Automated Assessment Tools Theory & Practice

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

August 2017

http://hdl.handle.net/2022/21723

Overview

- Very dangerous: Injection Attacks.
- Introduction to automated assessment tools.
- The SWAMP.
- Hands-on exercise in Java and the SWAMP.

Injection Attacks

Objectives

- Understand the general problem of injections.
- Understand what are SQL injections, and how to mitigate them.
- Understand what are Command injections, and how to mitigate them.

Injection Attacks

- Description
  - A string constructed with user input, that is then interpreted by another function, where the string is not parsed as expected
    - Command injection (in a shell)
    - Format string attacks (in printf/scanf)
    - SQL injection
    - Cross-site scripting or XSS (in HTML)
- General causes
  - Allowing metacharacters
  - Not properly neutralizing user data if metacharacters are allowed

SQL Injections

- User supplied values used in SQL command must be validated, quoted, or prepared statements must be used
- Signs of vulnerability
  - Uses a database mgmt system (DBMS)
  - Creates SQL statements at run-time
  - Inserts user supplied data directly into statement without validation

SQL Injections:
attacks and mitigations

- Dynamically generated SQL without validation or quoting is vulnerable

```
$u = " '; drop table t --";
$sth = $dbh->do("select * from t where u = '$u'");
```

Database sees <u>two</u> statements:

```
select * from t where u = ' '; drop table t --'
```

- Use ***prepared statements*** to mitigate

```
$sth = $dbh->do("select * from t where u = ?", $u);
```

  - SQL statement template and value sent to database
  - No mismatch between intention and use

Successful SQL Injection Attack

2. DB Queried

```
SELECT * FROM members
WHERE u='admin' AND p='' OR 'x'='x'
```

3. Returns all row of table members

1. User sends malicious data

```
user="admin"; pwd="'OR 'x'='x"
```

```
boolean Login(String user, String pwd)  {
    boolean loggedIn = false;
    conn = pool.getConnection( );
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT * FROM members"
                           + "WHERE u='" + user
                           + "' AND p='" + pwd + "'");
    if (rs.next())
        loggedIn = true;
}
```

4. System grants access

```
Login() returns true
```

Mitigated SQL Injection Attack

```
SELECT * FROM members WHERE u = ?1 AND p = ?2
        ?1 = "admin"    ?2 = "' OR 'x'='x"
```

2. DB Queried

3. Returns null set

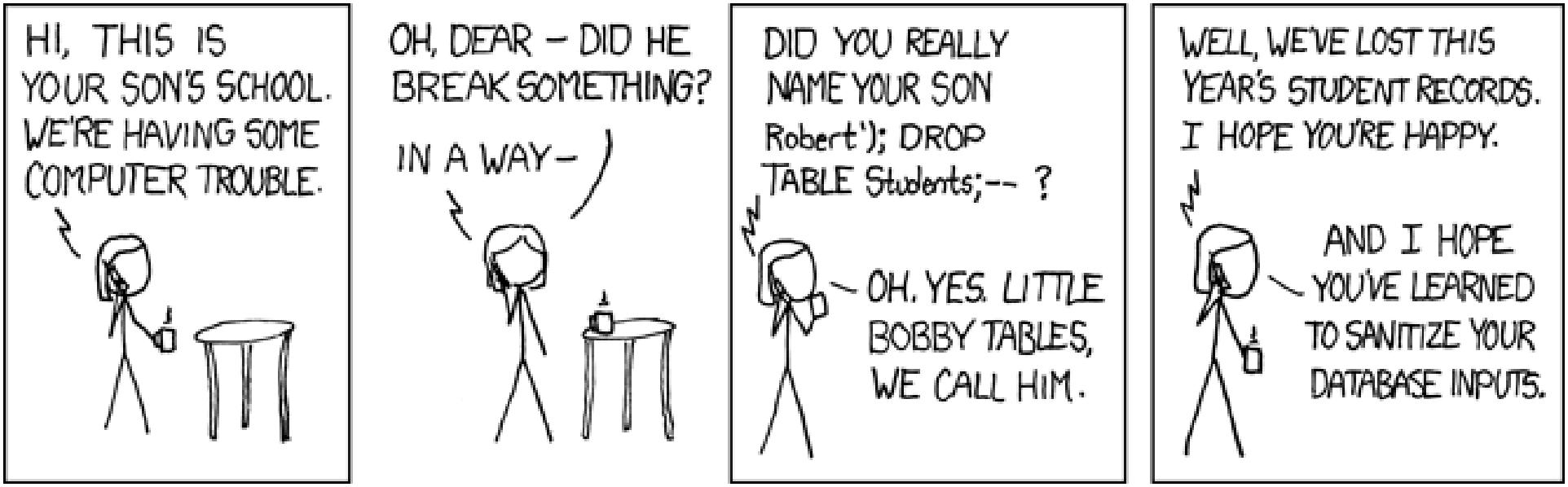
1. User sends malicious data

```
user="admin"; pwd="' OR 'x'='x"
```

```
boolean Login(String user, String pwd)  {
    boolean loggedIn = false;
    conn = pool.getConnection( );
    PreparedStatement pstmt = conn.prepareStatement(
        "SELECT * FROM members WHERE u = ? AND p = ?");
    pstmt.setString( 1, user);
    pstmt.setString( 2, pwd);
    ResultSet results = pstmt.executeQuery( );
    if (rs.next())
        loggedIn = true;
}
```

4. System does not grant access

```
Login() returns false
```



http://xkcd.com/327

Command Injections

- User supplied data used to create a string that is the interpreted by command shell such as `/bin/sh`
- Signs of vulnerability
  - Use of `popen`, or `system`
  - `exec` of a shell such as `sh`, or `csh`
  - Argument injections, allowing arguments to begin with `"-"` can be dangerous
- Usually done to start another program
  - That has no C API
  - Out of laziness

Command Injection Mitigations

- Check user input for metacharacters
- Neutralize those that can't be eliminated or rejected
  - replace single quotes with the four characters, `'\''`, and enclose each argument in single quotes
- Use `fork`, drop privileges and `exec` for more control
- Avoid if at all possible
- Use C API if possible

Command Argument Injections

- A string formed from user supplied input that is used as a command line argument to another executable
- Does not attack shell, attacks command line of program started by shell
- Need to fully understand command line interface
- If value should not be an option
  - Make sure it doesn't start with a -
  - Place after an argument of -- if supported

Perl Command Injection Danger Signs

- `open(F, $filename)`
  - Filename is a tiny language besides opening
    - Open files in various modes
    - Can start programs
    - `dup` file descriptors
  - If `$filename` is `"rm -rf /|"`, you probably won't like the result
  - Use separate mode version of open to eliminate vulnerability

Perl Command Injection Danger Signs

- **Vulnerable to shell interpretation**

```
open(C, "$cmd|")          open(C, "-|", $cmd)
open(C, "|$cmd")          open(C, "|-", $cmd)
`$cmd`                    qx/$cmd/
system($cmd)
```

- **Safe from shell interpretation**

```
open(C, "-|", @argList)
open(C, "|-", @cmdList)
system(@argList)
```

Perl Command Injection Examples

- `open(CMD, "|/bin/mail -s $sub $to");`
  - Bad if `$to` is `"badguy@evil.com; rm -rf /"`
- `open(CMD, "|/bin/mail -s '$sub' '$to'");`
  - Bad if `$to` is `"badguy@evil.com'; rm -rf /'"`
-
```
($qSub = $sub) =~ s/'/'\\''/g;
($qTo = $to)   =~ s/'/'\\''/g;
open(CMD, "|/bin/mail -s '$qSub' '$qTo'");
```
  - Safe from command injection
- `open(cmd, "|-", "/bin/mail", "-s", $sub, $to);`
  - Safe and simpler: use this whenever possible.

Eval Injections

- A string formed from user supplied input that is used as an argument that is interpreted by the language running the code
- Usually allowed in scripting languages such as Perl, sh and SQL
- In Perl `eval($s)` and `s/$pat/$replace/ee`
  - `$s` and `$replace` are evaluated as perl code

Ruby Command Injection Danger Signs

Functions prone to injection attacks:

- `Kernel.system(os command)`
- `Kernel.exec(os command)`
- `` `os command` `` # back tick operator
- `%x[os command]`
- `eval(ruby code)`

Python Command Injection Danger Signs

Functions prone to injection attacks:

- `exec()` # dynamic execution of Python code
- `eval()` # returns the value of an expression or # code object
- `os.system()` # execute a command in a subshell
- `os.popen()` # open a pipe to/from a command
- `execfile()` # reads & executes Python script from # a file.
- `input()` # equivalent to eval(raw_input())
- `compile()` # compile the source string into a code # object that can be executed

Successful OS Injection Attack

1. User sends malicious data

```
hostname="x.com;rm -rf /*"
```

2. Application uses nslookup to get DNS records

```
String rDomainName(String hostname)  {
  …
  String cmd = "/usr/bin/nslookup " + hostname;
  Process p = Runtime.getRuntime().exec(cmd);
  …
```

3. System executes `nslookup x.com;rm -rf /*`

4. All files possible are deleted

Mitigated OS Injection Attack

JAVA

1. User sends malicious data

```
hostname="x.com;rm -rf /*"
```

2. Application uses nslookup only if input validates

```
String rDomainName(String hostname)  {
  …
  if (hostname.matches("[A-Za-z][A-Za-z0-9.-]*")) {
    String cmd = "/usr/bin/nslookup " + hostname);
    Process p = Runtime.getRuntime().exec(cmd);
  } else {
    System.out.println("Invalid host name");
    …
```

3. System returns error `"Invalid host name"`

Code Injection

Cause

- Program generates source code from template
- User supplied data is injected in template
- Failure to neutralized user supplied data
  - Proper quoting or escaping
  - Only allowing expected data
- Source code compiled and executed

Very dangerous – high consequences for getting it wrong: arbitrary code execution

Code Injection Vulnerability

1. logfile – name's value is user controlled

```
name = John Smith
name = ');import os;os.system('evilprog');#
```

Read logfile

2. Perl log processing code – uses Python to do real work

```
%data = ReadLogFile('logfile');
PH = open("|/usr/bin/python");
print PH "import LogIt\n";w
while (($k, $v) = (each %data)) {
  if ($k eq 'name')  {
  print PH "LogIt.Name('$v')";
}
```

Start Python, program sent on stdin

3. Python source executed – 2nd LogIt executes arbitrary code

```
import LogIt;
LogIt.Name('John Smith')
LogIt.Name('');import os;os.system('evilprog');#')
```

Code Injection Mitigated

1. logfile – name's value is user controlled

```
name = John Smith
name = ');import os;os.system('evilprog');#
```

2. Perl log processing code – use QuotePyString to safely create string literal

```
%data = ReadLogFile('logfile');
PH = open("|/usr/bin/python");
print PH "import LogIt\n";w
while (($k, $v) = (each %data)) {
  if ($k eq 'name')  {
    $q = QuotePyString($v);
    print PH "LogIt.Name($q)";
}
```

```
sub QuotePyString  {
  my $s = shift;
  $s =~ s/\\/\\\\/g;     # \  ➔ \\
  $s =~ s/'/\\'/g;       # '  ➔ \'
  $s =~ s/\n/\\n/g;      # NL ➔ \n
  return "'$s'";         # add quotes
}
```

3. Python source executed – 2nd LogIt is now safe

```
import LogIt;
LogIt.Name('John Smith')
LogIt.Name('\');import os;os.system(\'evilprog\');#')
```

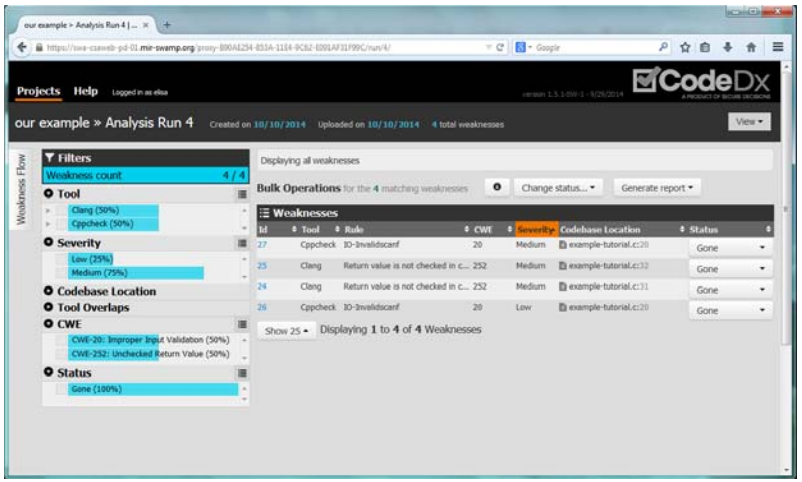
Introduction to Automated Assessment Tools

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

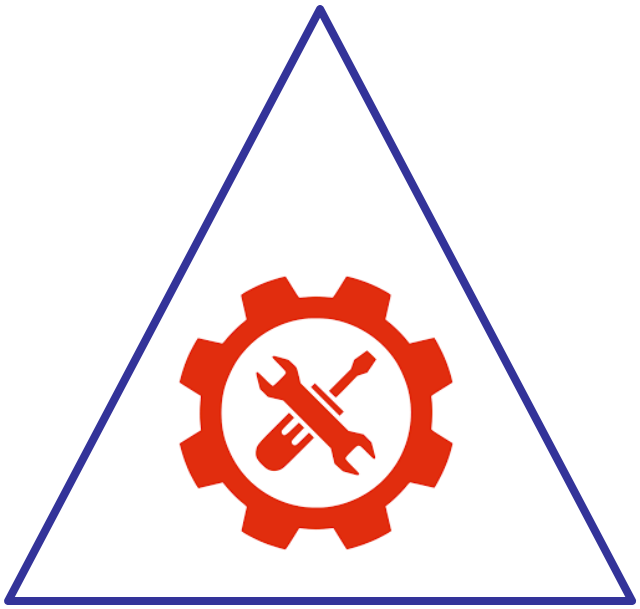
- Designed to analyze source code or binaries to help find security flaws.
- The source code may contain inadvertent or deliberate weaknesses that could lead to security vulnerabilities in the executable versions of the application program.
- Better to use them from the beginning of the software development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

```
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;
}
```

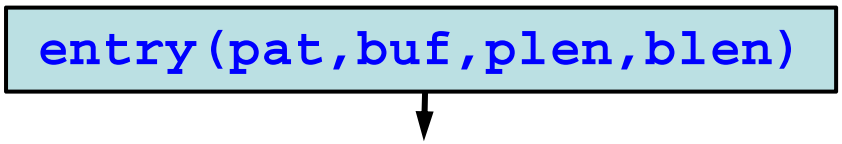
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

    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;
}
```



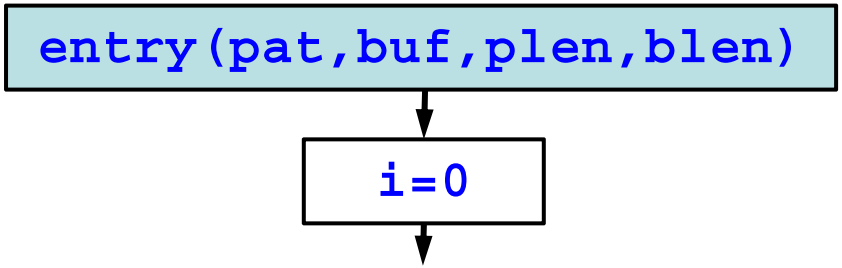
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

   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;
}
```



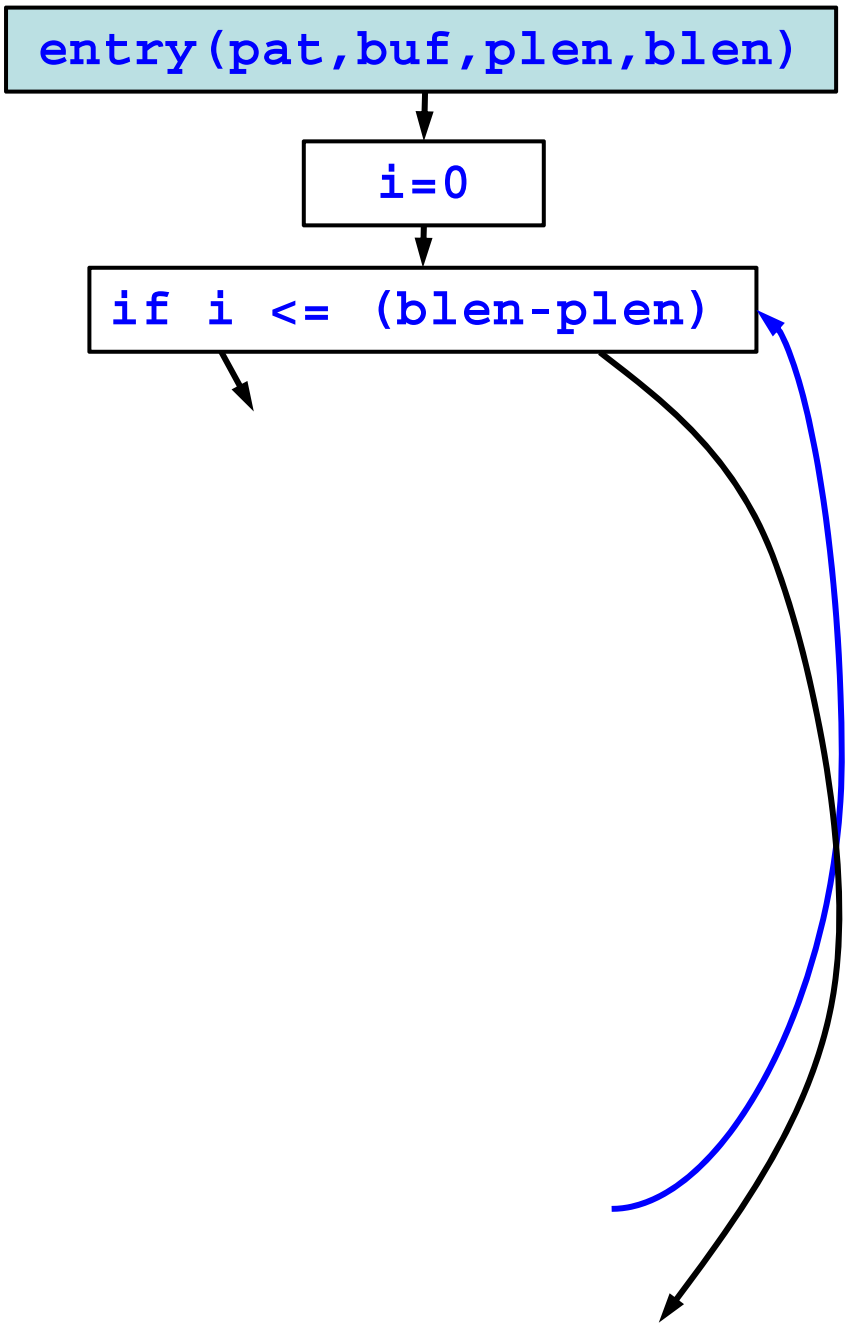
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

    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;
}
```



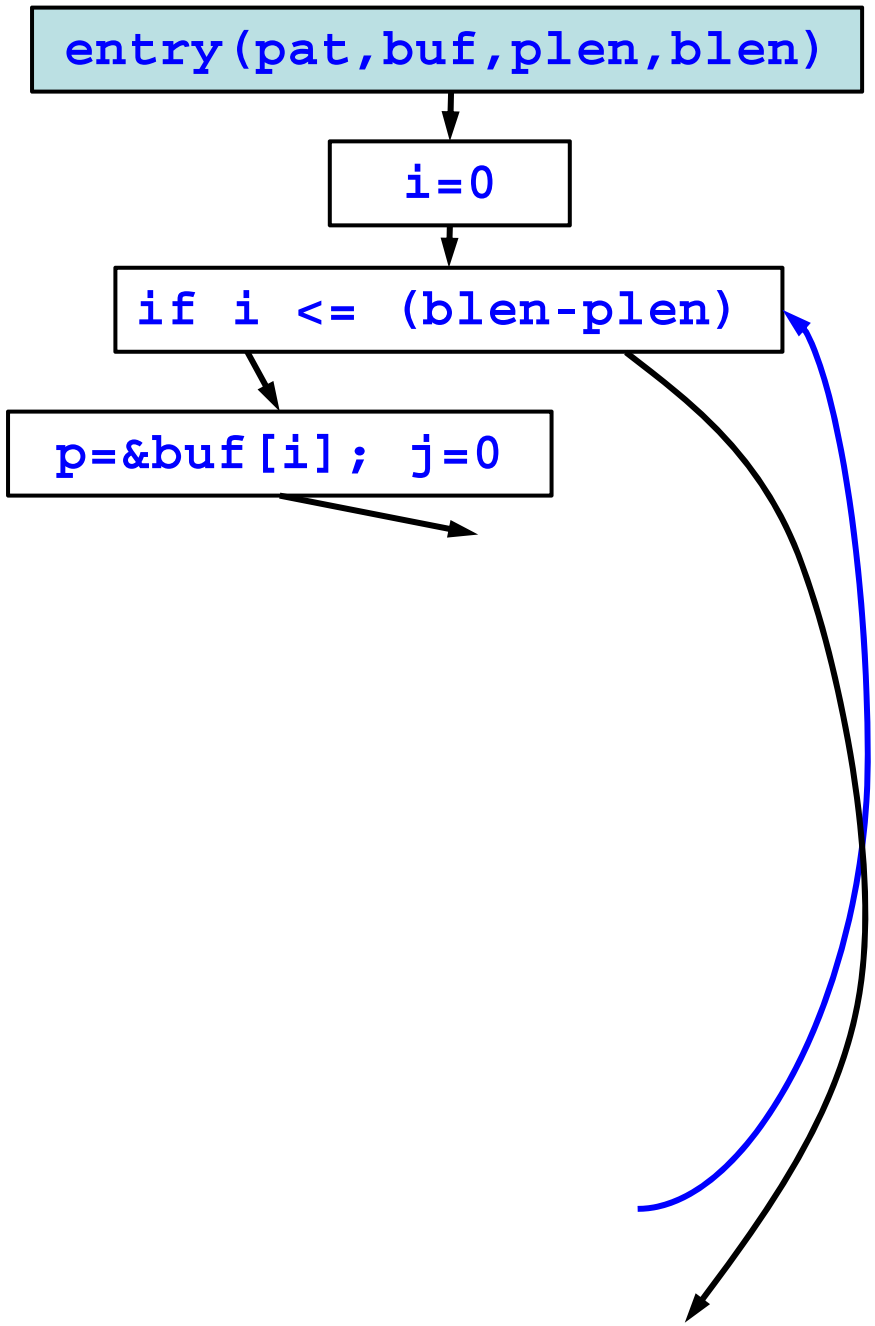
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

    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;
}
```



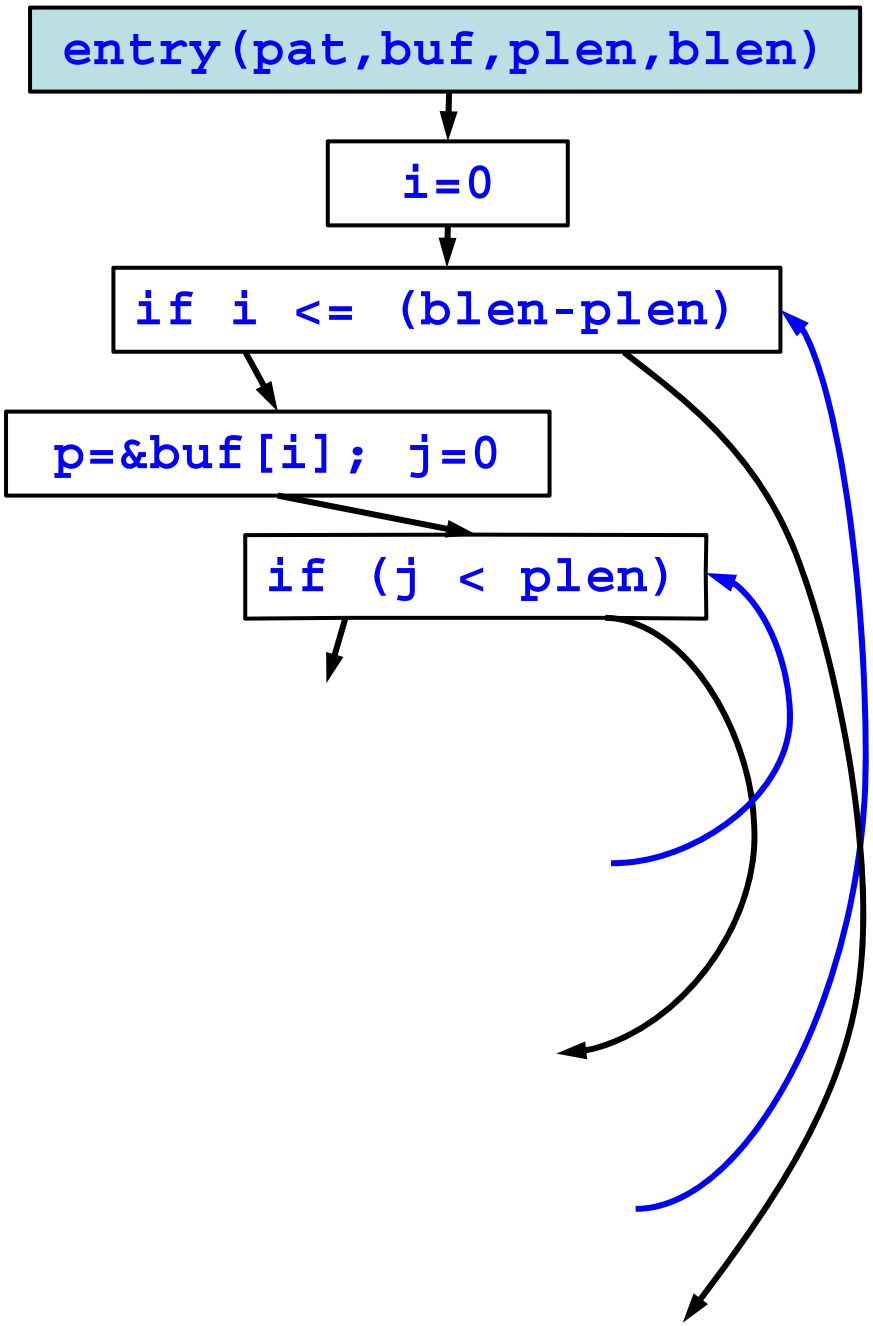
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

