Secure Coding Practices and Automated Assessments 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

2016 Cybersecurity Summit for Large Facilities and Cyberinfrastructure
Arlington, VA, August 16 2016

What do we do

- **Assess Middleware:** Make cloud/grid software more secure
- **Train:** We teach tutorials for users, developers, sys admins, and managers
- **Research:** Make in-depth assessments more automated and improve quality of automated code analysis

http://www.cs.wisc.edu/mist/papers/VAshort.pdf

Overview

Thinking like an attacker

Thinking like a programmer/designer

- Secure programming techniques

Understanding and using automated assessment tools

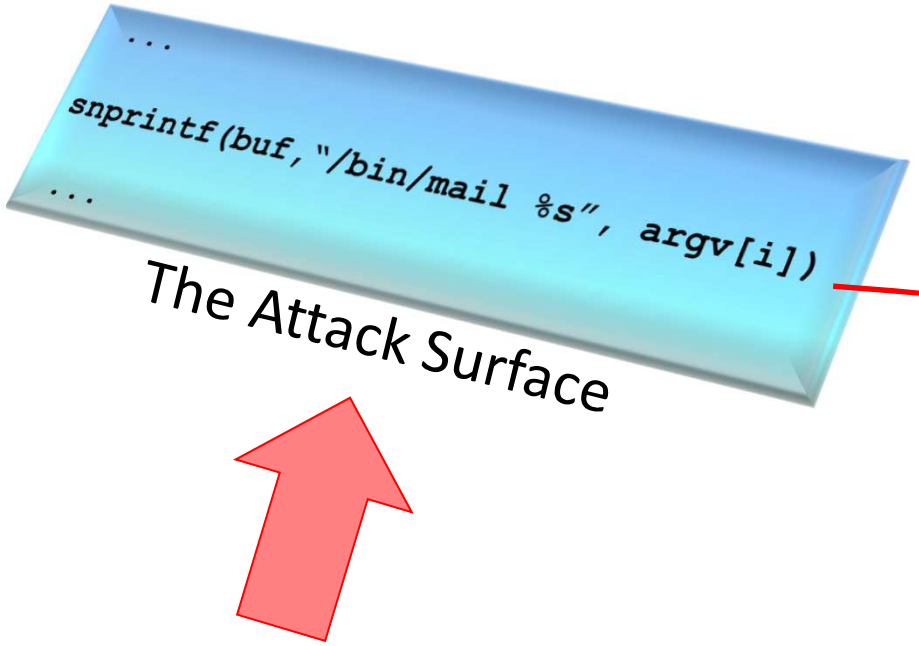
Thinking Like an Attacker

Thinking about an Attack: *Owning* the Bits

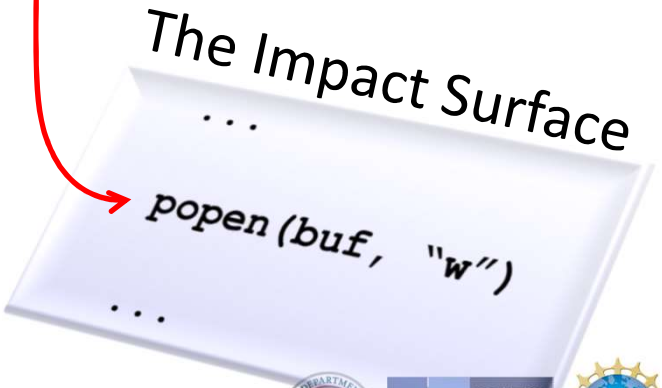
"Dark Arts"
and
"Defense Against the Dark Arts"

Learn to Think Like an Attacker

The Path of an Attack




```
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 == PUTSERVER)
      {
         found = !(strcmp (her, p->me));
      }
      if (found)
         return (p);
      else
         p = p->next;
   }
   return ((struct table *) 0);
```



THE UNIVERSITY of WISCONSIN MADISON

Universitat Autònoma de Barcelona

NATO OTAN

NSF

An Exploit through the Eyes of an Attacker

Exploit:

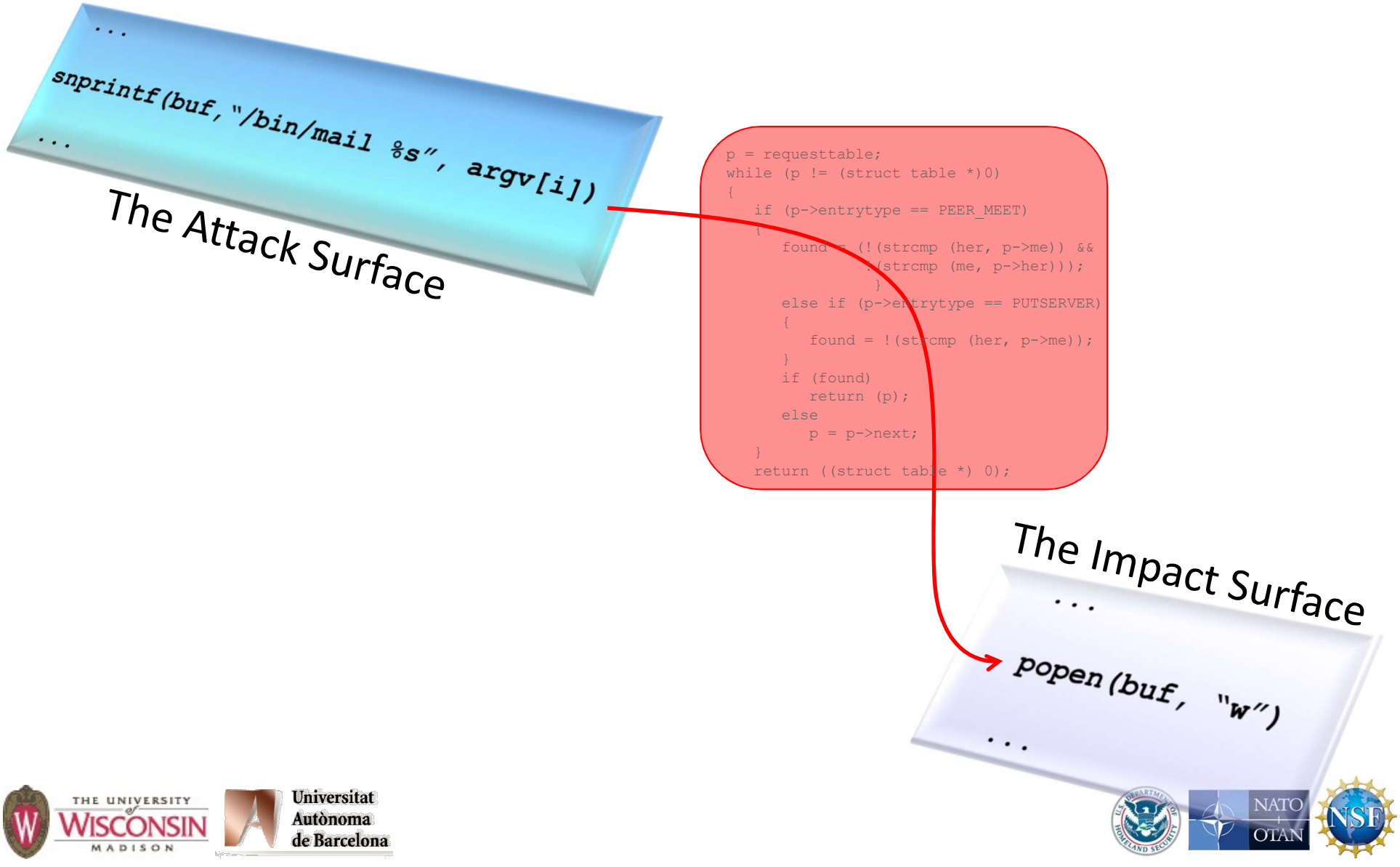
- A manipulation of a program's internal state in a way not anticipated (or desired) by the programmer.

Start at the user's entry point to the program: the *attack surface:*

- Network input buffer
- Field in a form
- Line in an input file
- Environment variable
- Program option
- Entry in a database
- …

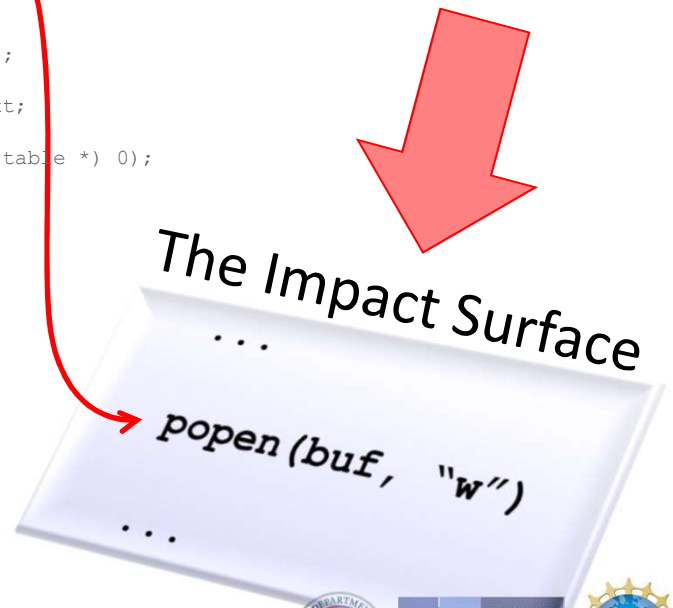
Attack surface: the set of points in the program's interface that can be controlled by the user.

The Path of an Attack



The Path of an Attack


```
p = requesttable;
while (p != (struct table *)0)
{
   if (p->entrytype == PEER_MEET)
   {
      found = (!(strcmp (buf, p->me)) &&
               !(strcmp (me, p->her)));
               }
      else if (p->entrytype == PUTSERVER)
      {
         found = !(strcmp (buf, p->me));
      }
      if (found)
         return (p);
      else
         p = p->next;
   }
   return ((struct table *) 0);
```



THE UNIVERSITY of WISCONSIN MADISON

Universitat Autònoma de Barcelona

NATO OTAN

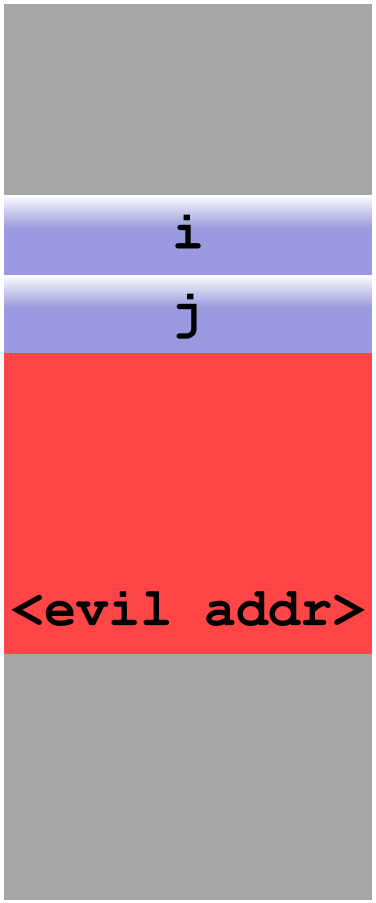
NSF

The Classic: A Stack Smash

```
int foo()
{
   char buffer[100];
   int i, j;
   …

   gets(buffer);

   …
   return(strlen(buffer));
}
```



Thinking Like a Programmer/Designer

Secure Programming: Roadmap

- Pointers and Strings
- Numeric Errors
- Exceptions
- Injection Attacks
- Web Attacks

Discussion of the Practices

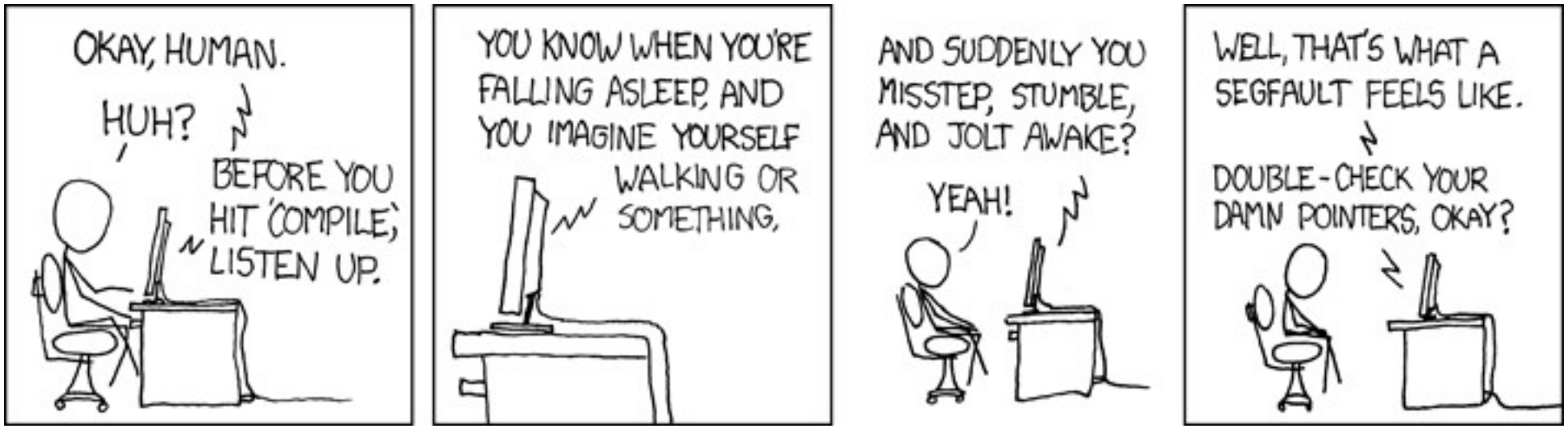
- Description of vulnerability
- Signs of presence in the code
- Mitigations
- Safer alternatives

Pointers and Strings

Buffer Overflows

http://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.html#Listing

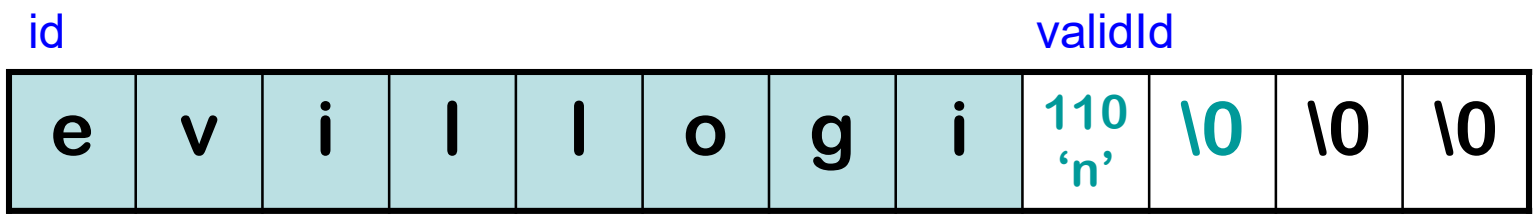
1. Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
2. Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
3. Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
4. Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
5. Missing Authentication for Critical Function
6. Missing Authorization
7. Use of Hard-coded Credentials
8. Missing Encryption of Sensitive Data
9. Unrestricted Upload of File with Dangerous Type
10. Reliance on Untrusted Inputs in a Security Decision


OKAY, HUMAN.
HUH?
BEFORE YOU HIT 'COMPILE', LISTEN UP.
YOU KNOW WHEN YOU'RE FALLING ASLEEP, AND YOU IMAGINE YOURSELF WALKING OR SOMETHING,
AND SUDDENLY YOU MISSTEP, STUMBLE, AND JOLT AWAKE?
YEAH!
WELL, THAT'S WHAT A SEGFAULT FEELS LIKE.
DOUBLE-CHECK YOUR DAMN POINTERS, OKAY?

Buffer Overflow of User Data Affecting Flow of Control


```
char id[8];
int  validId = 0;    /* not valid */
```


```
gets(id);       /* reads "evillogin"*/
```



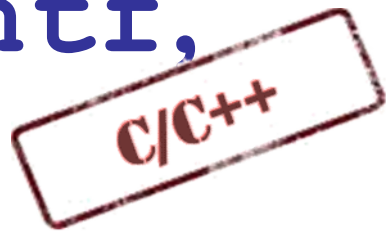

```
/* validId is now 110 decimal */
if (IsValid(id)) validId = 1; /* not true */
if (validId)                  /* is true  */
   {DoPrivilegedOp();}    /* gets executed */
```

Buffer Overflow Danger Signs:
Missing Buffer Size

- `gets`, `getpass`, `getwd`, and `scanf` family (with `%s` or `%[...]` specifiers without width)
  - Impossible to use correctly: size comes solely from user input
  - Source of the first (1987) stack smash attack.
  - Alternatives:

Unsafe	Safer
`gets(s)`	`fgets(s, sLen, stdin)`
`getcwd(s)`	`getwd(s, sLen)`
`scanf("%s", s)`	`scanf("%100s", s)`

strcat, strcpy, sprintf, vsprintf

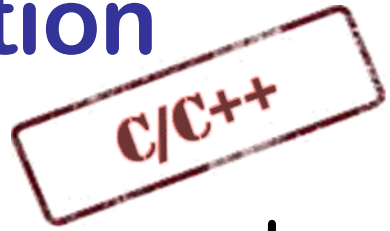


- Impossible for function to detect overflow
  - Destination buffer size not passed
- Difficult to use safely w/o pre-checks
  - Checks require destination buffer size
  - Length of data formatted by printf
  - Difficult & error prone
  - Best incorporated in a safe replacement function

Proper usage: concat s1, s2 into dst

```
If (dstSize < strlen(s1) + strlen(s2) + 1)
        {ERROR("buffer overflow");}
strcpy(dst, s1);
strcat(dst, s2);
```

Buffer Overflow Danger Signs:
Difficult to Use and Truncation

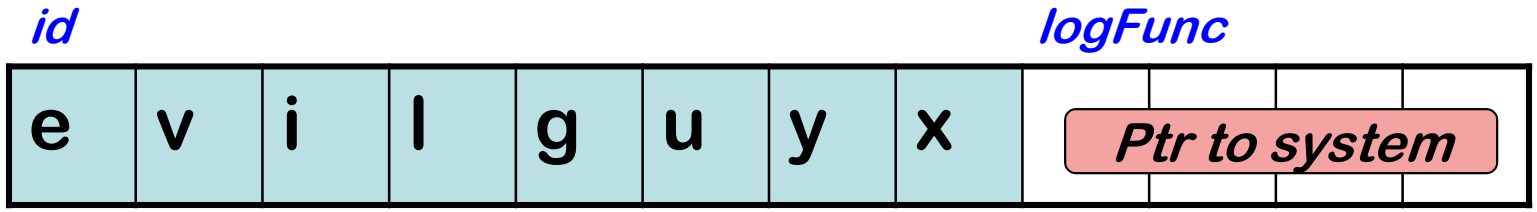


- **`strncat`**(*dst, src, n*)
  - ***`n`*** **is the maximum number of chars of** ***`src`*** **to append (trailing null also appended)**
  - ***can overflow if*** **`n >=(dstSize-strlen(dst))`**
- **`strncpy`**(*dst, src, n*)
  - **Writes** ***`n`*** **chars into** ***`dst`***, **if** **`strlen(src)<n`**, **it fills the other** **`n-strlen(src)`** **chars with 0's**
  - **If** **`strlen(src)>=n`**, ***`dst`*** **is not null terminated**
- **Truncation detection not provided**
- **Deceptively insecure**
  - **Feels safer but requires same careful use as** **`strcat`**

Buffer Overflow of a User Pointer


```
{
  char id[8];
  int  (*logFunc)(char*) = MyLogger;
```


```
  gets(id);      /* reads "evilguyx Ptr to system */
```




```
  /* equivalent to system(userMsg) */
  logFunc(userMsg);
```

Buffer Overflow

Some people believe that buffer overflows are ancient history …

Heartbleed:

- Failure of the OpenSSL library to validate the length field (as compared to the size of the actual message).
- The heartbeat protocol is supposed to echo back the data sent in the request where the amount is given by the payload length.
- Since the length field is not checked, `memcpy` can read up to 64KB of memory.

```
memcpy(bp, pl, payload);
```

Destination. Allocated, used, and freed. OK.

Source. Buffer with the heartbeat record. Improperly used.

Length field. Supplied by an untrusted source.

Buffer Overflow

Some people believe that buffer overflows are ancient history …

Heartbleed:

- Failure of the OpenSSL library to validate the length field (as compared to the size of the actual message).
- The heartbeat protocol is supposed to echo back the data sent in the request where the amount is given by the payload length.
- Since the length field is not checked, `memcpy` can read up to 64KB of memory.

… but they would be wrong.

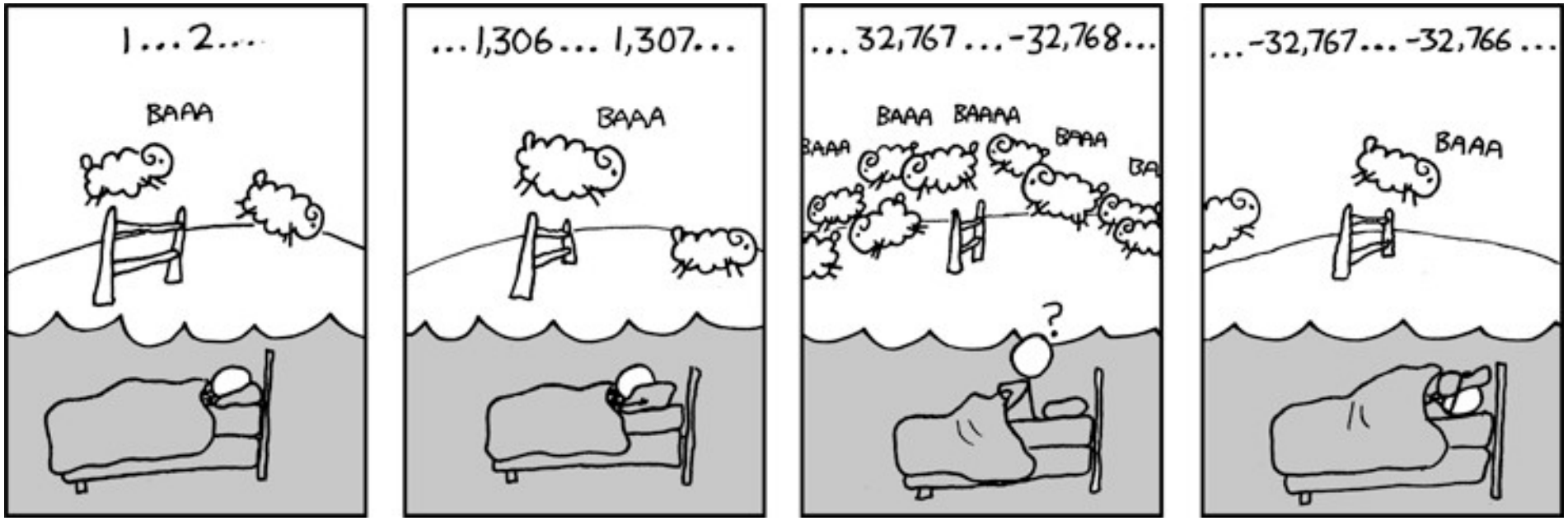
Buffer Overflow

Validation to remediate Heartbleed

Read type and payload length

```
if ((1+2+payload+16)>InputLength)
    return 0; // silently discard
```

Numeric Errors


1...2...
BAAA
...1,306... 1,307...
BAAA
...32,767...-32,768...
BAAA BAAAA
BAAA
BAAA
BA
?
...-32,767... -32,766...
BAAA

Motivation

This is a classic overflow from OpenSSH 3.3.

```
nresp = packet_get_int();
if (nresp > 0) {
    response = xmalloc(nresp*sizeof(char*));
    for (i = 0; i < nresp; i++)
        response[i] = packet_get_string(NULL);
}
```

If **nresp** has the value 1,073,741,824 (**0x40000000**) and **sizeof(char*)** has a value of **4**, then the result of the operation:

```
nresp * sizeof(char*) = 0x100000000
```

overflows, and the argument to **xmalloc()** will be **0**.

From https://www.owasp.org

Integer Vulnerabilities

Description

- Most programming languages allow silent loss of integer data without warning due to:
  - Overflow
  - Truncation
  - Signed vs. unsigned representations
- Code may be secure on one platform, but silently vulnerable on another due to different underlying integer types.

