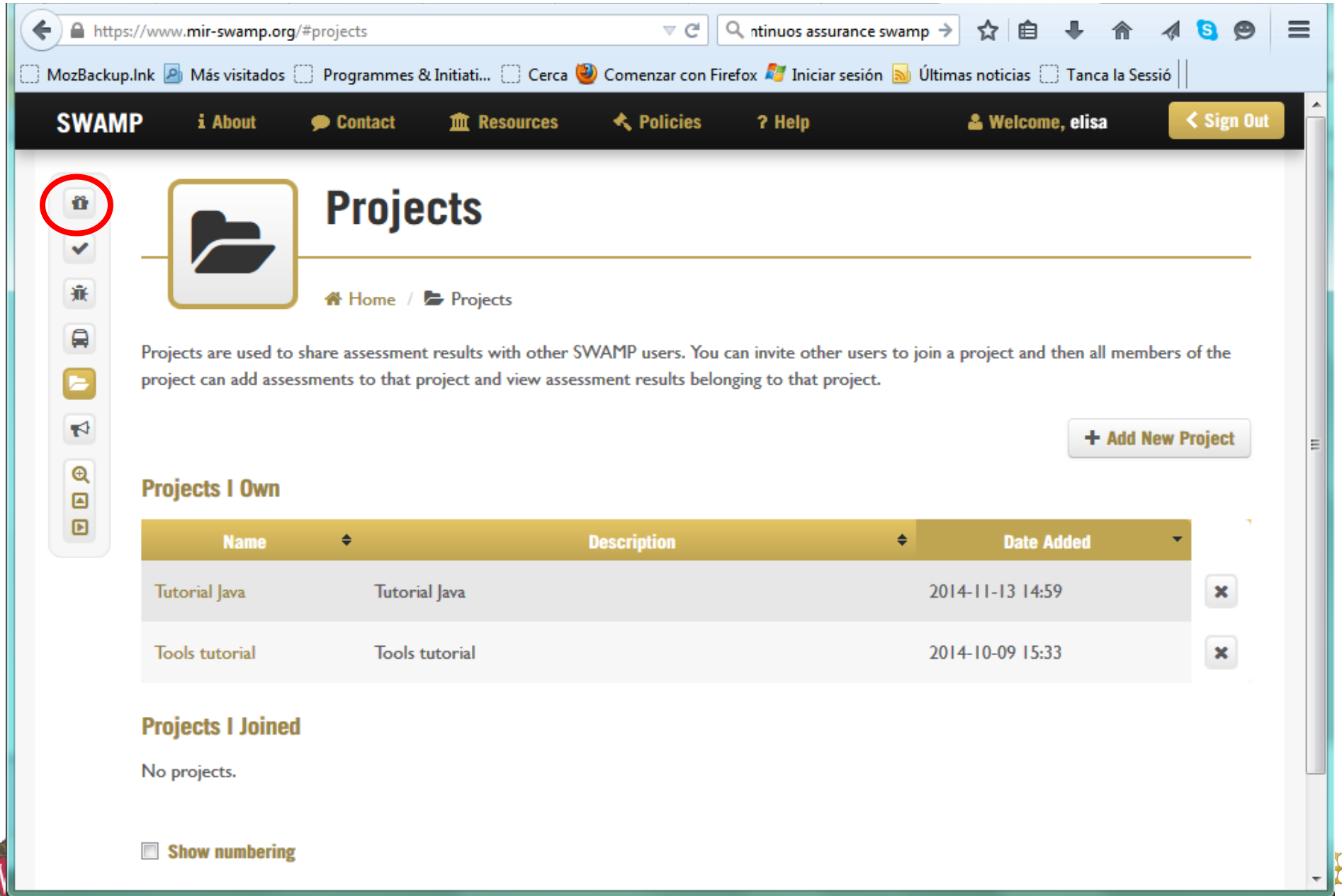


What can I do in the SWAMP?