   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;
}
```



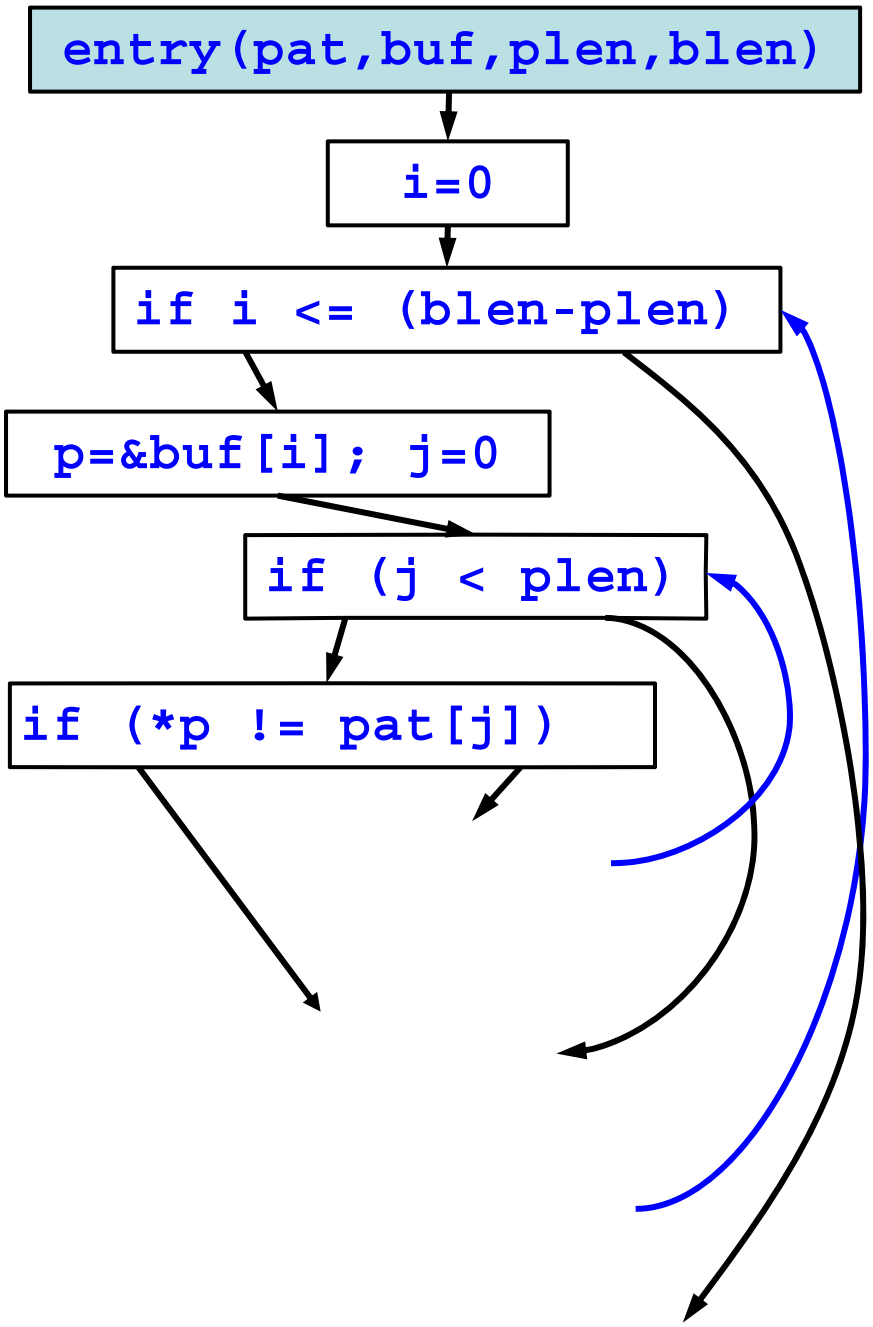
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

    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;
        i++;
    }

    return -1;
}
```



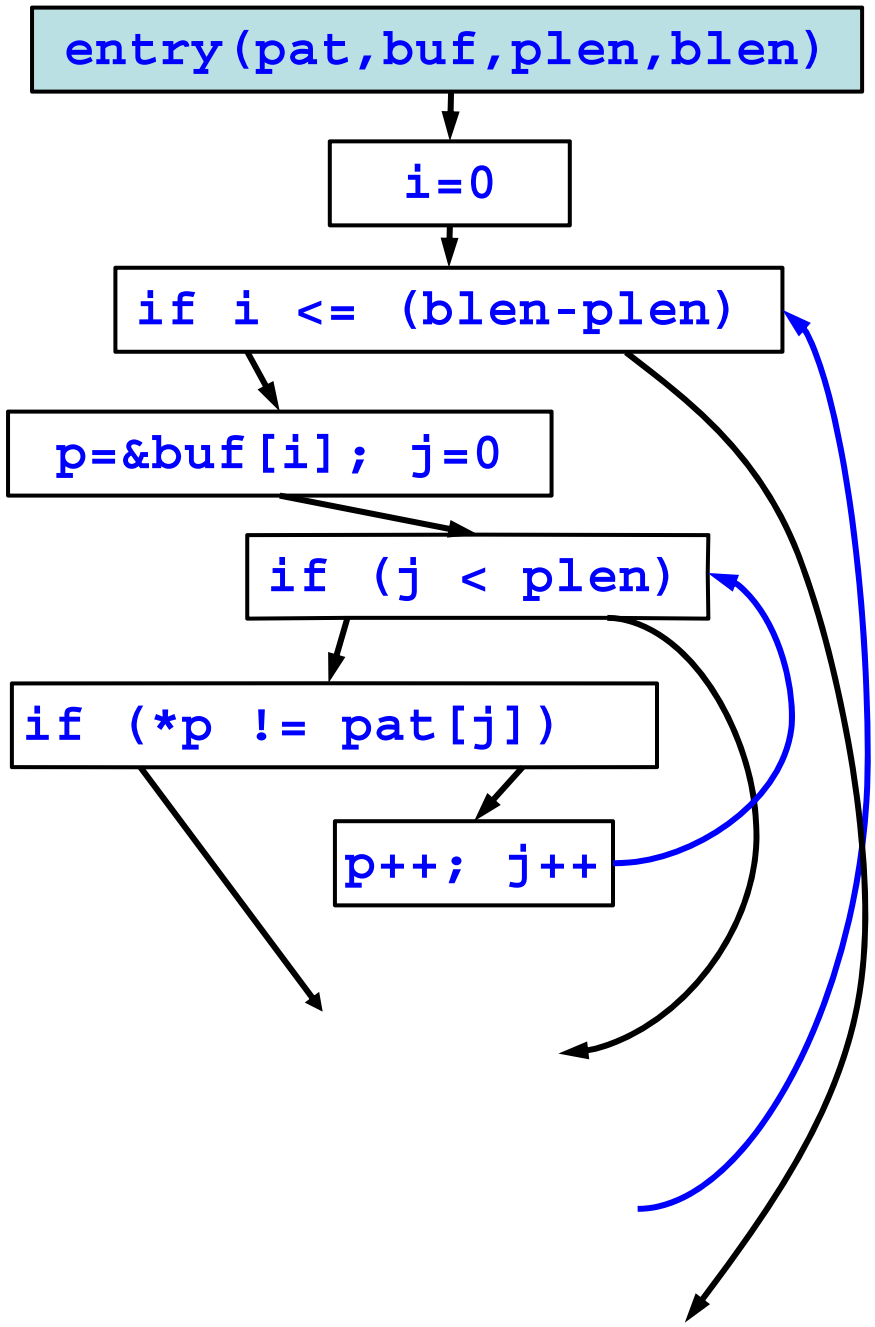
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

    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;
}
```



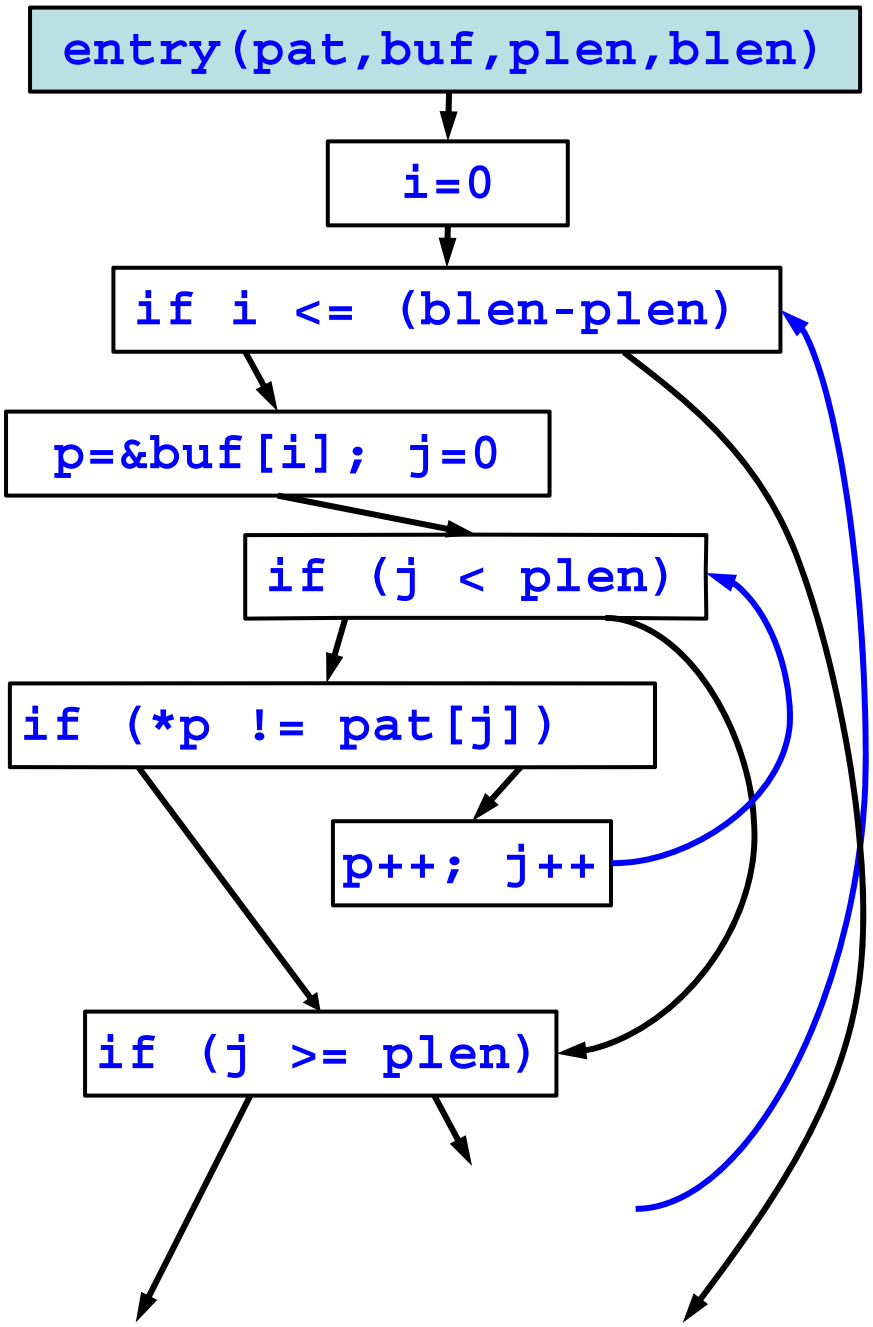
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

    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;
}
```



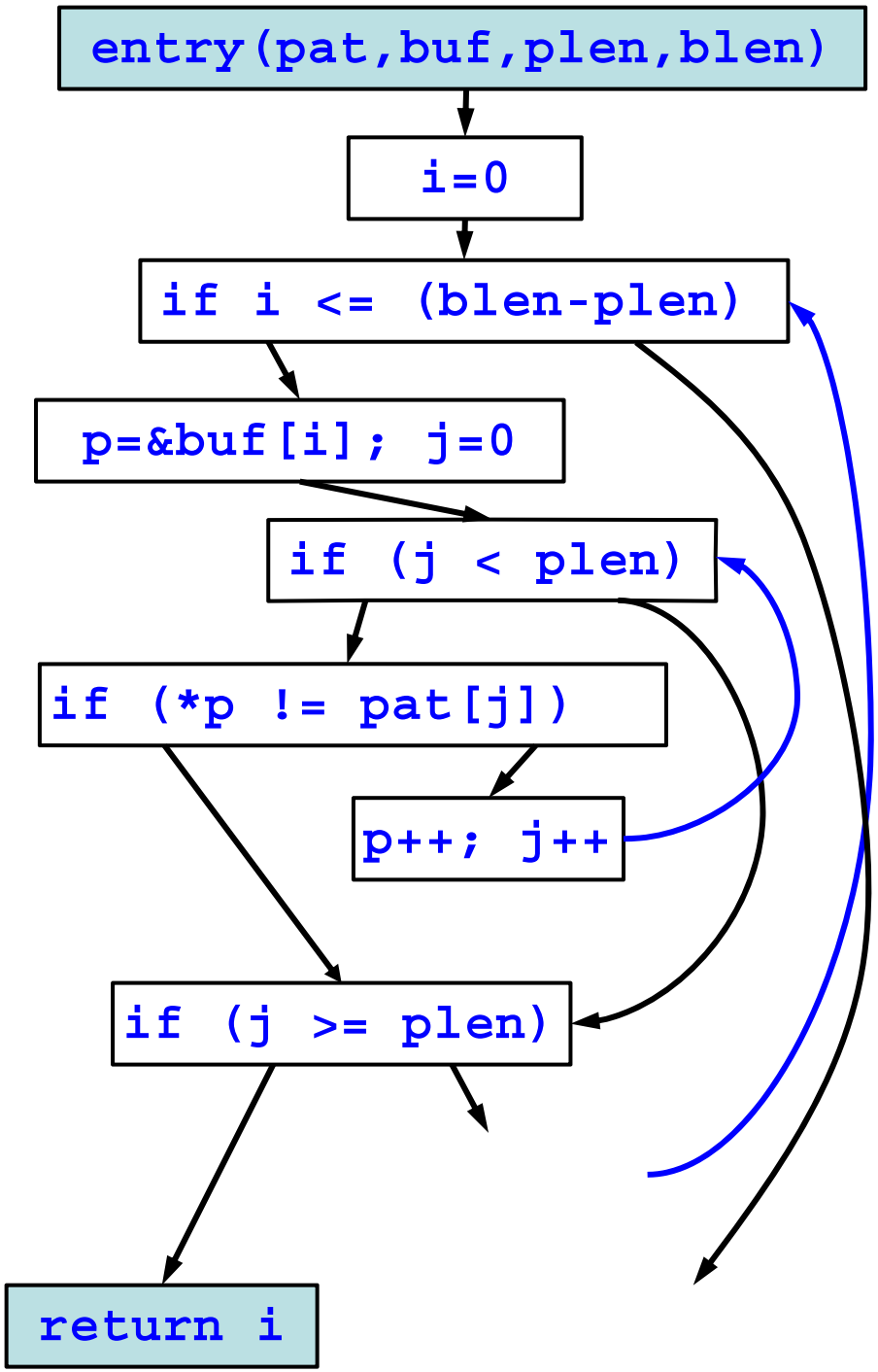
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

    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;
}
```



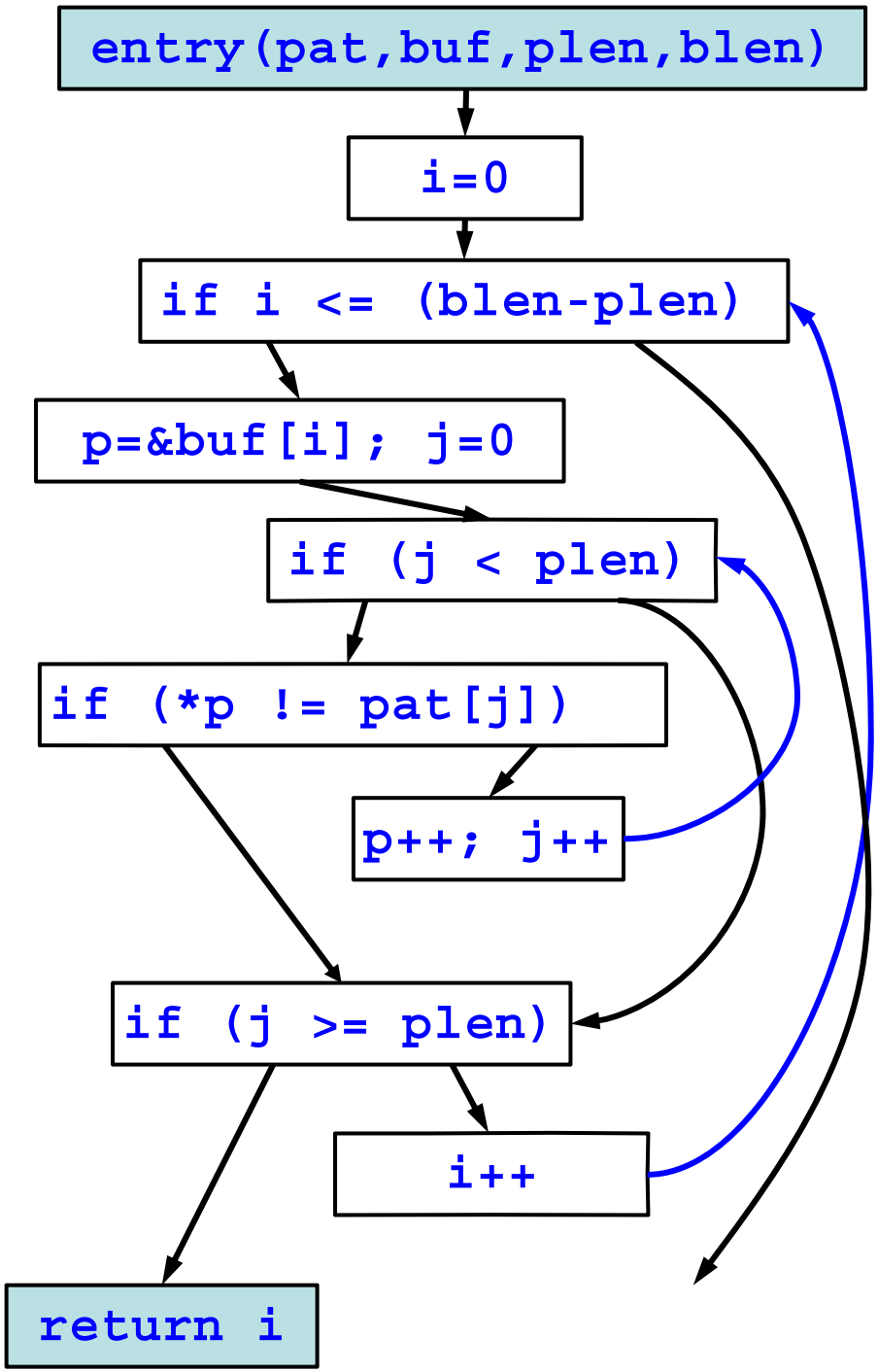
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

    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;
}
```



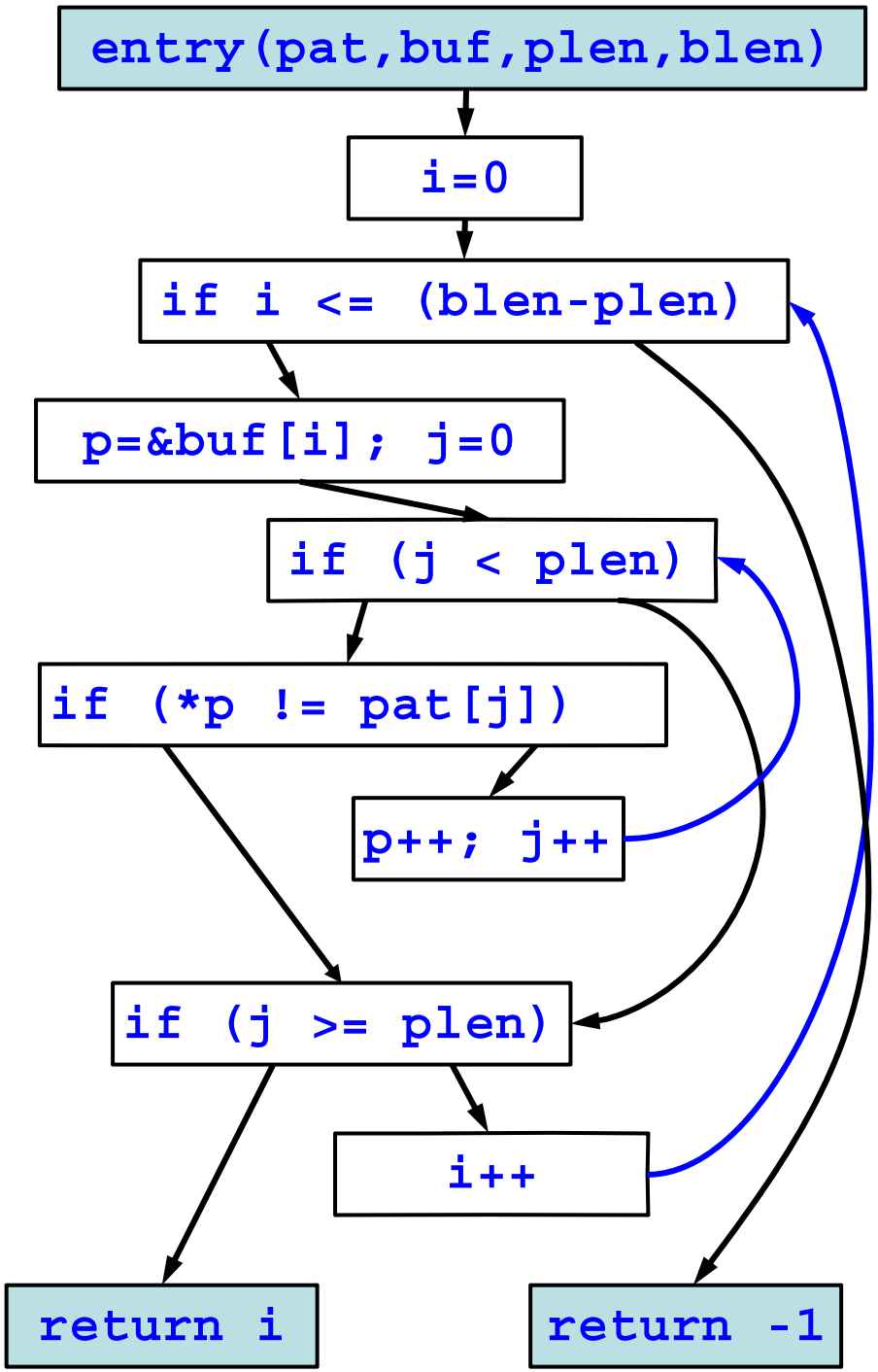
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

    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;
}
```



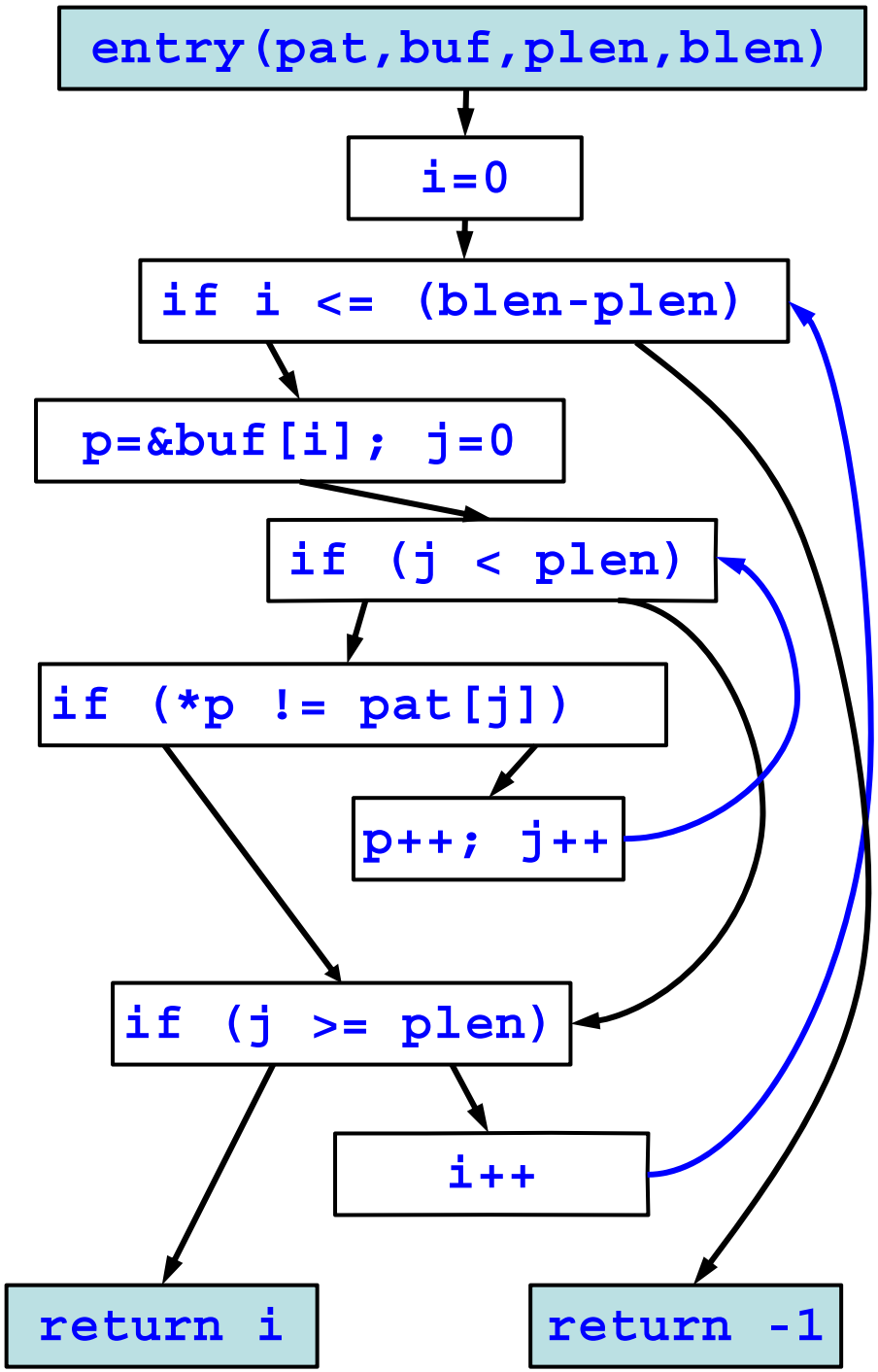
```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