Numeric Parsing Unreported Errors

`atoi`, `atol`, `atof`, `scanf` family (with `%u`, `%i`, `%d`, `%x` and `%o` specifiers)

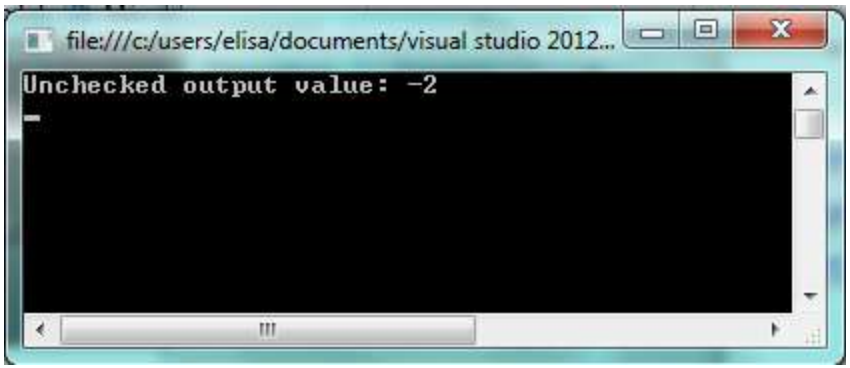
- Out of range values results in unspecified behavior.
- Non-numeric input returns 0.
- Use `strtol`, `strtoul`, `strtoll`, `strtoull`, `strtof`, `strtod`, `strtold` which allow error detection.

Numeric Error

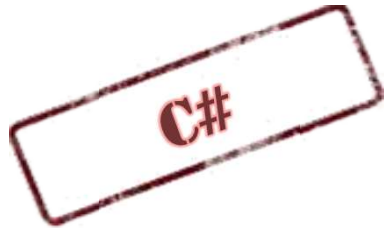
`unchecked` to bypass integer overflow control.

```
const int x = 2147483647;   // Maxint
const int y = 2;
static void UnCheckedMethod() {
        int z=0;
        unchecked {
                z = x * y;
        }
        Console.WriteLine("Unchecked output value: {0}", z);
}
```

http://msdn.microsoft.com/es-es/library/a569z7k8%28v=vs.90%29.aspx

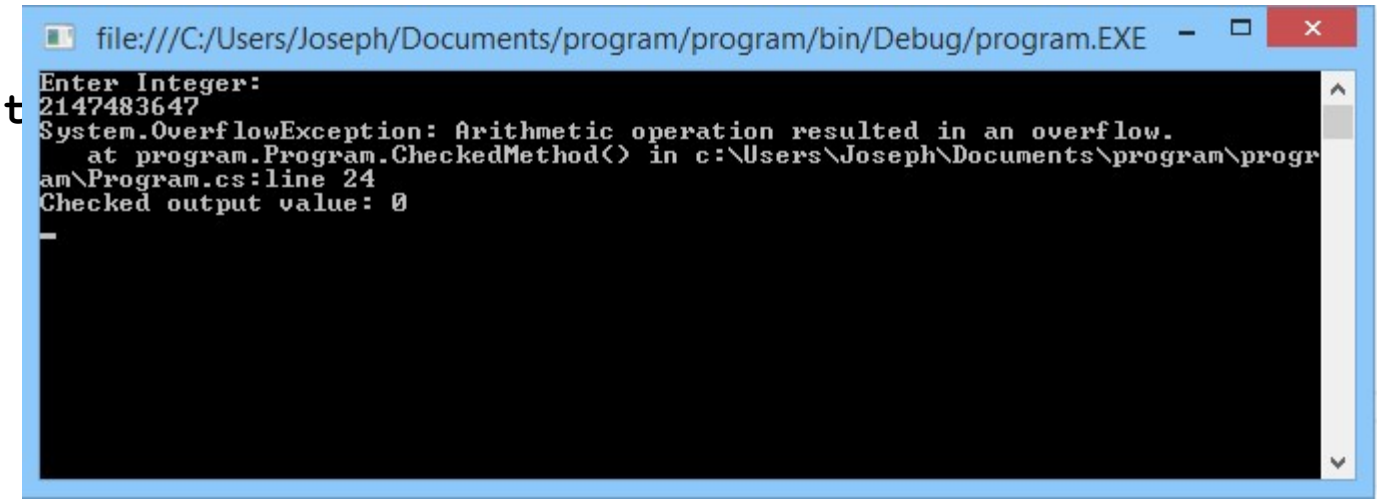


Numeric Error



`checked` for integer overflow control.

```
const int y = 2;
static void CheckedMethod() {
        int z=0;
        Console.WriteLine("Enter Integer:");
        int x = int.Parse(Console.ReadLine());
        try {
                z = checked (x * y);
        }
        catch (System.OverflowException e) {
                Console.WriteLine(e.ToString());
        }
        Console.Writ
}
```



THE UNIVERSITY of WISCONSIN MADISON

Integer Mitigations

- Use correct types, before validation.
- Validate range of data.
- Add code to check for overflow, or use safe integer libraries or large integer libraries.
- Not mixing signed and unsigned integers in a computation.
- Compiler options for signed integer run-time exceptions, and integer warnings.
- Use `strtol`, `strtoul`, `strtoll`, `strtoull`, `strtof`, `strtod`, `strtold`, which allow error detection.

The Cost of Not Checking…

4 Jun 1996: An unchecked 64 bit floating point number assigned to a 16 bit integer

Cost: Development cost: $7 billion
Lost rocket and payload $500 million

Exceptions

Exception Vulnerabilities

- **Exception are a nonlocal control flow mechanism**, usually used to propagate error conditions in languages such as Java, C#, C++, Python, and Ruby.

```
try {
    // code that generates exception
} catch (Exception e) {
    // perform cleanup and error recovery
}
```

- Common Vulnerabilities include:
  - **Ignoring** (program terminates)
  - **Suppression** (catch, but do not handled)
  - **Information leaks** (sensitive information in error messages)

Proper Use of Exceptions

- Add proper exception handling:
  - Handle expected exceptions (i.e. check for errors)
  - Don't suppress:
    - Do not catch just to make them go away.
    - Recover from the error or rethrow exception.
  - Include top level exception handler to avoid exiting: catch, log, and restart
- Do not disclose sensitive information in messages:
  - Only report non-sensitive data.
  - Log sensitive data to secure store, return id of data.
  - Don't report unnecessary sensitive internal state:
    - Stack traces.
    - Variable values.
    - Configuration data.

Exception Suppression

1. User sends malicious data | `user="admin",pwd=null`

```
boolean Login(String user, String pwd){
    boolean loggedIn = true;
    String realPwd = GetPwdFromDb(user);
    try {
        if (!GetMd5(pwd).equals(realPwd))
        {
            loggedIn = false;
        }
    } catch (Exception e) {
        //this can not happen, ignore
    }
    return loggedIn;
}
```

2. System grants access | `Login() returns true`

Unusual or Exceptional Conditions Mitigation

1. User sends malicious data | `user="admin",pwd=null`

```
boolean Login(String user, String pwd){
    boolean loggedIn = true;
    String realPwd = GetPwdFromDb(user);
    try {
        if (!GetMd5(pwd).equals(realPwd))
        {
            loggedIn = false;
        }
    } catch (Exception e) {
        loggedIn = false;
    }
    return loggedIn;
}
```

2. System does not grant access | `Login() returns false`

Exception Suppression

C#

1. User sends malicious data | `user="admin",pwd=null`

```
bool Login(string user, string pwd){
    bool loggedIn = true;
    string realPwd = GetPwdFromDb(user);
    try {
        using (MD5 md5Hash = MD5.Create()){
          if (!string.Equals(realPwd,
               md5Hash.ComputeHash(pwd)));
          {
              loggedIn = false;
          }
        }
    } catch (Exception e) {
        //this can not happen, ignore
    }
     return loggedIn;
```

2. System grants access | `Login() returns true`

Exception Suppression

C#

1. User sends malicious data | `user="admin",pwd=null`

```
bool Login(string user, string pwd){
    bool loggedIn = true;
    string realPwd = GetPwdFromDb(user);
    try {
        using (MD5 md5Hash = MD5.Create()){
```

```
file:///c:/users/elisa/documents/visual studio 2013/Projects/ConsoleApplication1/ConsoleApplicati...
System.ArgumentNullException: String reference not set to an instance of a Strin
g.
Parameter name: s
   at System.Text.Encoding.GetBytes(String s)
   at ConsoleApplication1.Program.Login(String user, String pwd) in c:\Users\Eli
sa\Documents\Visual Studio 2013\Projects\ConsoleApplication1\ConsoleApplication1
\Program.cs:line 21
True
Done!
```

```
        return loggedIn;
    }
```

2. System grants access | `Login() returns true`

Unusual or Exceptional Conditions Mitigation

C#

1. User sends malicious data | `user="admin",pwd=null`

```
bool Login(string user, string pwd){
    bool loggedIn = true;
    string realPwd = GetPwdFromDb(user);
    try {
        using (MD5 md5Hash = MD5.Create()){
          if (!string.Equals(realPwd,
              md5Hash.ComputeHash(pwd)));
          {
              loggedIn = false;
          }
        }
    } catch (Exception e) {
        loggedIn = false;
    }
    return loggedIn;
```

2. System does not grant access | `Login() returns false`

Unusual or Exceptional Conditions Mitigation

C#

1. User sends malicious data

```
user="admin",pwd=null
```

```
bool Login(string user, string pwd){
    bool loggedIn = true;
    string realPwd = GetPwdFromDb(user);
    try {
```

```
file:///c:/users/elisa/documents/visual studio 2013/Projects/ConsoleApplication1/ConsoleApplicati...
System.ArgumentNullException: String reference not set to an instance of a Strin
g.
Parameter name: s
   at System.Text.Encoding.GetBytes(String s)
   at ConsoleApplication1.Program.Login(String user, String pwd) in c:\Users\Eli
sa\Documents\Visual Studio 2013\Projects\ConsoleApplication1\ConsoleApplication1
\Program.cs:line 21
False
Done!
```

```
    }
    return loggedIn;
```

2. System does not grant access

```
Login() returns false
```

WTMI (Way Too Much Info)


```
Login(… user, … pwd)   {
  try {
    ValidatePwd(user, pwd);
  } catch (Exception e) {
    print("Login failed.\n");
    print(e + "\n");
    e.printStackTrace();
    return;
  }
}
```

```
void ValidatePwd(… user, … pwd)
        throws BadUser, BadPwd  {
  realPwd = GetPwdFromDb(user);
  if (realPwd == null)
    throw BadUser("user=" + user);
  if (!pwd.equals(realPwd))
    throw BadPwd("user=" + user
          + " pwd=" + pwd
          + " expected=" + realPwd);
…
```

User exists

Entered pwd

```
Login failed.
BadPwd: user=bob pwd=x expected=password
BadPwd:
 at Auth.ValidatePwd (Auth.java:92)
 at Auth.Login (Auth.java:197)
 …
 com.foo.BadFramework(BadFramework.java:71)
 …
```

User's actual password ?!?
(passwords aren't hashed)

Reveals internal structure
(libraries used, call structure, version information)

WTMI (Way Too Much Info)


```
#!/usr/bin/ruby

def ValidatePwd(user, password)
        if wrong password
                raise "Bad passwd for user #{user} expected #{password}"
        end
end

def Login(user, password)
        ValidatePwd(user, password);
rescue Exception => e
        puts "Login failed"
        puts e.message
        puts e.backtrace.inspect
end
```

User exists

User's actual password ?!?

Reveals internal structure

```
Login failed.
Bad password for user Elisa expected pwd
["./test3:4:in `ValidatePwd'", "./test3:8:in `Login'", "./test3:15"]
```

The Right Amount of Information


```
Login {
    try {
        ValidatePwd(user, pwd);
    } catch (Exception e) {
        logId = LogError(e); // write exception and return log ID.
        print("Login failed, username or password is invalid.\n");
        print("Contact support referencing problem id " + logId
                + " if the problem persists");
        return;
   }
}
```

Log sensitive information

Generic error message
(id links sensitive information)

```
void ValidatePwd(… user, … pwd) throws BadUser, BadPwd  {
 realPwdHash = GetPwdHashFromDb(user)
 if (realPwdHash == null)
   throw BadUser("user=" + HashUser(user));
 if (!HashPwd(user, pwd).equals(realPwdHash))
   throw BadPwdExcept("user=" + HashUser(user));
 …
}
```

User and password are hashed
(minimizes damage if breached)

Injection Attacks

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

JAVA

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

JAVA

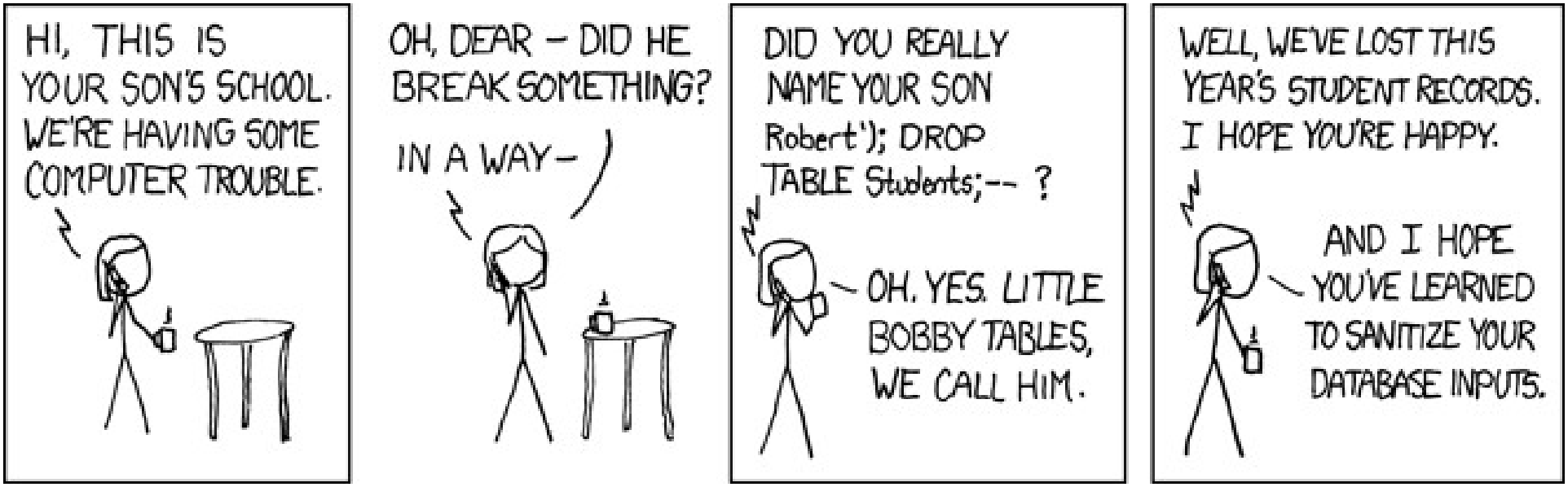
1. User sends malicious data

```
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

JAVA

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)";
}
```

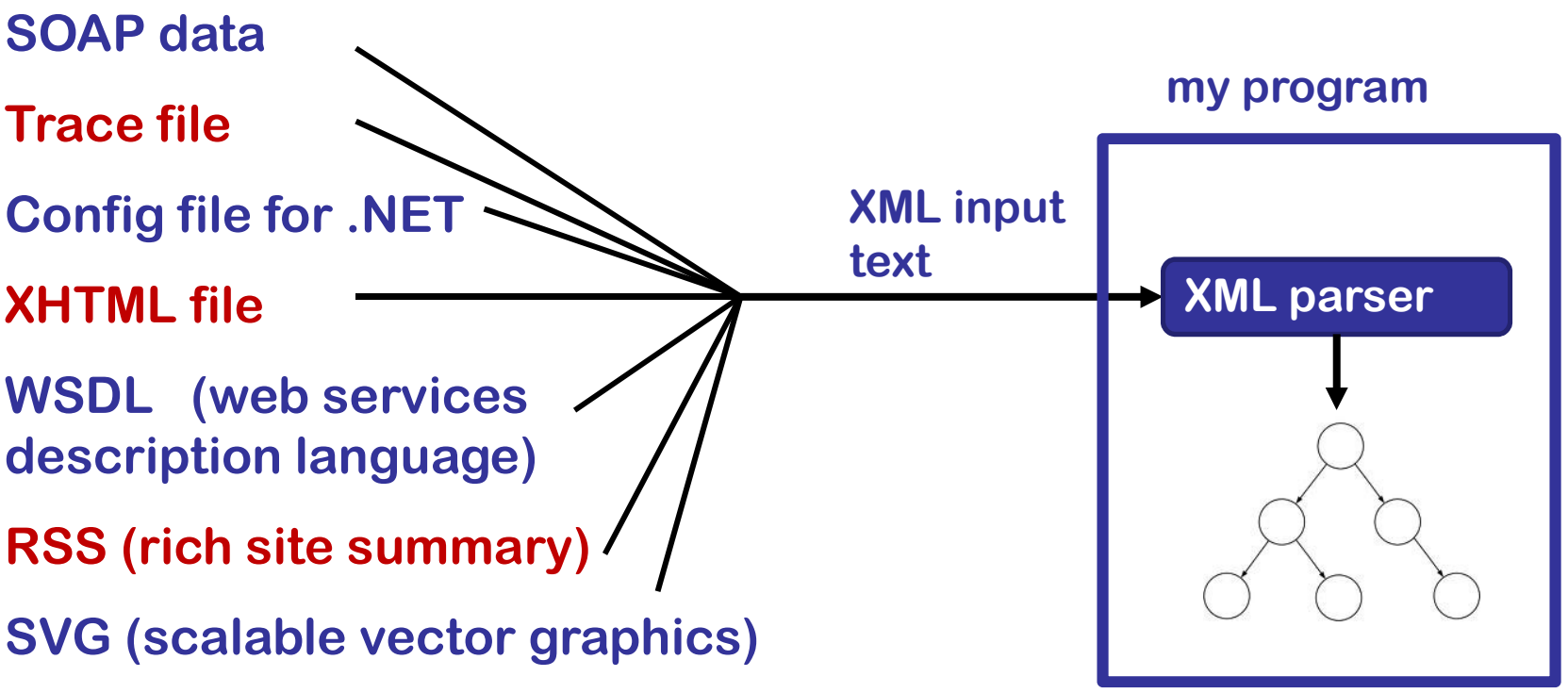
```
sub QuotePyString  {
  my $s = shift;
  $s =~ s/\\/\\\\/g;     # \  ➔ \\
  $s =~ s/'/\\'/g;       # '  ➔ \'
  $s =~ s/\n/\\n/g;      # NL ➔ \n
  return "'$s'";         # add quotes
}
```

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

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

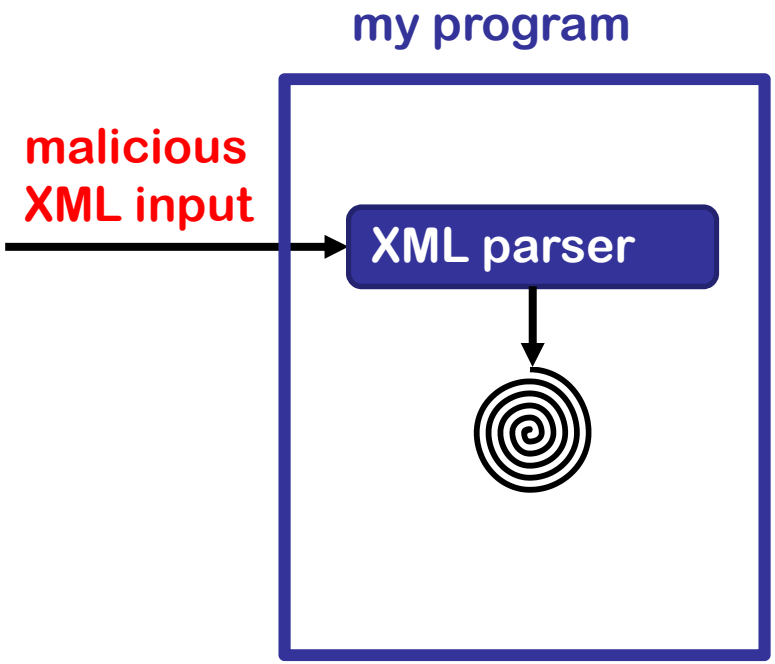
XML Injection

XML Injection



XML Injection

- Attack an application that parses XML input.
- The processing is carried out by a weakly configured XML parser.
- The XML parser crashes or executes incorrectly on the input data.
- Can cause a DoS or leak of sensitive information.



XML Injection

Two kinds of attacks:

- **XML Bombs.**

  Block of XML that is valid, but crashes the program that attempts to parse it.

- **XML External Entity (XEE).**

  Entity replacement values come from external URIs causing information disclosure or other undesirable behaviors.

XML Bombs

XML bombs

- Block of XML that is both well-formed and valid.
- Crashes or hangs a program when that program attempts to parse it.
- Example: the *Billion Laughs Attack:*

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

XML Bombs

XML bombs

- Block of XML that is both well-formed and valid.
- Crashes or hangs a program when that program attempts to parse it.
- Example: the *Billion Laughs Attack:*

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

XML Bombs

XML bombs

- Block of XML that is both well-formed and valid.
- Crashes or hangs a program when that program attempts to parse it.
- Example: the *Billion Laughs Attack:*

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

XML Bombs

XML bombs

- Block of XML that is both well-formed and valid.
- Crashes or hangs a program when that program attempts to parse it.
- Example: the *Billion Laughs Attack:*

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Expansion: A billion "lol"s almost 3GB of memory!

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

XML Bombs

XML bomb

- Quadratic Blowup Attack:

```
<?xml version="1.0"?>
<!DOCTYPE kaboom [
  <!ENTITY a "aaaaaaaaaaaaaaaaaa...">
]>
<kaboom>&a;&a;&a;&a;&a;&a;&a;&a;&a;...</kaboom>
```

50,000 character long

50,000 times

- XML bomb attack payload slightly over 200 KB
- Expands to 2.5 GB when parsed

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

Mitigated XML Bombs

- Disable inline expansion of entities.
- If that is not possible, limit the size of expanded entities.

Mitigated XML Bombs

Examples in .NET 4.0:

Disable inline DTDs

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.DtdProcessing = DtdProcessing.Prohibit;
XmlReader reader = XmlReader.Create(stream, settings);
```

Limit the size of expanded entities

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.ProhibitDtd = false;
settings.MaxCharactersFromEntities = 1024;
XmlReader reader = XmlReader.Create(stream, settings);
```

Mitigated XML Bombs

Example in Ruby:

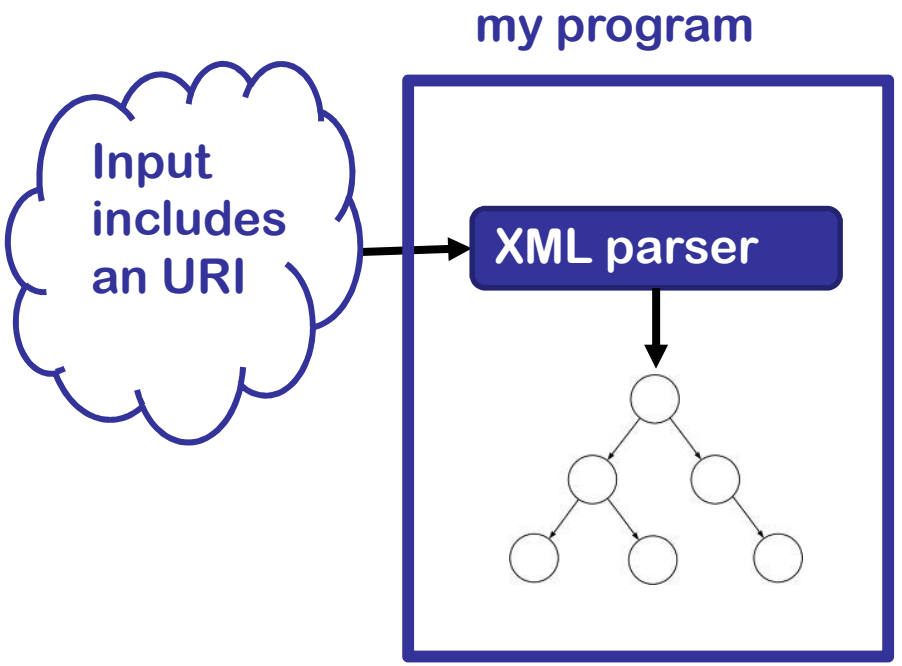
Disable entity expansion (and limit the size of expanded entities):

Disable entity expansion in Ruby's REXML document parser.

```
REXML::Document.entity_expansion_limit = 0
```

XML External Entity (XXE) Attack

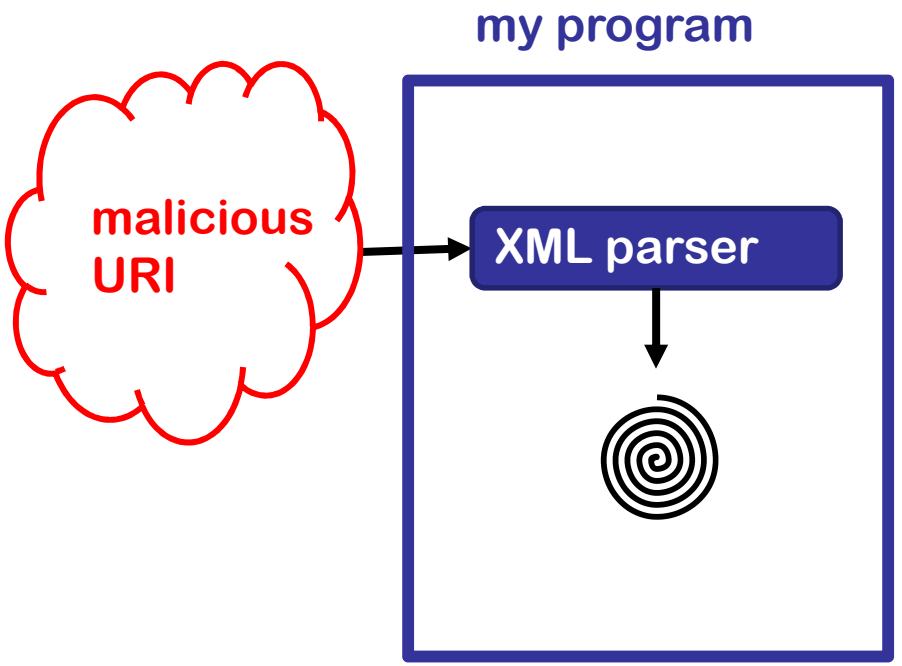
- An XML input containing a reference to an external entity is processed by a weakly configured XML parser.
- Entity replacement values are pulled from external URIs.