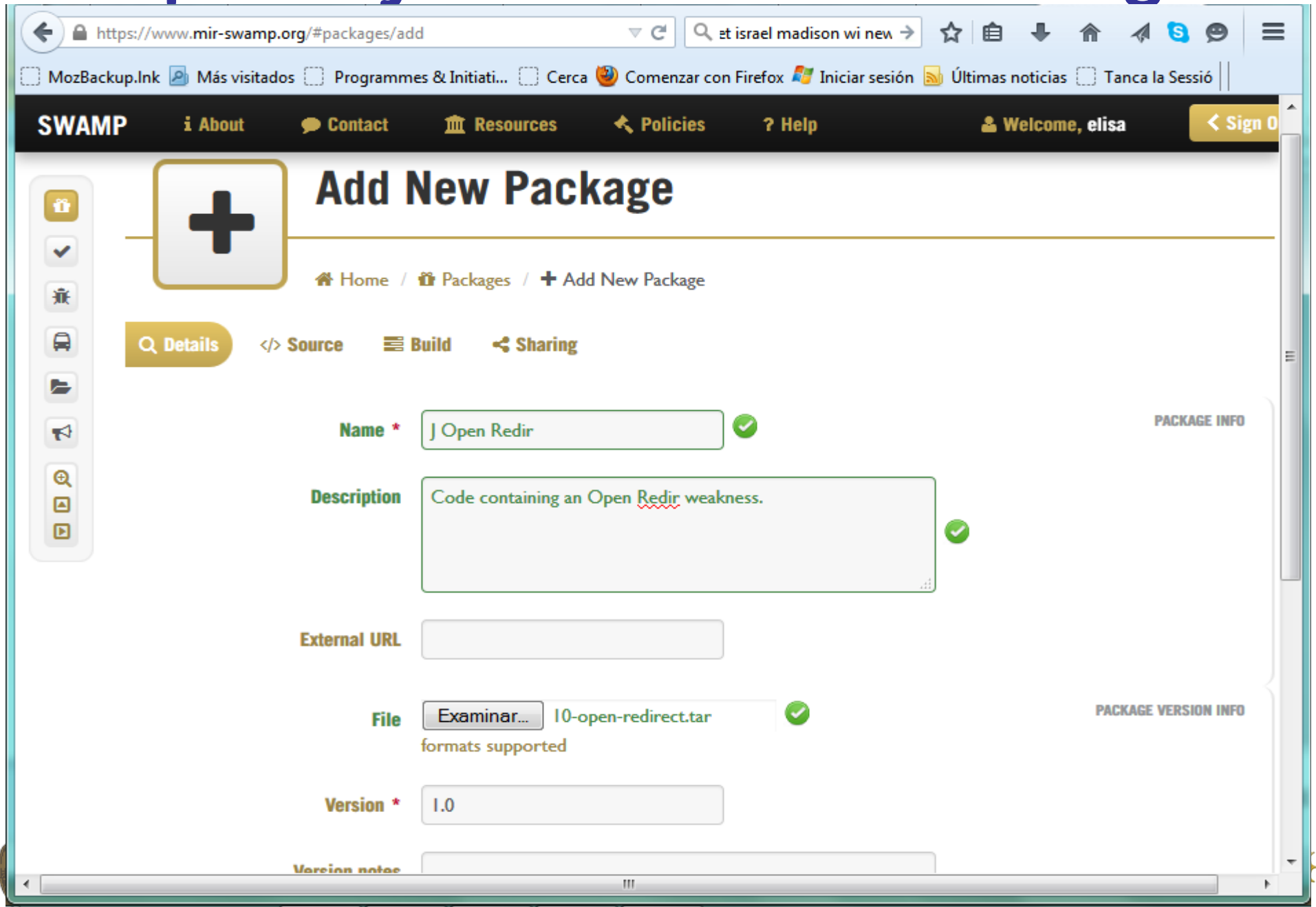
Create a Project



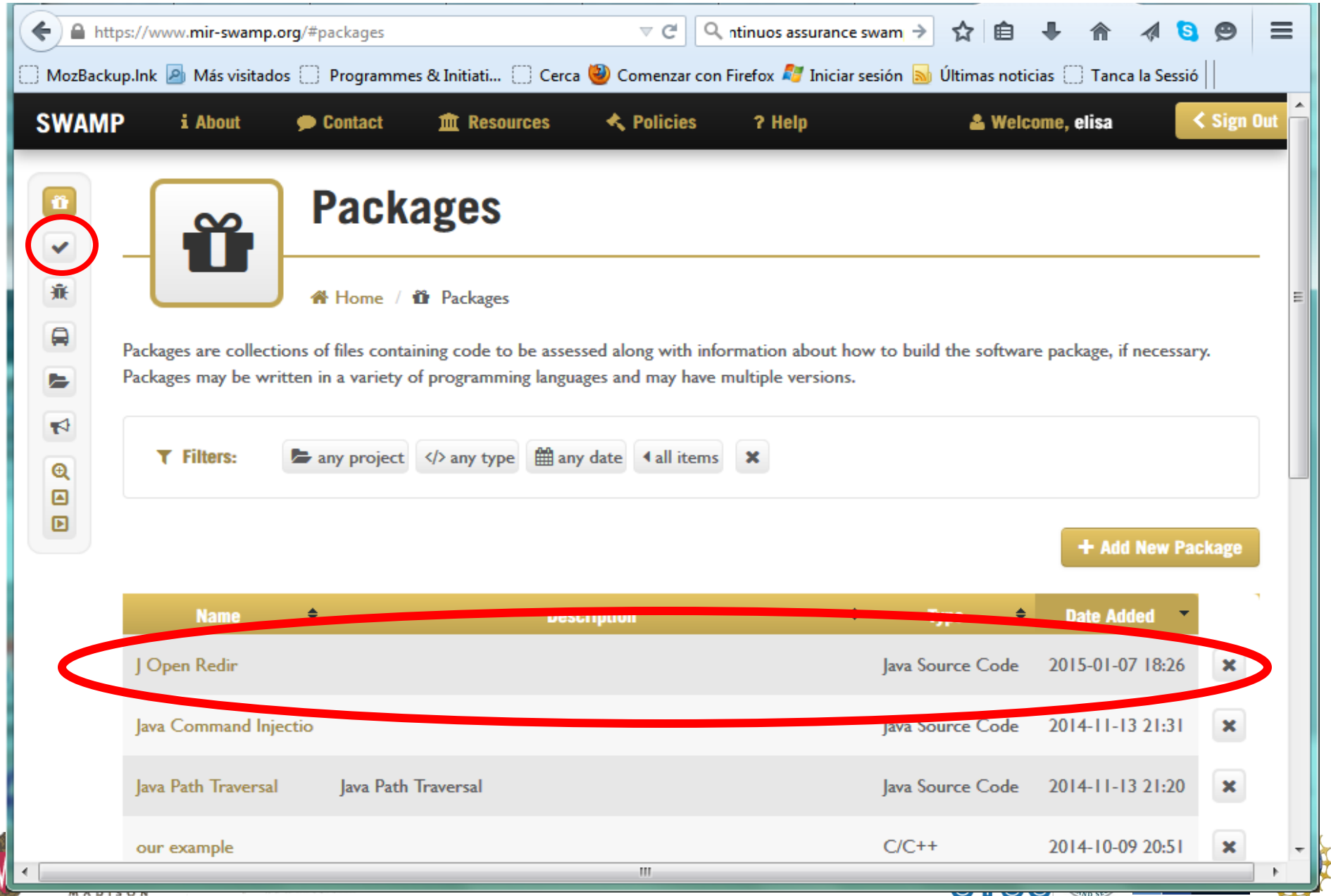
My Projects


https://www.mir-swamp.org/#projects
ntinuos assurance swamp
MozBackup.lnk
Más visitados
Programmes & Initiati...
Cerca
Comenzar con Firefox
Iniciar sesión
Últimas noticias
Tanca la Sessió
SWAMP
About
Contact
Resources
Policies
Help
Welcome, elisa
Sign Out
Projects
Home / 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.
+ Add New Project
Projects I Own
Name
Description
Date Added
Tutorial Java
Tutorial Java
2014-11-13 14:59
Tools tutorial
Tools tutorial
2014-10-09 15:33
Projects I Joined
No projects.
Show numbering