    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;
       i++;
    }

    return -1;
}
```



Data Flow Analysis

Goal: Is this code safe?

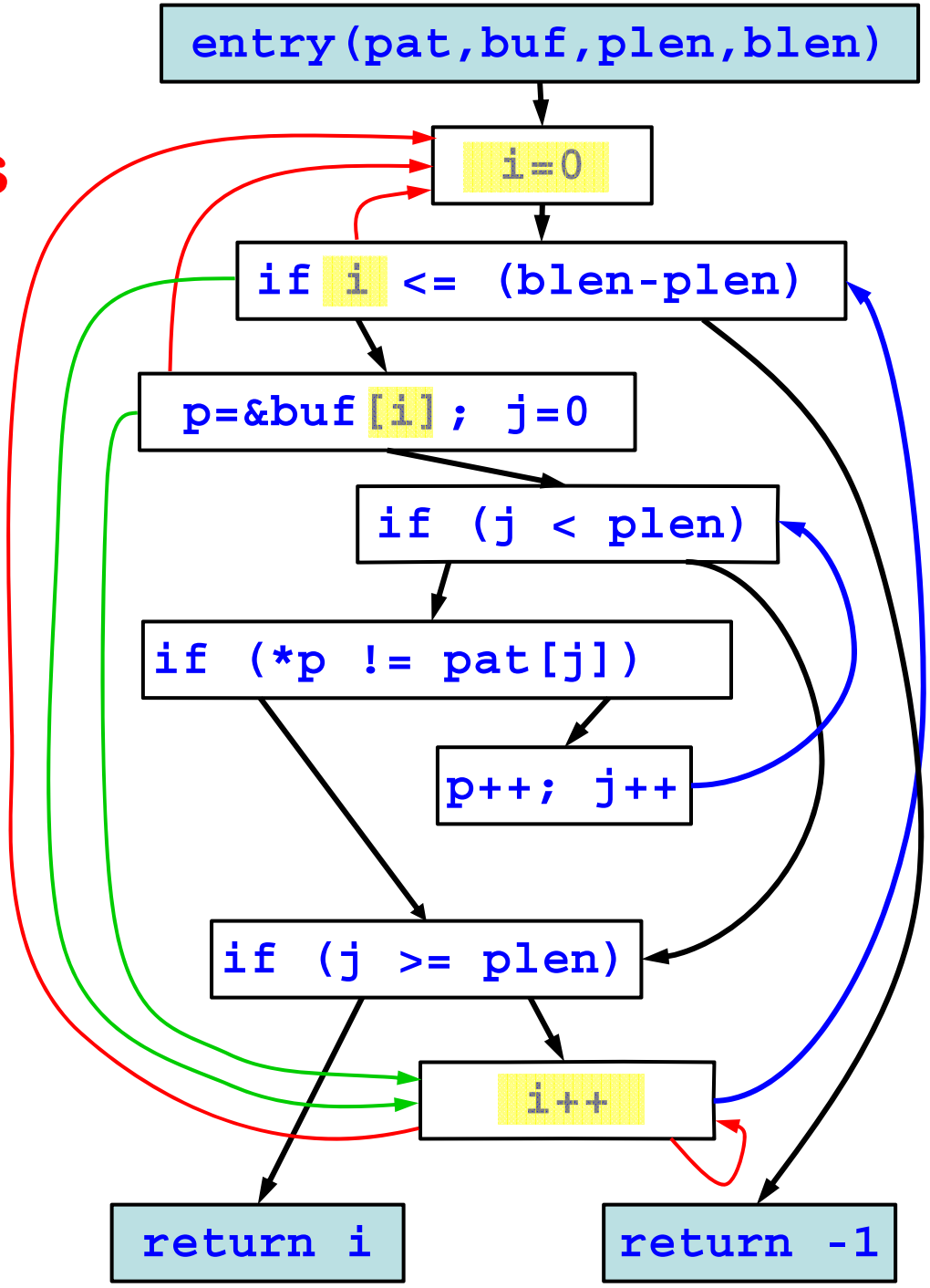
Subgoal:

Do we violate the borders of `buf` and `pat`?

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

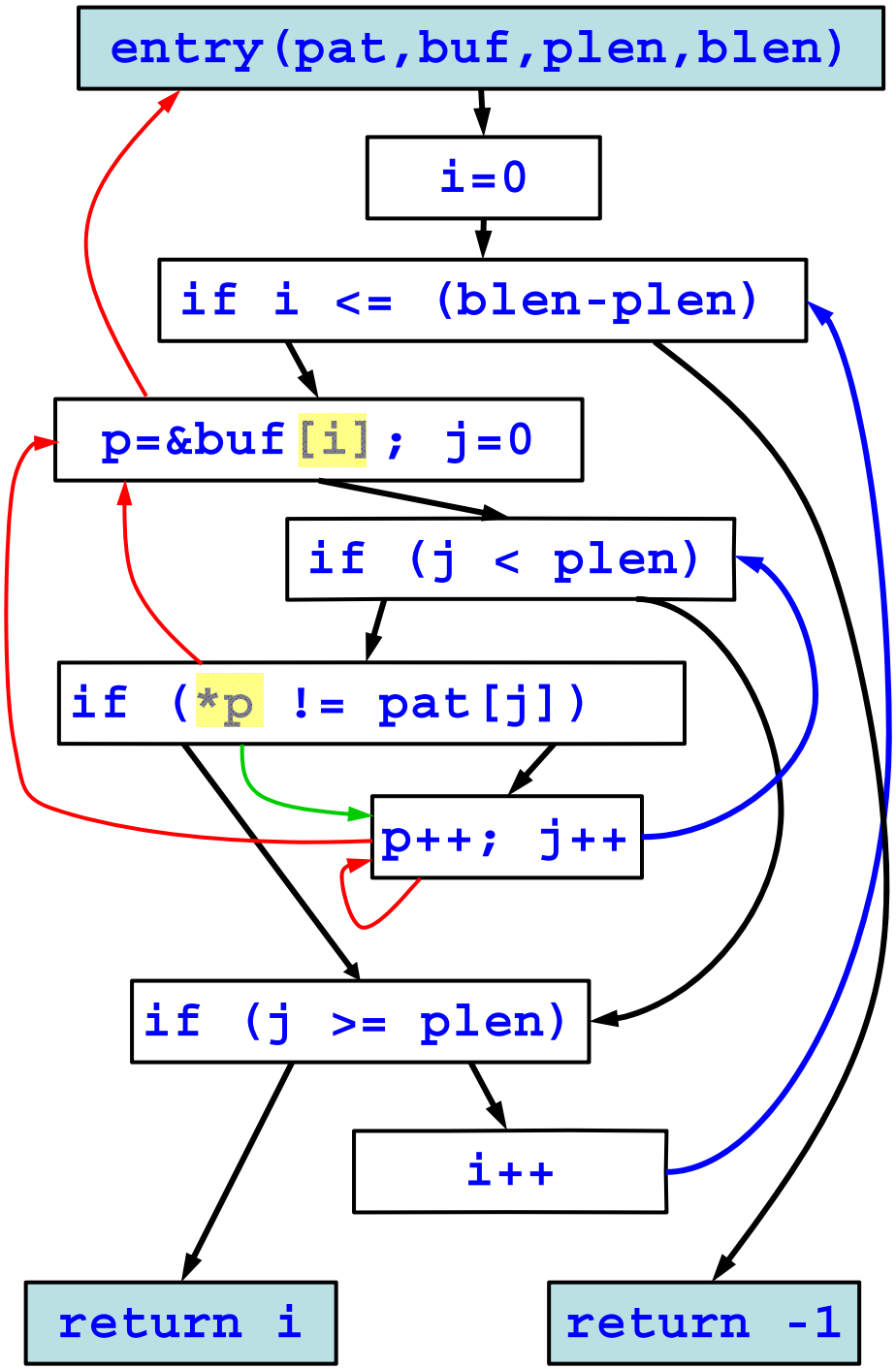
Data Flow Analysis

- **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]`.**



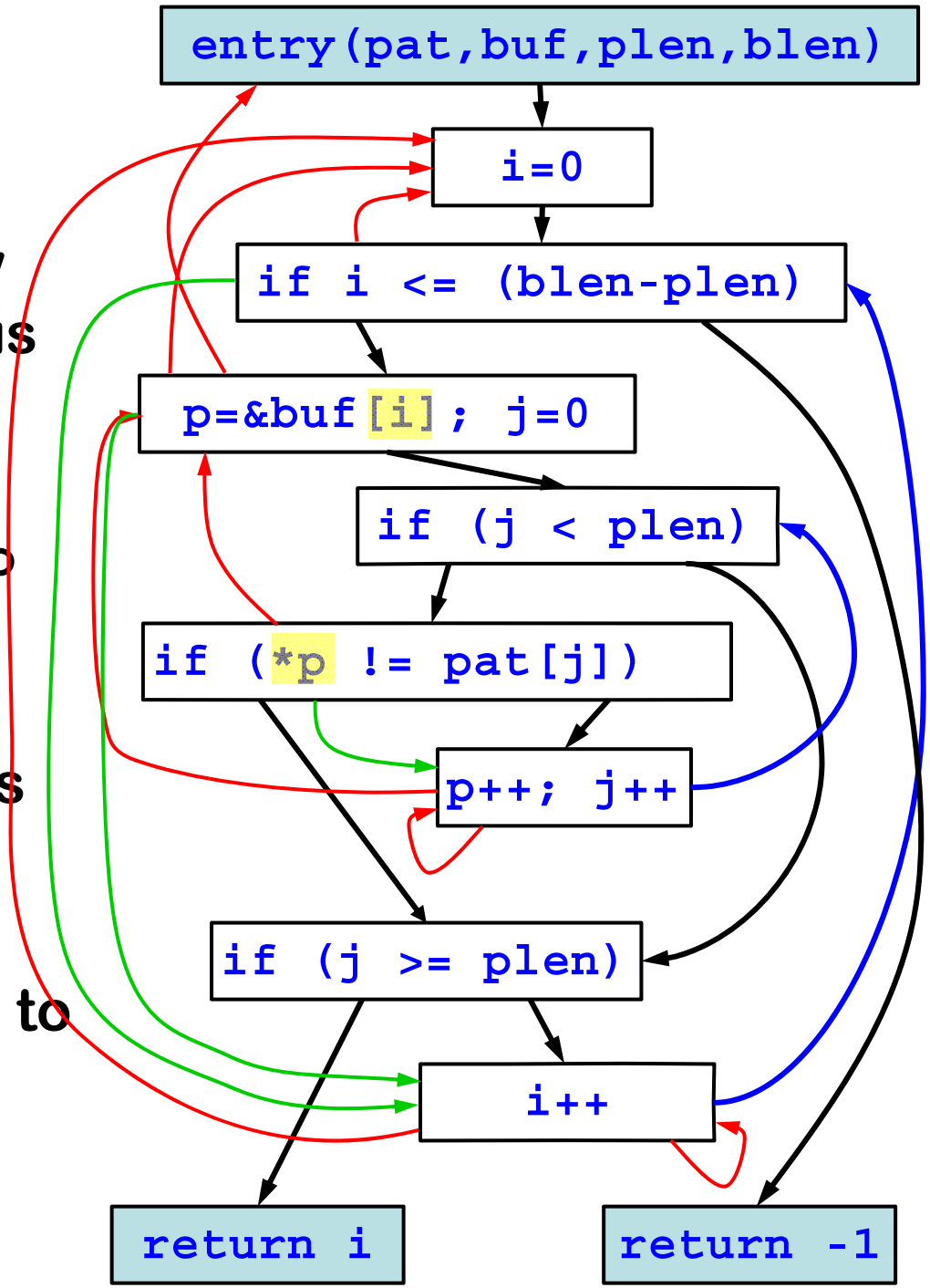
Data Flow Analysis

- 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.



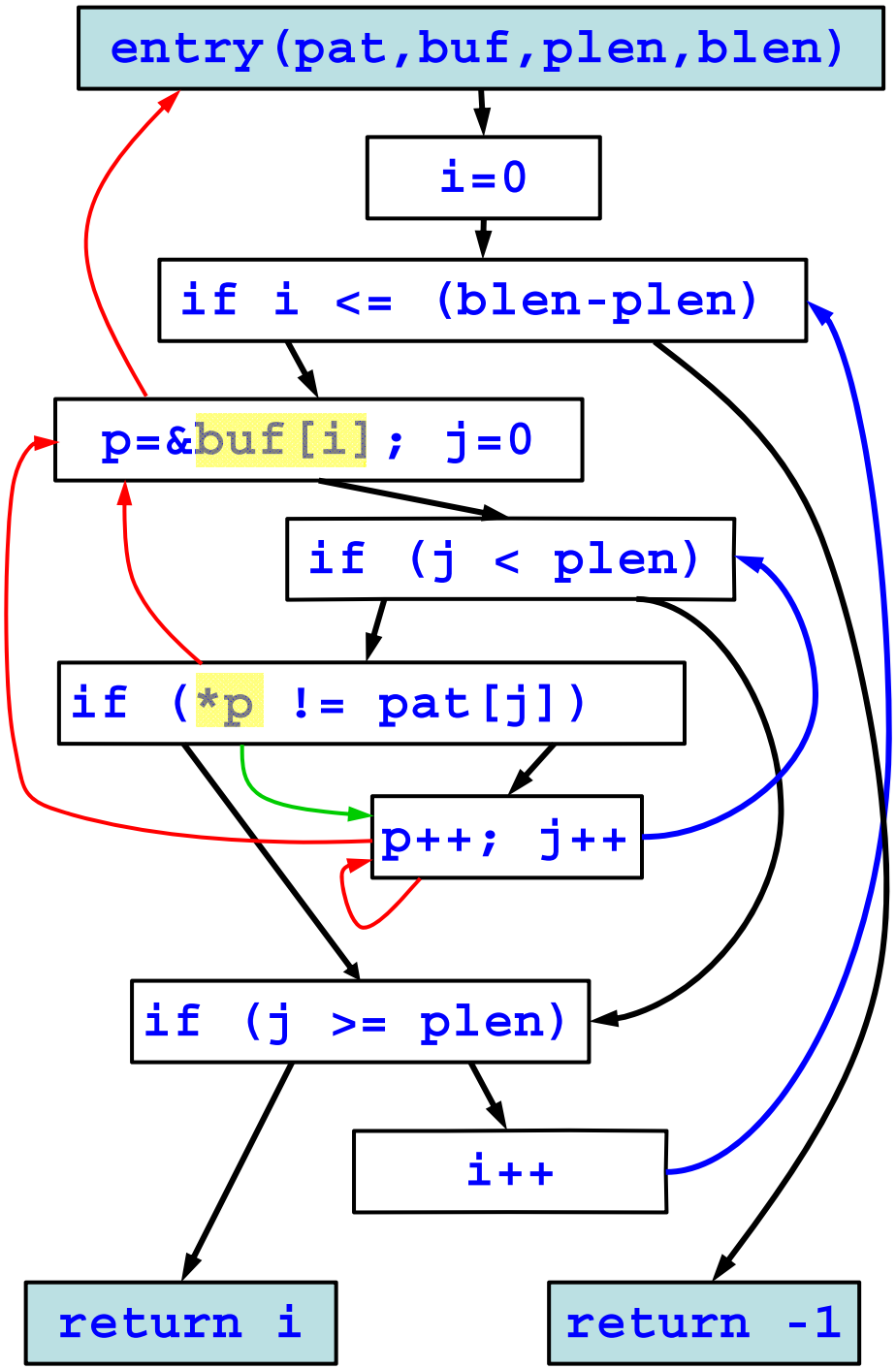
Data Flow Analysis

- 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.



```
int Find(char *pat, char *buf,
         unsigned int plen,
         unsigned int blen) {

   int i, j;
   char *p;

   i = 0;                                  i:[0,0]