- This may lead to the disclosure of confidential data, denial of service, port scanning on the machine where the parser is located.

XML External Entity (XXE) Attack

- An XML input containing a reference to an external entity is processed by a weakly configured XML parser.
- Entity replacement values are pulled from external URIs.



- This may lead to the disclosure of confidential data, denial of service, port scanning on the machine where the parser is located.

XML External Entity (XXE) Attack

Accessing a local resource that may not return:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >
```

Disclosing sensitive information:

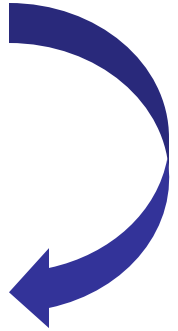
```
<!ENTITY xxe SYSTEM "file:///etc/passwd" >
```

XML External Entity (XXE) Attack

Attacker controlled server can cause a DoS:

```
<!ENTITY xxe SYSTEM "http://www.attacker.com/dos.ashx" >

public void ProcessRequest(HttpContext context) {
      context.Response.ContentType = "text/plain";
      byte[] data = new byte[1000000];
      for (int i = 0; i<data.Length; i++)
            data[i] = (byte)'A';
      while (true) {
            context.Response.OutputStream.Write
                                    (data, 0, data.Length);
            context.Response.Flush();
      }
}
```



Mitigated XML External Entity (XXE) Attacks

- Configure the XML parser to avoid resolving external references
- If not possible, must modify the XML parser:
  - Timeout to prevent infinite delay attacks.
  - Limit the amount of data to be retrieved.
  - Restrict the XmlResolver from retrieving resources on the local host.

Mitigated XML External Entity (XXE) Attacks

Example of configuring the XML parser to avoid resolving external references

In .NET 4.0:

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.XmlResolver = null;
XmlReader reader = XmlReader.Create(stream, settings);
```

In PHP when using the default XML parser:

```
libxml_disable_entity_loader(true);
```

Web Attacks

Cross Site Scripting (XSS)

- Injection into an HTML page
  - HTML tags
  - JavaScript code
- Reflected (from URL) or persistent (stored from prior attacker visit)
- Web application fails to neutralize special characters in user supplied data
- Mitigate by preventing or encoding/escaping special characters
- Special characters and encoding depends on context
  - HTML text
  - HTML tag attribute
  - HTML URL

Reflected Cross Site Scripting (XSS)

3. Generated HTML displayed by browser

```
<html>
...
You searched for:
widget
...
</html>
```

1. Browser sends request to web server

```
http://example.com?q=widget
```

2. Web server code handles request

```
...
String query = request.getParameter("q");
if (query != null) {
    out.writeln("You searched for:\n" + query);
}
...
```

Reflected Cross Site Scripting (XSS)

3. Generated HTML displayed by browser

```
<html>
…
You searched for:
<script>alert('Boo!')</script>
…
</html>
```

1. Browser sends request to web server

```
http://example.com?q=<script>alert('Boo!')</script>
```

2. Web server code handles request

```
…
String query = request.getParameter("q");
if (query != null) {
    out.writeln("You searched for:\n" + query);
}
…
```

XSS Mitigation

JAVA

3. Generated HTML displayed by browser

```
<html>
...
Invalid query
...
</html>
```

1. Browser sends request to web server

```
http://example.com?q=<script>alert('Boo!')</script>
```

2. Web server code correctly handles request

```
...
String query = request.getParameter("q");
if (query != null) {
    if (query.matches("^\\w*$"))  {
        out.writeln("You searched for:\n" + query);
    }  else  {
        out.writeln("Invalid query");
    }
}
...
```

Cross Site Request Forgery (CSRF)

- CSRF is when loading a web pages causes a malicious request to another server
- Requests made using URLs or forms (also transmits any cookies for the site, such as session or auth cookies)
  - http://bank.com/xfer?amt=1000&toAcct=joe HTTP GET method
  - HTTP POST method

```
<form action=/xfer method=POST>
  <input type=text name=amt>
  <input type=text name=toAcct>
</form>
```

- Web application fails to distinguish between a user initiated request and an attack
- Mitigate by using a large random nonce

Cross Site Request Forgery (CSRF)

0. User has a session already open with their bank.
1. User loads bad page from web server
   - XSS
   - Bad guy's server
   - Fake server
   - Compromised server
2. Web browser makes a request to the victim web server directed by bad page
   - Tags such as
     <img src='http://bank.com/xfer?amt=1000&toAcct=evil37'>
   - JavaScript
3. Victim web server processes request and assumes request from browser is valid
   - Session IDs in cookies are automatically sent along

SSL does not help – channel security is not an issue here

Successful CSRF Attack

1. User visits evil.com

```
http://evil.com
```

2. evil.com returns HTML

```
<html>
…
<img src='http://bank.com/xfer?amt=1000&toAcct=evil37'>
…
</html>
```

3. Browser sends attack

```
http://bank.com/xfer?amt=1000&toAcct=evil37
```

4. bank.com server code handles request

```
…
String id = response.getCookie("user");
userAcct = GetAcct(id);
If (userAcct != null)  {
    deposits.xfer(userAcct, toAcct, amount);
}
```

CSRF Mitigation

1. User visits evil.com

2. evil.com returns HTML

3. Browser sends attack

JAVA

Very unlikely attacker will provide correct nonce

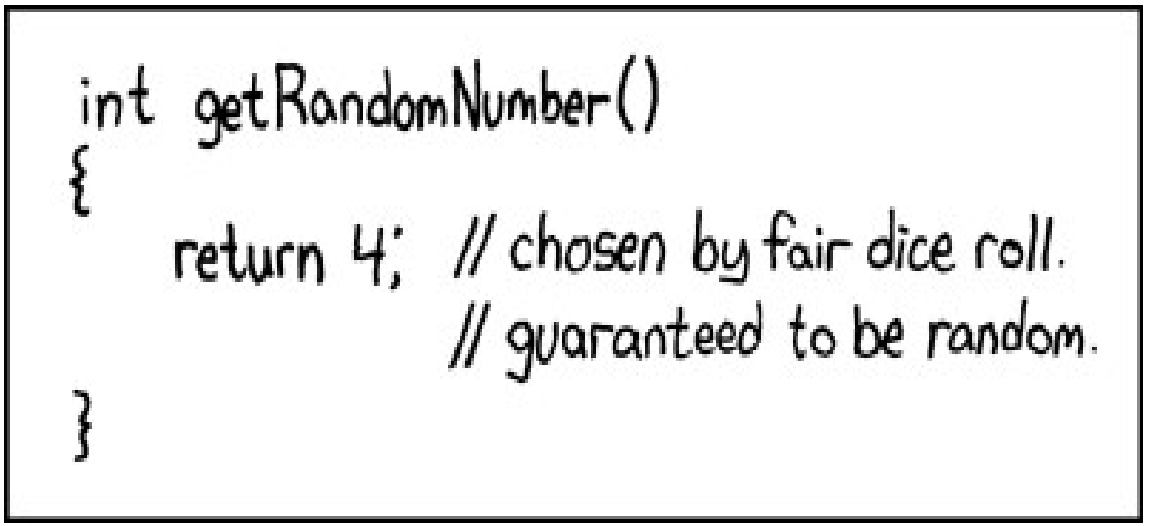
4. bank.com server code correctly handles request

```
...
String nonce = (String)session.getAttribute("nonce");
String id = response.getCookie("user");
if (Utils.isEmpty(nonce)
      || !nonce.equals(getParameter("nonce")   {
    Login();   // no nonce or bad nonce, force login
    return;    //     do NOT perform request
}              // nonce added to all URLs and forms
userAcct = GetAcct(id);
if (userAcct != null)   {
    deposits.xfer(userAcct, toAcct, amount);
}
```

Session Hijacking

- Session IDs identify a user's session in web applications.
- Obtaining the session ID allows impersonation
- Attack vectors:
  - Intercept the traffic that contains the ID value
  - Guess a valid ID value (weak randomness)
  - Discover other logic flaws in the sessions handling process

Good Session ID Properties



http://xkcd.com/221

- **Hard to guess**
  - Large entropy (big random number)
  - No patterns in IDs issued
- **No reuse**

Session Hijacking Mitigation

- Create new session id after
  - Authentication
  - switching encryption on
  - other attributes indicate a host change (IP address change)
- Encrypt to prevent obtaining session ID through eavesdropping
- Expire IDs after short inactivity to limit exposure of guessing or reuse of illicitly obtained IDs
- Entropy should be large to prevent guessing
- Invalidate session IDs on logout and provide logout functionality

Session Hijacking Example

1. An insecure web application accepts and reuses a session ID supplied to a login page.
2. Attacker tricked user visits the web site using attacker chosen session ID
3. User logs in to the application
4. Application creates a session using attacker supplied session ID to identify the user
5. The attacker uses session ID to impersonate the user

Successful Hijacking Attack

1. Tricks user to visit

```
http://bank.com/login;JSESSIONID=123
```

2. User Logs In

```
http://bank.com/login;JSESSIONID=123
```

3. Creates the session

```
HTTP/1.1 200 OK
Set-Cookie:
JSESSIONID=123
```

4. Impersonates the user

```
http://bank.com/home
Cookie: JSESSIONID=123
```

```
if(HttpServletRequest.getRequestedSessionId() == null)
{
      HttpServletRequest.getSession(true);
}
...
```

Mitigated Hijacking Attack

1. Tricks user to visit

```
http://bank.com/login;JSESSIONID=123
```

2. User Logs In

```
http://bank.com/login;JSESSIONID=123
```

3. Creates the session

```
HTTP/1.1 200 OK
Set-Cookie:
JSESSIONID=XXX
```

4. Impersonates the user

```
http://bank.com/home
Cookie: JSESSIONID=123
```

```
HttpServletRequest.invalidate();
HttpServletRequest.getSession(true);
...
```

Open Redirect

(AKA: URL Redirection to Untrusted Site, and Unsafe URL Redirection)

- Description
  - Web app redirects user to malicious site chosen by attacker
    - URL parameter (reflected)

      http://bank.com/redir?url=http://evil.com
    - Previously stored in a database (persistent)
  - User may think they are still at safe site
  - Web app uses user supplied data in redirect URL
- Mitigations
  - Use white list of tokens that map to acceptable redirect URLs
  - Present URL and require explicit click to navigate to user supplied URLs

Open Redirect Example

1. User receives phishing e-mail with URL

   `http://www.bank.com/redir?url=http://evil.com`
2. User inspects URL, finds hostname valid for their bank
3. User clicks on URL
4. Bank's web server returns a HTTP redirect response to malicious site
5. User's web browser loads the malicious site that looks identical to the legitimate one
6. Attacker harvests user's credentials or other information

Successful Open Redirect Attack

1. User receives phishing e-mail

```
Dear bank.com costumer,
Because of unusual number of invalid login
attempts...
<a href="http://bank.com/redir?url=http://evil.com">
Sign in to verify</a>
```

2. Opens `http://bank.com/redir?url=http://evil.com`

```
String url = request.getParameter("url");
if (url != null)  {
    response.sendRedirect( url );
}
```

3. Web server redirects `Location: http://evil.com`

4. Browser requests http://evil.com

```
<h1>Welcome to bank.com<h1>
Please enter your PIN ID:
<from action="login">
...
```

5. Browser displays forgery

Open Redirect Mitigation

JAVA

1. User receives phishing e-mail

```
Dear bank.com costumer,
•••
```

2. Opens `http://bank.com/redir?url=http://evil.com`

```
boolean isValidRedirect(String url) {
   List<String> validUrls = new ArrayList<String>();
   validUrls.add("index");
   validUrls.add("login");
   return (url != null && validUrls.contains(url));
}
•••
if (!isValidRedirect(url)){
   response.sendError(response.SC_NOT_FOUND, "Invalid URL");
   •••
```

3. bank.com server code correctly handles request

`404 Invalid URL`

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

Questions?

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

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

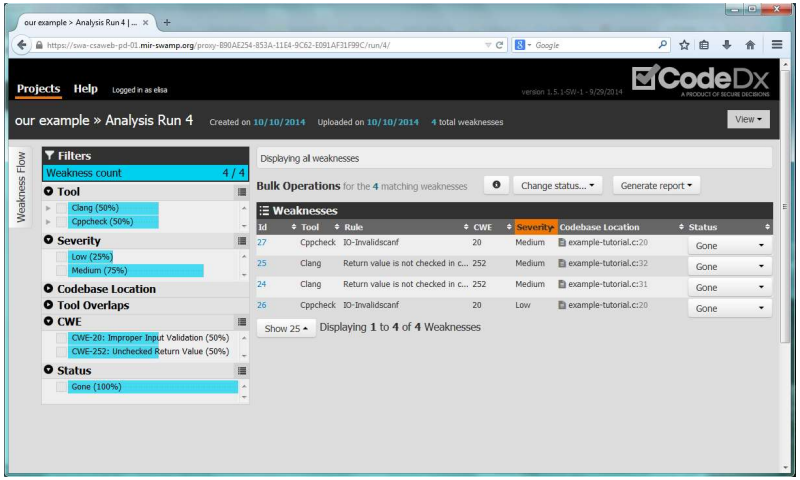
elisa@cs.wisc.edu

1. What You Need to Know about How Tools Work

2. The Tools And Their Use

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, -Unreachable-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++;
   }
}
```

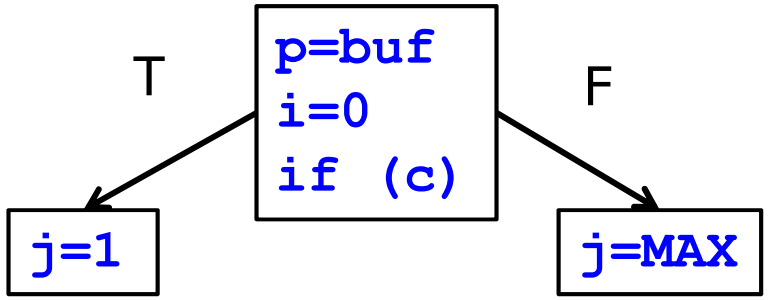
```
void foo() {
   char buf[MAX] = "example";
   int i, j, k;
   char a, b;
   char *p = buf;

   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)
```

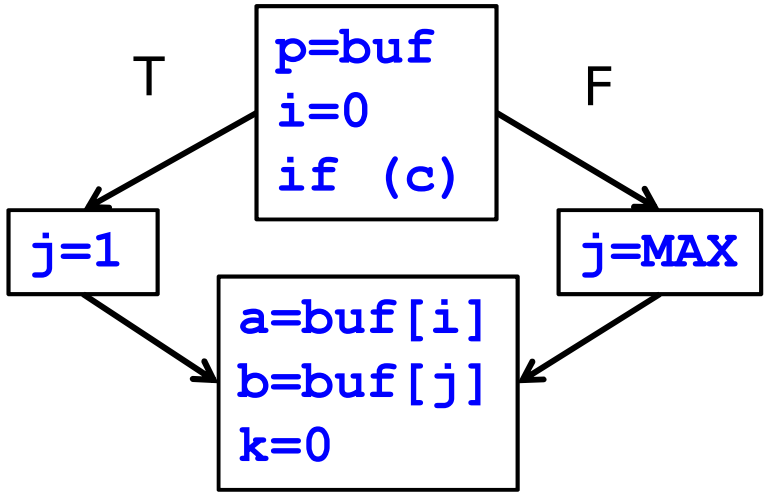
```
void foo() {
   char buf[MAX] = "example";
   int i, j, k;
   char a, b;
   char *p = buf;

   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++;
   }
}
```



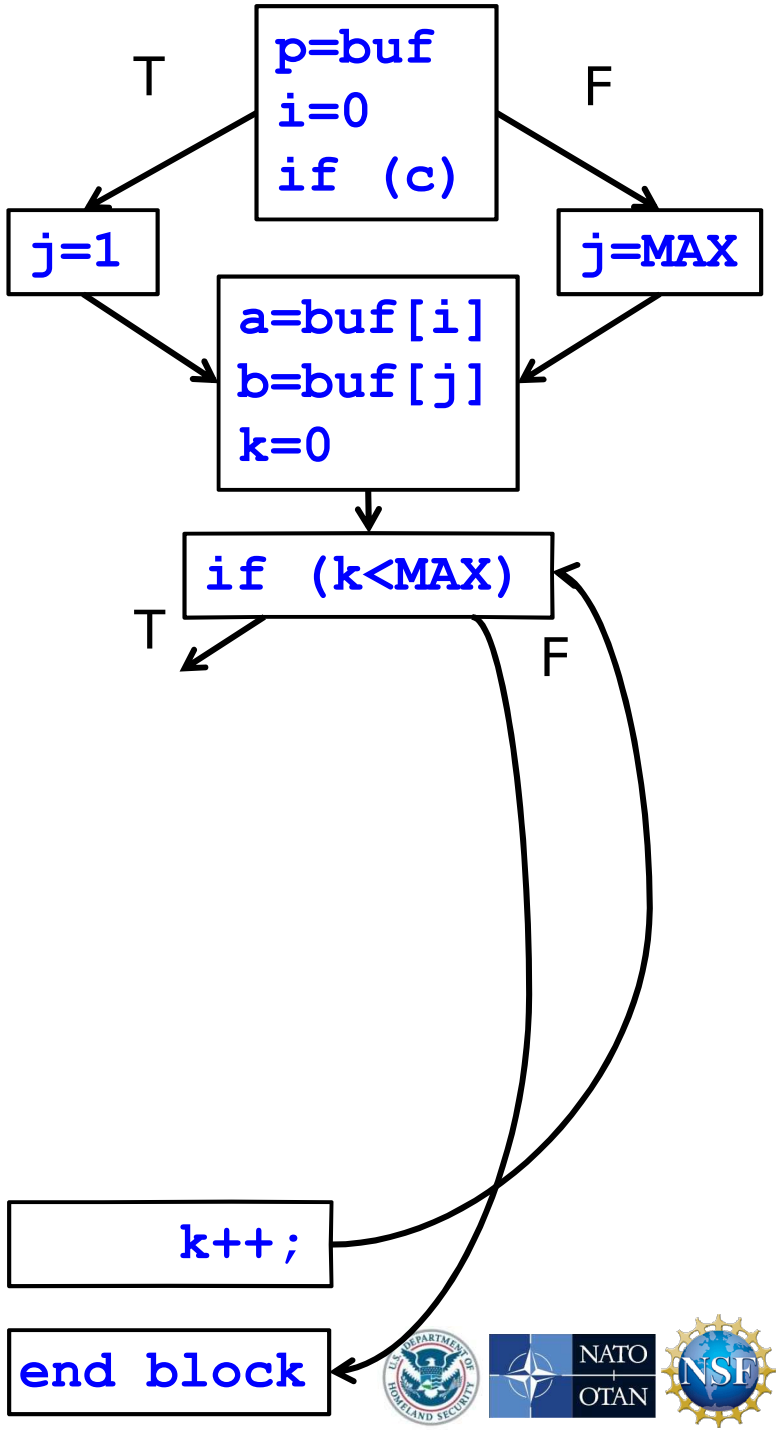
```
void foo() {
   char buf[MAX] = "example";
   int i, j, k;
   char a, b;
   char *p = buf;

   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++;
   }
```



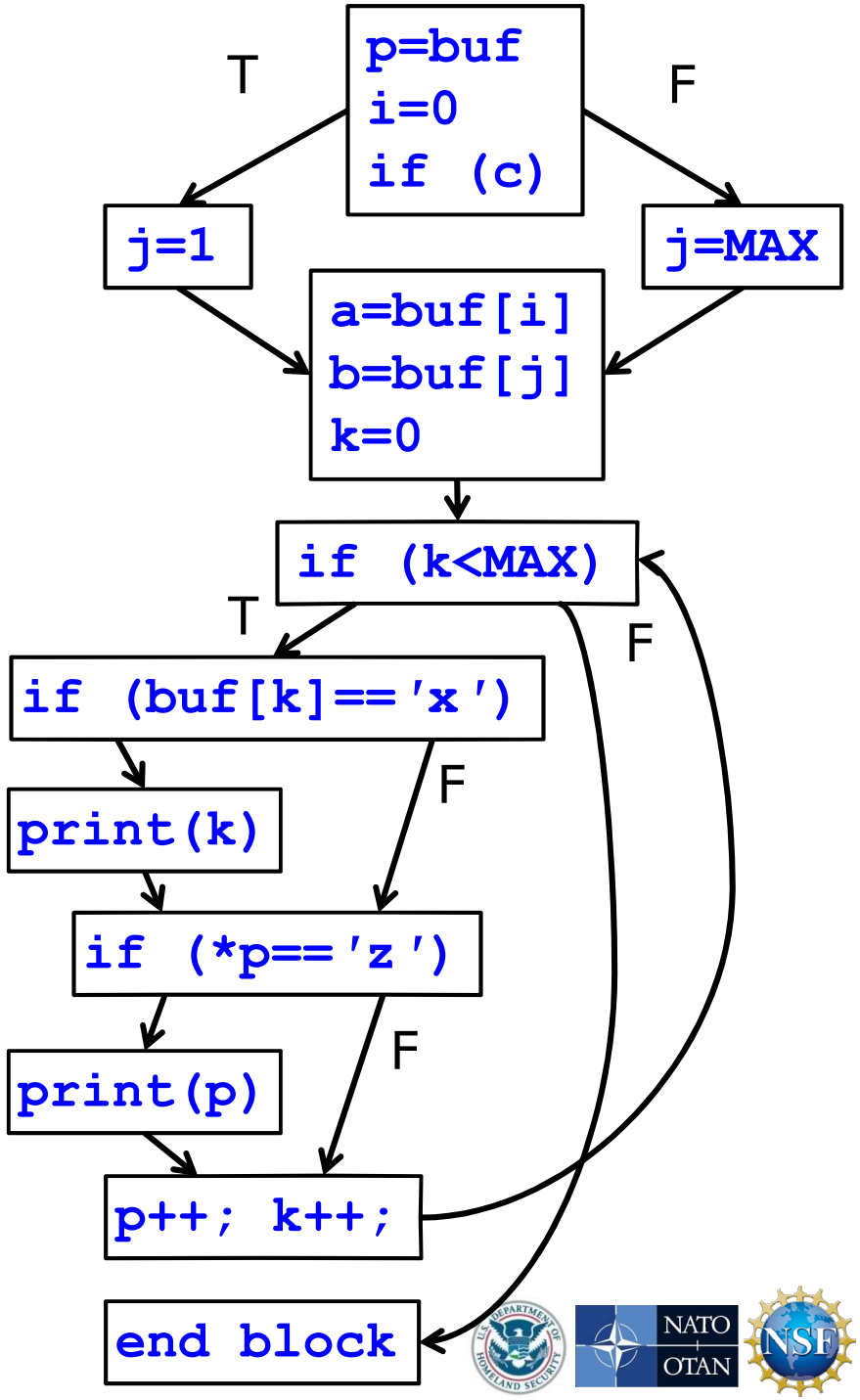
```
void foo() {
   char buf[MAX] = "example";
   int i, j, k;
   char a, b;
   char *p = buf;

   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?