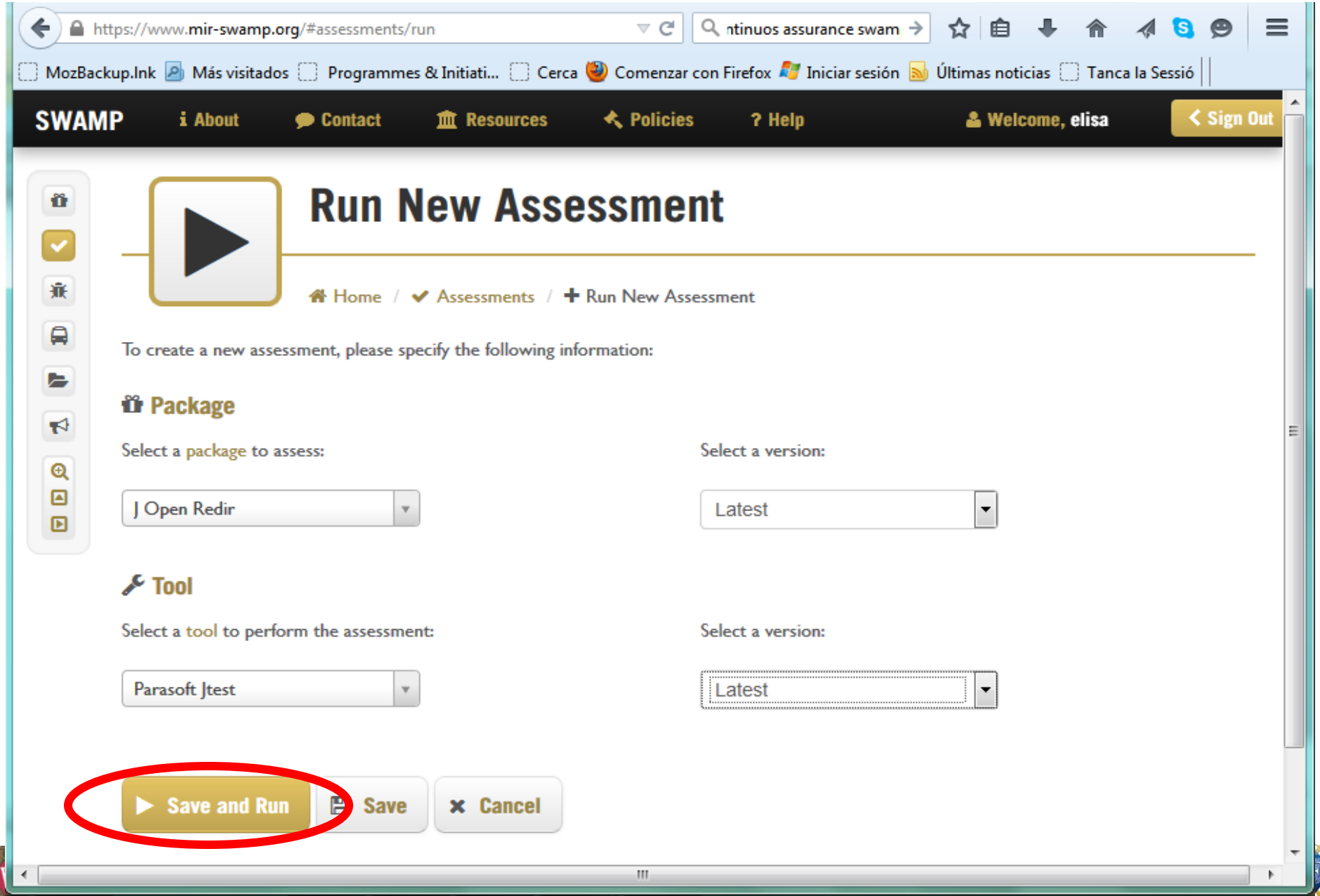
Upload your Software Package



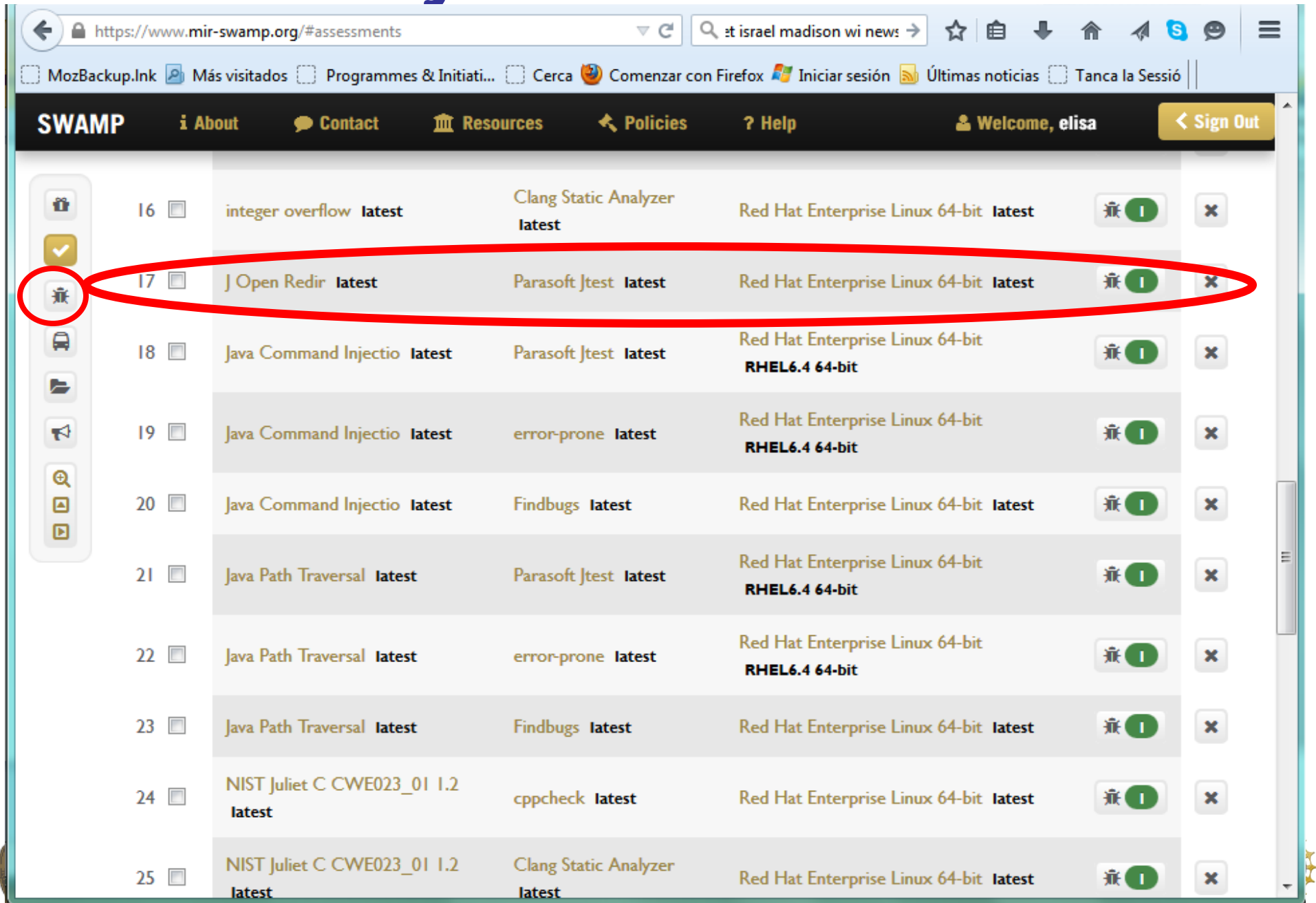
My software Packages



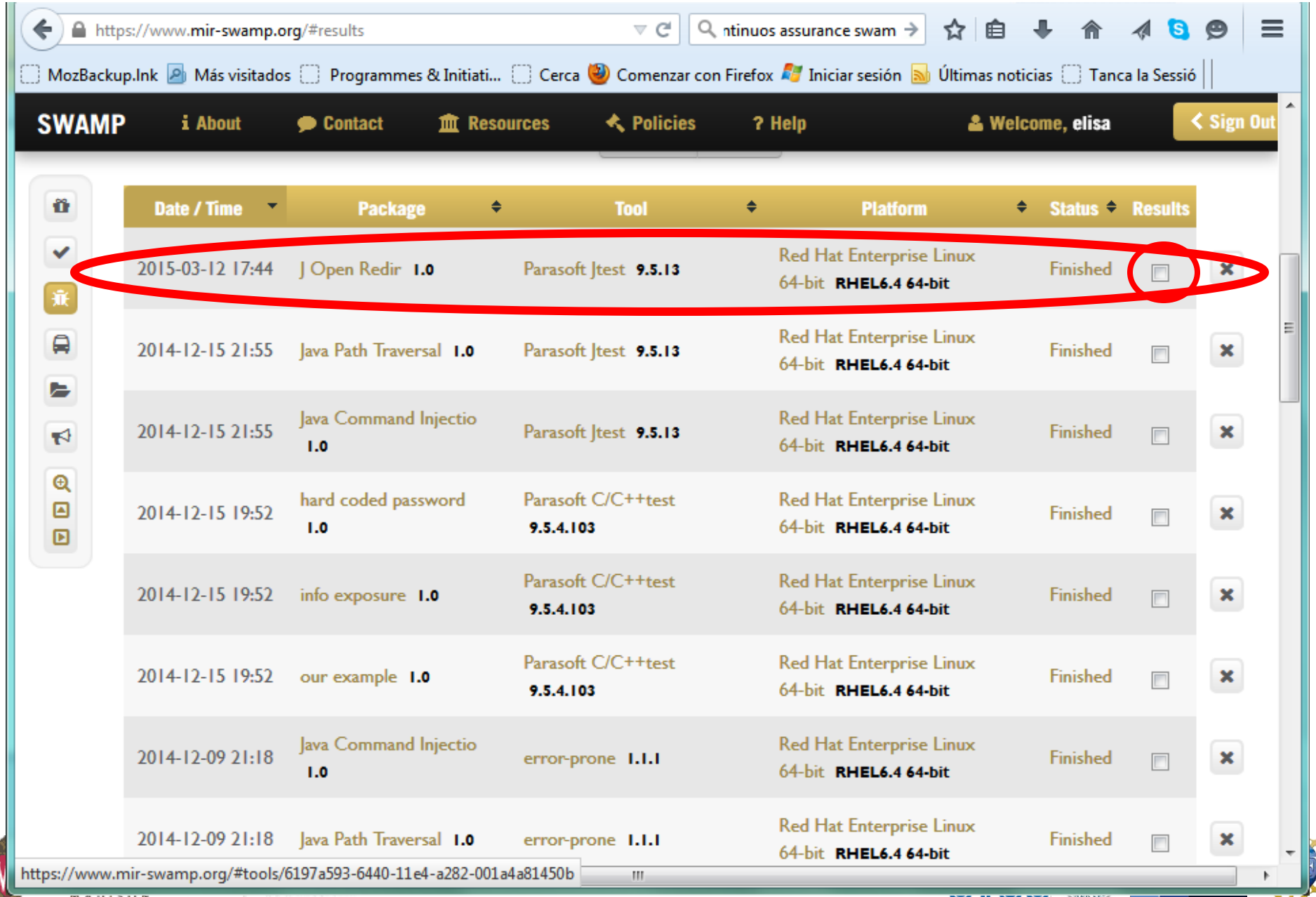
Run your Assessments



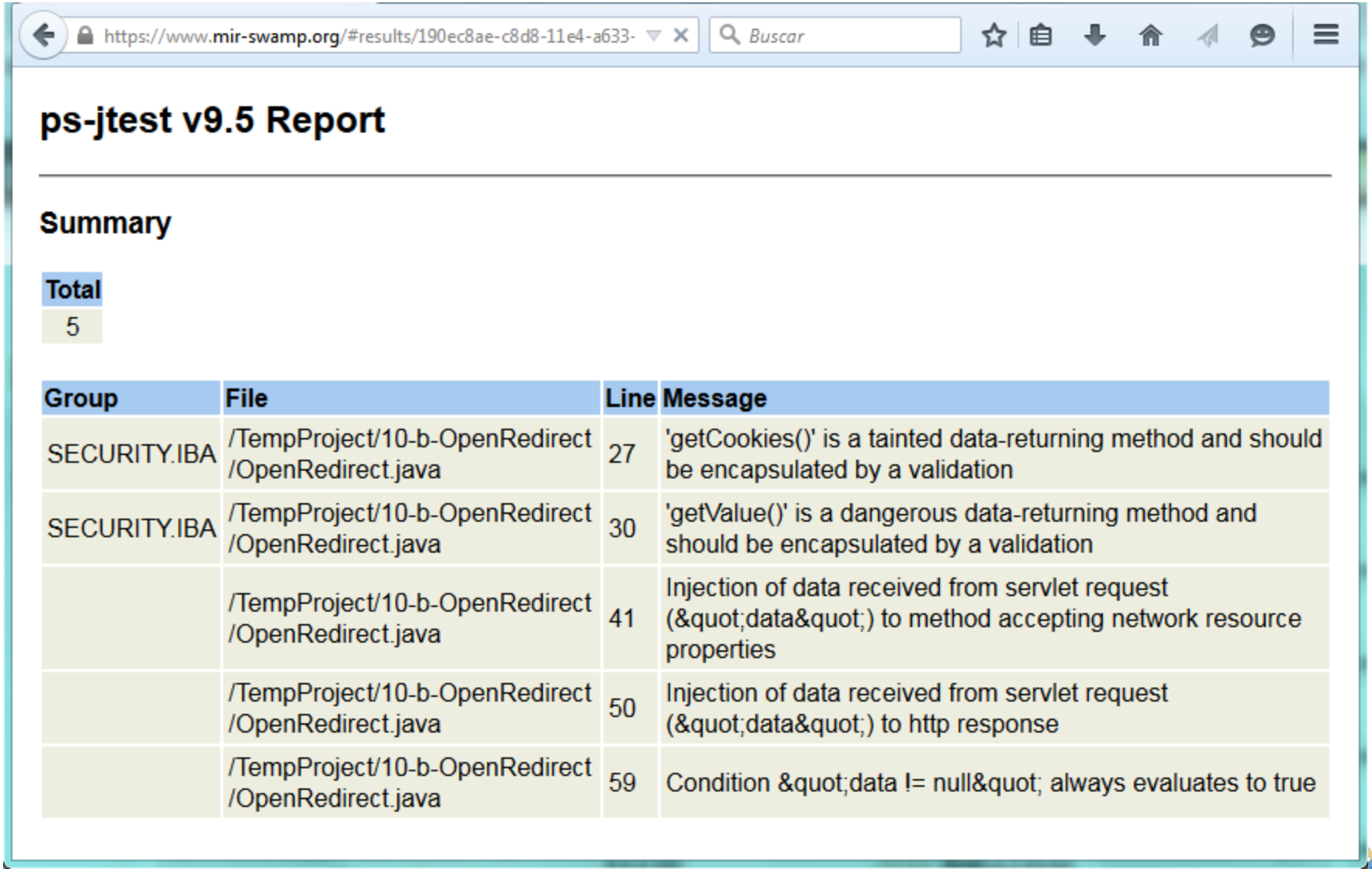
My Assessments



View your Results