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

   return -1;
}
```

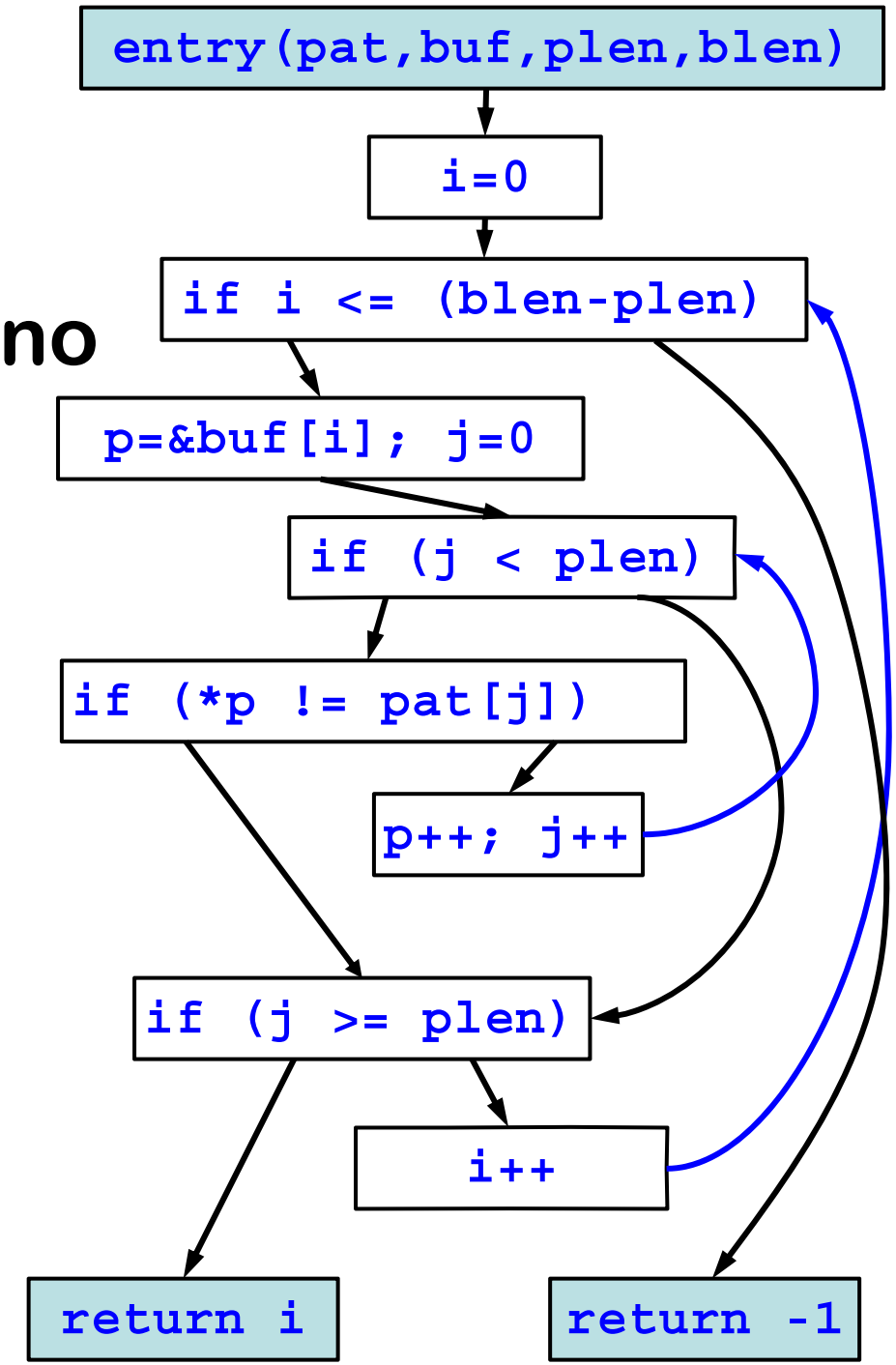
The goal is to understand the range of values for each variable:

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

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 the strong and weak points of using specific tools.

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)

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

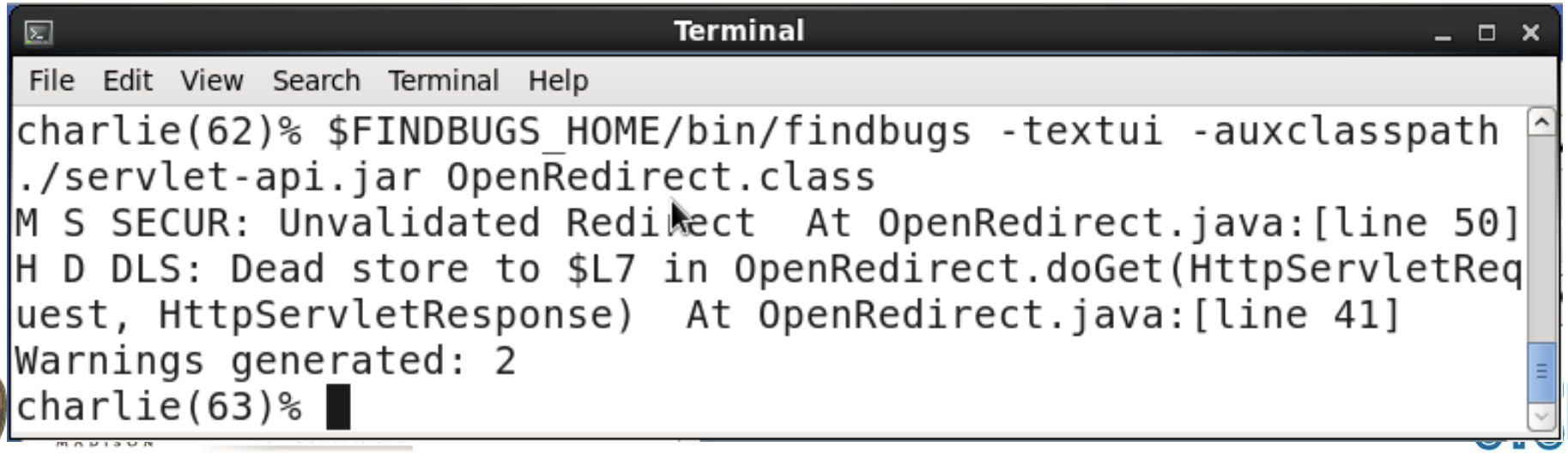
1. Define FINDBUGS_HOME in the environment.
2. Install the Find Security Bugs plugin.
3. Learn the command line instructions and also use the graphical interface.
4. Command line interface:

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

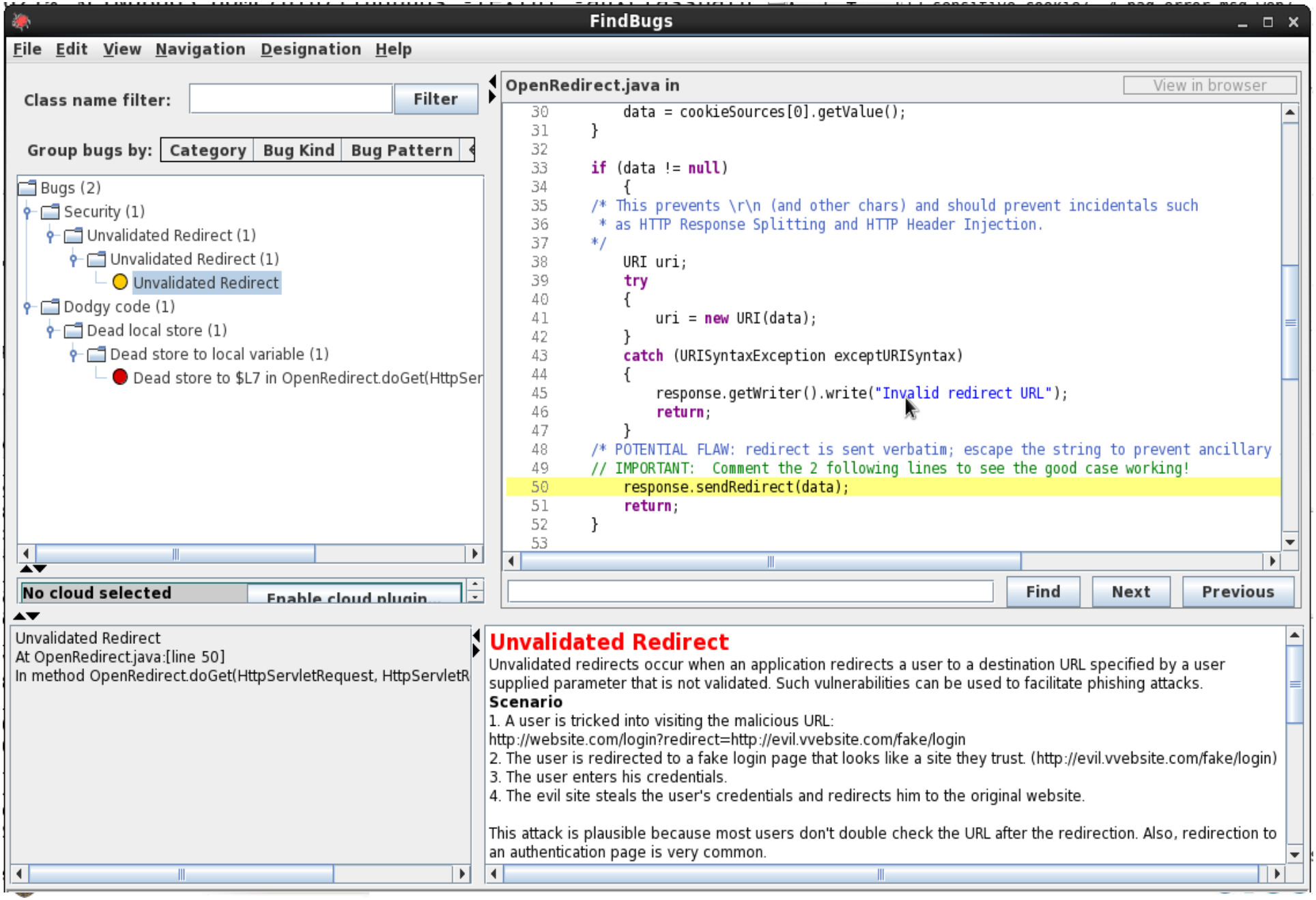
5. Graphic Interface: `java -jar $FINDBUGS_HOME/lib/findbugs.jar -gui`

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

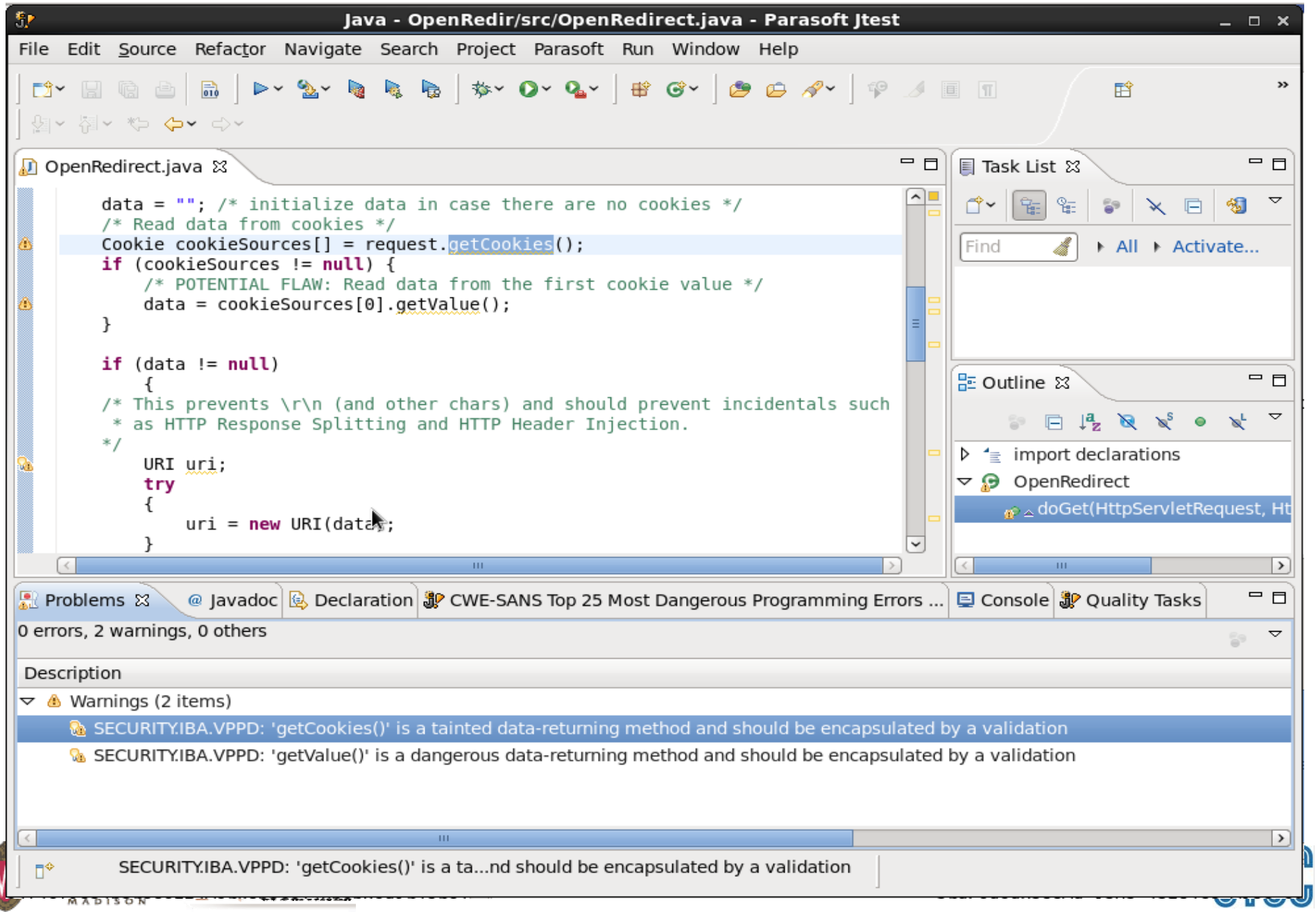
Create the OpenRedir project.

Include servlet-api.jar in the OpenRedir project.

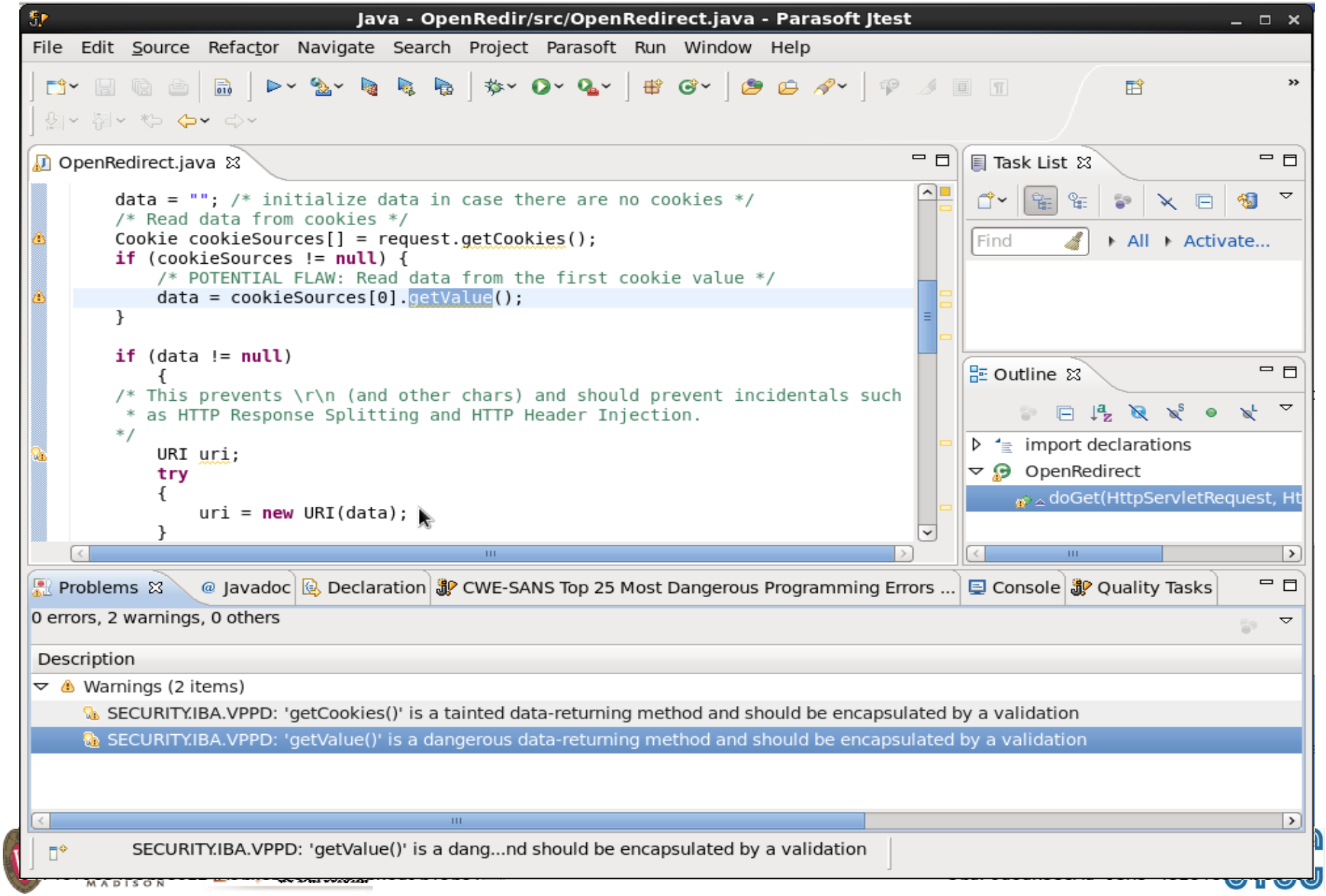
```
cp OpenRedirect.java
~elisa/parasoft/workspace1/OpenRedir/src
```

- **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.**

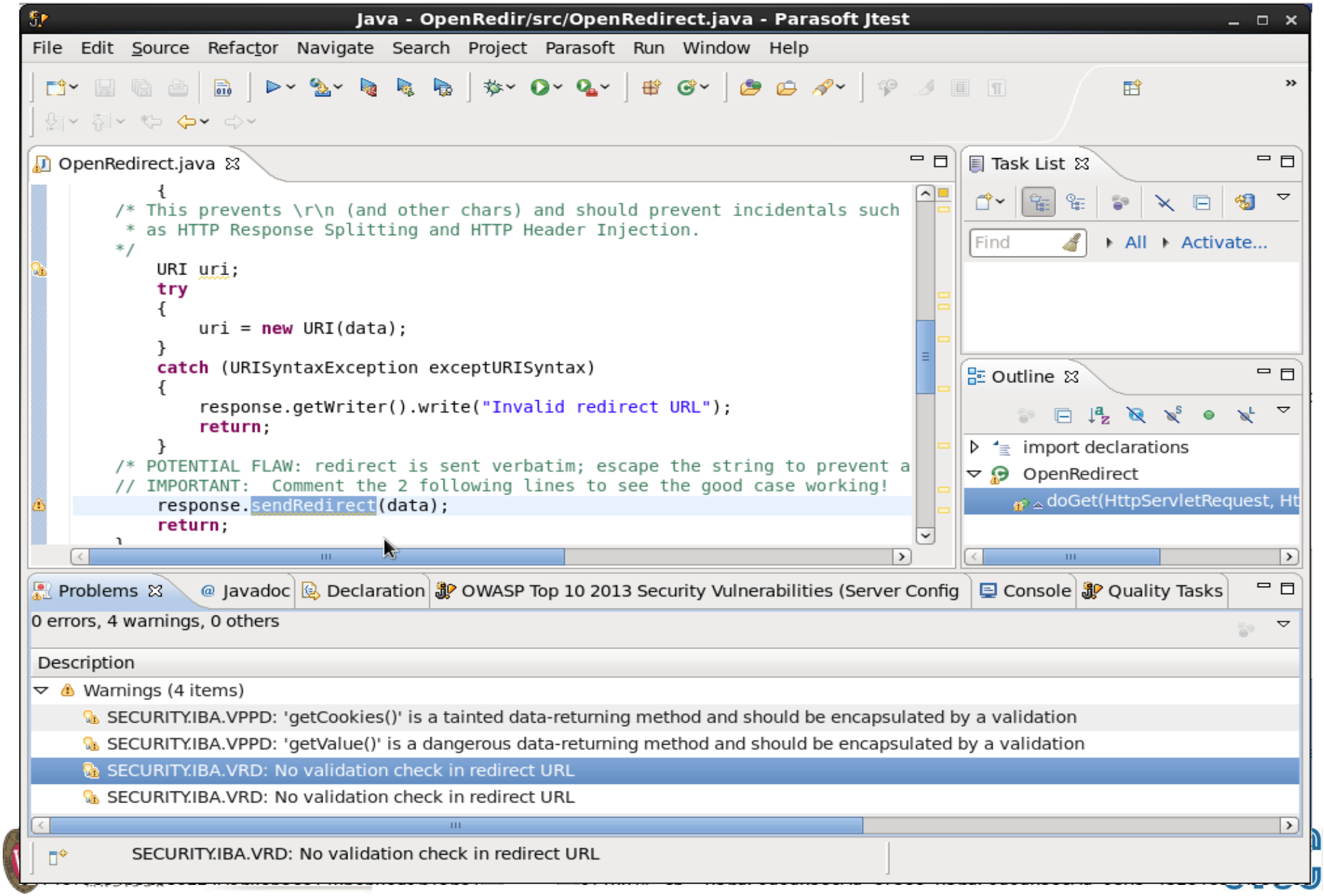
Jtest. Open Redirect



Jtest. Open Redirect



Jtest. Open Redirect



Roadmap

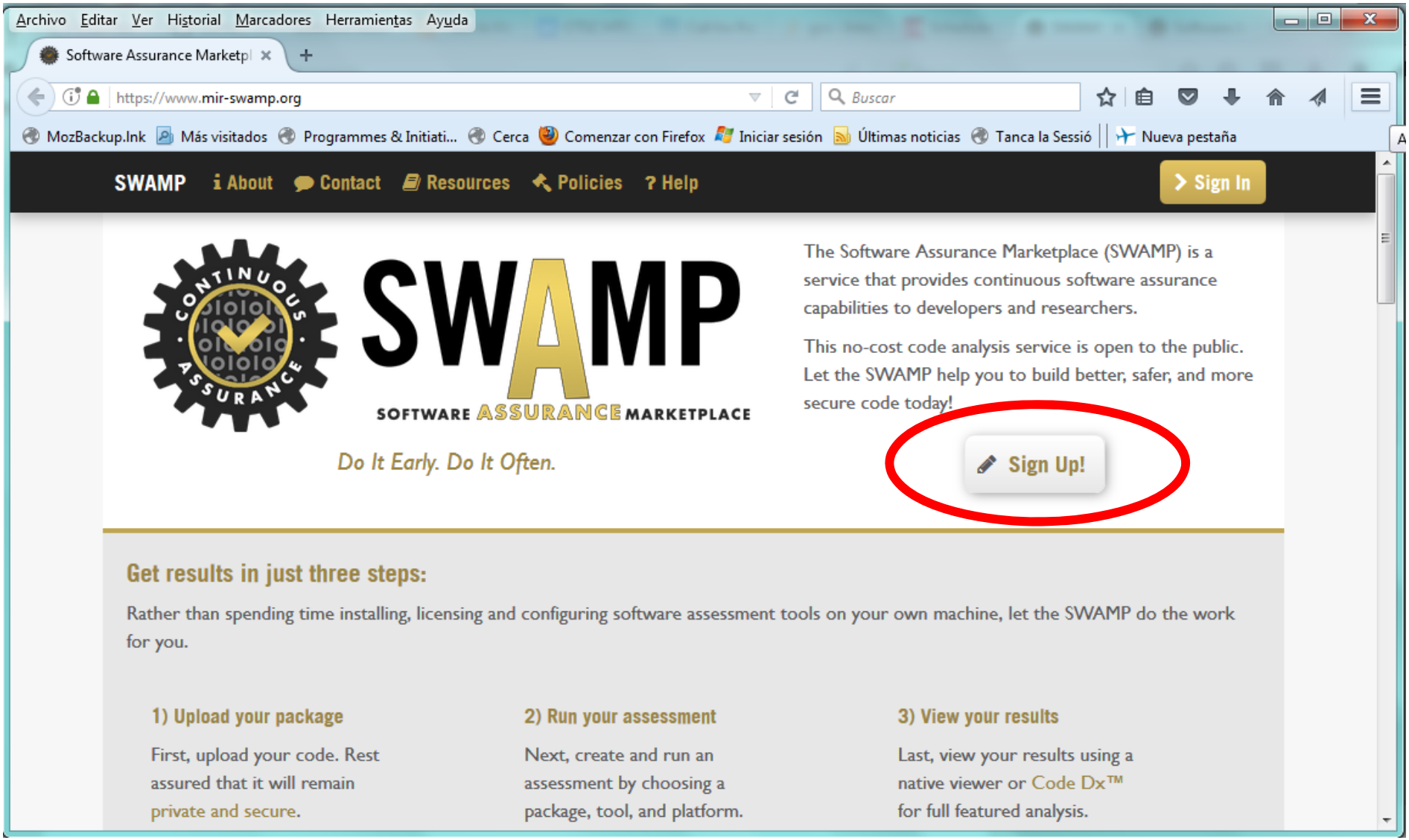
- What is the SWAMP?
- Using the SWAMP
  - Register
  - Create a project
  - Upload your software package
  - Run your assessment
  - View the results

https://www.swampinabox.org/doc/SWAMP-User-Manual.pdf

Getting Started with the SWAMP

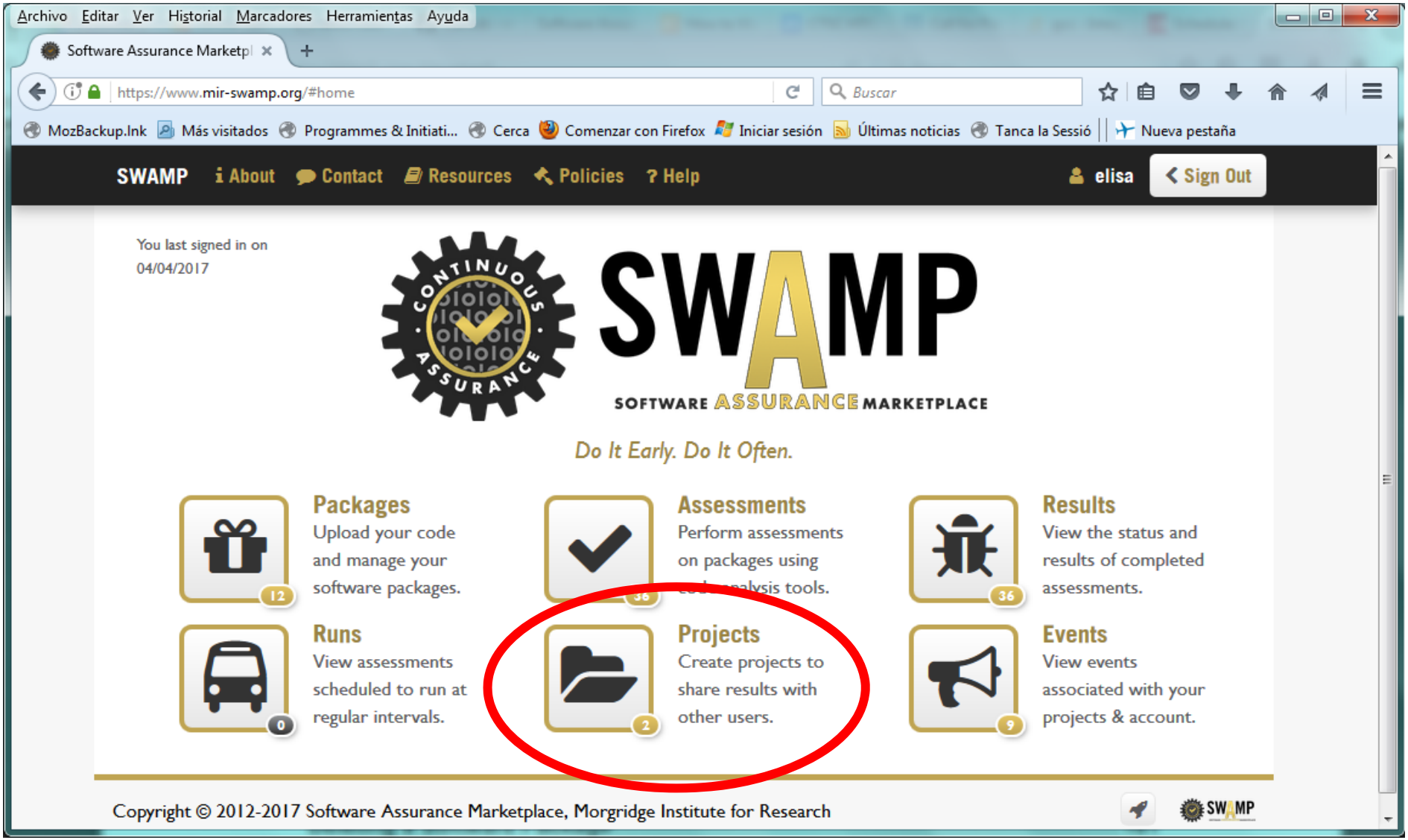
- **Software Assurance Market Place.**
- **Objective: Automate and simplify the use of (multiple) tools.**
- **A national, no-cost resource for software assurance (SwA) technologies used across research institutions, non-governmental organizations, and civilian agencies and their communities as both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