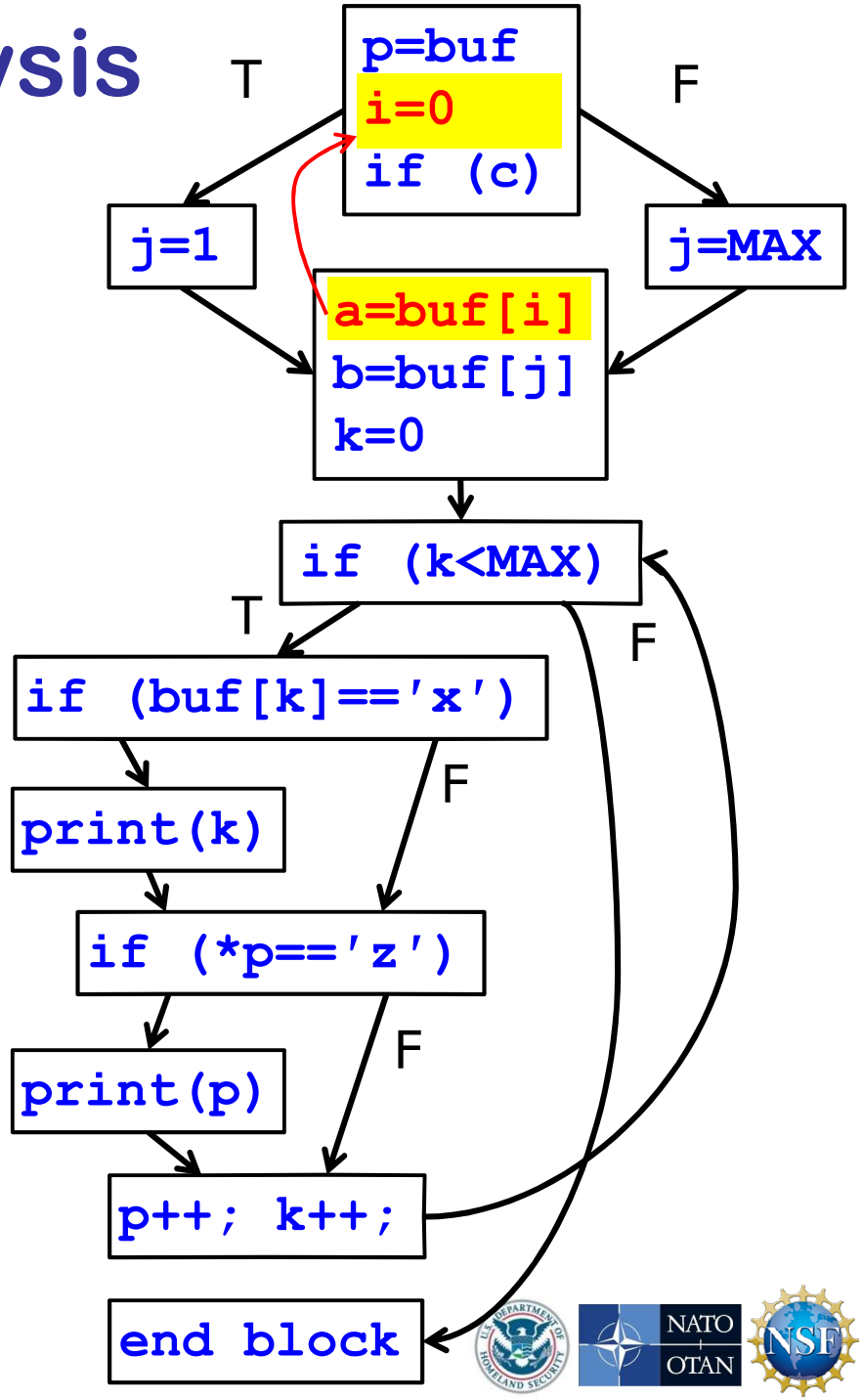
Subgoal:

Do we ever violate the borders of `buf`?