My Results for Java: Jtest – J Open Redir - Native



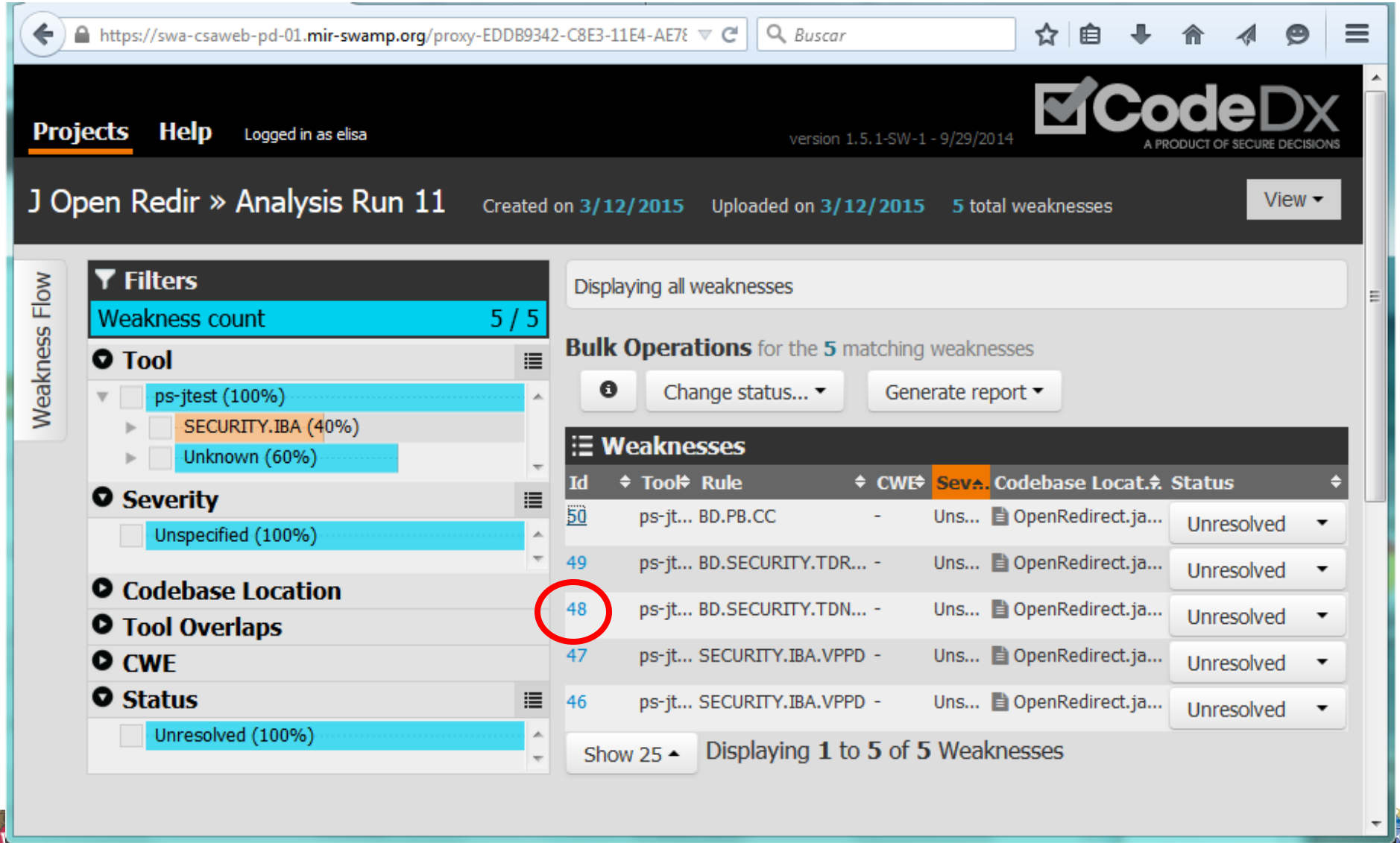
ps-jtest v9.5 Report

Summary

Total
5

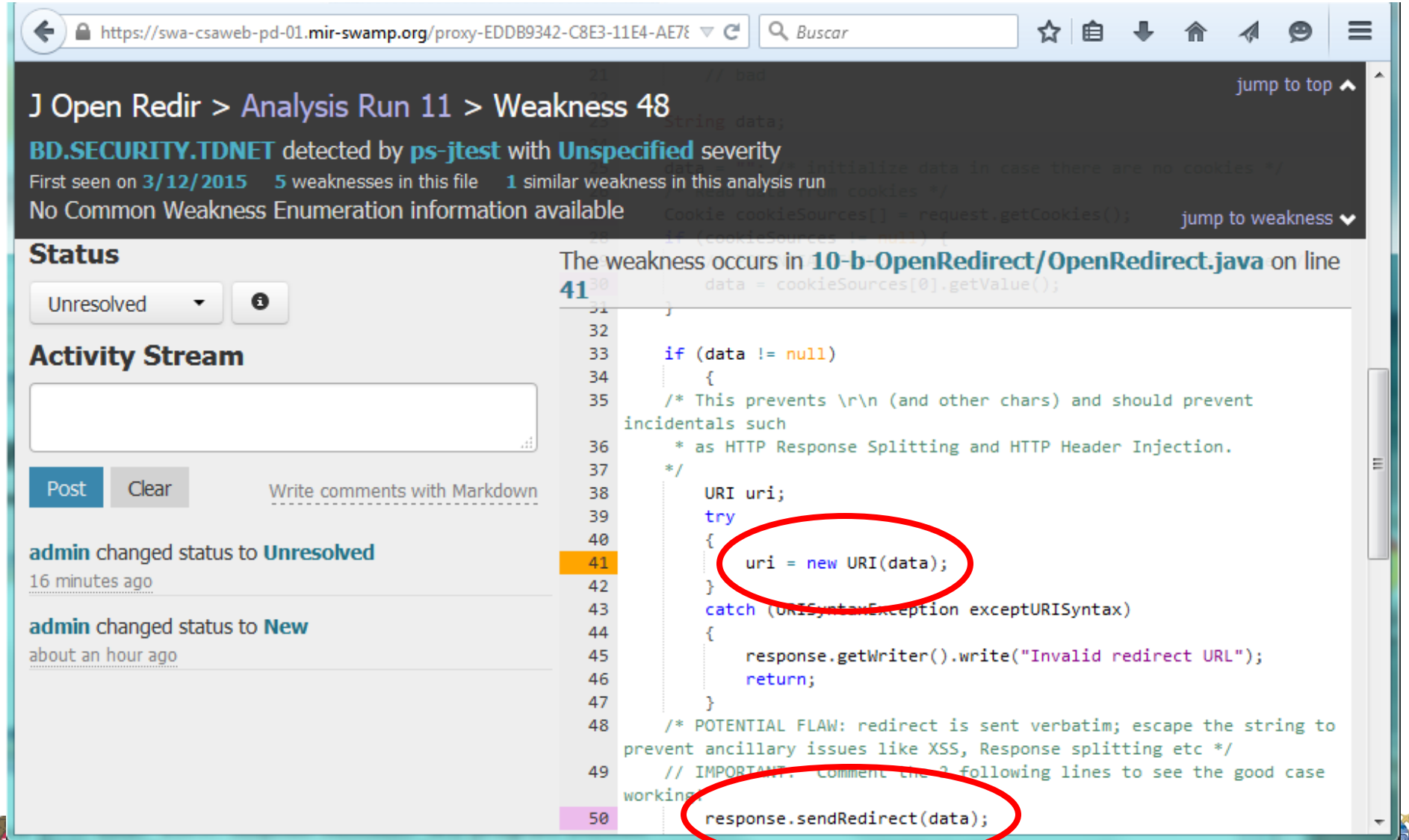
Group	File	Line	Message
SECURITY.IBA	/TempProject/10-b-OpenRedirect /OpenRedirect.java	27	'getCookies()' is a tainted data-returning method and should be encapsulated by a validation
SECURITY.IBA	/TempProject/10-b-OpenRedirect /OpenRedirect.java	30	'getValue()' is a dangerous data-returning method and should be encapsulated by a validation
	/TempProject/10-b-OpenRedirect /OpenRedirect.java	41	Injection of data received from servlet request ("data") to method accepting network resource properties
	/TempProject/10-b-OpenRedirect /OpenRedirect.java	50	Injection of data received from servlet request ("data") to http response
	/TempProject/10-b-OpenRedirect /OpenRedirect.java	59	Condition "data != null" always evaluates to true

My Results for Java: Jtest – J Open Redir - CodeDx



My Results for Java: Jtest – J Open Redir - CodeDx

My Results for Java: Jtest – J Open Redir - CodeDx



Questions?

Elisa Heymann

Elisa.Heymann@uab.es

Barton P. Miller

bart@cs.wisc.edu

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