Register to use the SWAMP

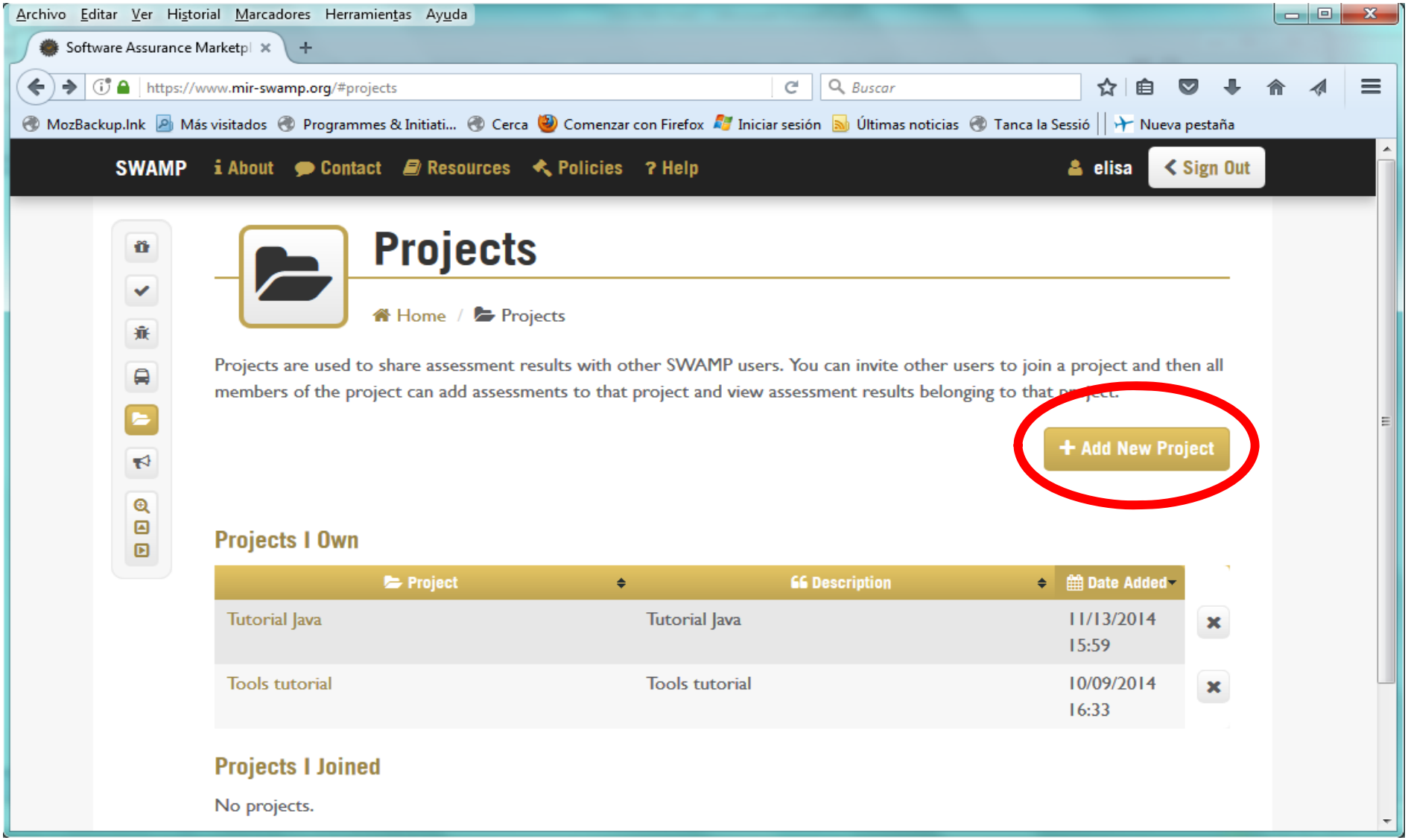


THE UNIVERSITY of WISCONSIN MADISON
Universitat Autònoma de Barcelona
CTSC

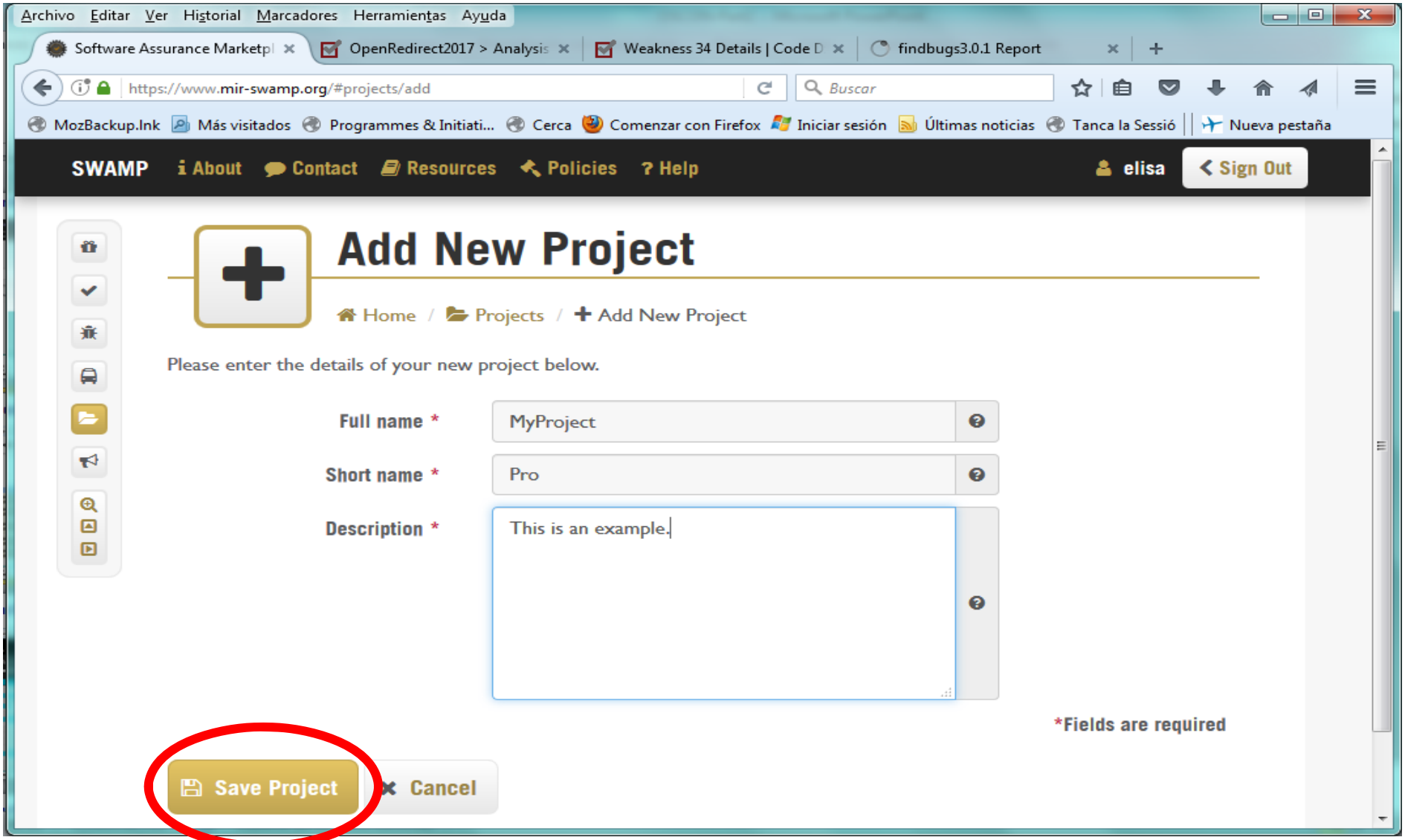
What can I do in the SWAMP?



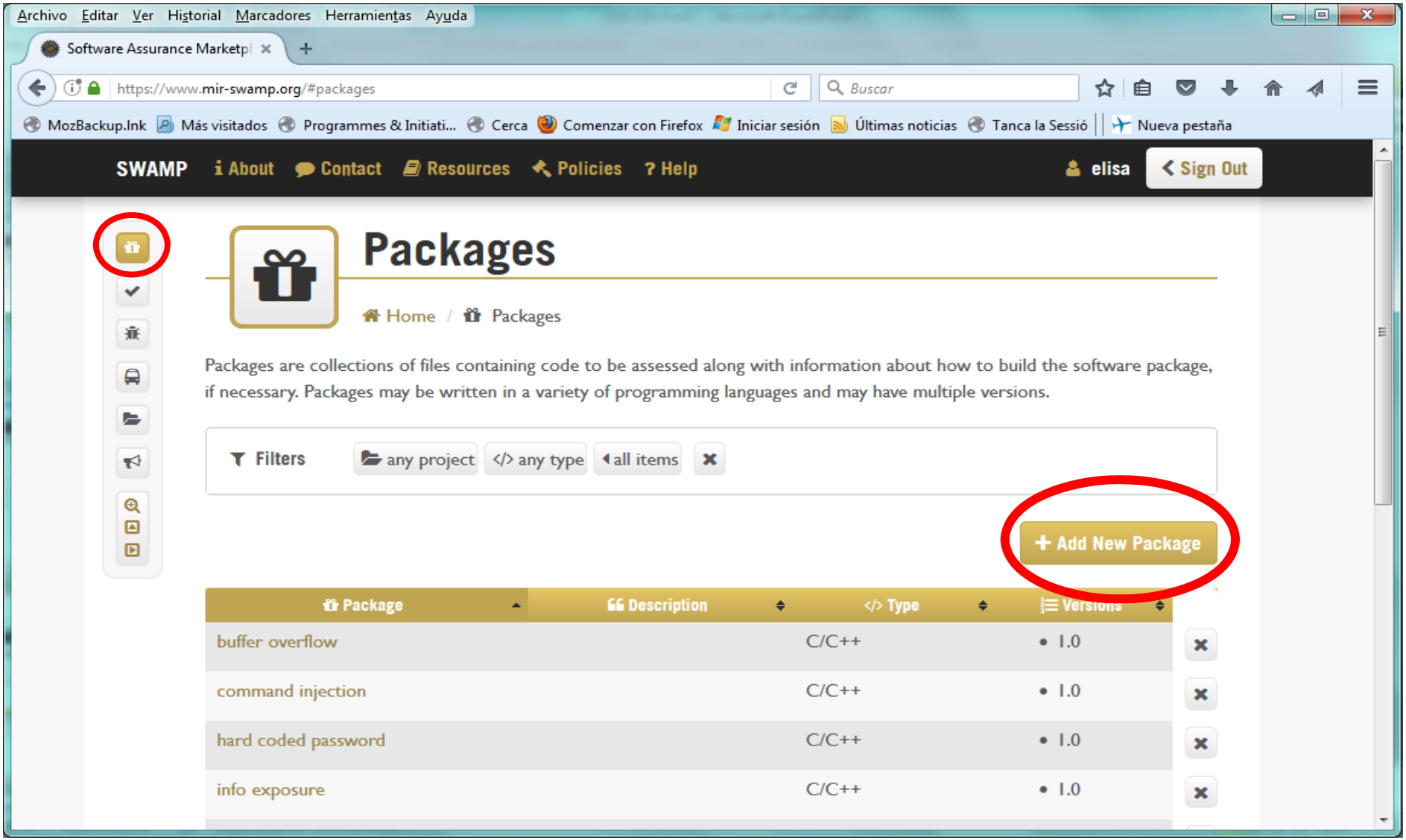
Create a Project



Create a Project



Packages

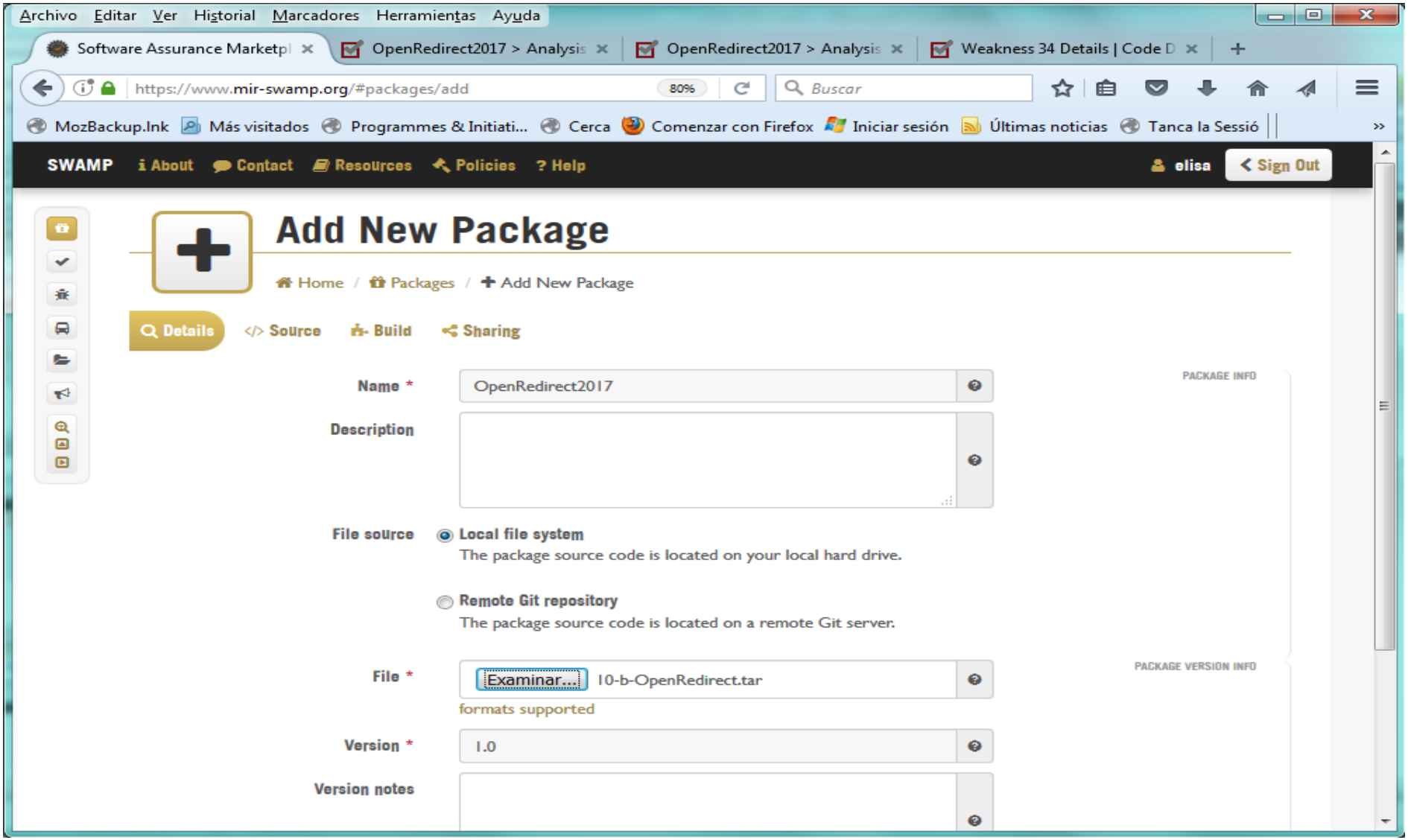


THE UNIVERSITY of WISCONSIN MADISON

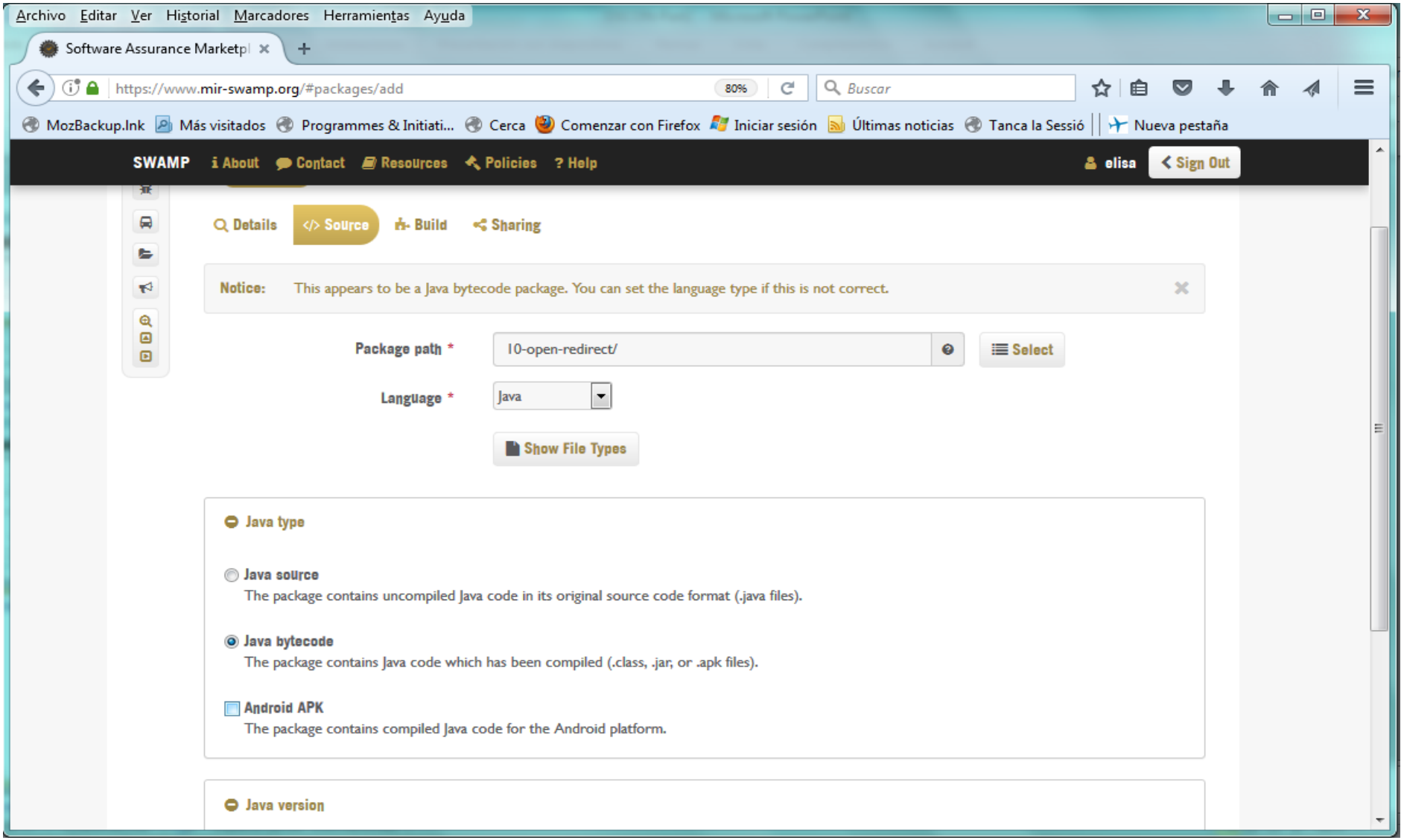
Universitat Autònoma de Barcelona

CTSC

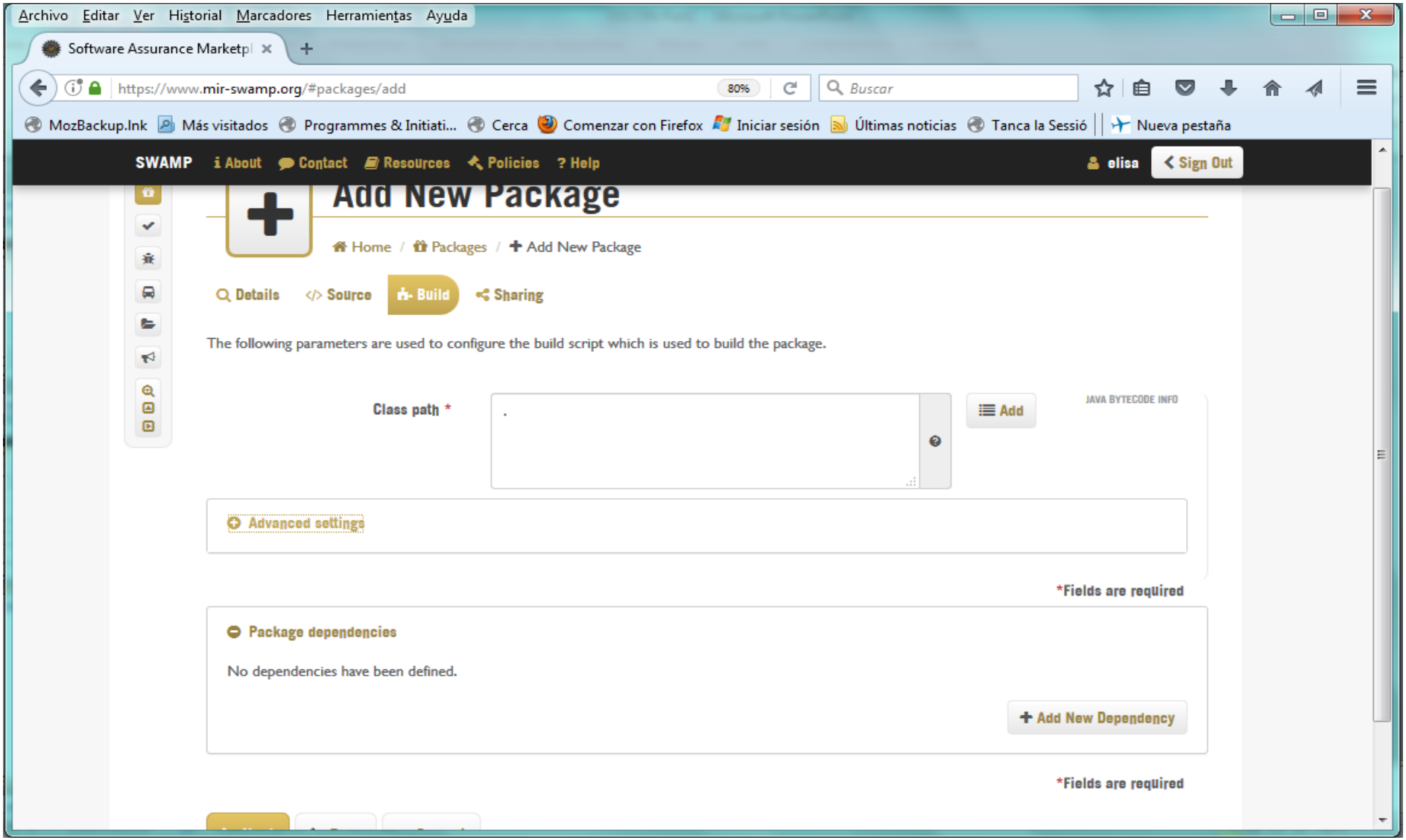
Upload your Software Package



Upload your Software Package



Upload your Software Package

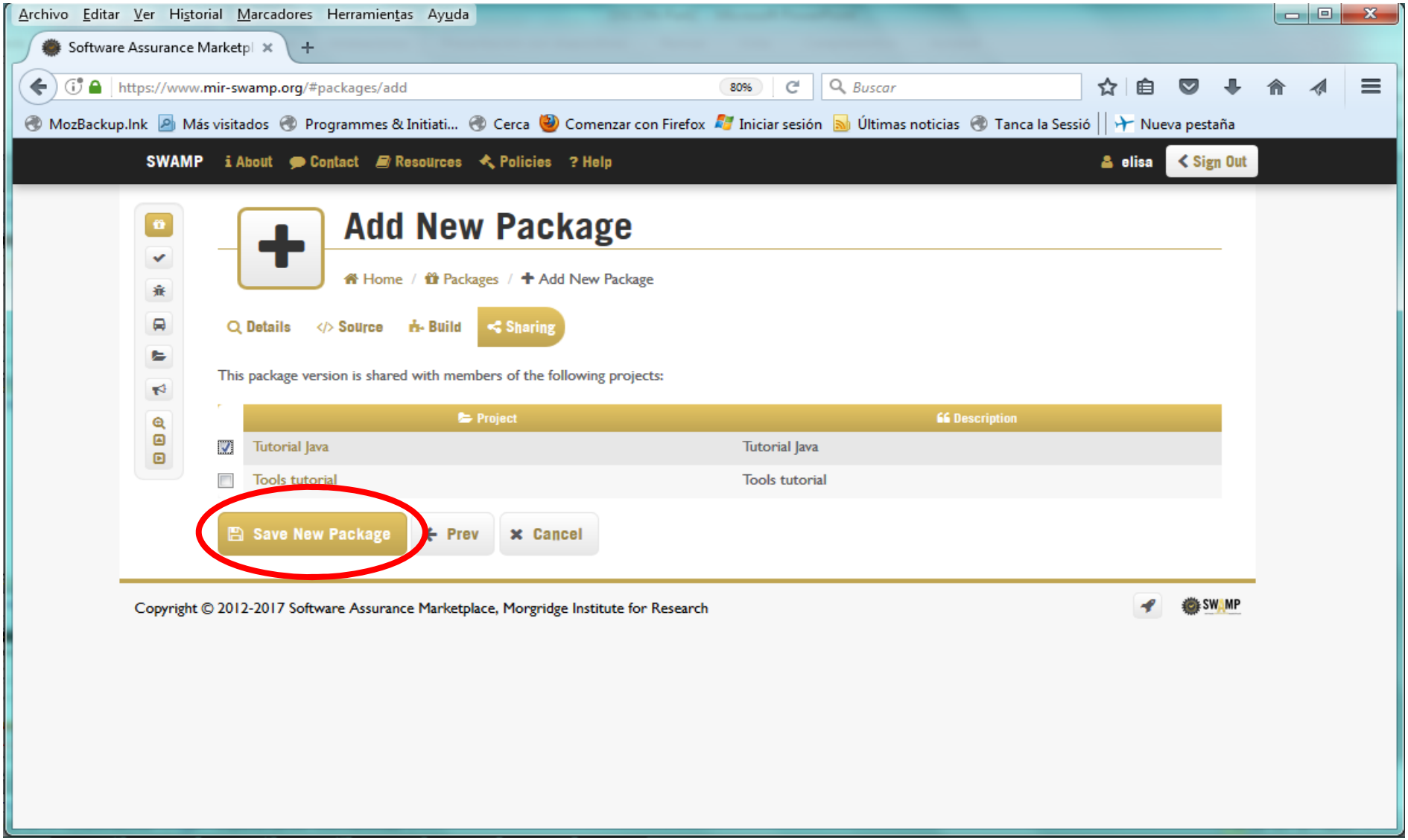


THE UNIVERSITY of WISCONSIN MADISON

Universitat Autònoma de Barcelona

CTSC

Upload your Software Package

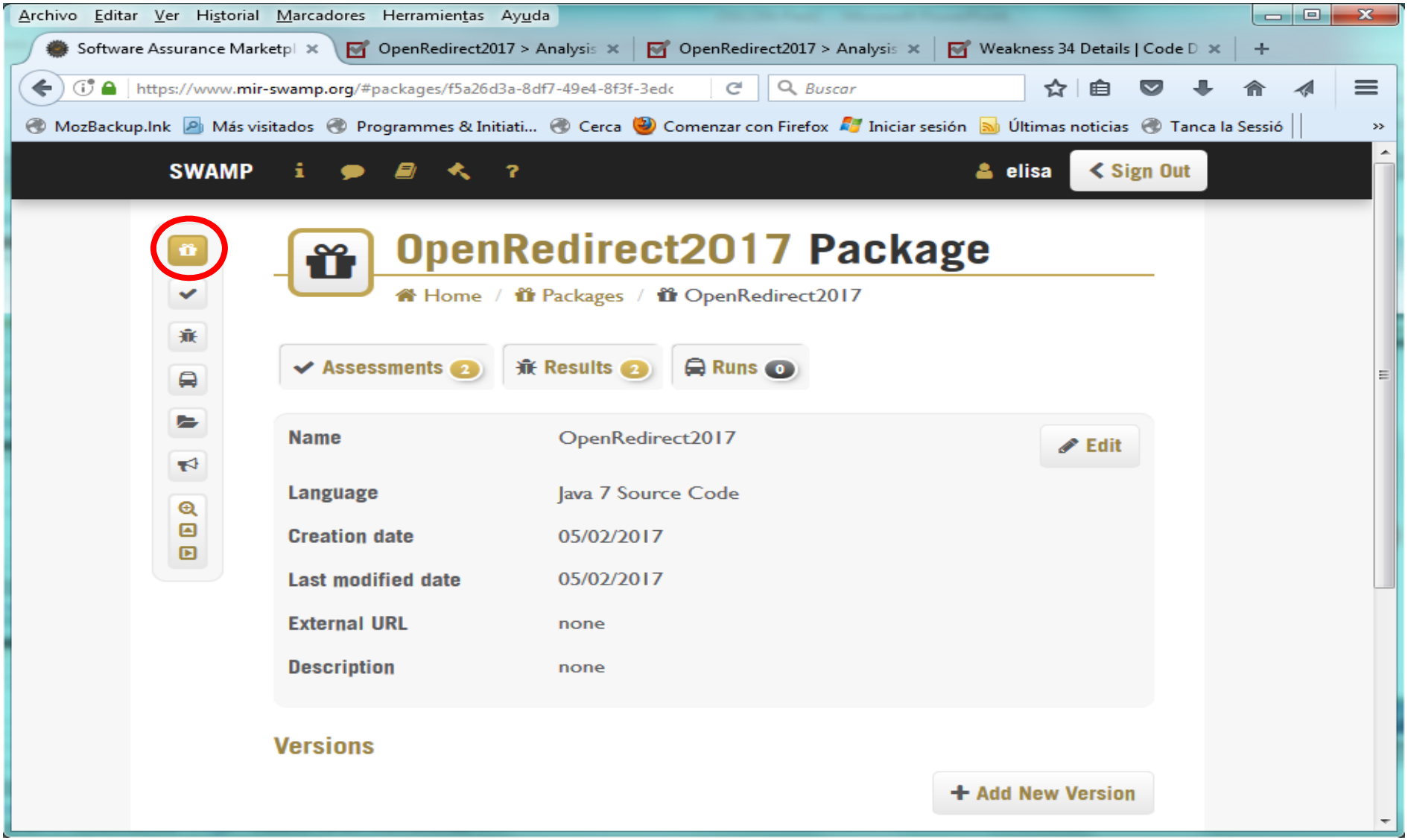


THE UNIVERSITY of WISCONSIN MADISON

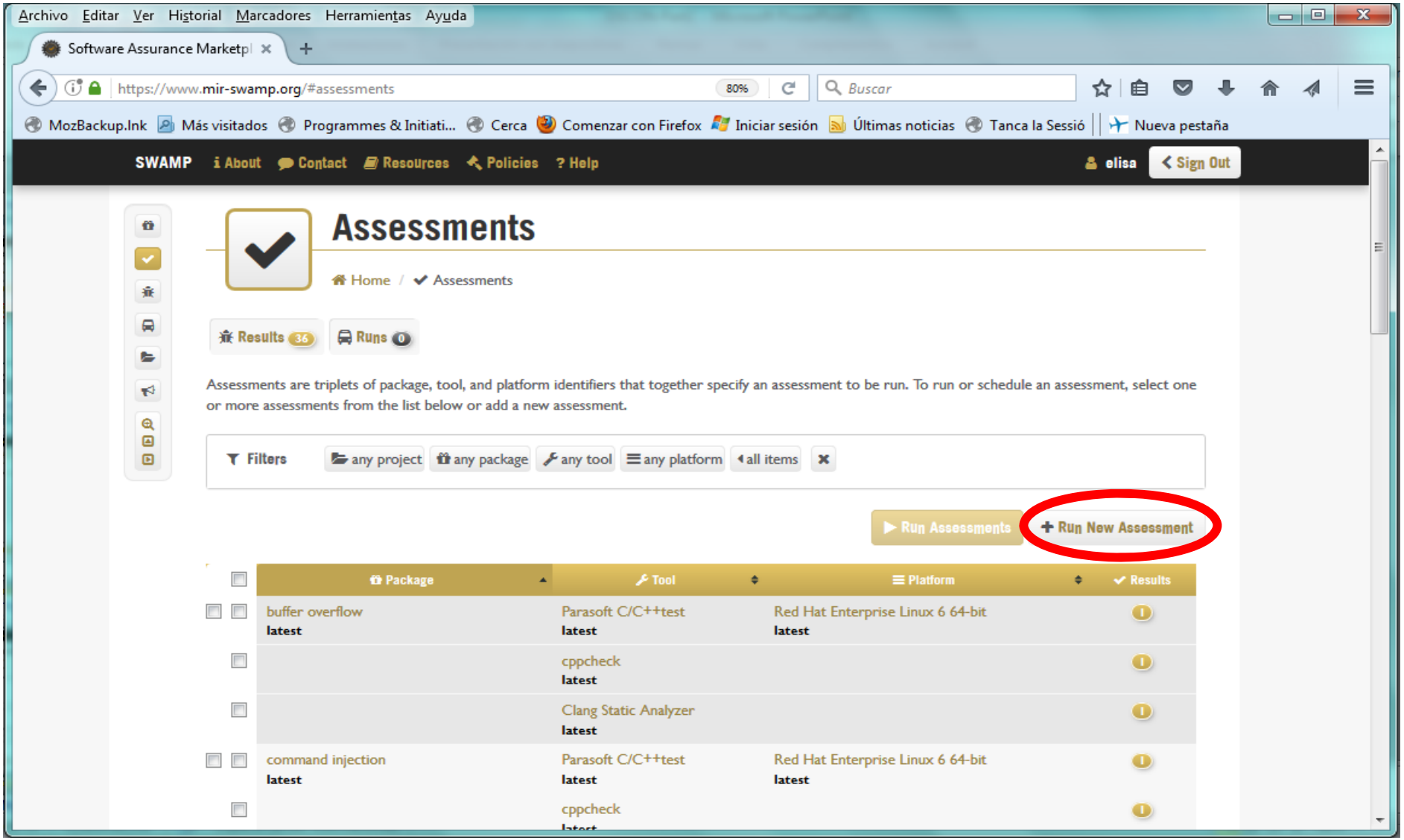
Universitat Autònoma de Barcelona

CTSC

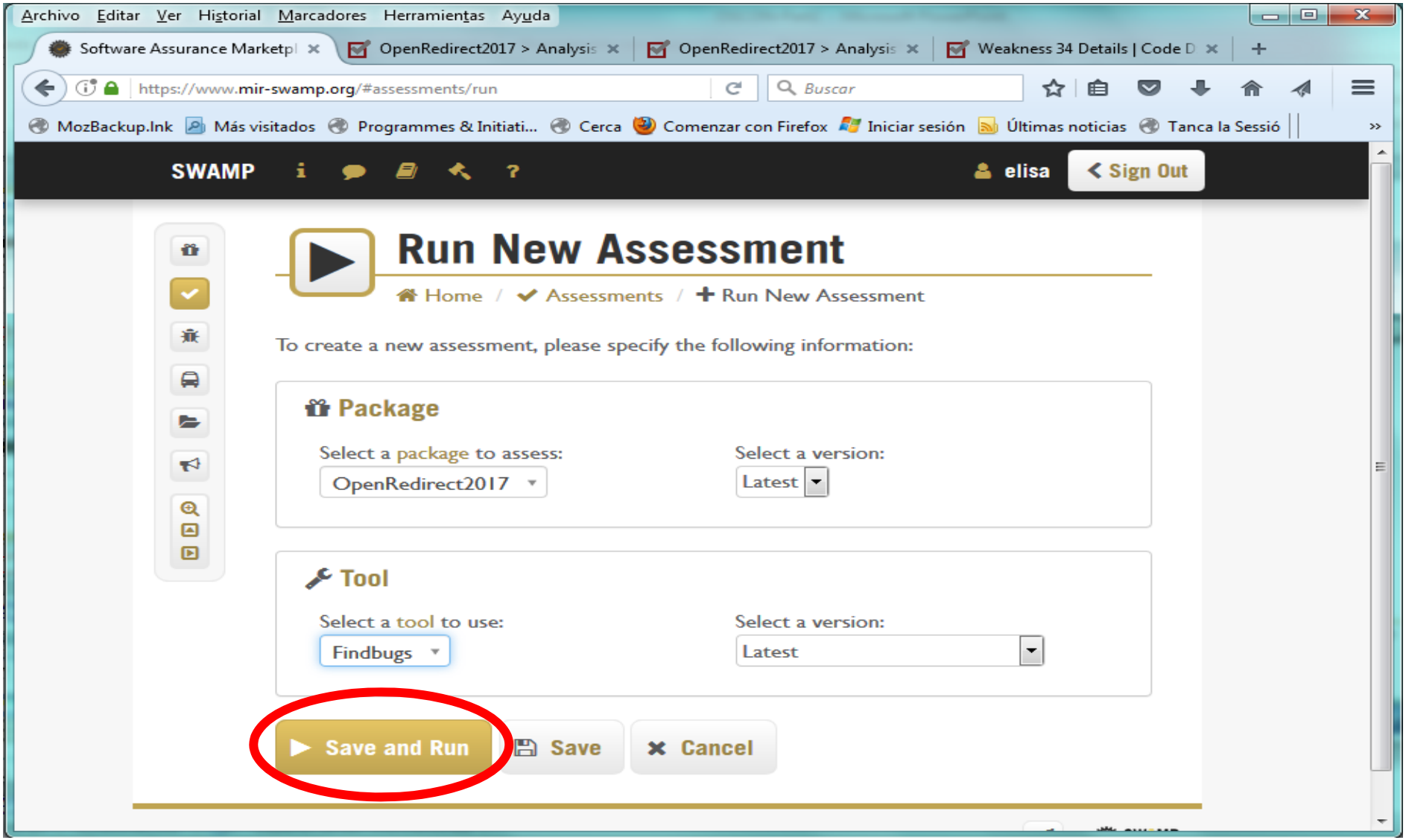
Upload your Software Package



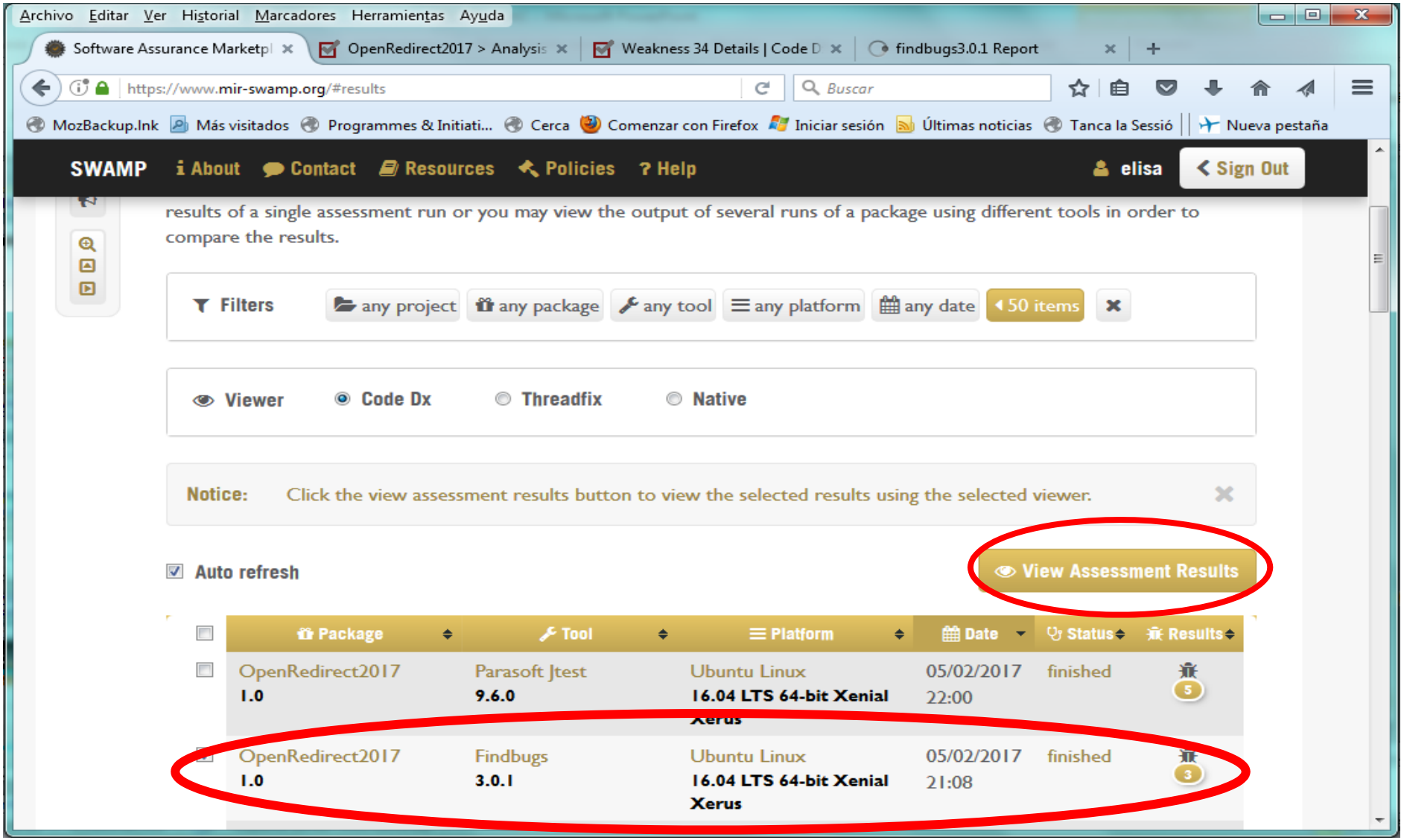
Run your Assessments



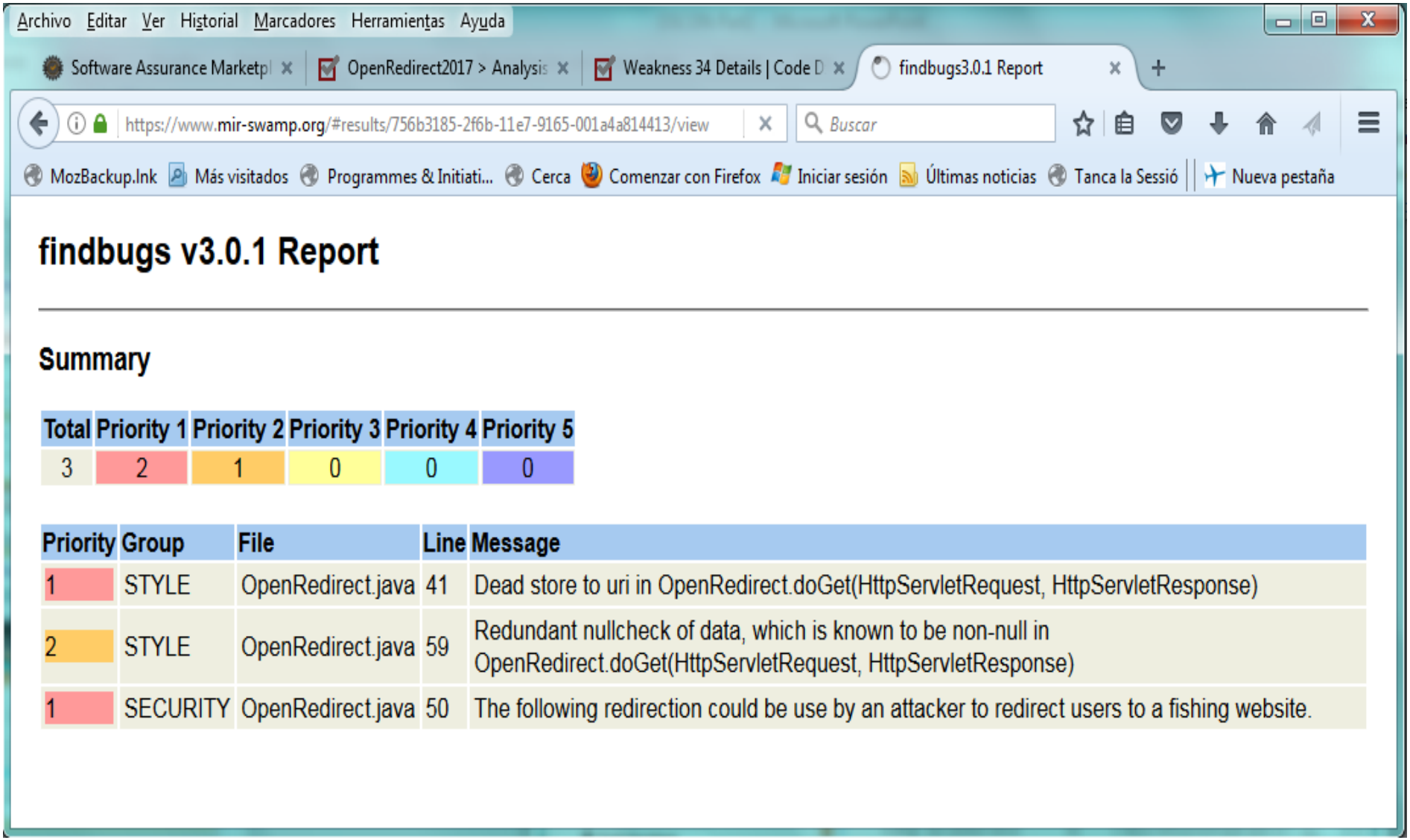
Run your Assessments



My Assessments



View your Results. FindBugs - Native



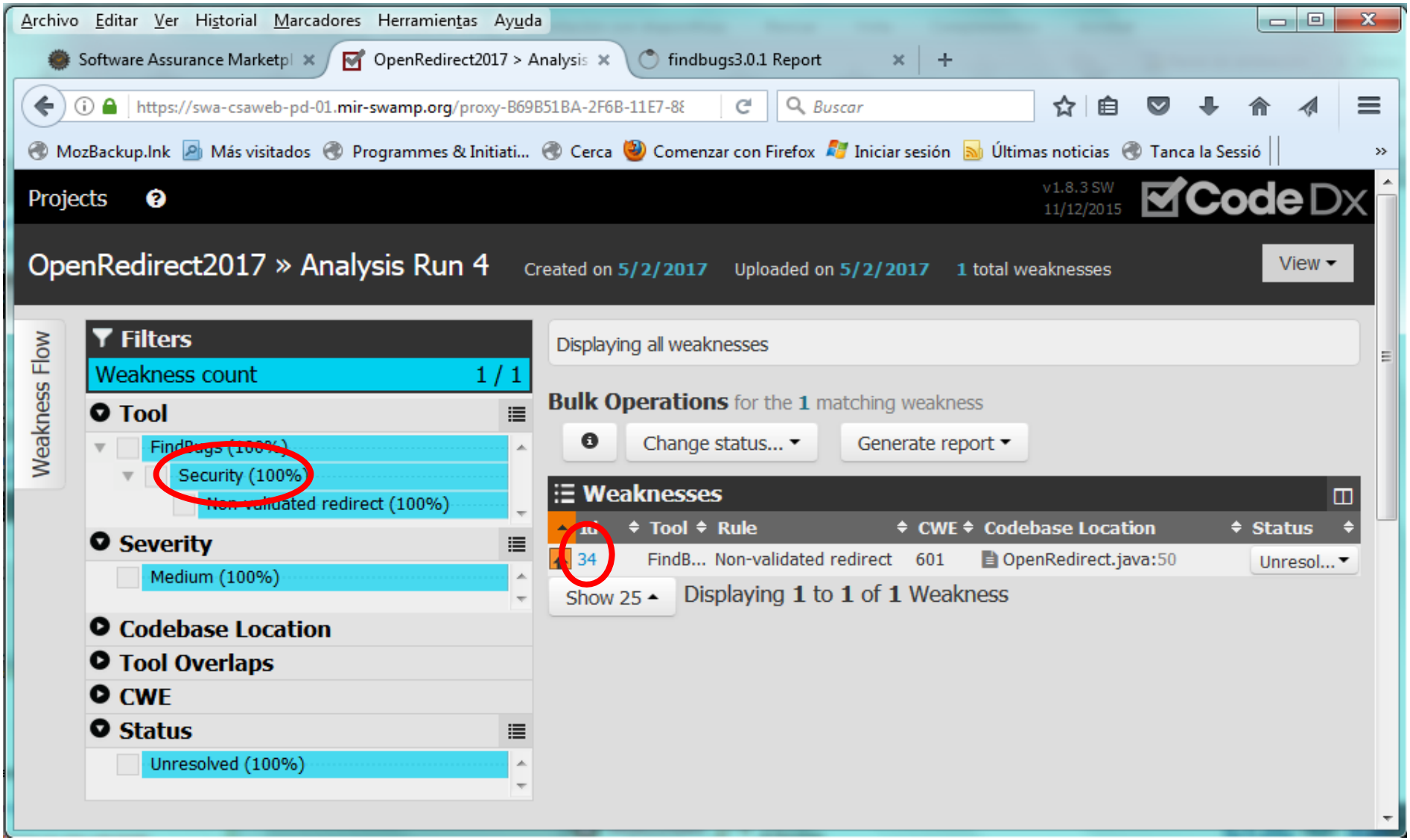
findbugs v3.0.1 Report

Summary

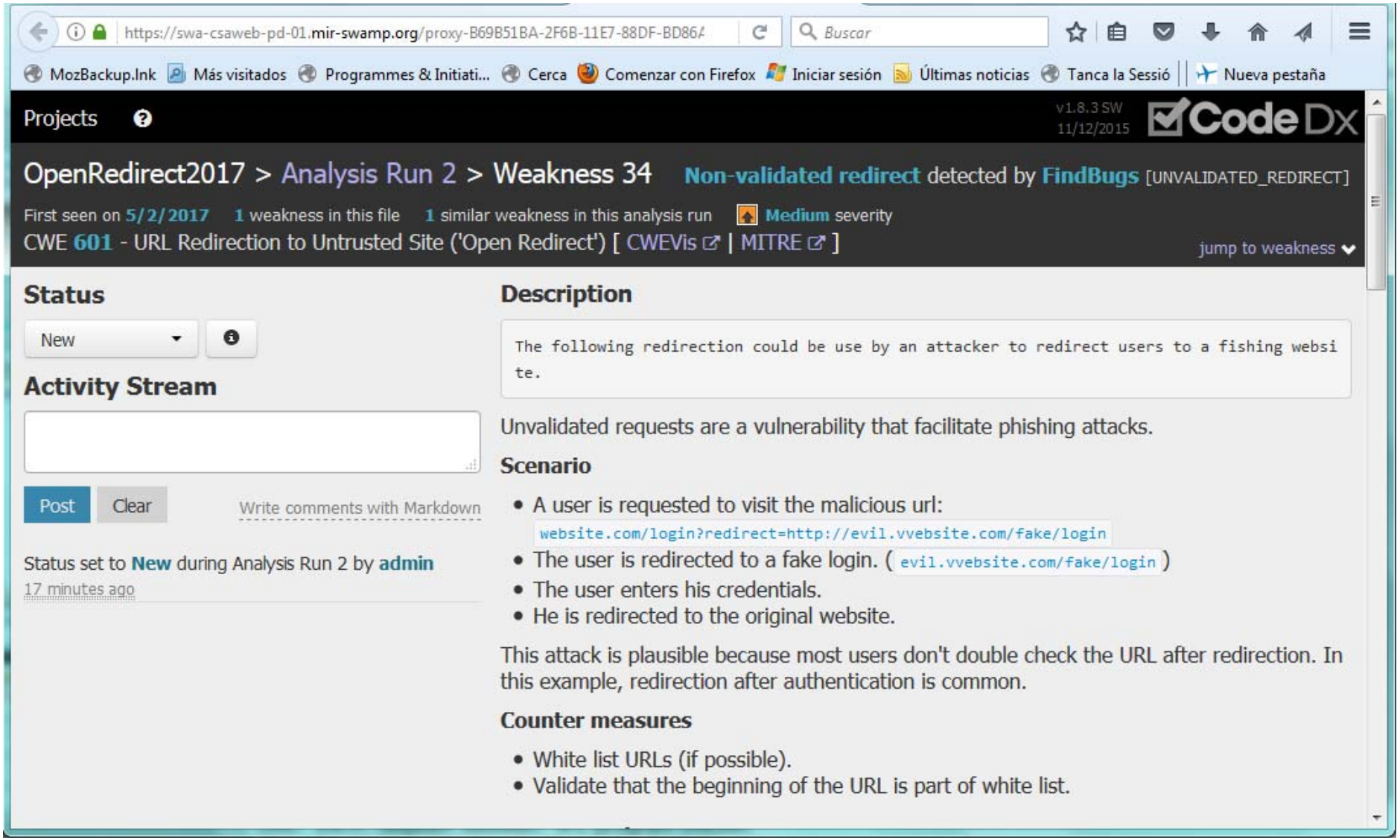
Total	Priority 1	Priority 2	Priority 3	Priority 4	Priority 5
3	2	1	0	0	0

Priority	Group	File	Line	Message
1	STYLE	OpenRedirect.java	41	Dead store to uri in OpenRedirect.doGet(HttpServletRequest, HttpServletResponse)
2	STYLE	OpenRedirect.java	59	Redundant nullcheck of data, which is known to be non-null in OpenRedirect.doGet(HttpServletRequest, HttpServletResponse)
1	SECURITY	OpenRedirect.java	50	The following redirection could be use by an attacker to redirect users to a fishing website.

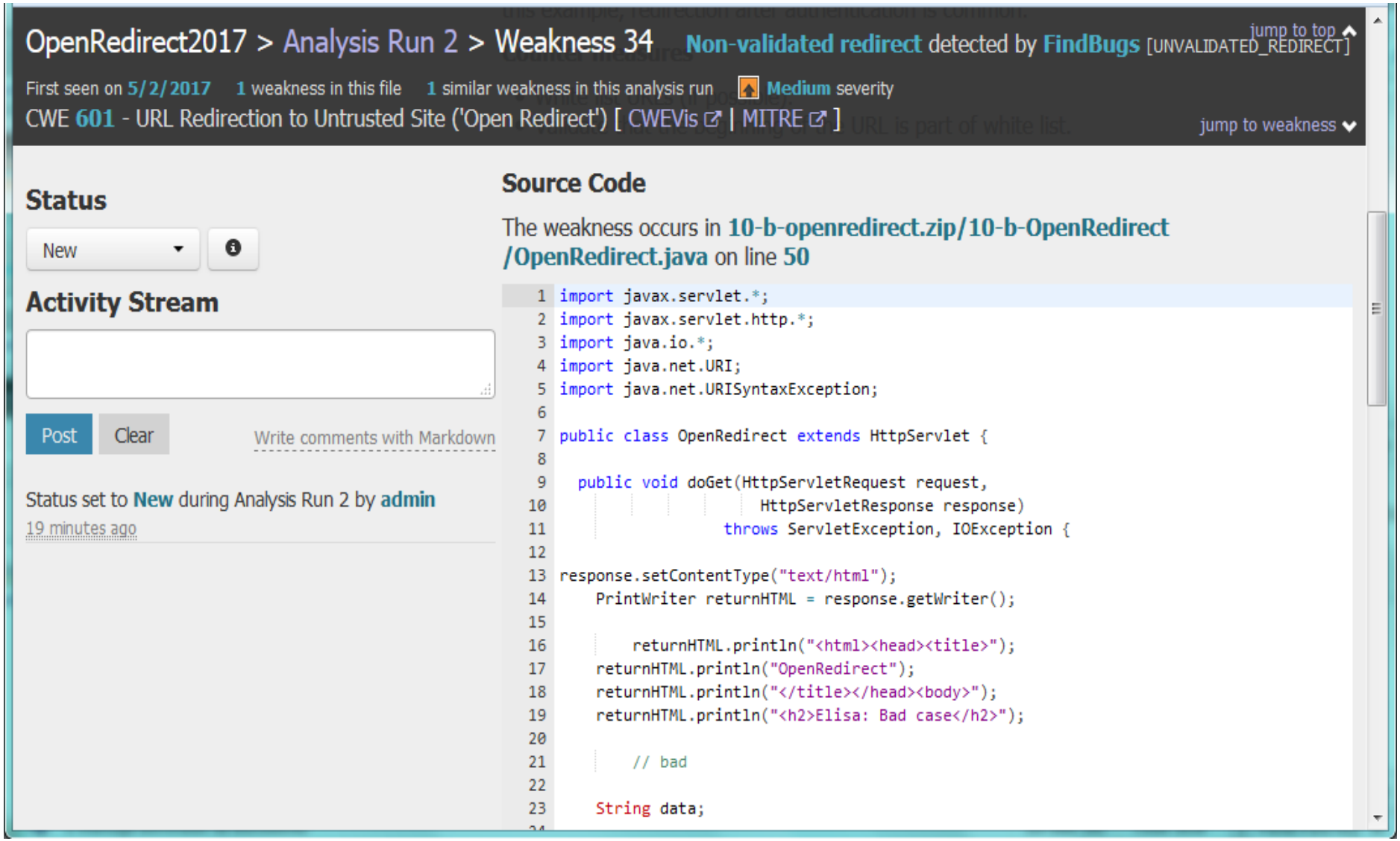