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

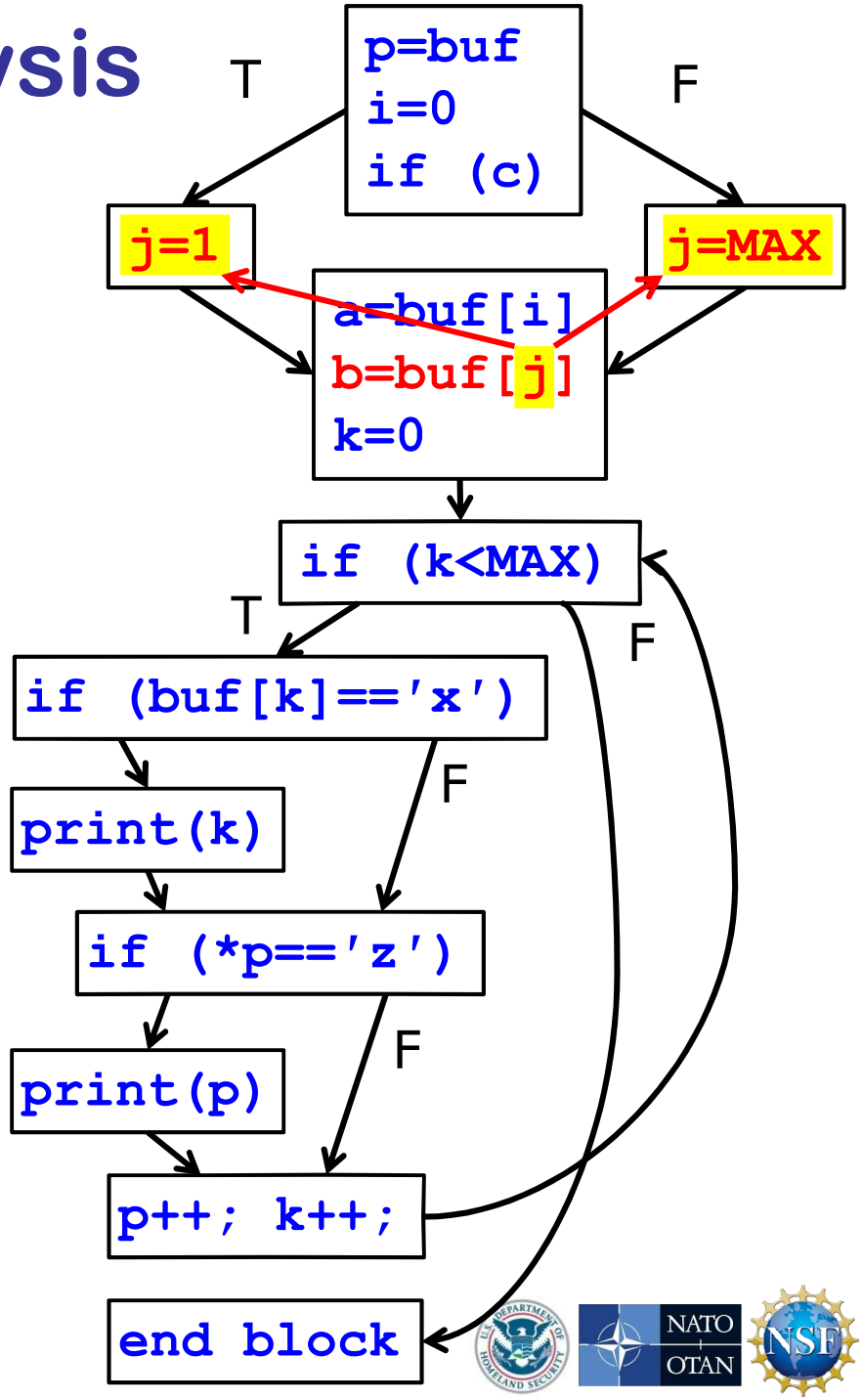
Data Flow Analysis

- Simple dependence



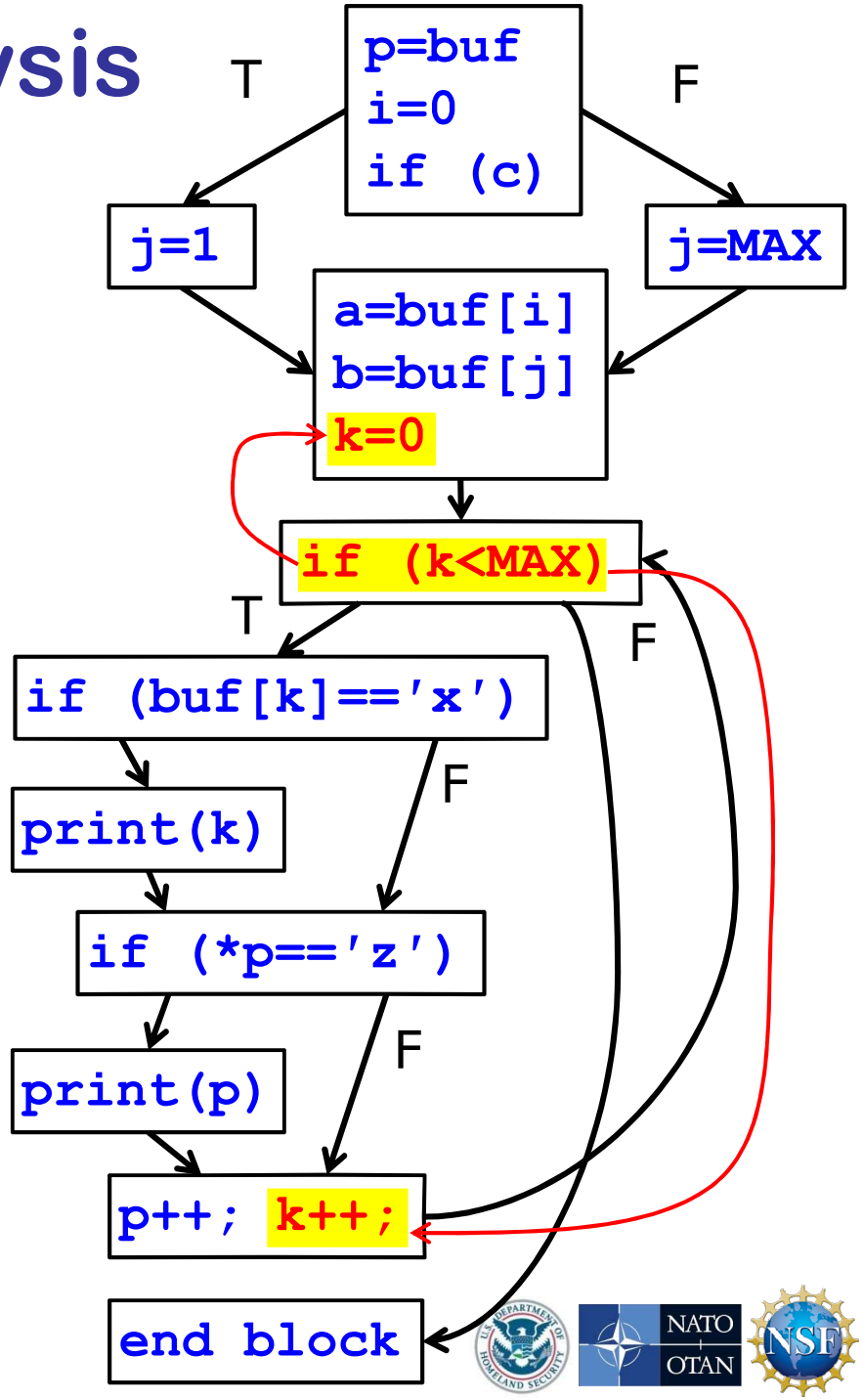
Data Flow Analysis

- Flow insensitive



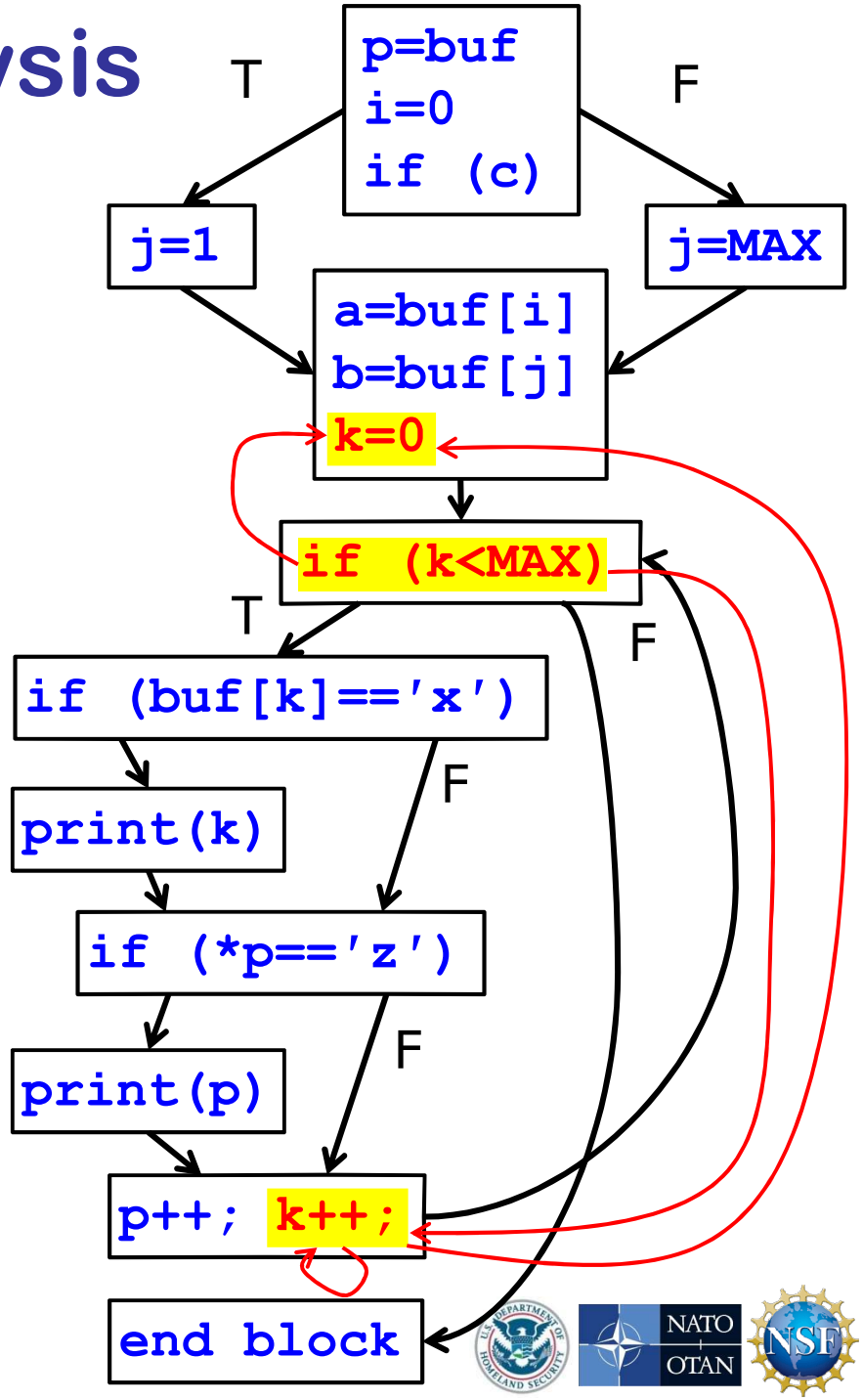
Data Flow Analysis

- Loops



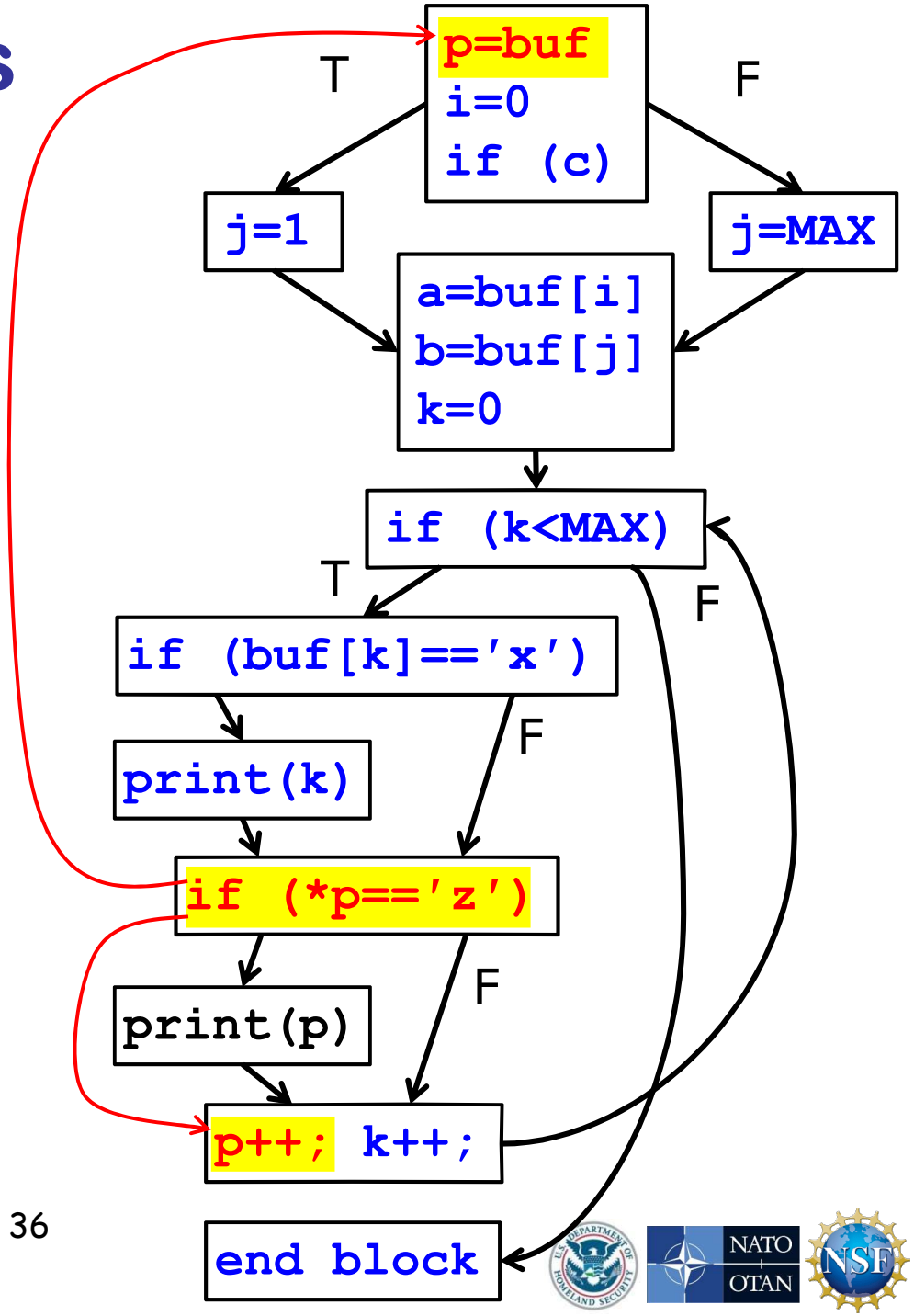
Data Flow Analysis

- Loops



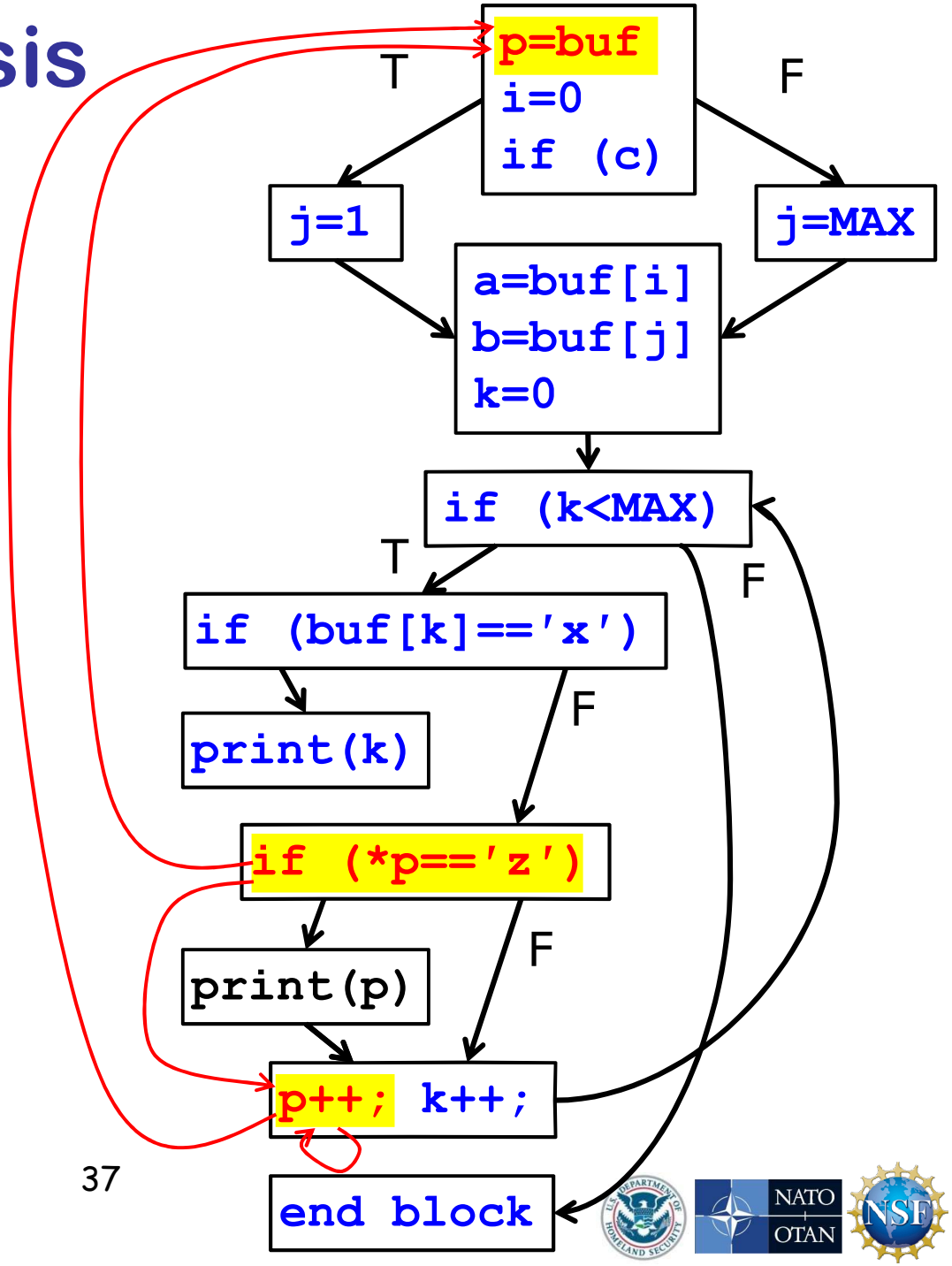
Data Flow Analysis

- Pointers



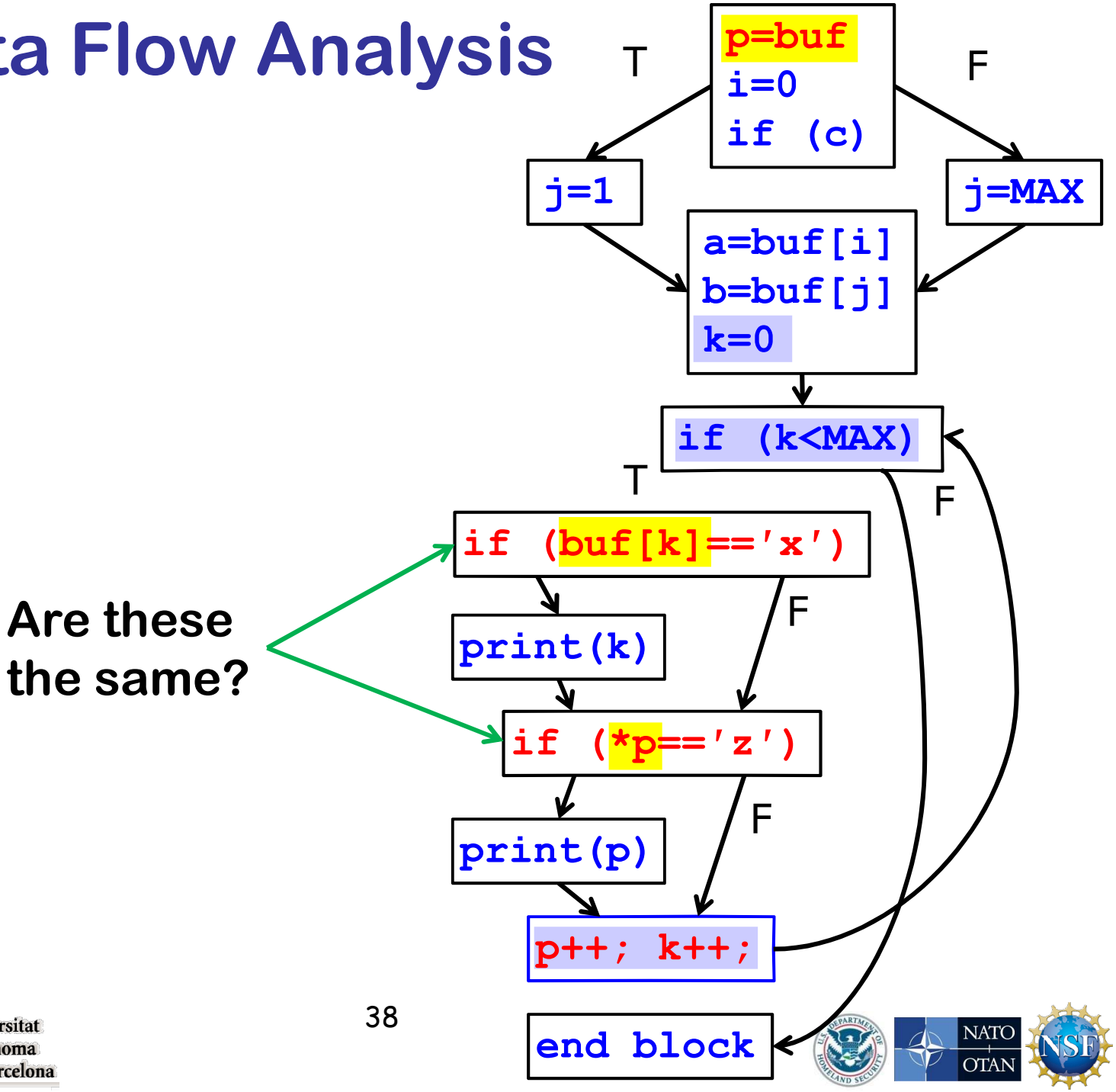
Data Flow Analysis

- Pointers



Data Flow Analysis

- Aliasing

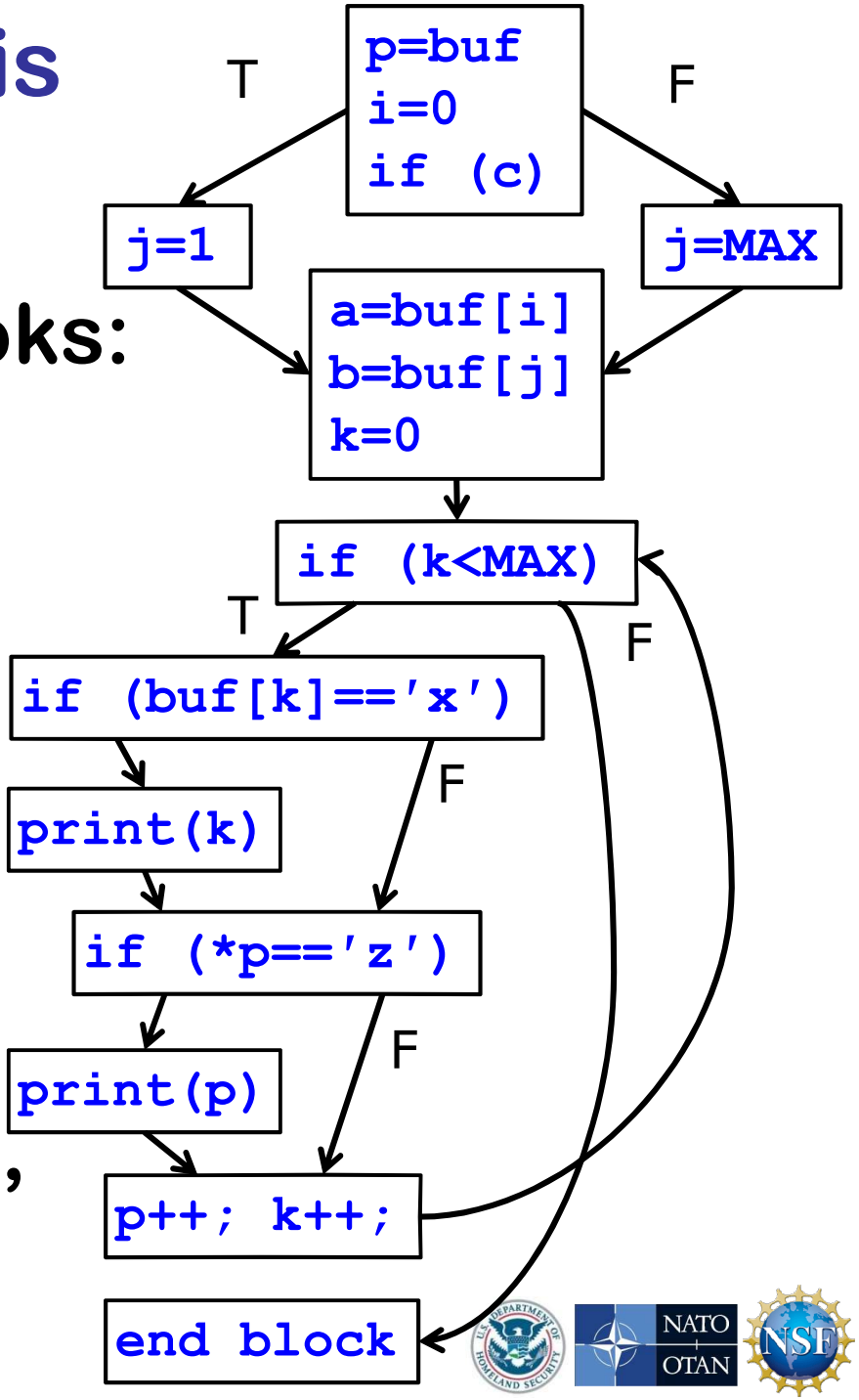


Semantic Analysis

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.

Background on Automated Assessment Tools

Elisa Heymann **Barton P. Miller**

Elisa.Heymann@uab.es bart@cs.wisc.edu

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

http://www.cs.wisc.edu/mist/papers/VAshort.pdf

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

```
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
```

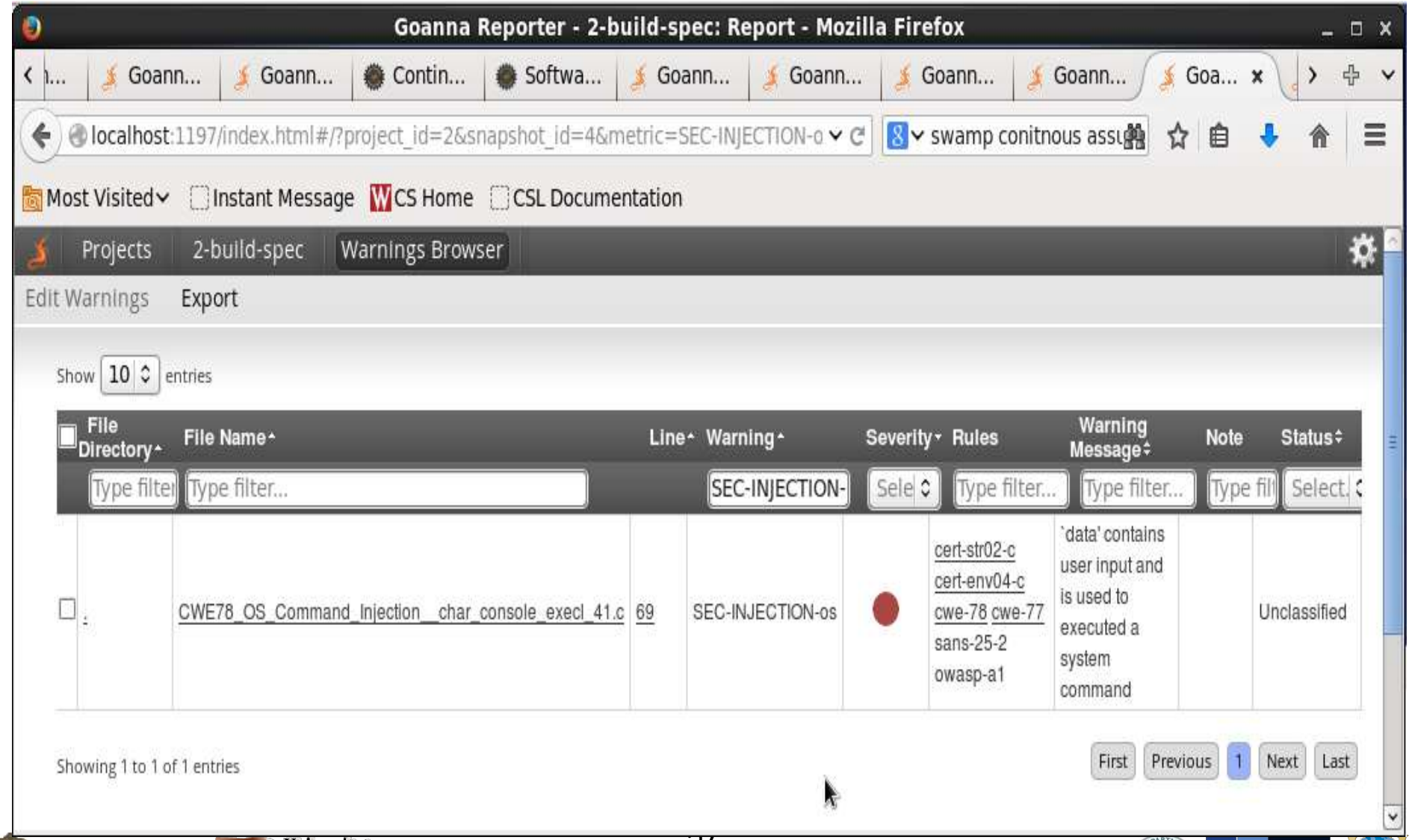
- Read and interact with the analysis results.
  - After goreporter is running, load the provided URL in a web browser.

Goanna. OS Command Injection

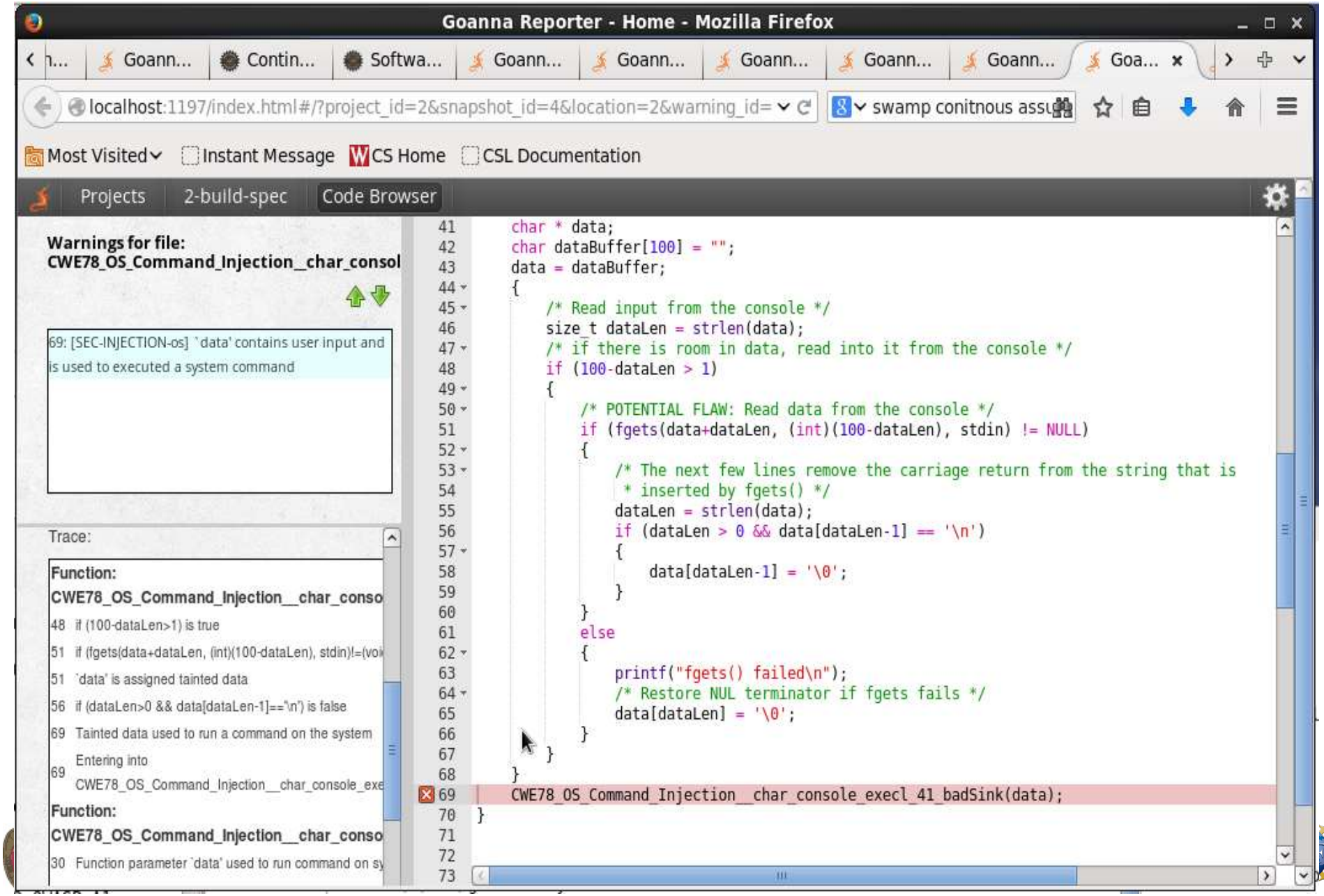
```
$ goanna-trace make
$ goanna-analyze
$ goanna-report
```

- **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:

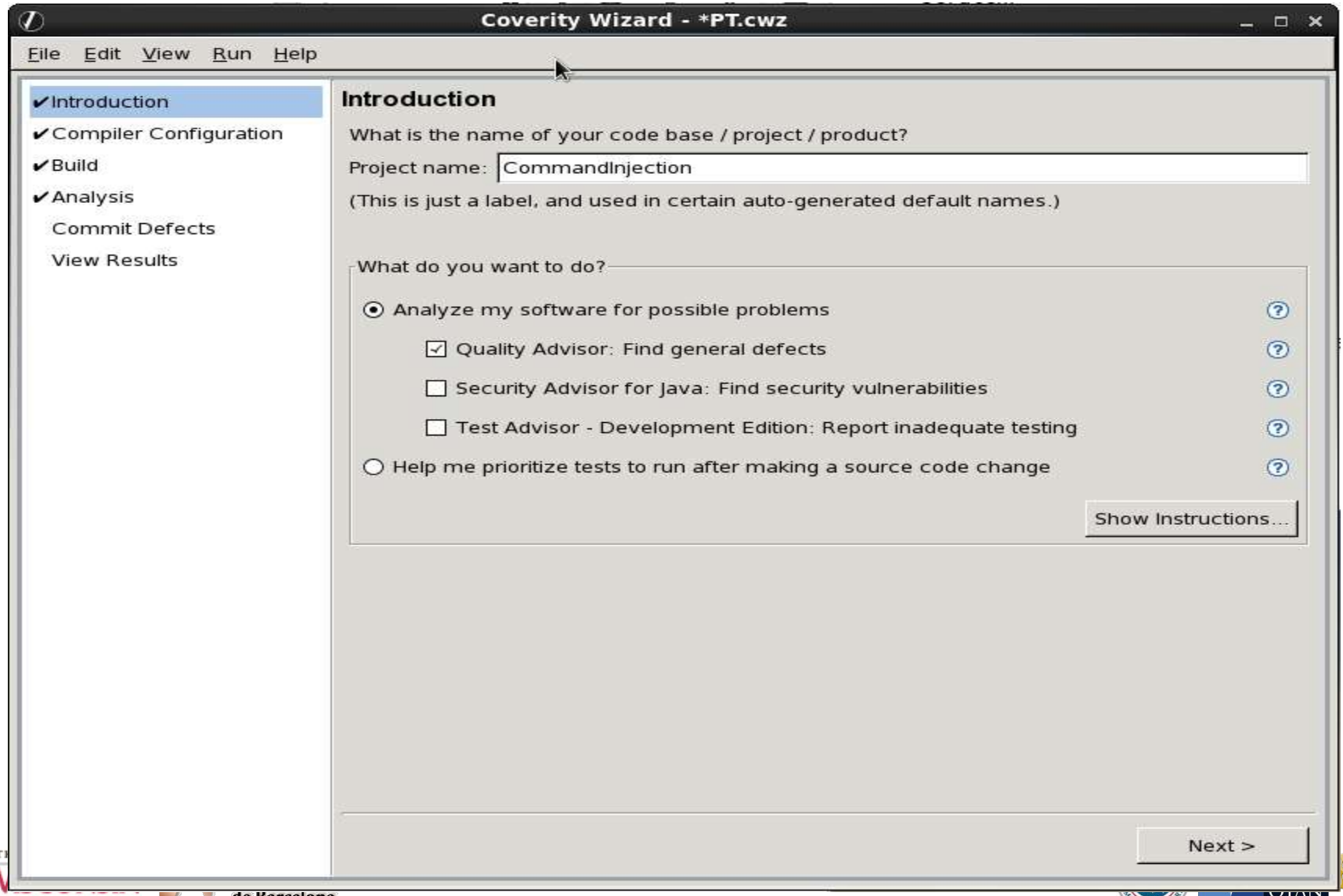
- Generate a configuration for the compiler:
  `cov-configure --gcc`
- Build the intermediate representation of the source code:
  `cov-build --dir <intermediate-dir> make`
- `cov-analyze --dir <intermediate-dir>`
- Check the checkers included by `cov-analize`:
  `cov-analyze --list-checkers`
- Read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

Coverity. OS Command Injection

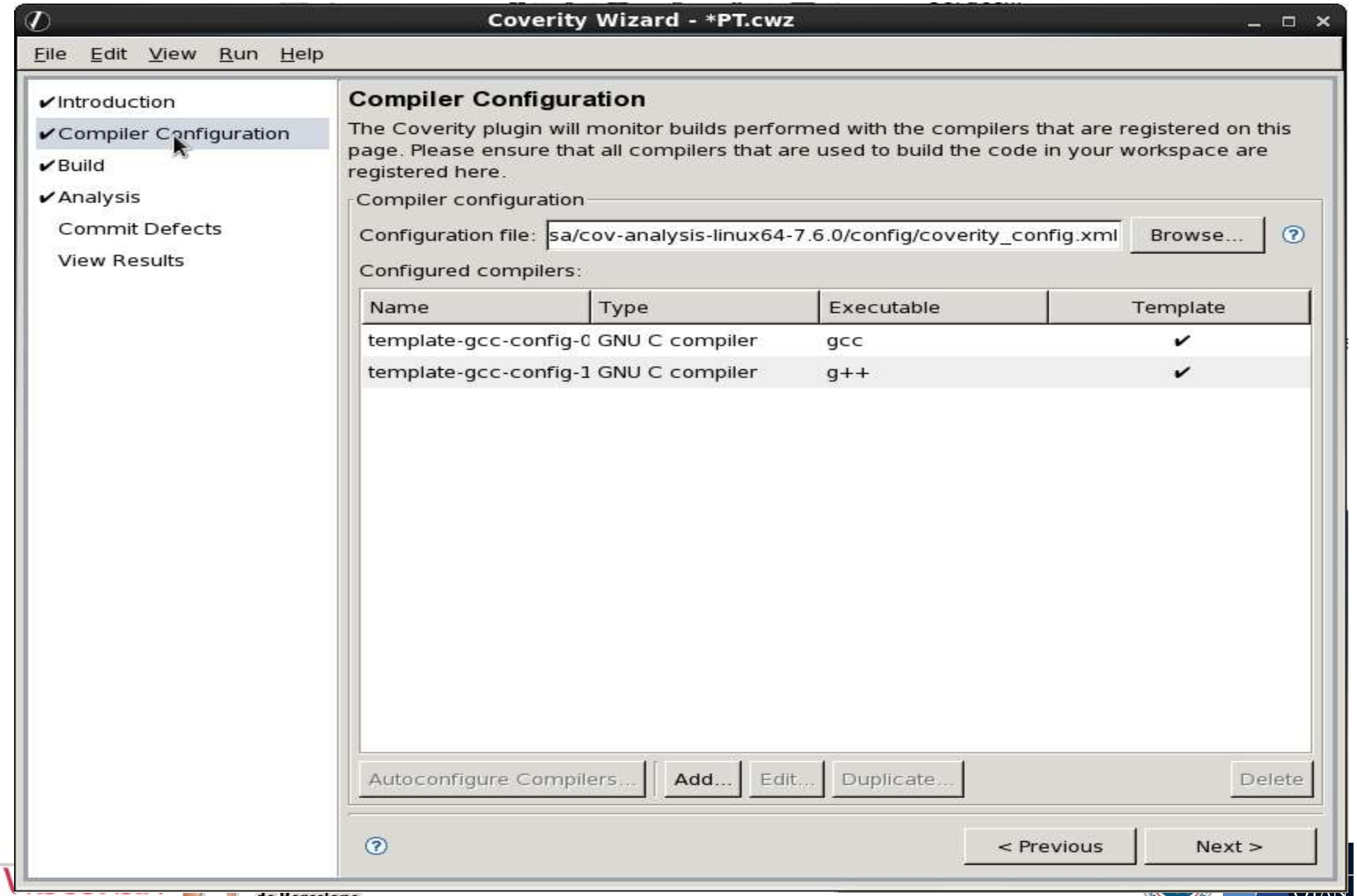
```
$ cov-build --dir cov-comm-injection make
$ cov-analyze --dir cov-comm-injection
              --security
```

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

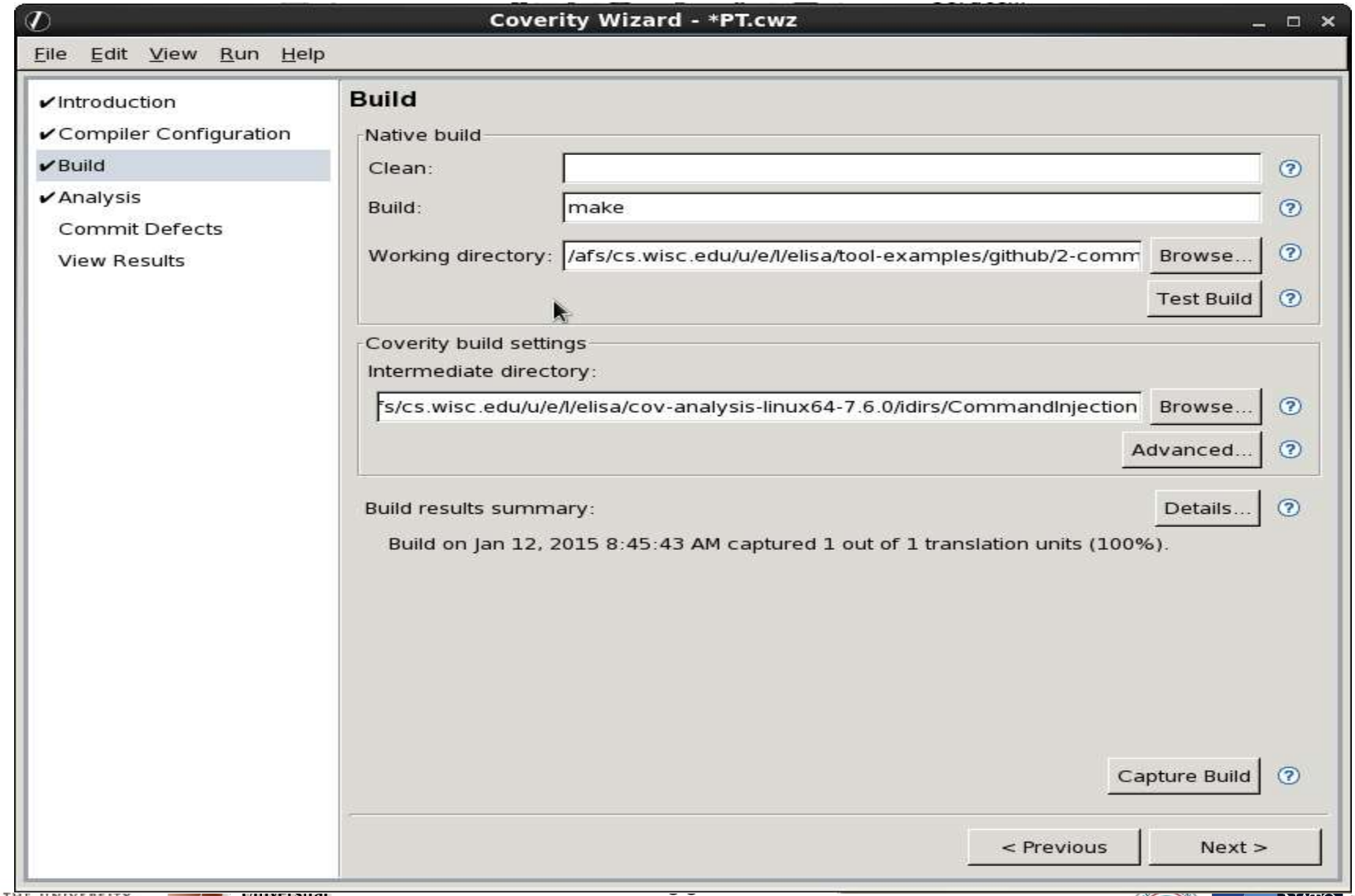
Coverity. OS Command Injection



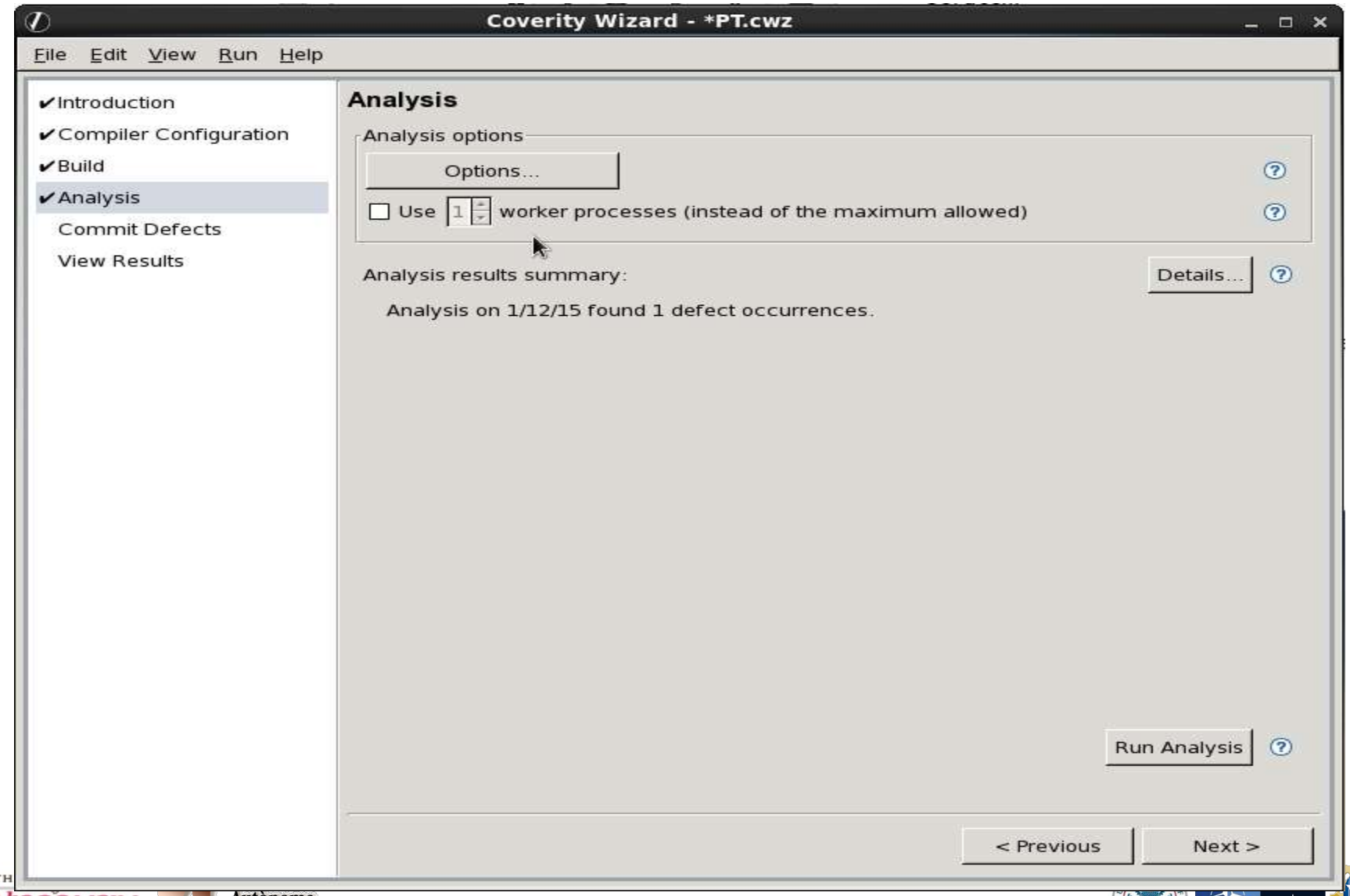
Coverity. OS Command Injection



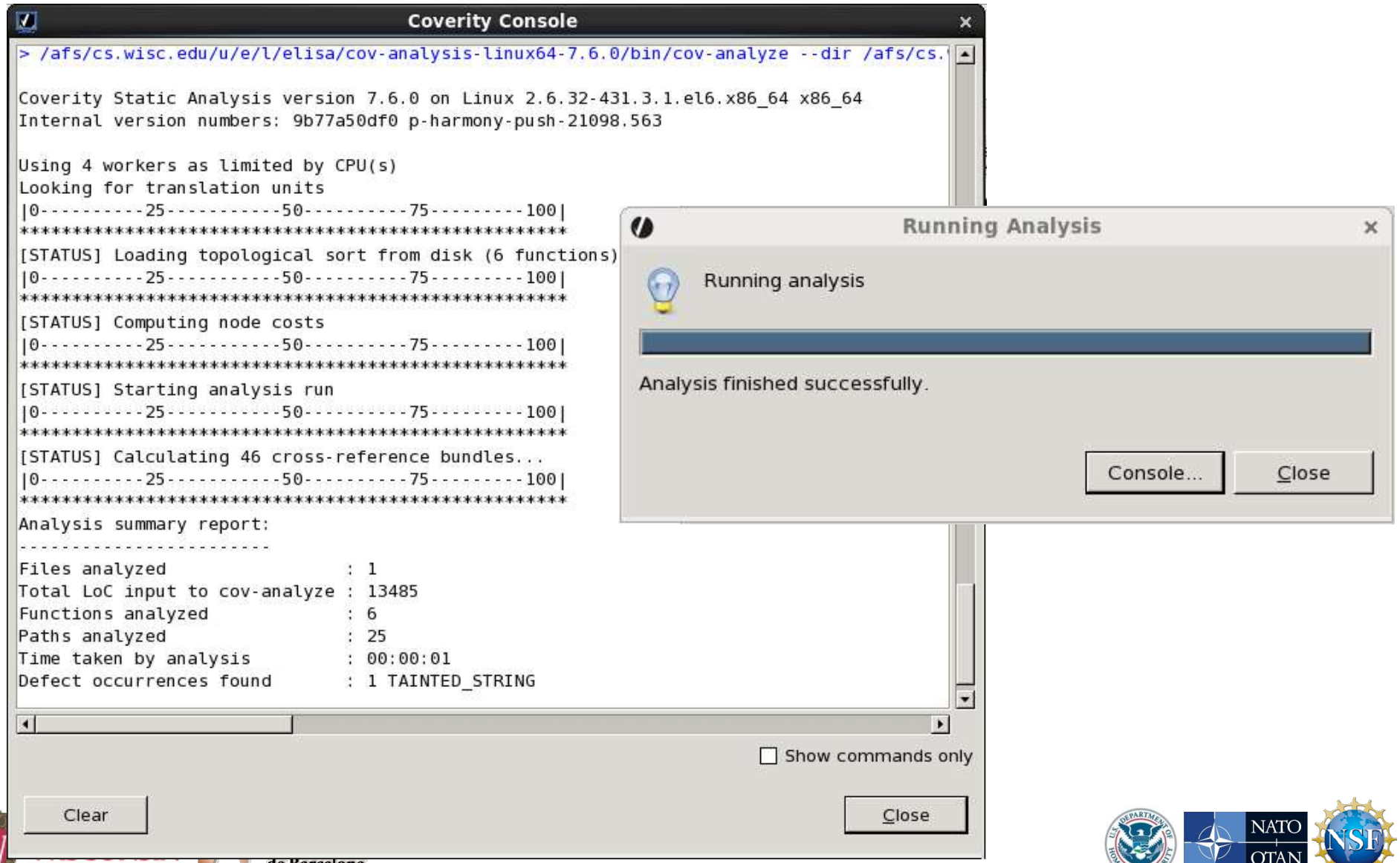
Coverity. OS Command Injection



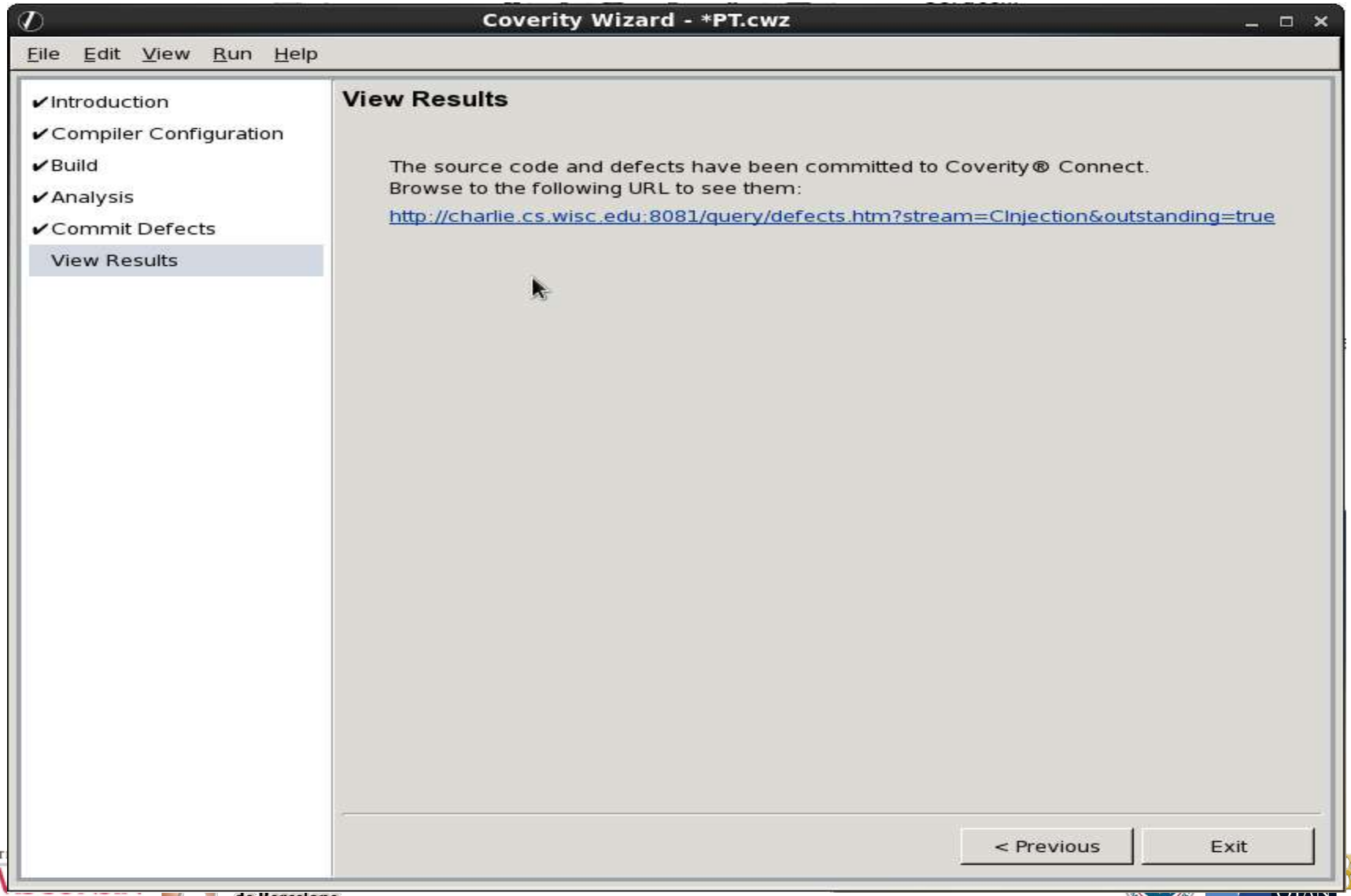
Coverity. OS Command Injection



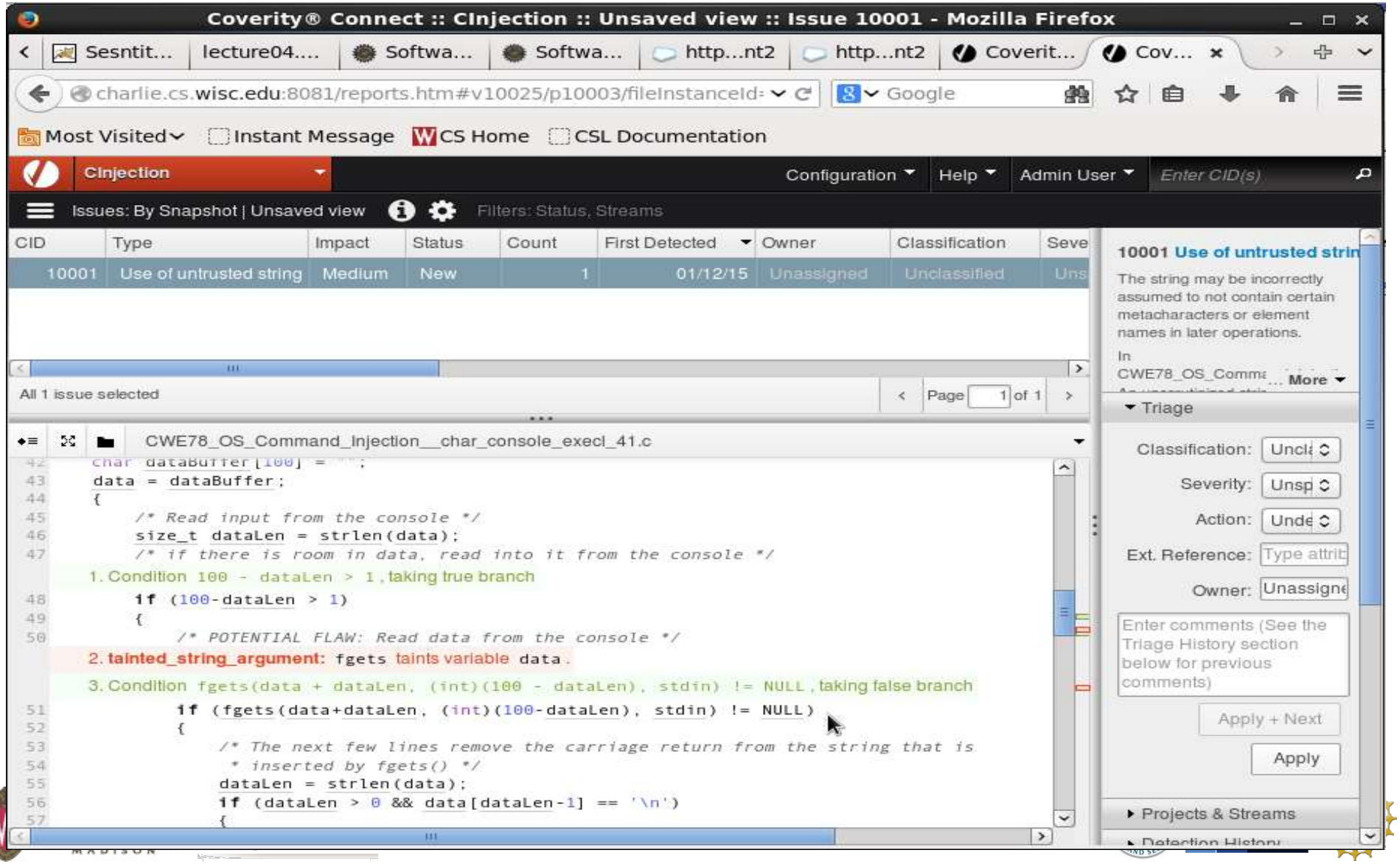
Coverity. OS Command Injection



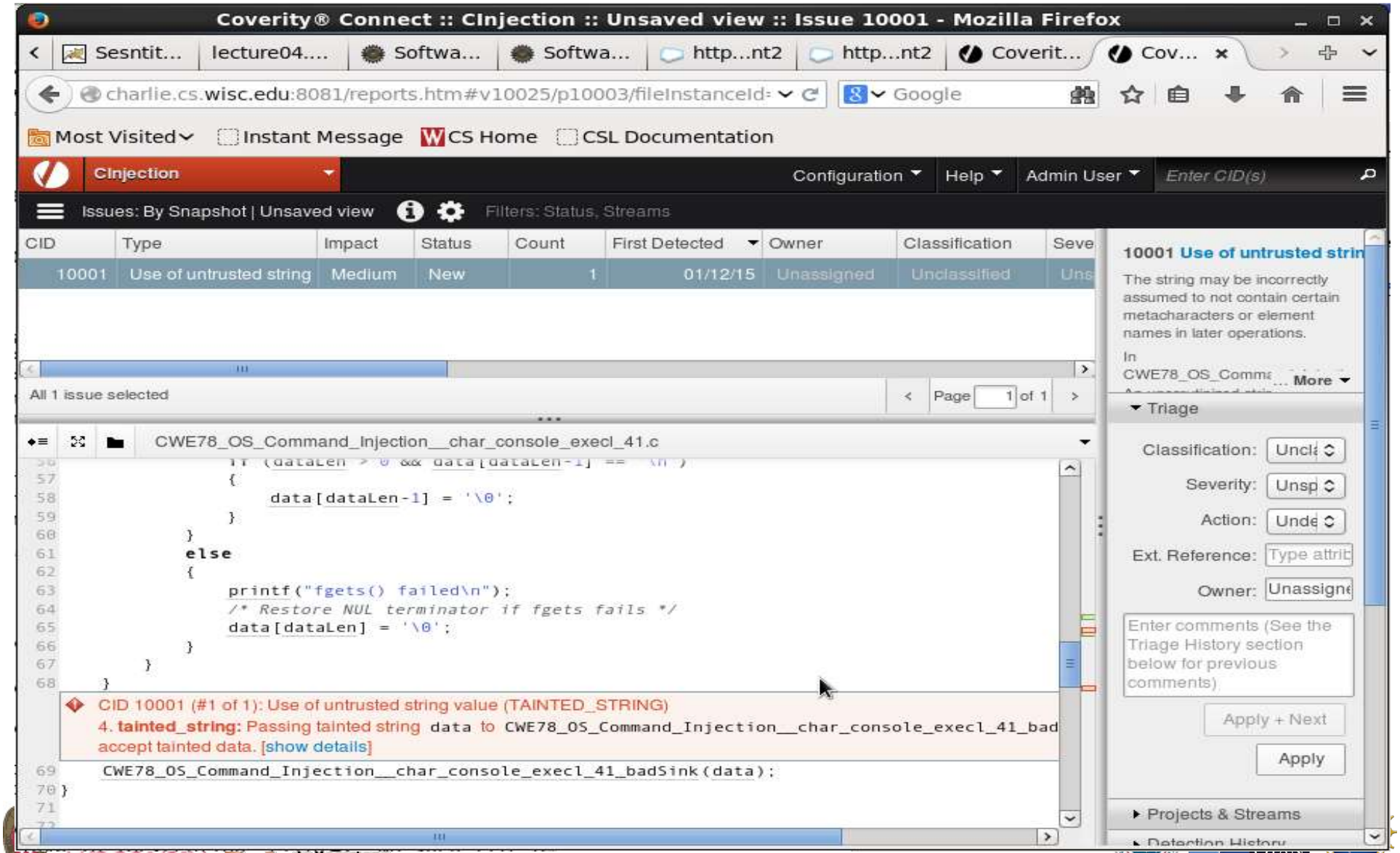
Coverity. OS Command Injection



Coverity. OS Command Injection



Coverity. OS Command Injection



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

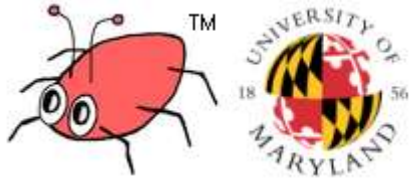
It's a Servlet

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

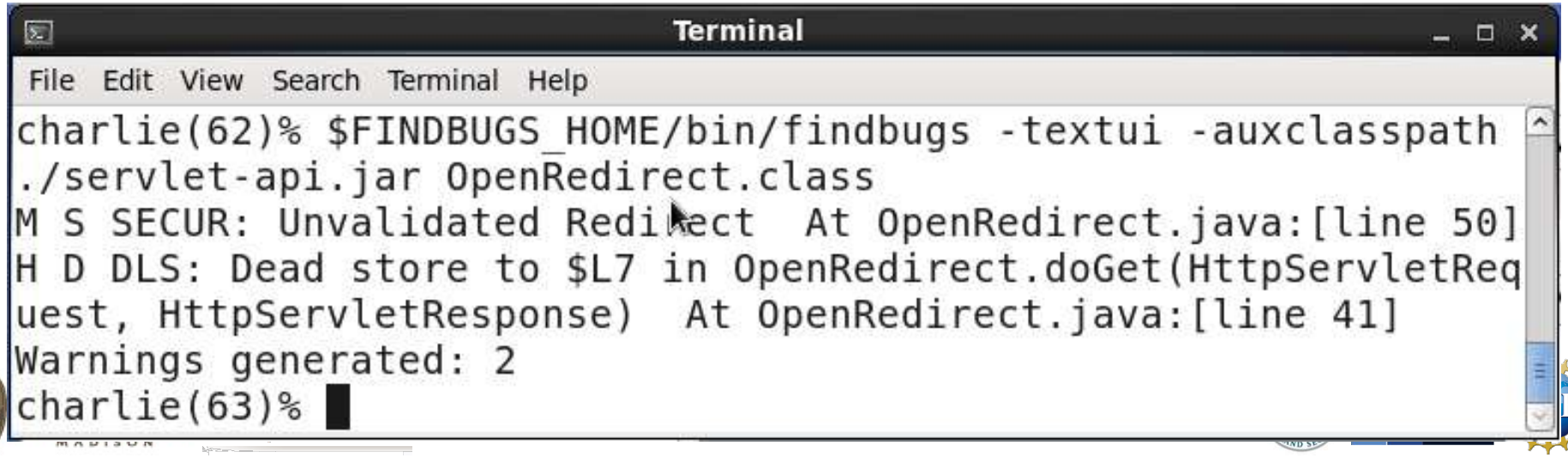
1. Define FINDBUGS_HOME in the environment.
2. Install the Find Security Bugs plugin.
3. Learn the command line instructions and also use the graphical interface.
4. Command line interface:

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

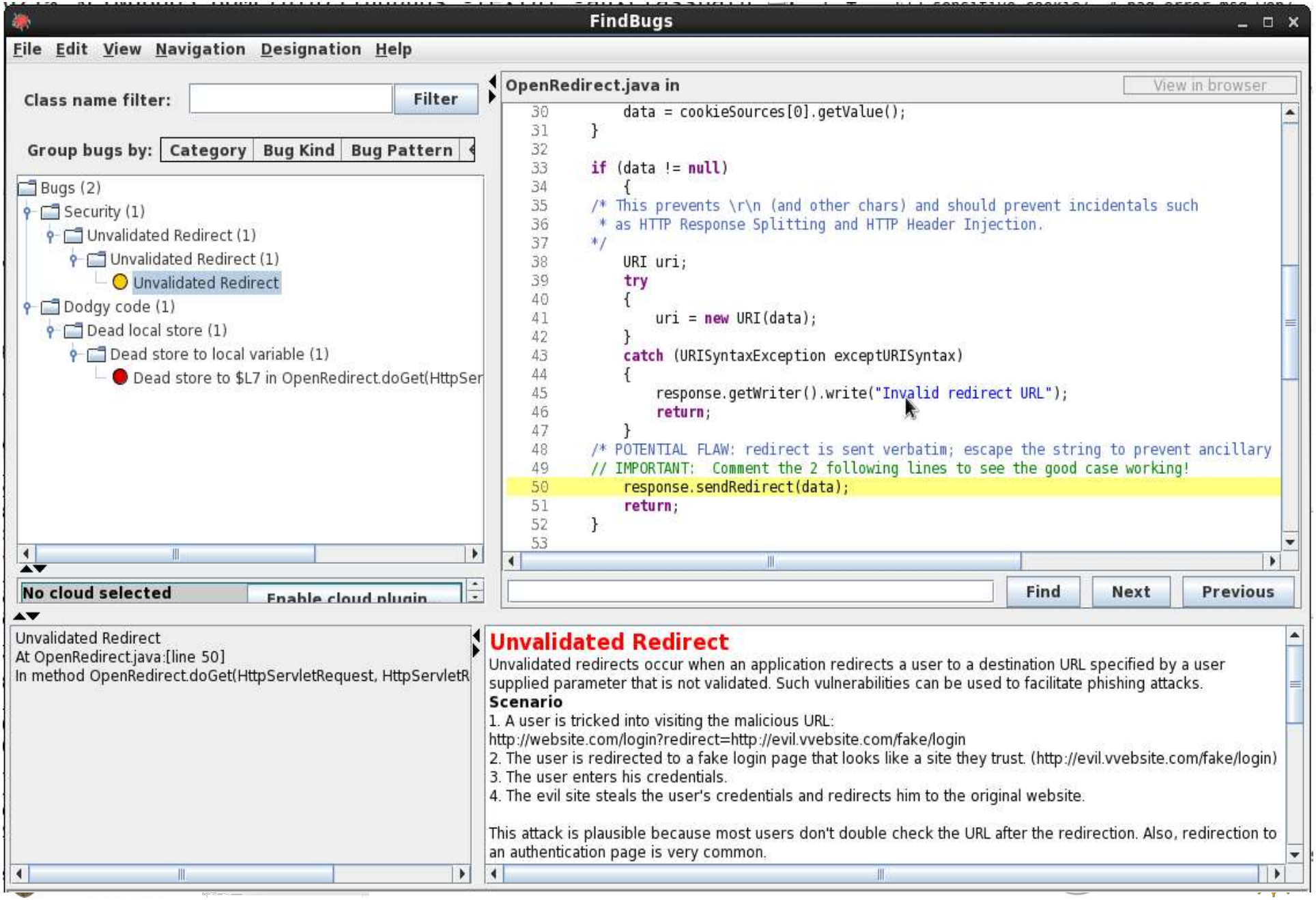
5. Graphic Interface: `java -jar $FINDBUGS_HOME/lib/findbugs.jar -gui`

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

- 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

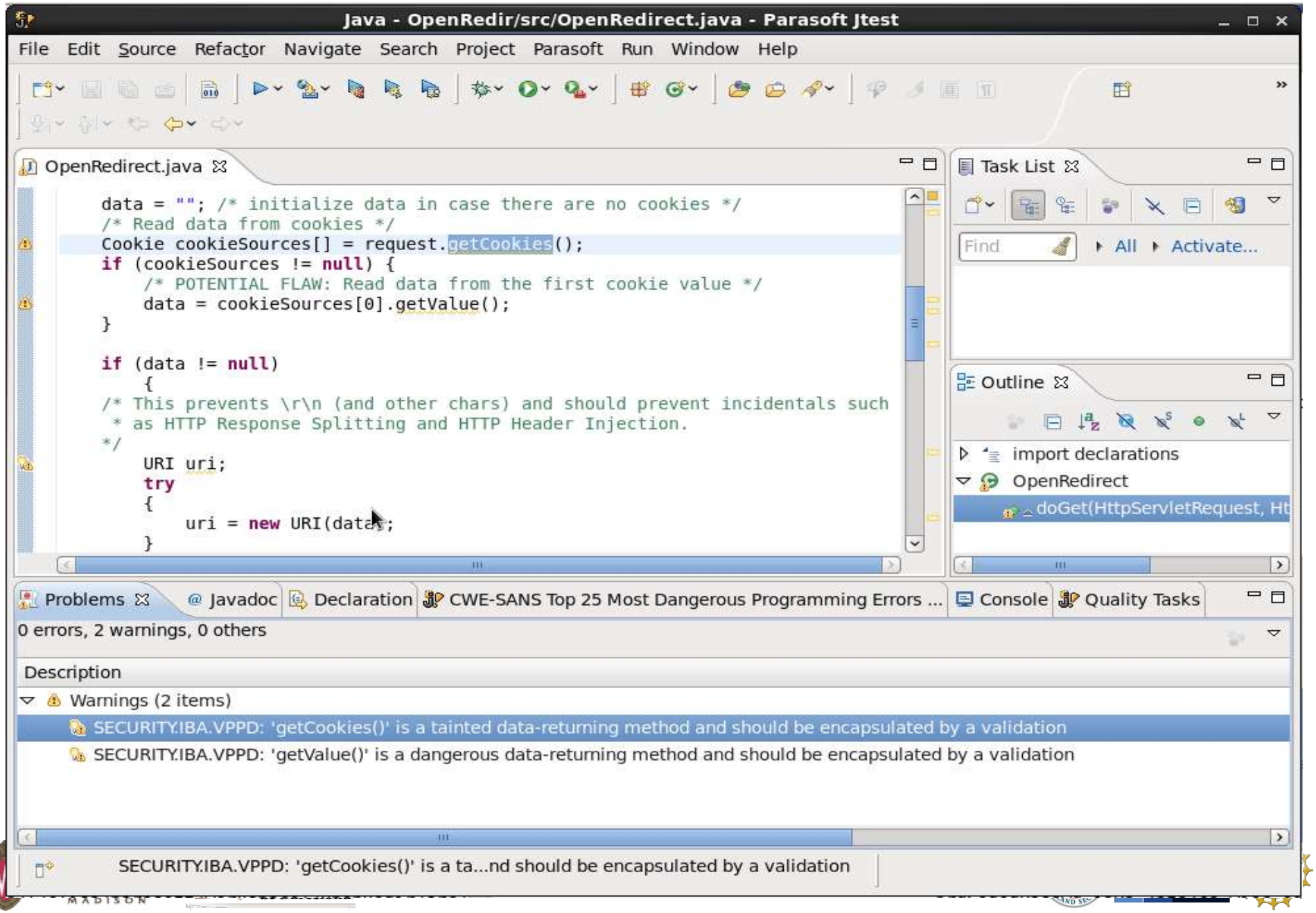
Create the OpenRedir project.

Include servlet-api.jar in the OpenRedir project.

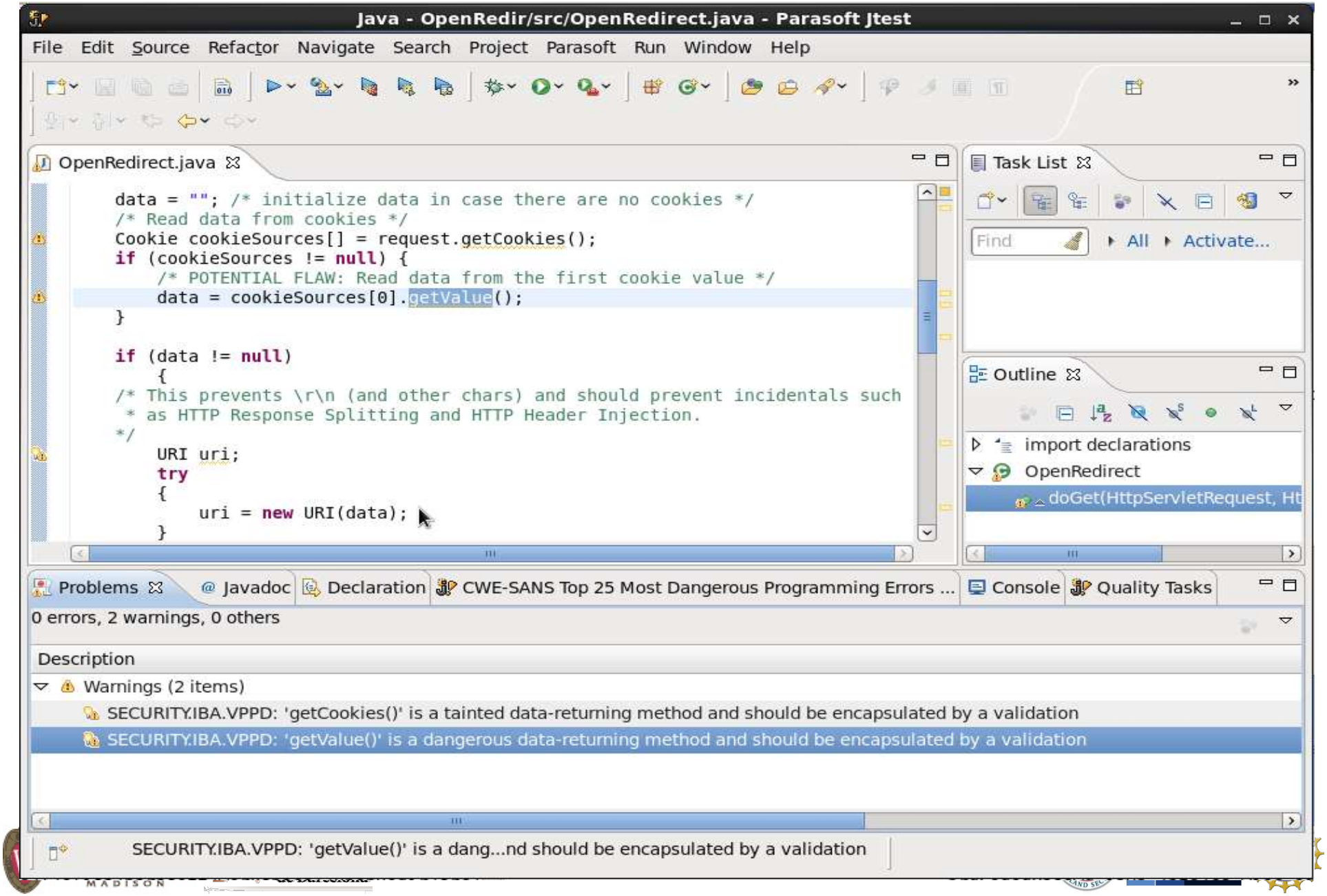
```
cp OpenRedirect.java
~elisa/parasoft/workspace1/OpenRedir/src
```

- **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.**

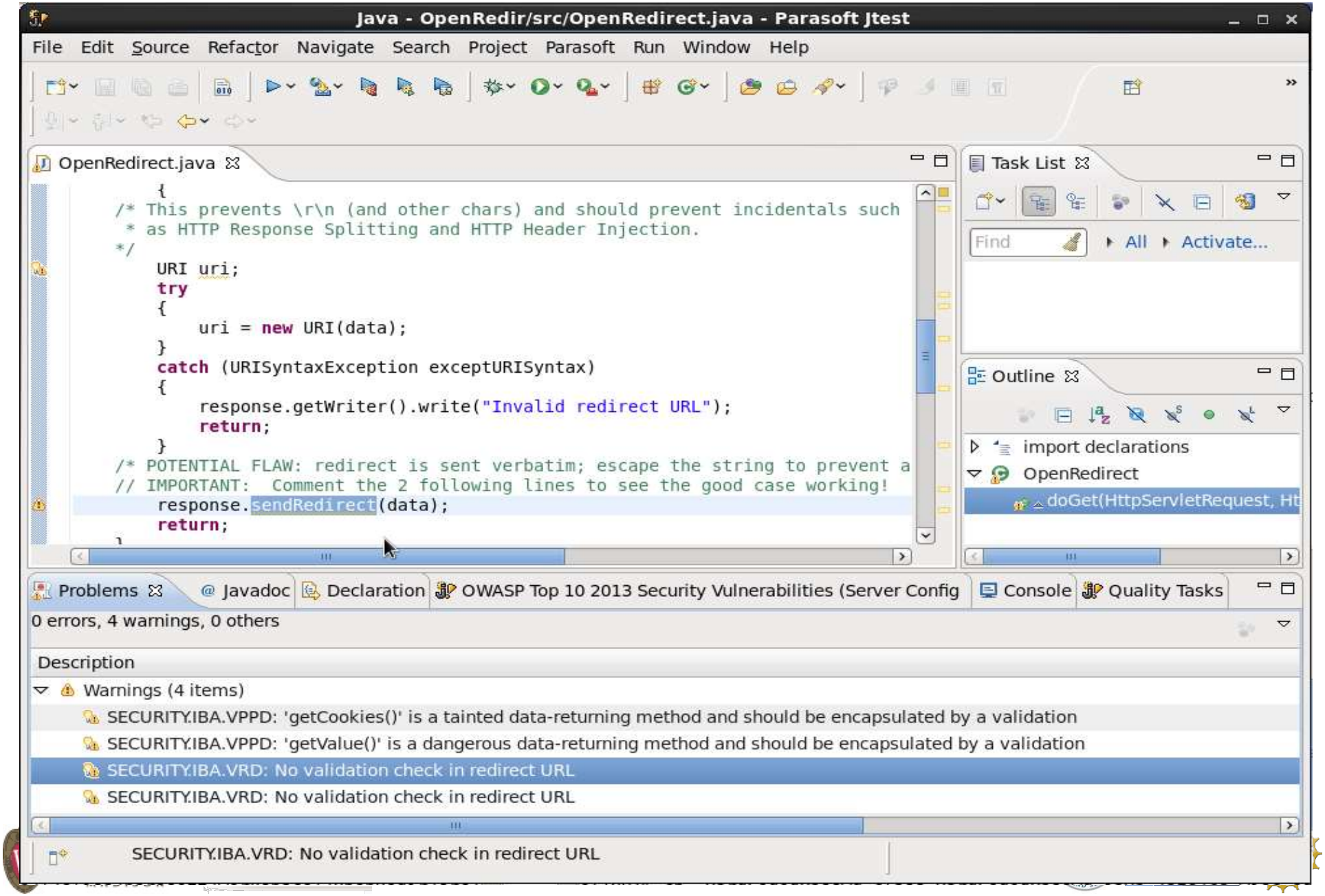
Jtest. Open Redirect



Jtest. Open Redirect



Jtest. Open Redirect



Roadmap

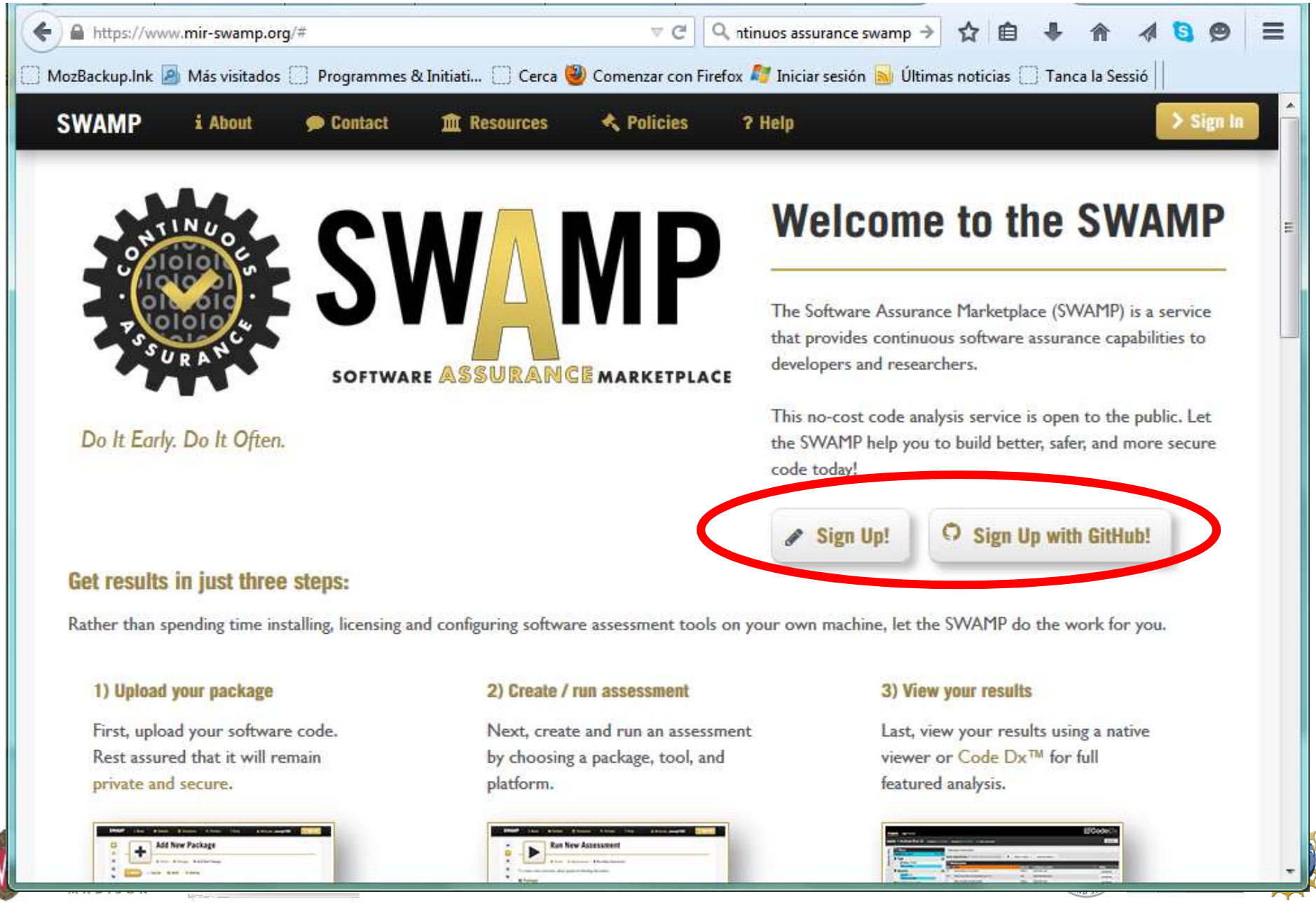
- What is the SWAMP?
- Using the SWAMP
  - Register
  - Create a project
  - Upload your software package
  - Run your assessment
  - View the results
    - Java
    - C/C++

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

Getting Started with the SWAMP

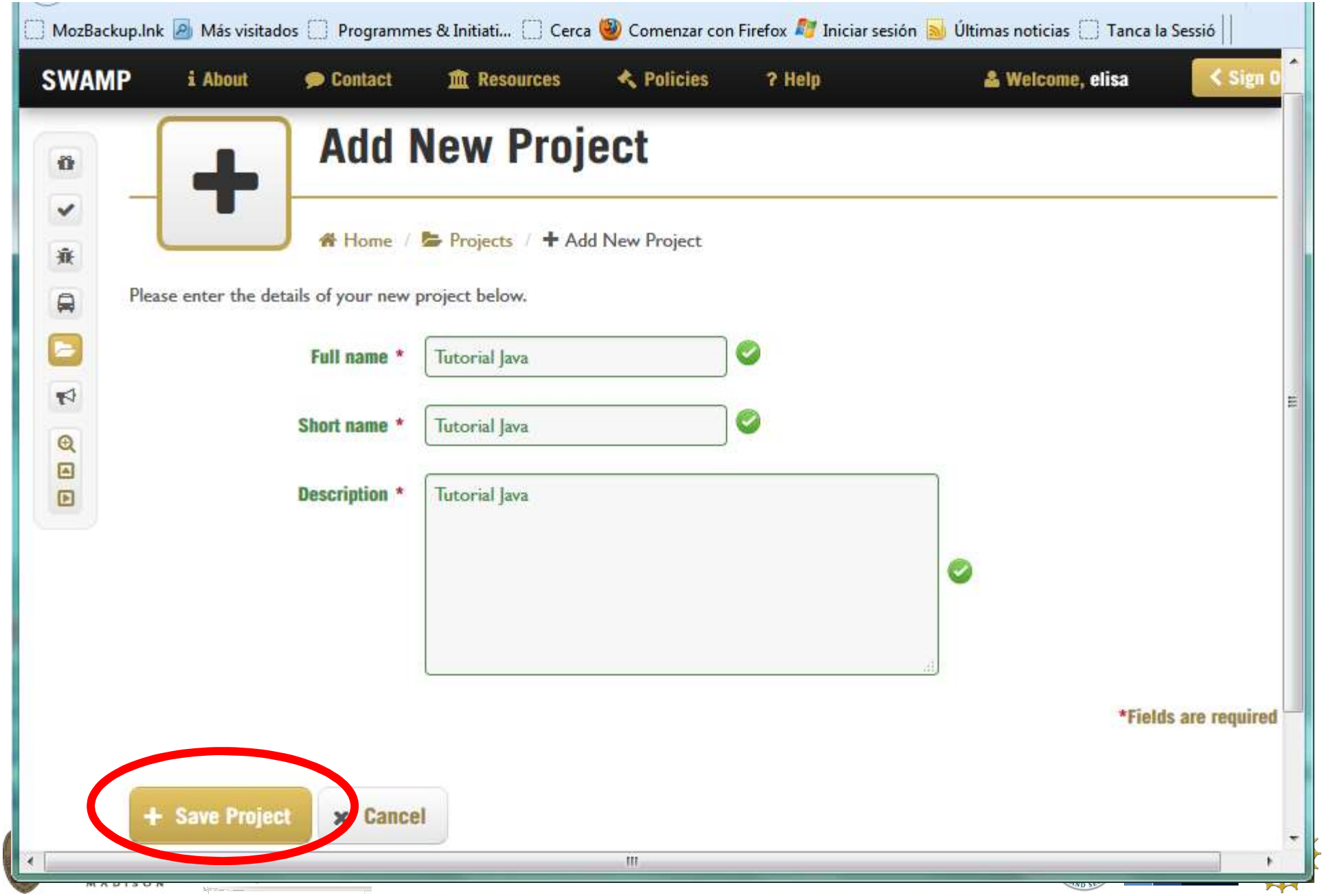
- **Software Assurance Market Place.**
- **Objective: Automate and simplify the use of (multiple) tools.**
- **A national, no-cost resource for software assurance (SwA) technologies used across research institutions, non-governmental organizations, and civilian agencies and their communities as both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

Register to use the SWAMP

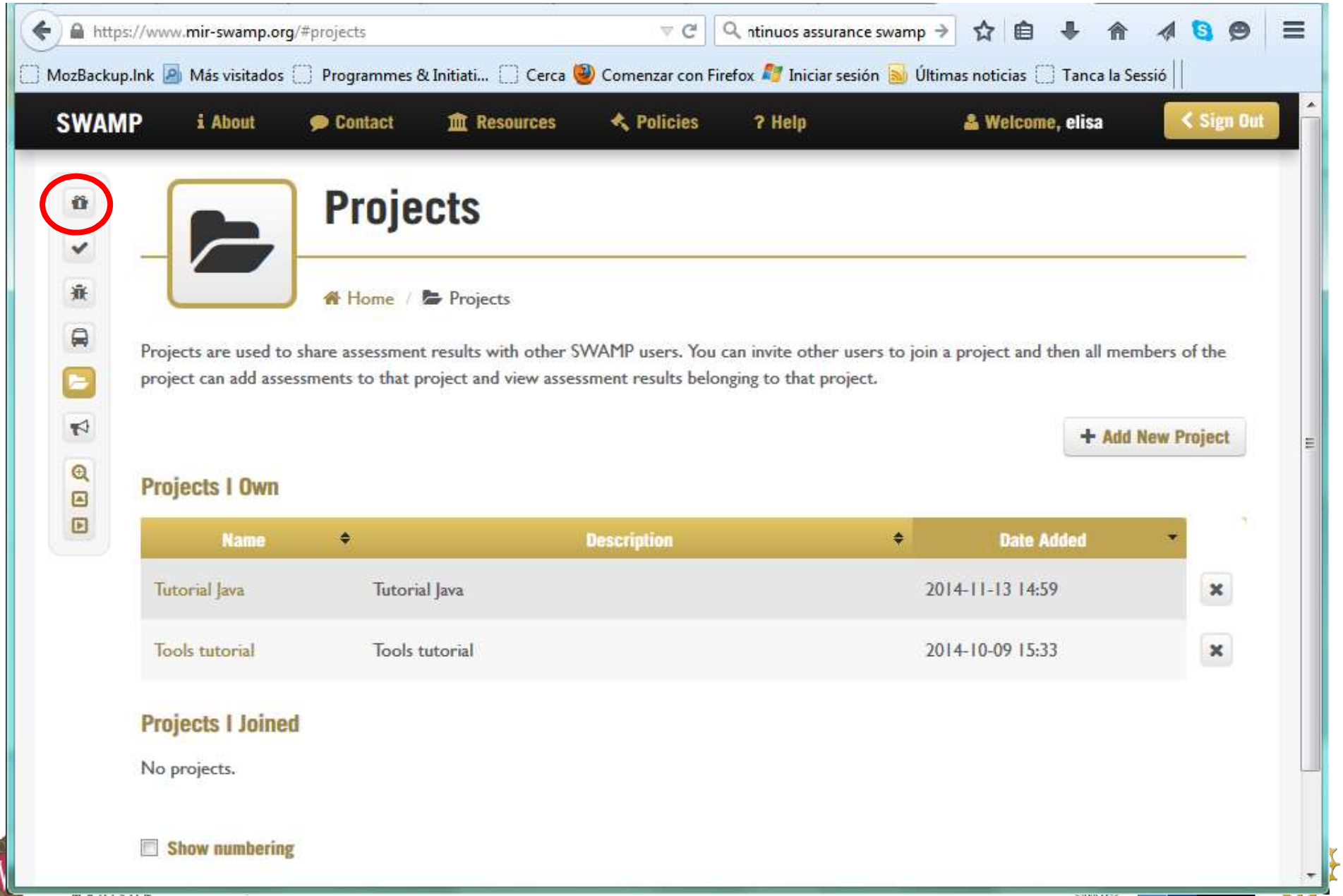


What can I do in the SWAMP?