View your Results. FindBugs - CodeDX



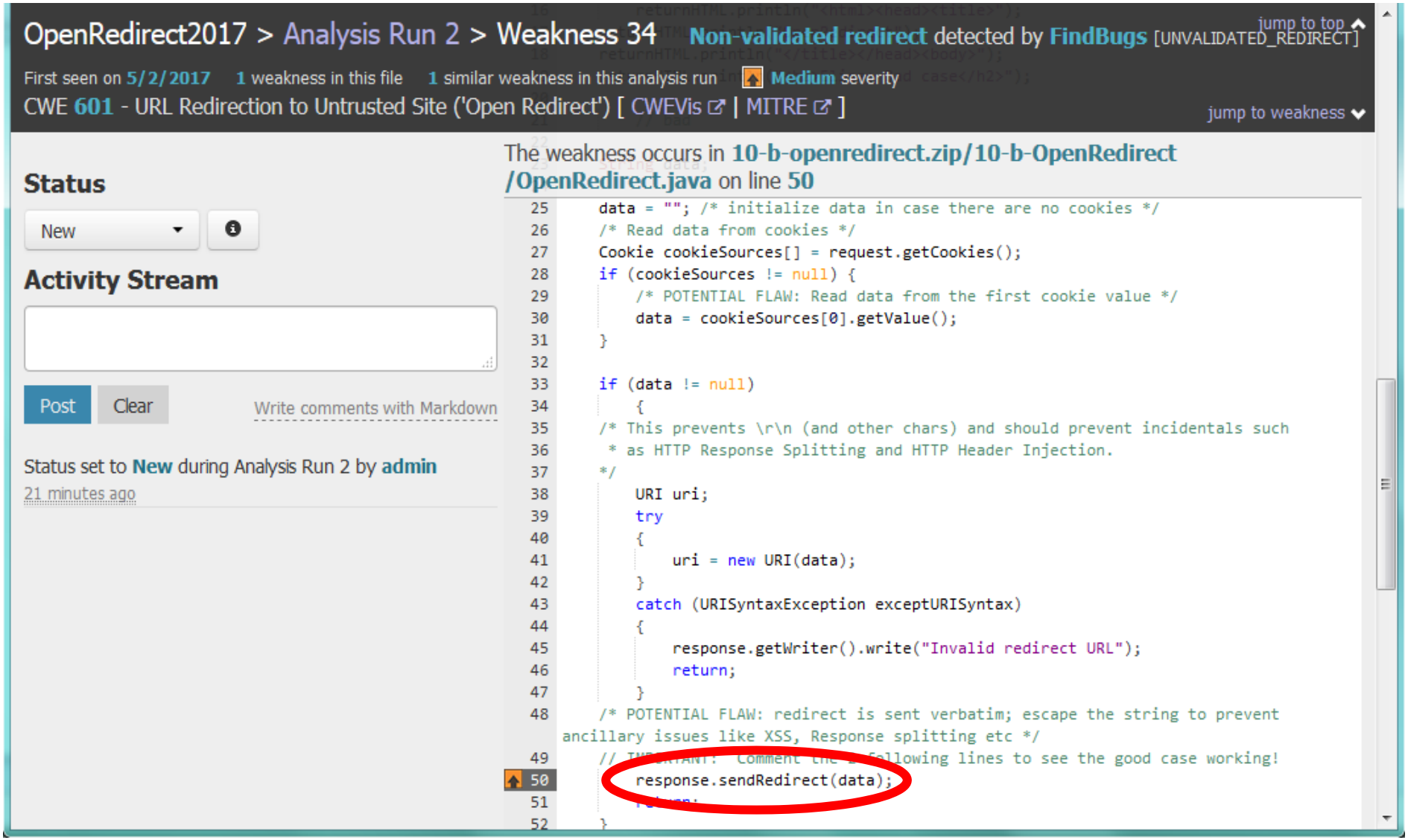
View your Results. FindBugs - CodeDX



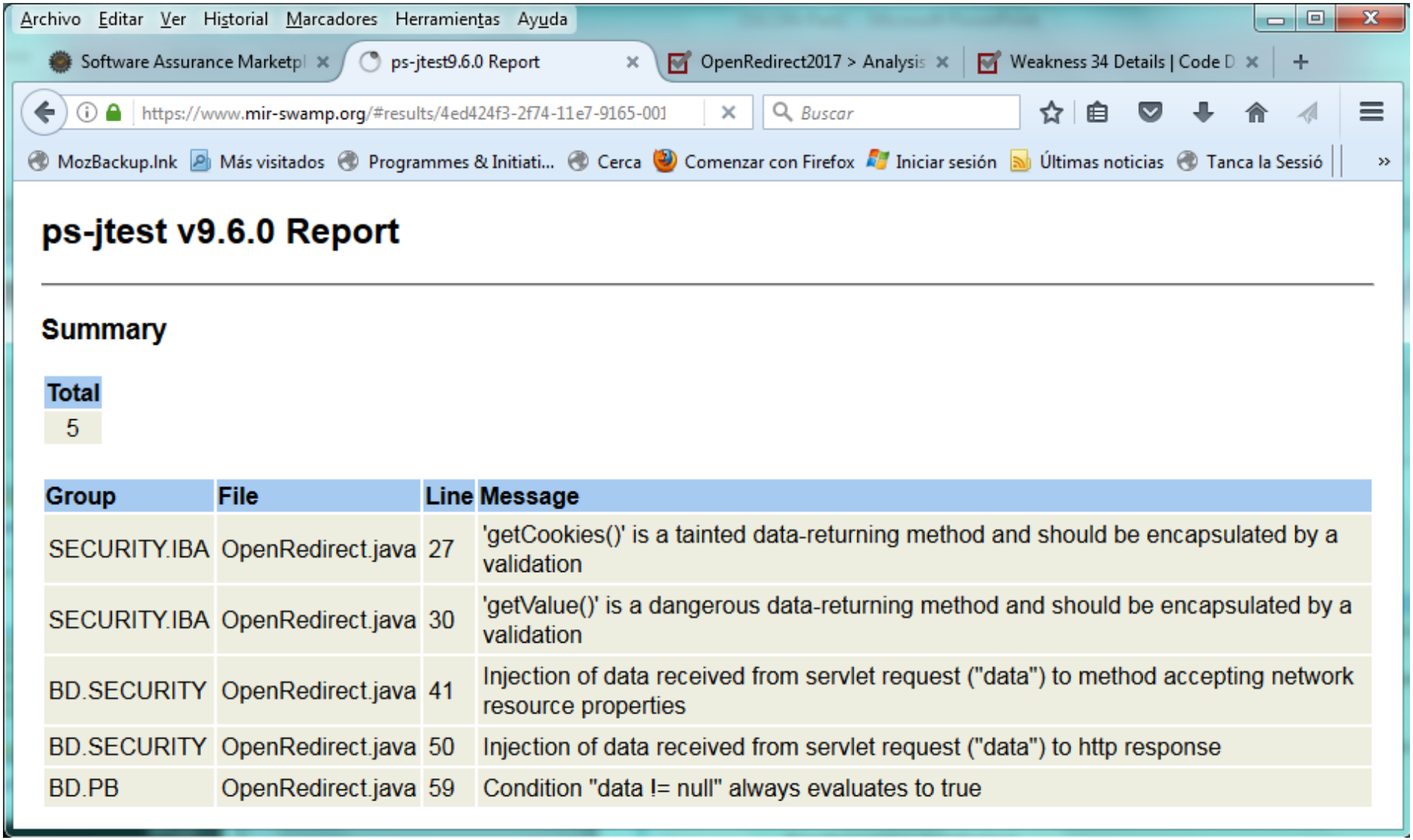
View your Results. FindBugs - CodeDX



View your Results. FindBugs - CodeDX



View your Results. JTest - Native



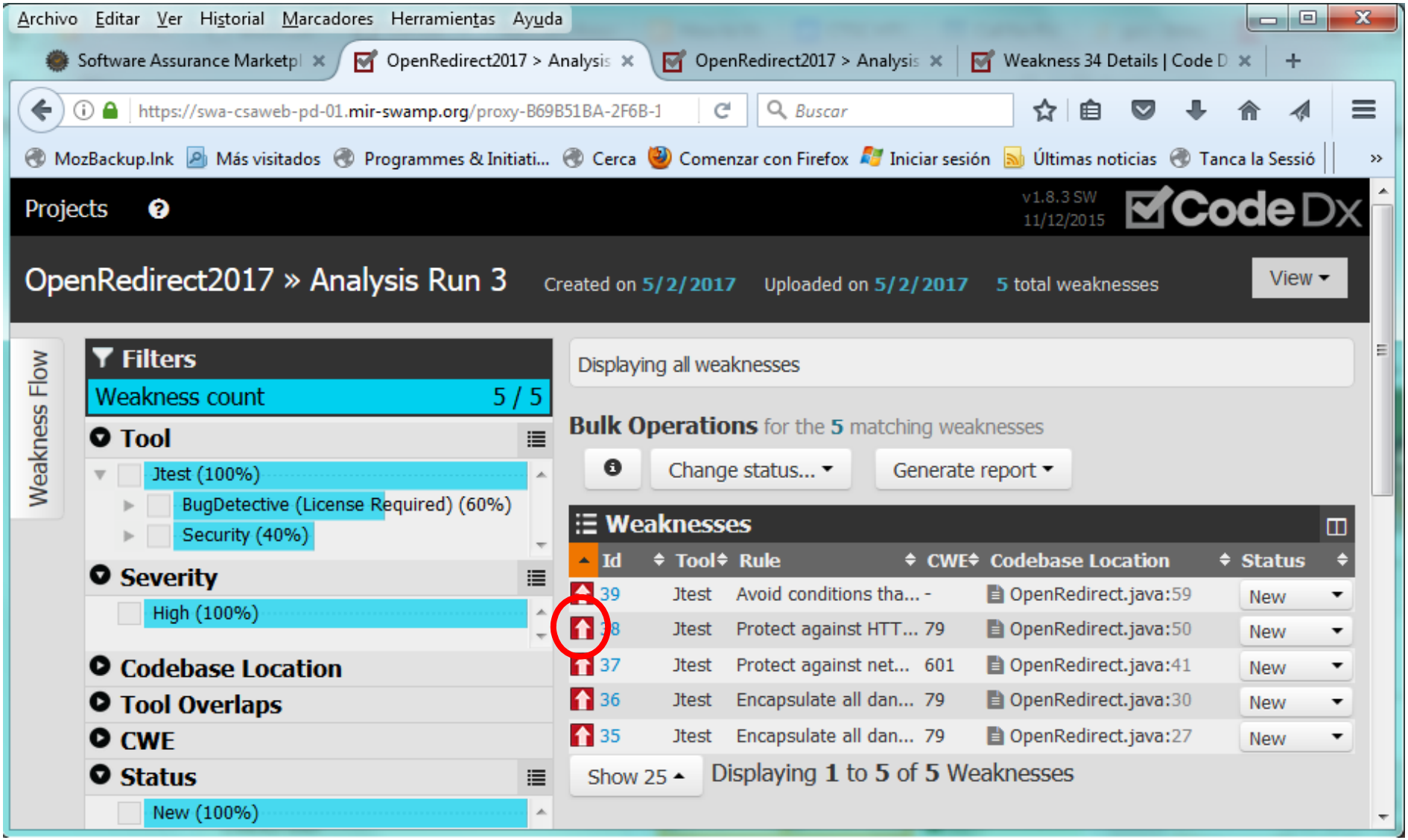
ps-jtest v9.6.0 Report

Summary

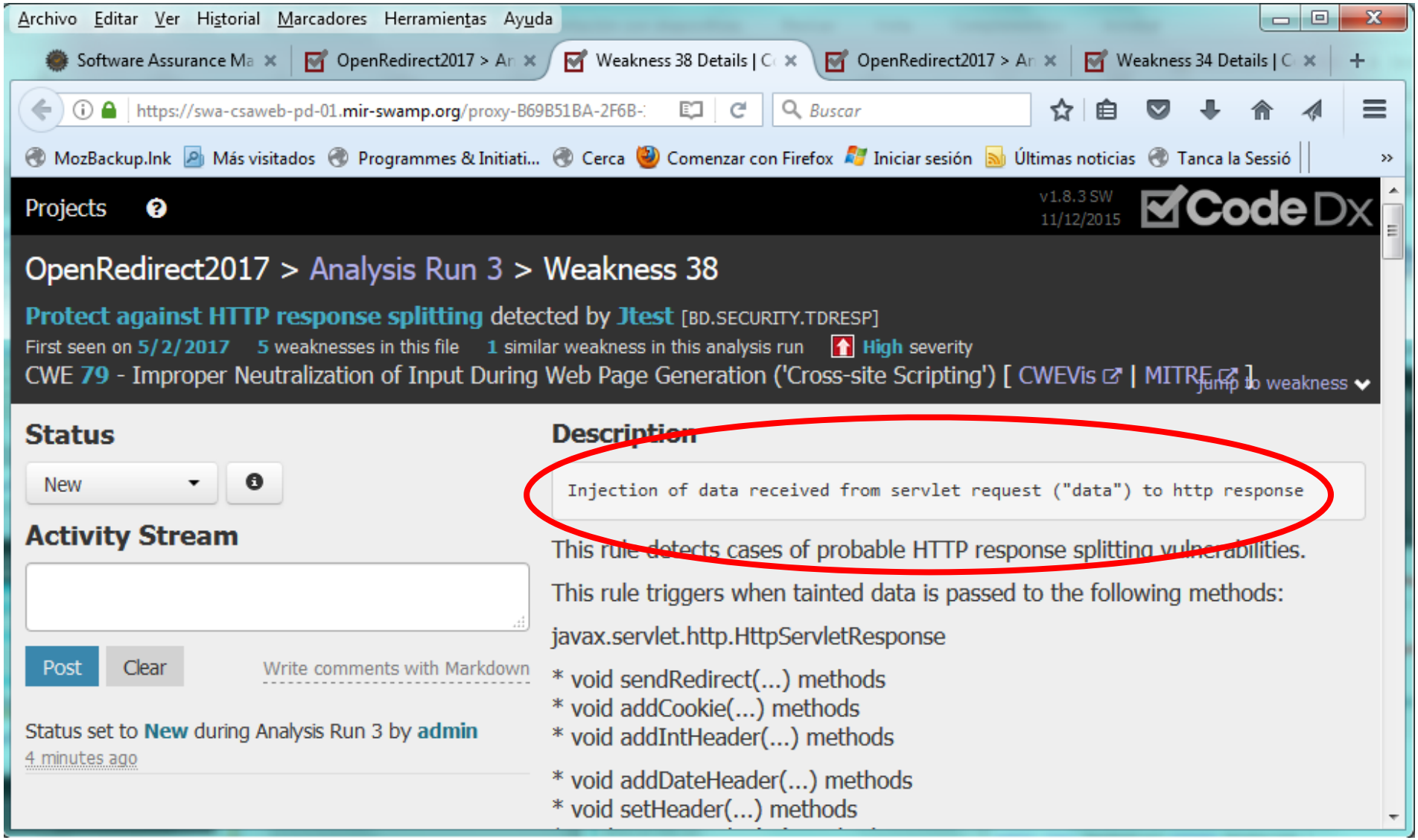
Total
5

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

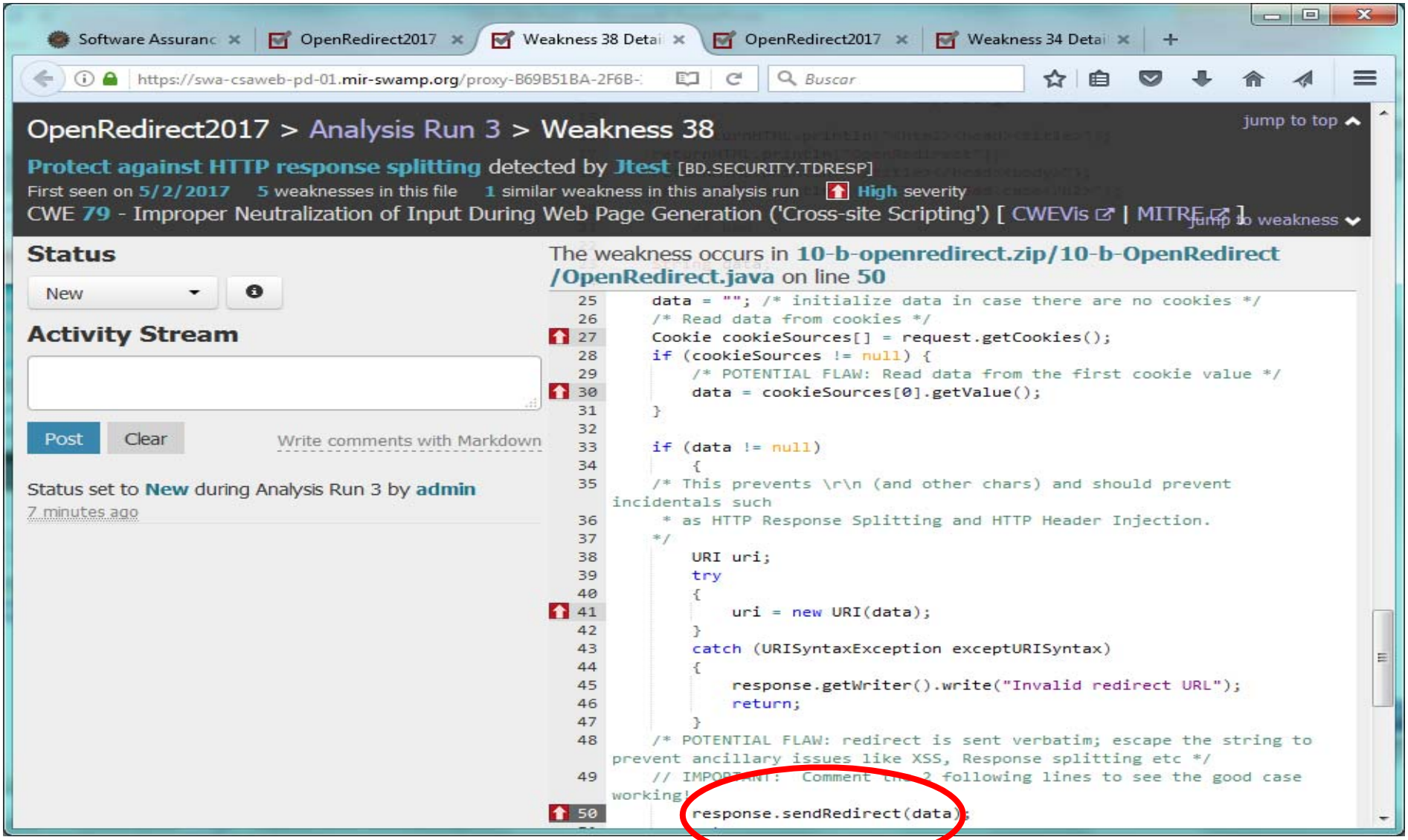
View your Results. JTest - CodeDX



View your Results. JTest - CodeDX



View your Results. JTest - CodeDX



THE UNIVERSITY of WISCONSIN MADISON

Universitat Autònoma de Barcelona

CTSC

Questions?

Elisa Heymann

Elisa.Heymann@uab.es

Barton P. Miller

bart@cs.wisc.edu

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

Java Hands-on Exercise

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

August 2017

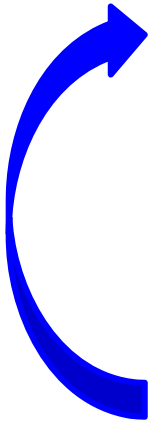
CTSC

Goals of the Assignment

- Learn to use software assessment tools and the SWAMP with a Java application.
- Understand the tools (and SWAMP) output:
  - Understand the interface.
  - Learn to locate in the source files the problems the tools are reporting security-wise.
  - Interpret the results.

Steps we'll go through

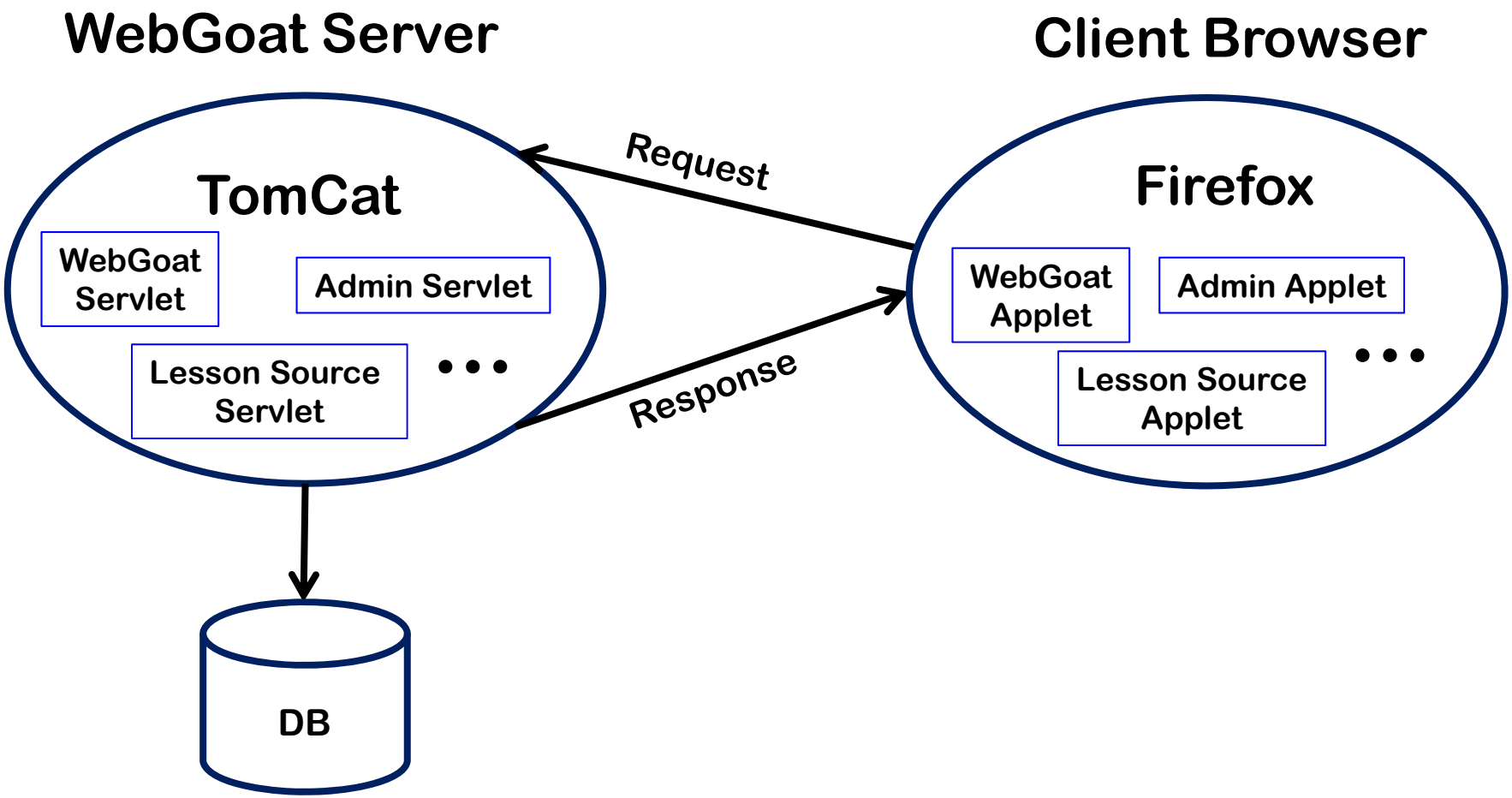
1. Introduction to a sample application and use it.
2. Upload the software package to the SWAMP.
3. Run assessments with the SWAMP tools.
4. Understand the output.
5. Modify the source code to fix a problem.



WebGoat

- A collection of unsecure web applications produced by OWASP.
- Built into a framework intended for learning to avoid common programming mistakes.
- Server-side application flaws.
- Designed to teach web applications security lessons.
- http://webgoat.github.io

WebGoat Basic Architecture



Run WebGoat

On your console run the WebGoat server:

```
cd ~/WebGoat/WebGoat-7.1
java -jar webgoat-standalone/target/webgoat-standalone-7.1-exec.jar --port 8888
```

(you can run the `~/server.sh` script)

On your web browser run the client code:

```
http://localhost:8888/WebGoat
```

Go to Injection Flaws-> String SQL injection

Assess WebGoat in the SWAMP

On your console package the code to send to the SWAMP:

```
cd ~/WebGoat/WebGoat-Lessons-develop
mvn clean
cd ..
tar cvf WebGoat-Lessons-develop.tar
        WebGoat-Lessons-develop/
```

On your web browser go to:

```
https://www.mir-swamp.org/
```

Assess WebGoat in the SWAMP

When creating the MyWebGoat package in the SWAMP:

- **Chose the `WebGoat-Lessons-develop.tar` file just created**
- **Select Java 8**
- **In "build target" type clean install**

When seeing the results:

- **Focus on security issues**
- **Focus on the SqlStringInjection.java file**

Edit & Compile WebGoat Lessons

```
cd ~/WebGoat/WebGoat-Lessons-develop/sql-string-injection/src/main/java/org/owasp/webgoat/plugin

vi SqlStringInjection.java

cd ~/WebGoat/WebGoat-Lessons-develop
mvn clean package
cp target/plugins/*.jar ../WebGoat-7.1/webgoat-container/src/main/webapp/plugin_lessons/
```

(you can run the `~/compile.sh` script)

Edit & Compile WebGoat Lessons

```
String query = "SELECT * FROM user_data WHERE last_name = ?";
ec.addElement(new PRE(query));
try
{
    PreparedStatement statement =
       connection.prepareStatement(query,
       ResultSet.TYPE_SCROLL_INSENSITIVE,
       ResultSet.CONCUR_READ_ONLY);
    statement.setString(1, accountName);
    ResultSet results = statement.executeQuery();
```

WebGoat Exercise 2. Code Injection

No need to run the assessment again.

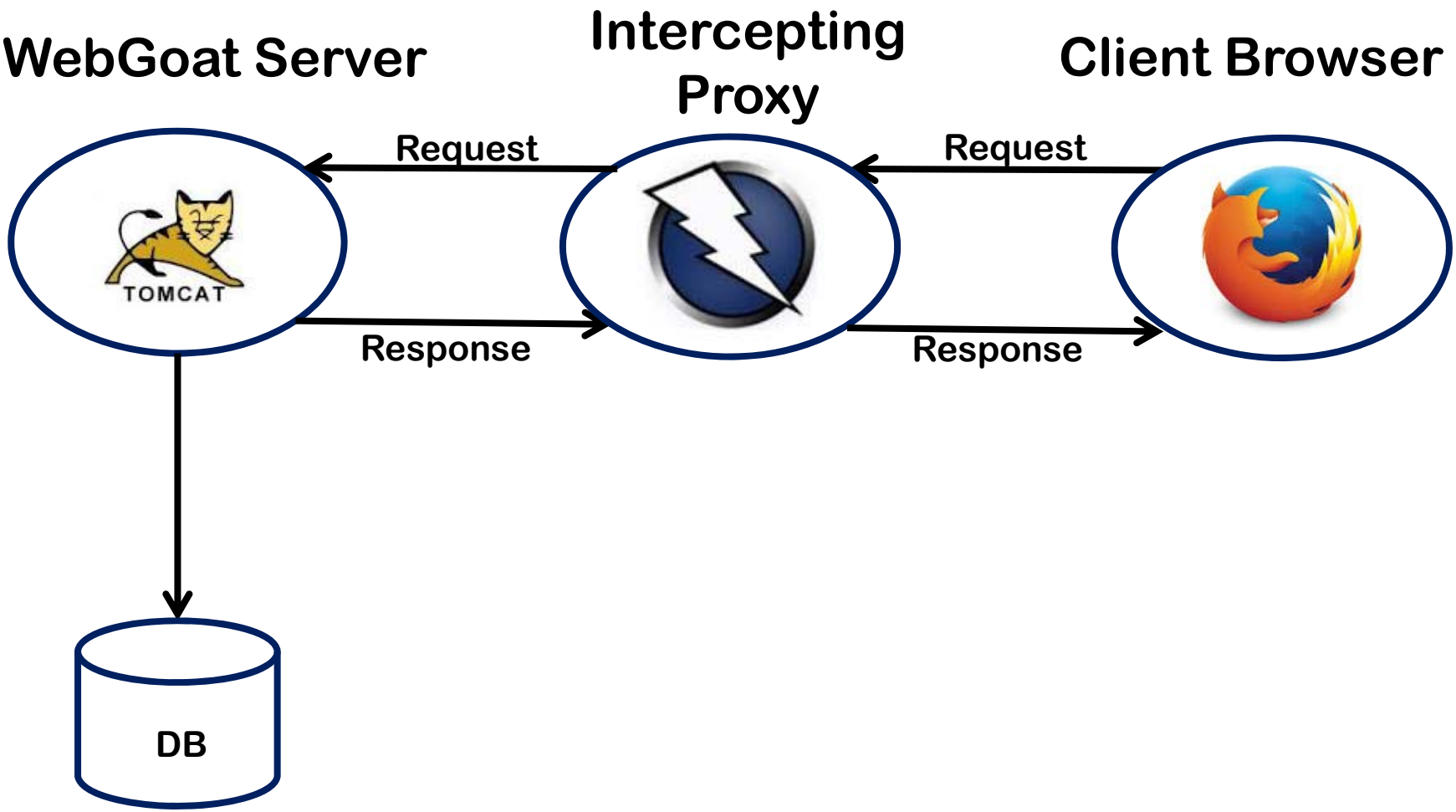
When seeing the results:

- Focus on security issues
- Focus on the CommandInjection.java file

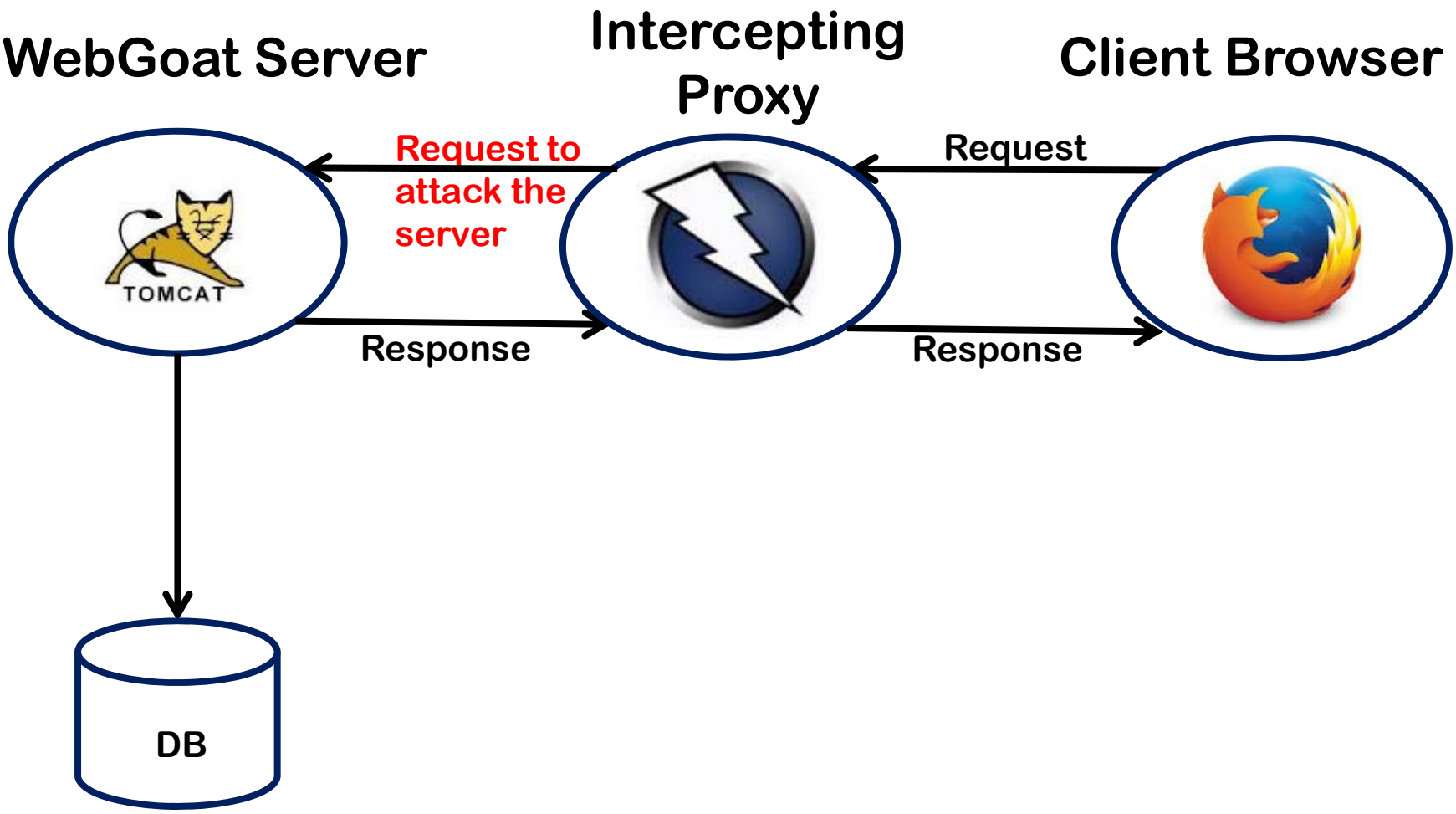
Options to attack the server:

- Modify the client code.
- Intercept and modify the request messages.

WebGoat Exercise 2. Code Injection



WebGoat Exercise 2. Code Injection



WebGoat Exercise 2

On your console start OWASP ZAP, the intercepting proxy:

```
cd /opt/zaproxy
./zap.sh
```

On your another terminal run the WebGoat server:

```
cd ~/WebGoat/WebGoat-7.1
java -jar webgoat-standalone/target/webgoat-standalone-7.1-exec.jar --port 8888
```

(you can run the `~/server.sh` script)

On your web browser run the WebGoat client code:

```
http://localhost:8888/WebGoat
```

Go to Injection Flaws-> Command injection

WebGoat Exercise 2

Agenda:

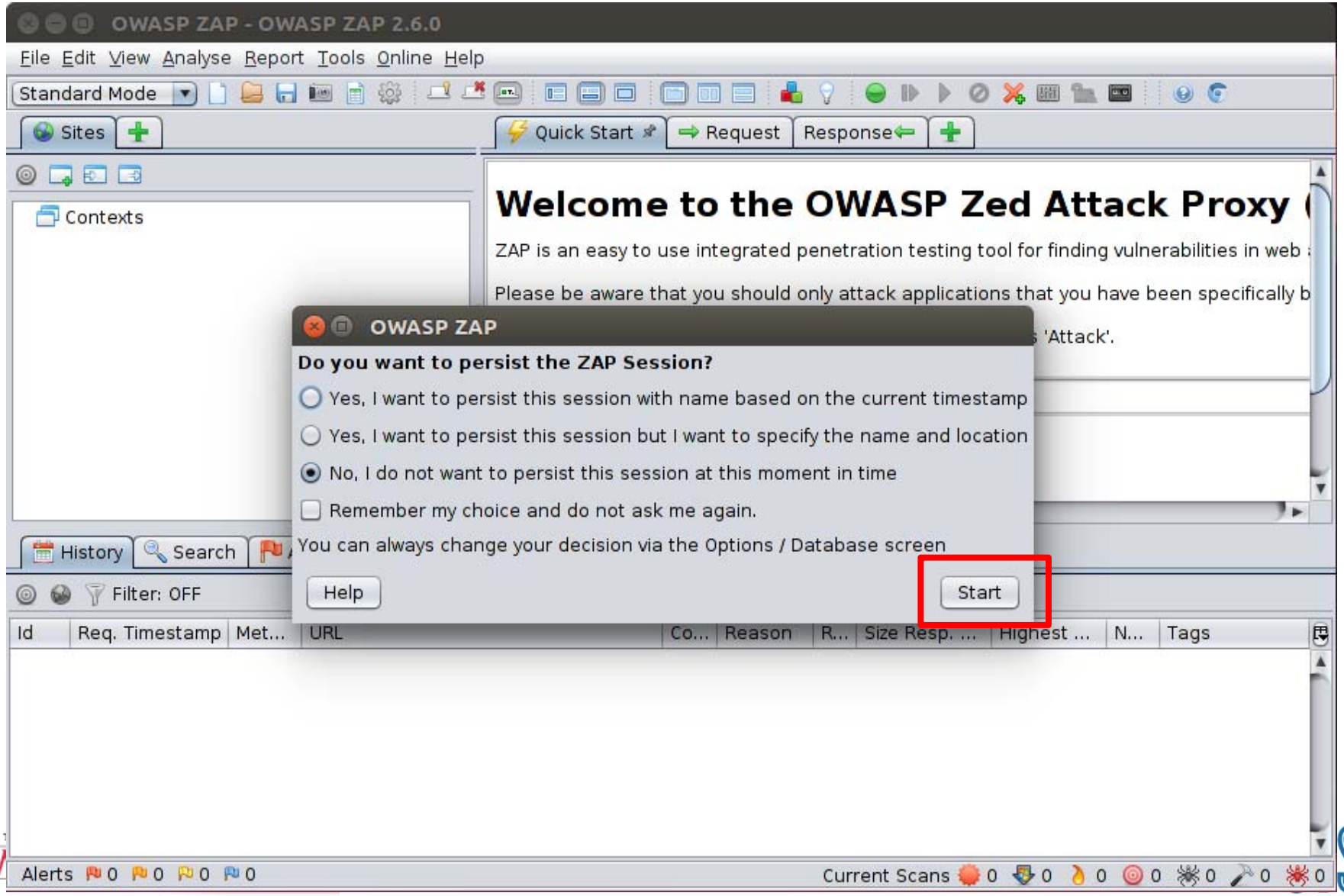
Try the lesson: Select the "lesson plan to view" and click "View".

Modify that request to attack the server:

- In ZAP, you can detect a pattern in the client-server communication (request or response), stop the communication, edit the contents and then continue it. Much like in a debugger, ZAP calls this ability a ***breakpoint***.
- Your job is to:
  - Define breakpoints in ZAP.
  - Modify the request to inject an OS command.
  - View the output.

WebGoat Exercise 2

```
./zap.sh
```

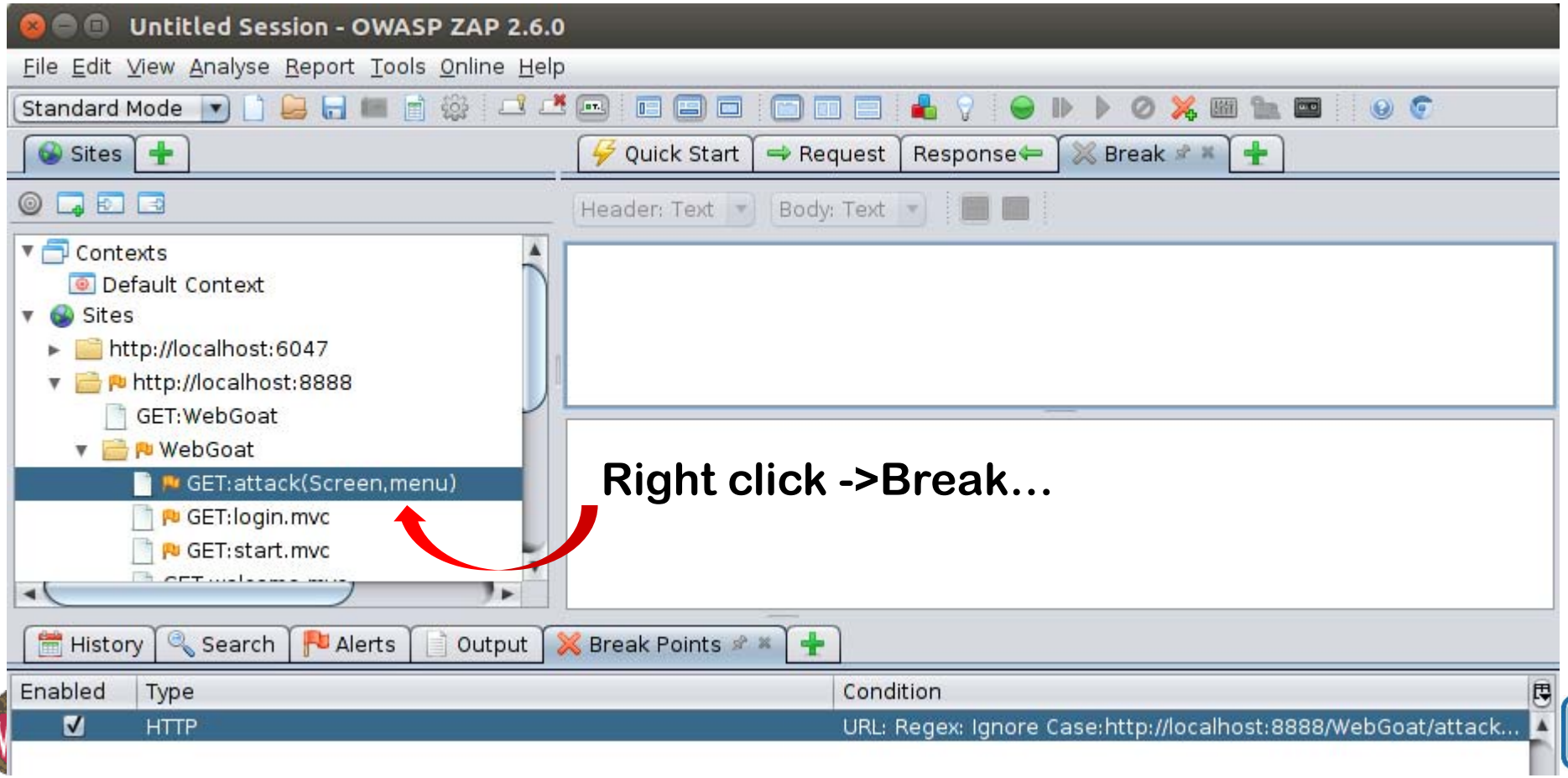


WebGoat Exercise 2

In your browser go to `localhost:8888/WebGoat`

Go to `Injection Flaws-> Command injection`**, choose a lesson plan and click on "View".**

Define a breakpoint in ZAP:

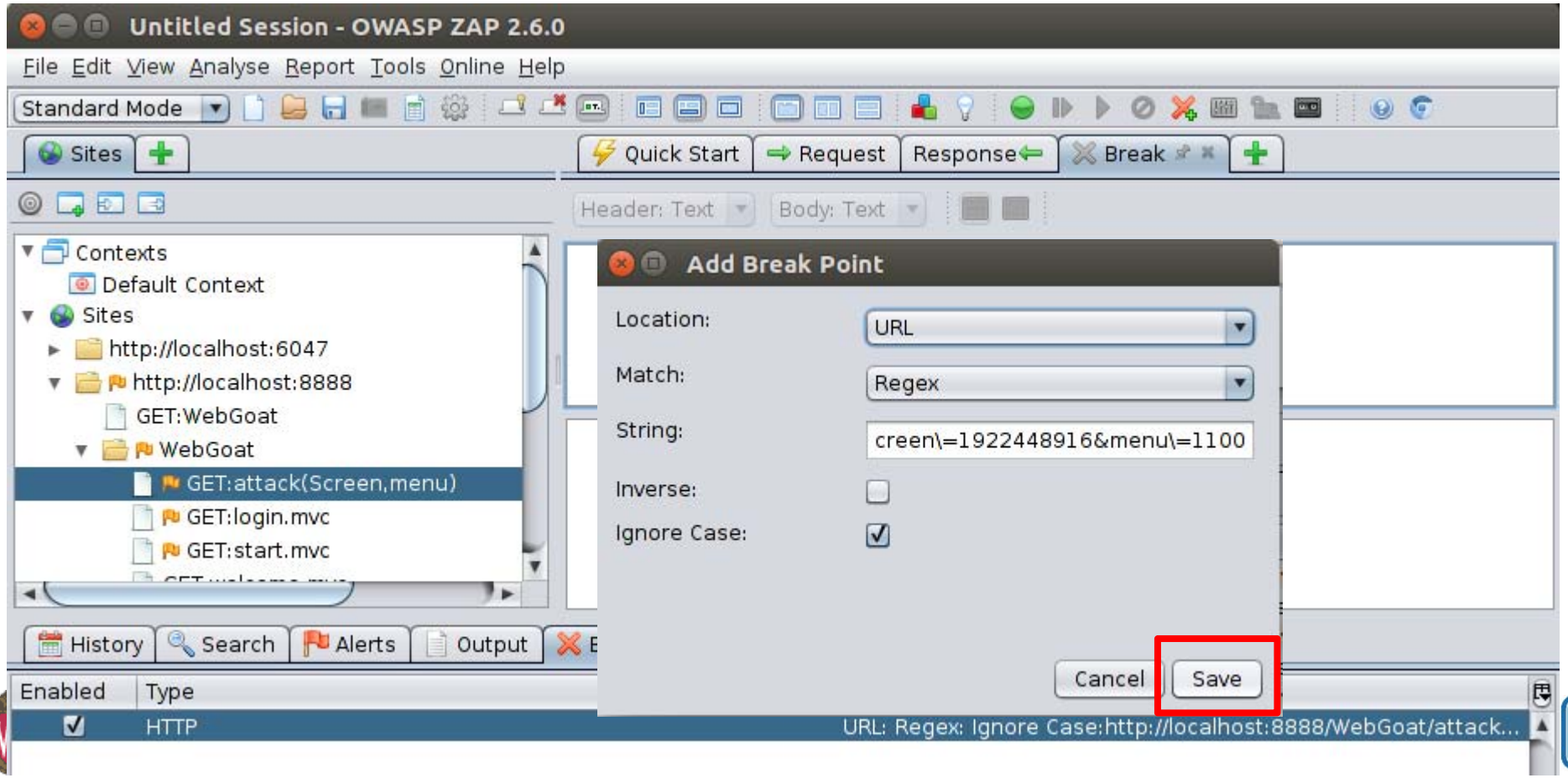


WebGoat Exercise 2

In your browser go to `localhost:8888/WebGoat`

Go to `Injection Flaws-> Command injection`**, choose a lesson plan and click on "View".**

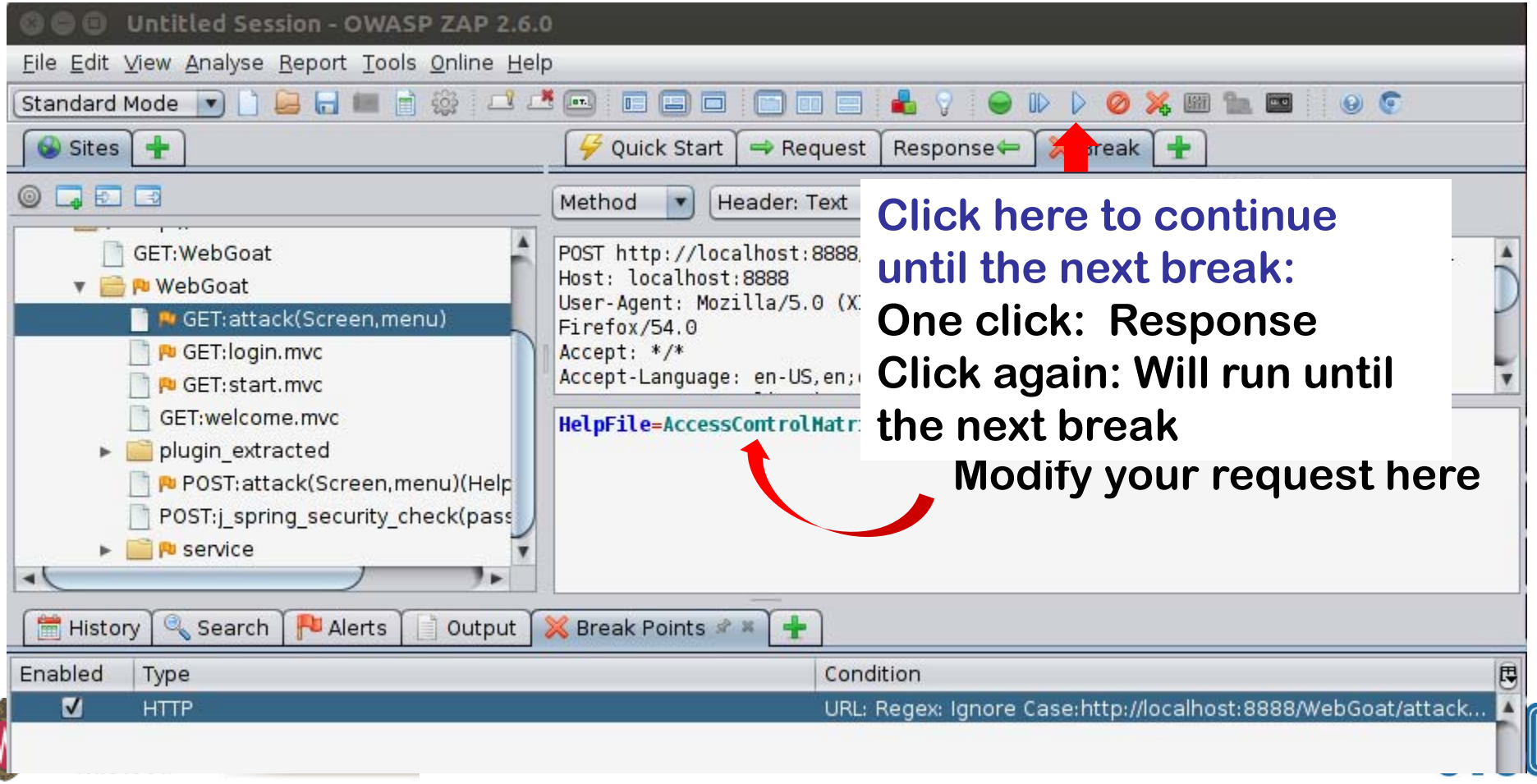
Define a breakpoint in ZAP:



WebGoat Exercise 2

Modify the request to inject an OS command:

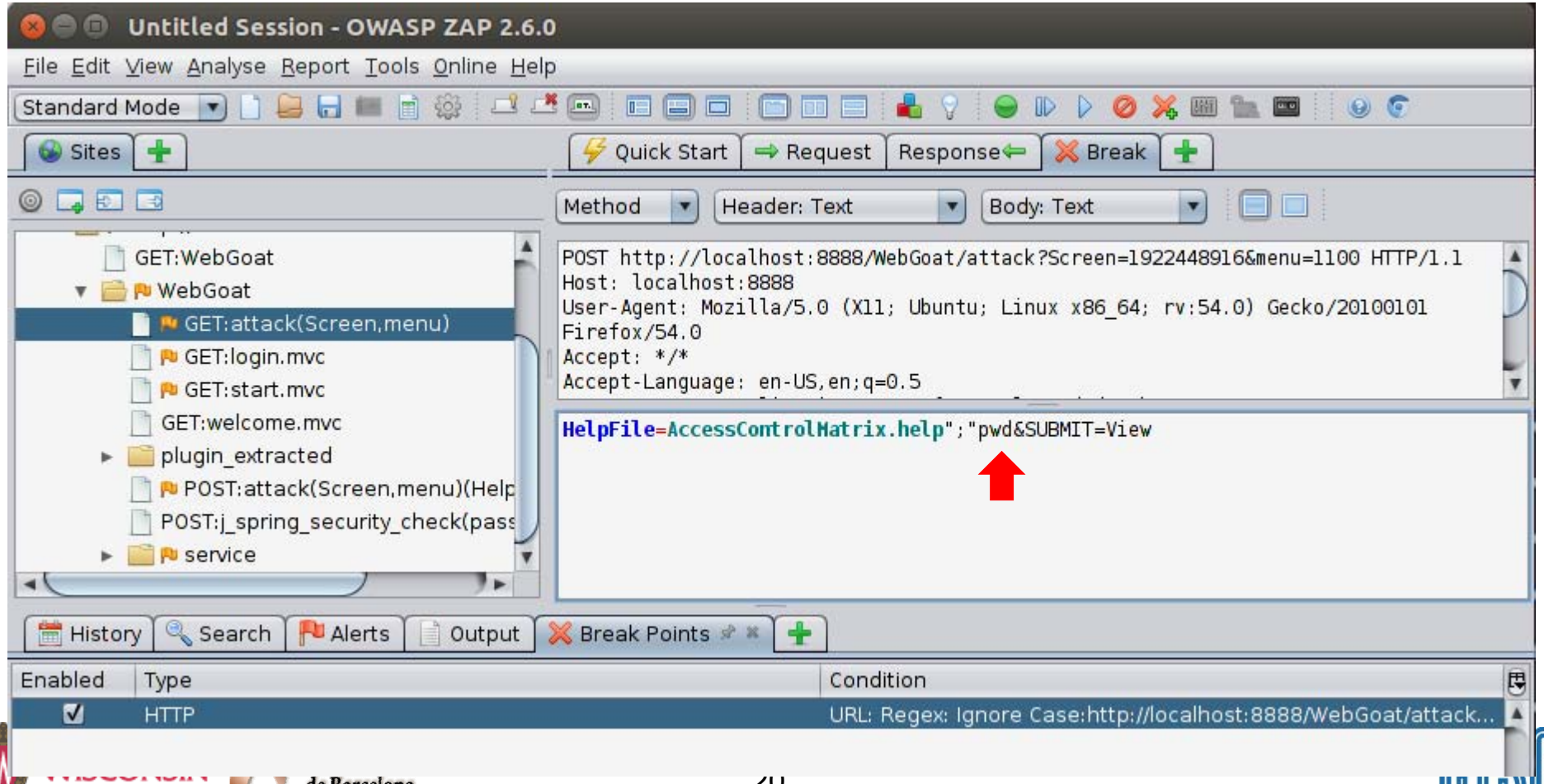
Again, go to `Injection Flaws-> Command injection`, choose a lesson plan and click on "View".



WebGoat Exercise 2

Modify the request to inject an OS command:

```
HelpFile=AccessMatrix.help";"pwd&SUBMIT=View
```



WebGoat Exercise 2

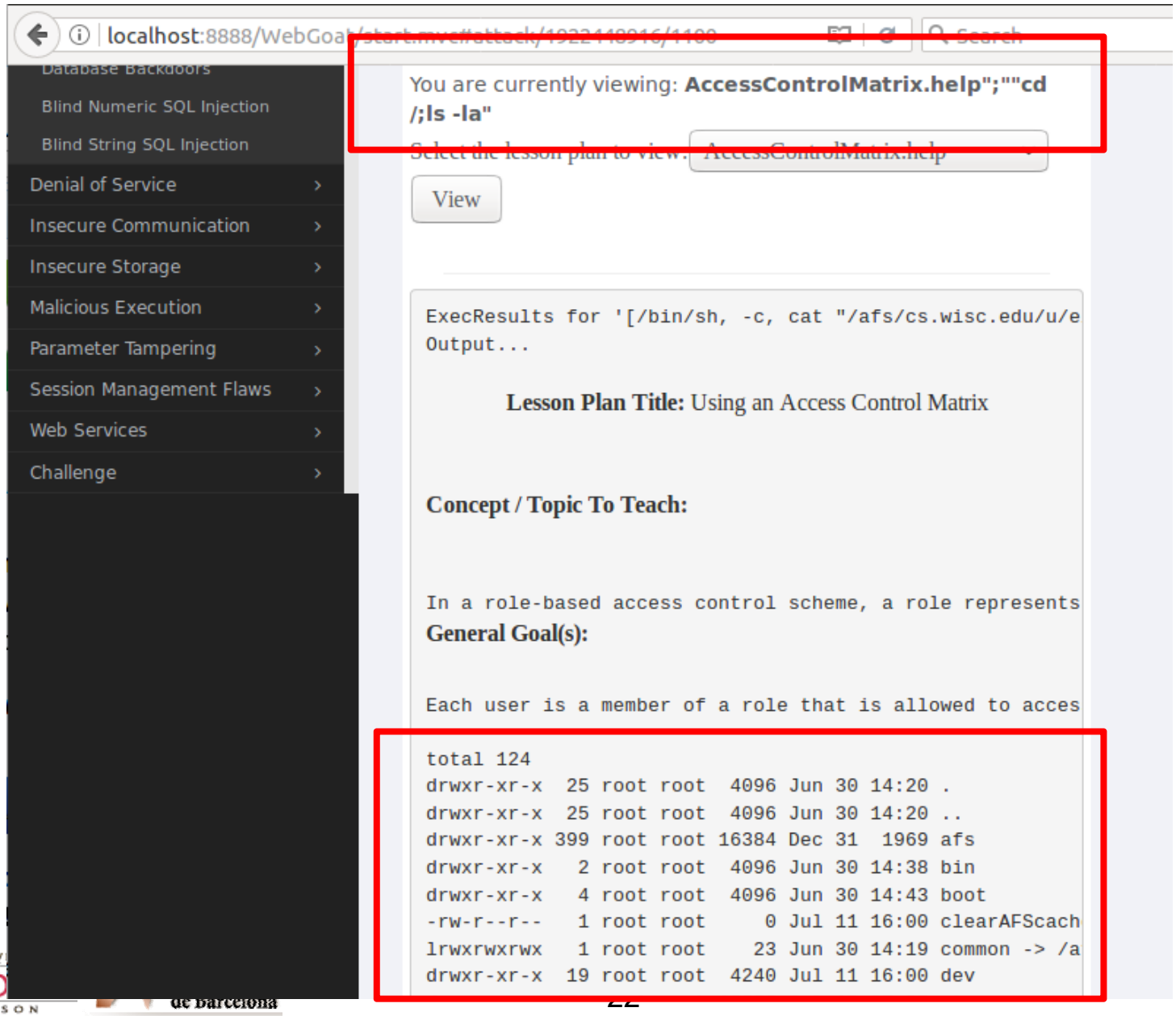
```
HelpFile=AccessMatrix.help”;”pwd&SUBMIT=View
```

What about if we want blank spaces in the command to inject? (out goal is to be able to execute `rm -r`)

```
HelpFile=AccessMatrix.help”;””ls -l”&SUBMIT=View
HelpFile=AccessMatrix.help”;””cd /;ls -la”&SUBMIT=View
```

Pay attention to the command being executed shown in the lesson output:

WebGoat Exercise 2



Edit & Compile WebGoat Lessons

```
cd ~/WebGoat/WebGoat-Lessons-develop/command-injection/src/main/java/org/owasp/webgoat/plugin

vi CommandInjection.java

cd ~/WebGoat/WebGoat-Lessons-develop
mvn clean package
cp target/plugins/*.jar ../WebGoat-7.1/webgoat-container/src/main/webapp/plugin_lessons/
```

(you can run the `~/compile.sh` script)

Edit & Compile WebGoat Lessons

In `CommandInjection.java`:

Sanitize `command` **in** `exec(WebSession s, String command)` **(line 218)**

```
cd ~/WebGoat/WebGoat-Lessons-develop
mvn clean package
cp target/plugins/*.jar ../WebGoat-7.1/webgoat-container/src/main/webapp/plugin_lessons/
```