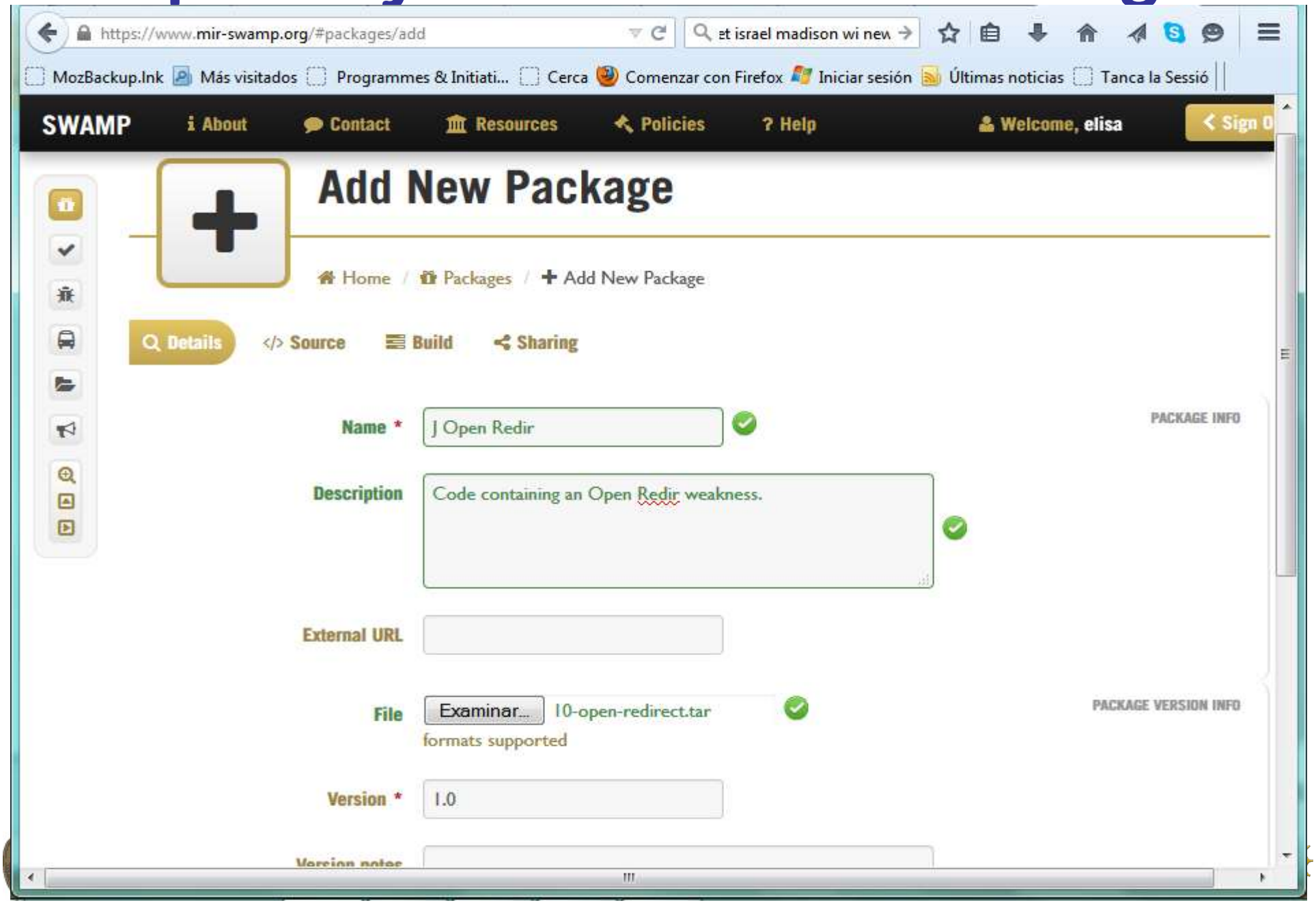
Create a Project



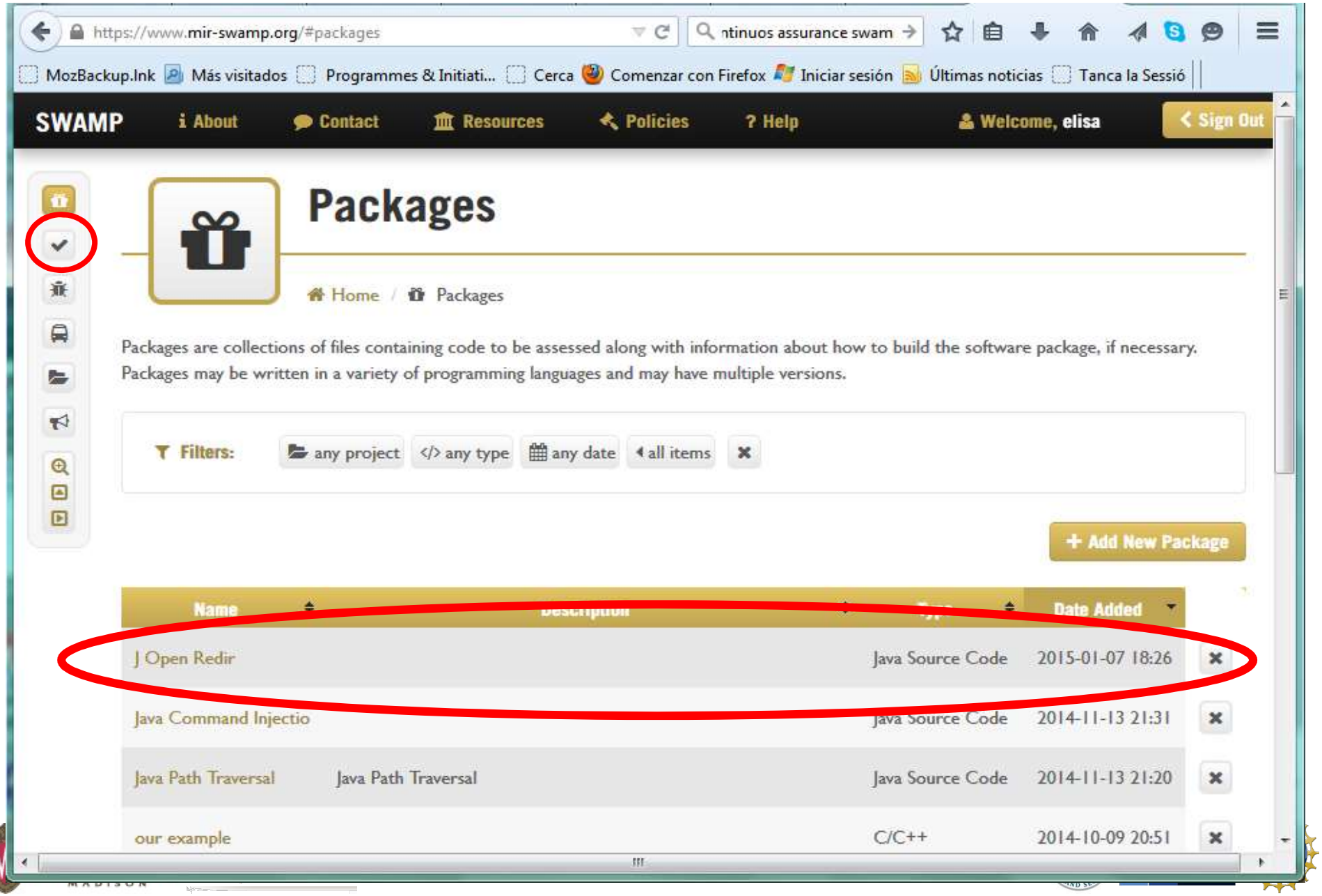
My Projects


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

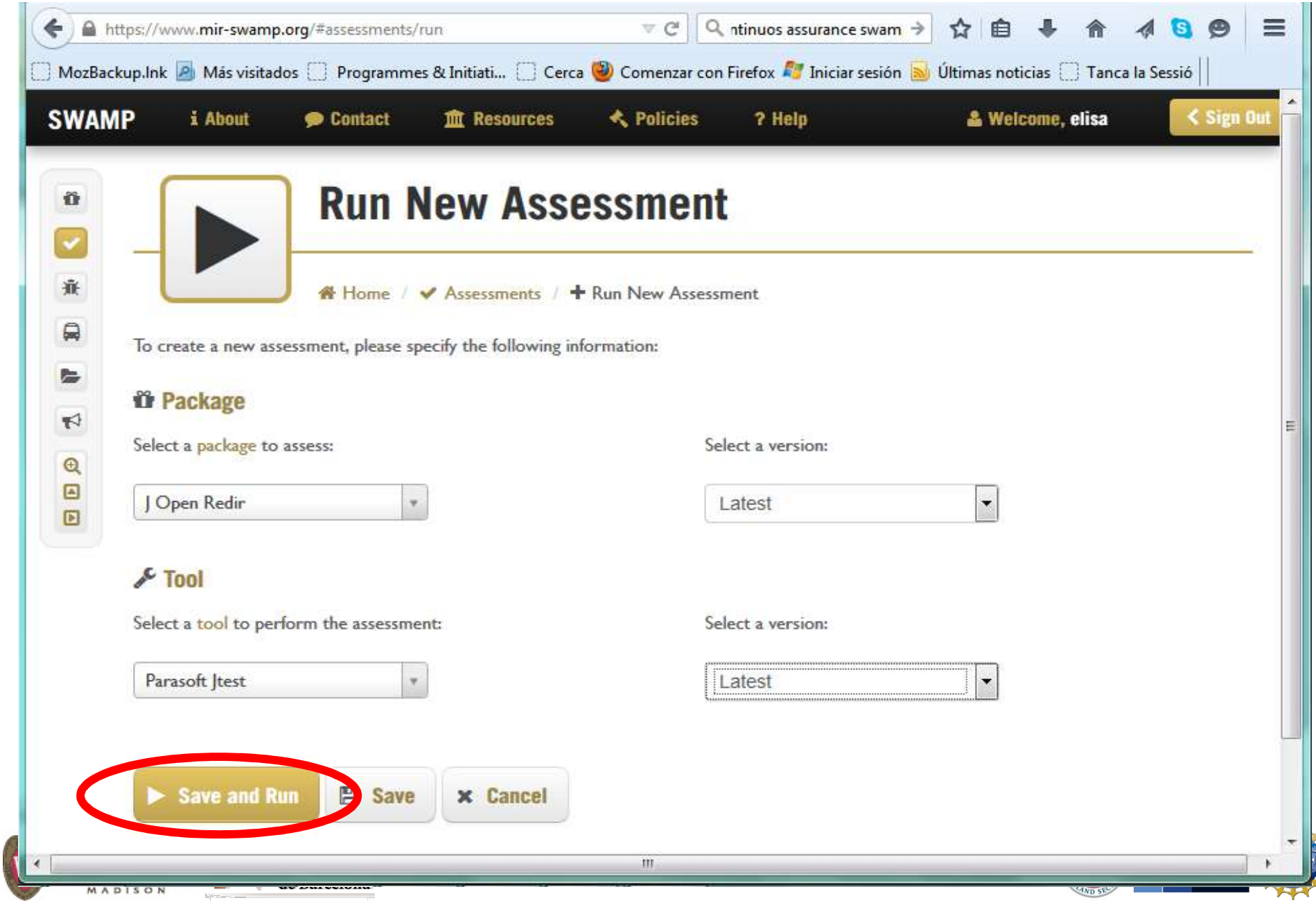
Upload your Software Package



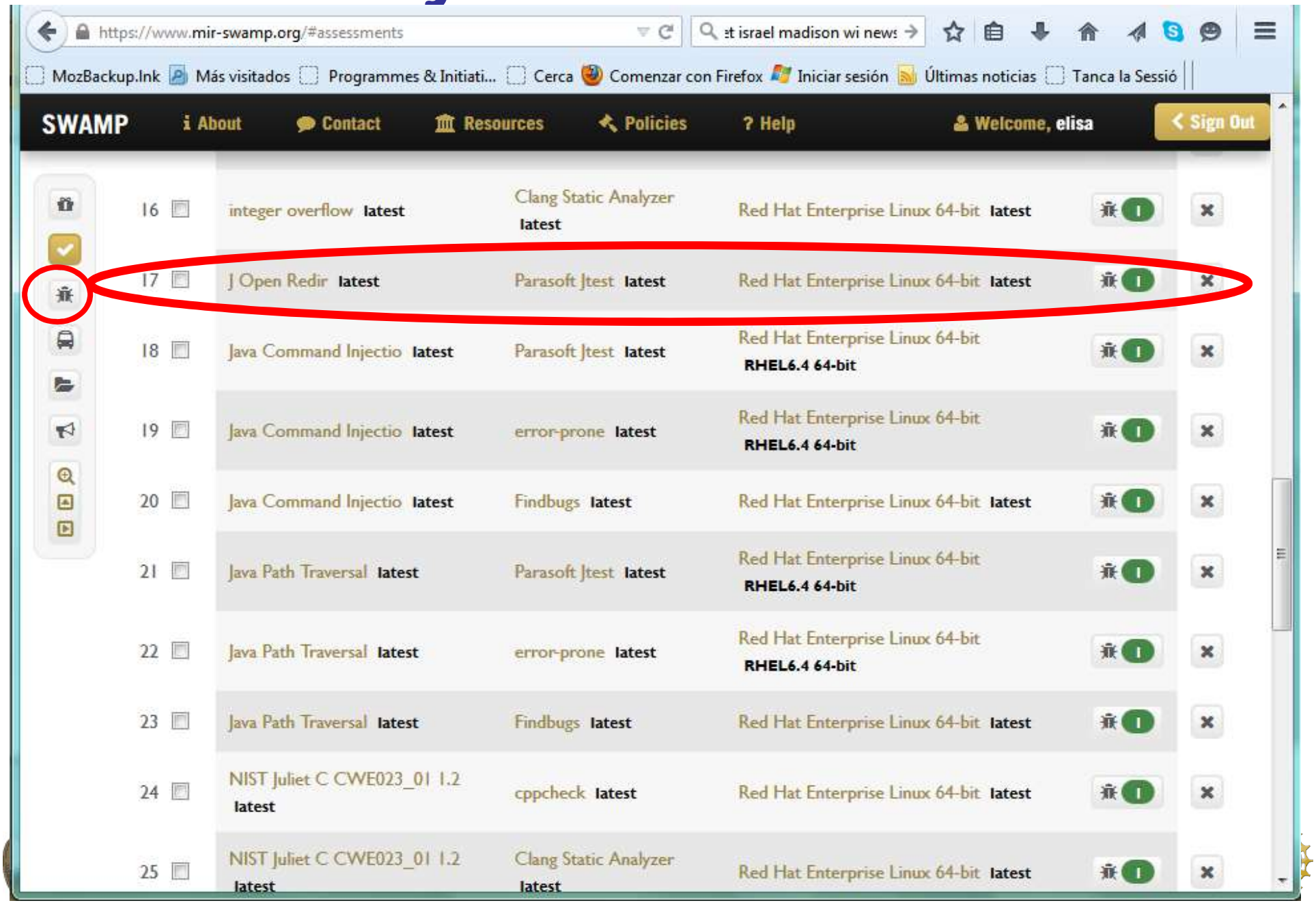
My software Packages



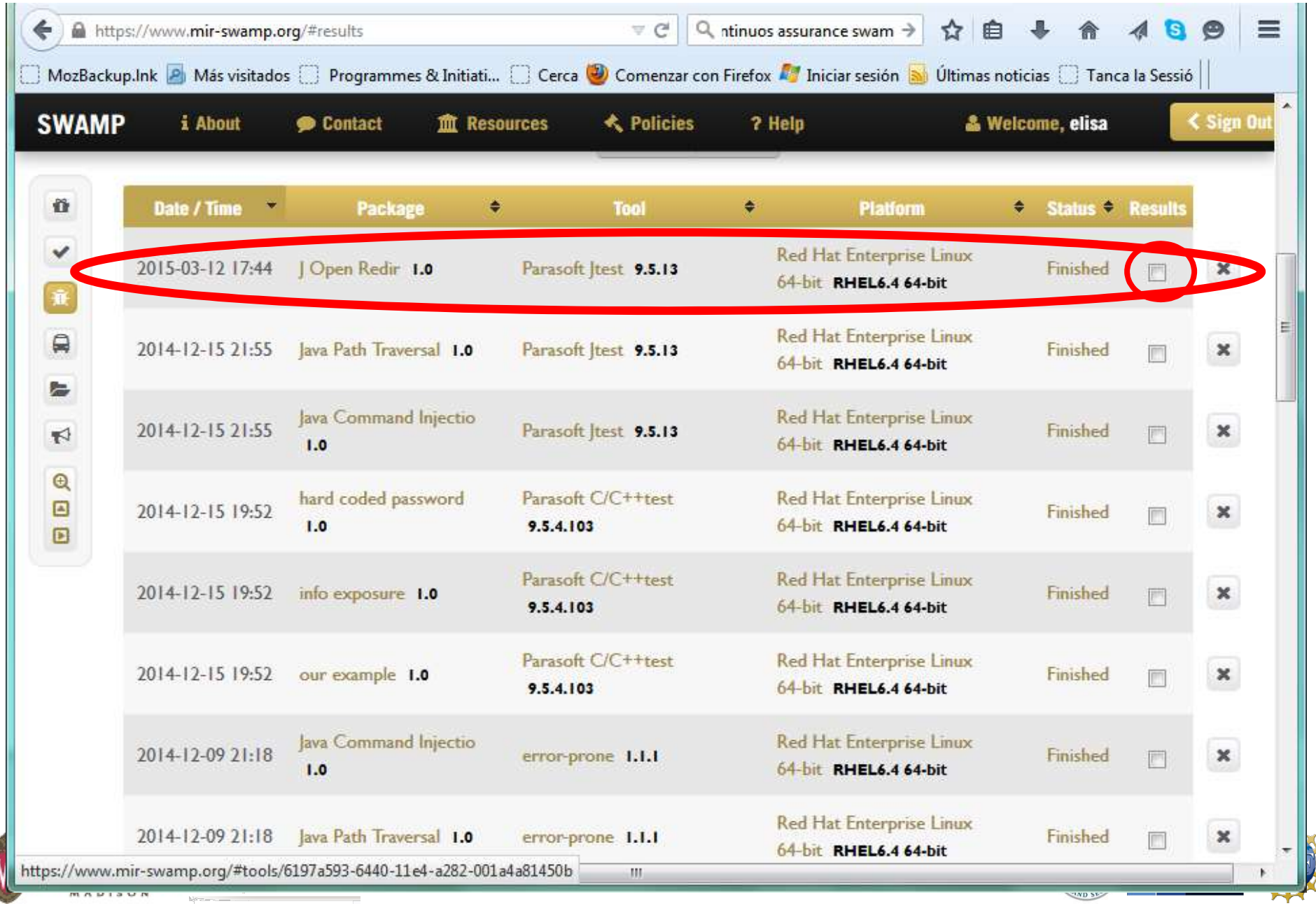
Run your Assessments



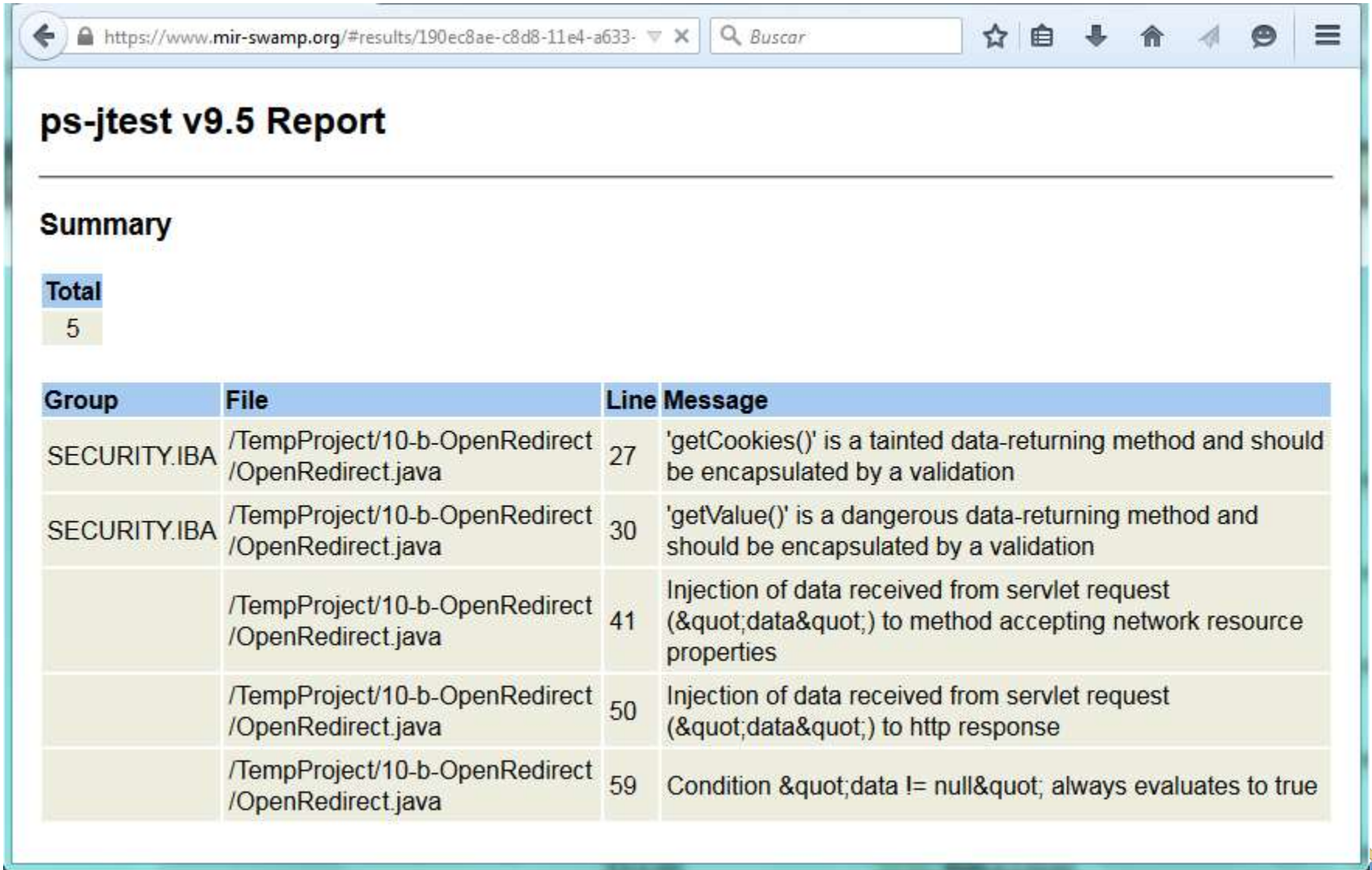
My Assessments



View your Results



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



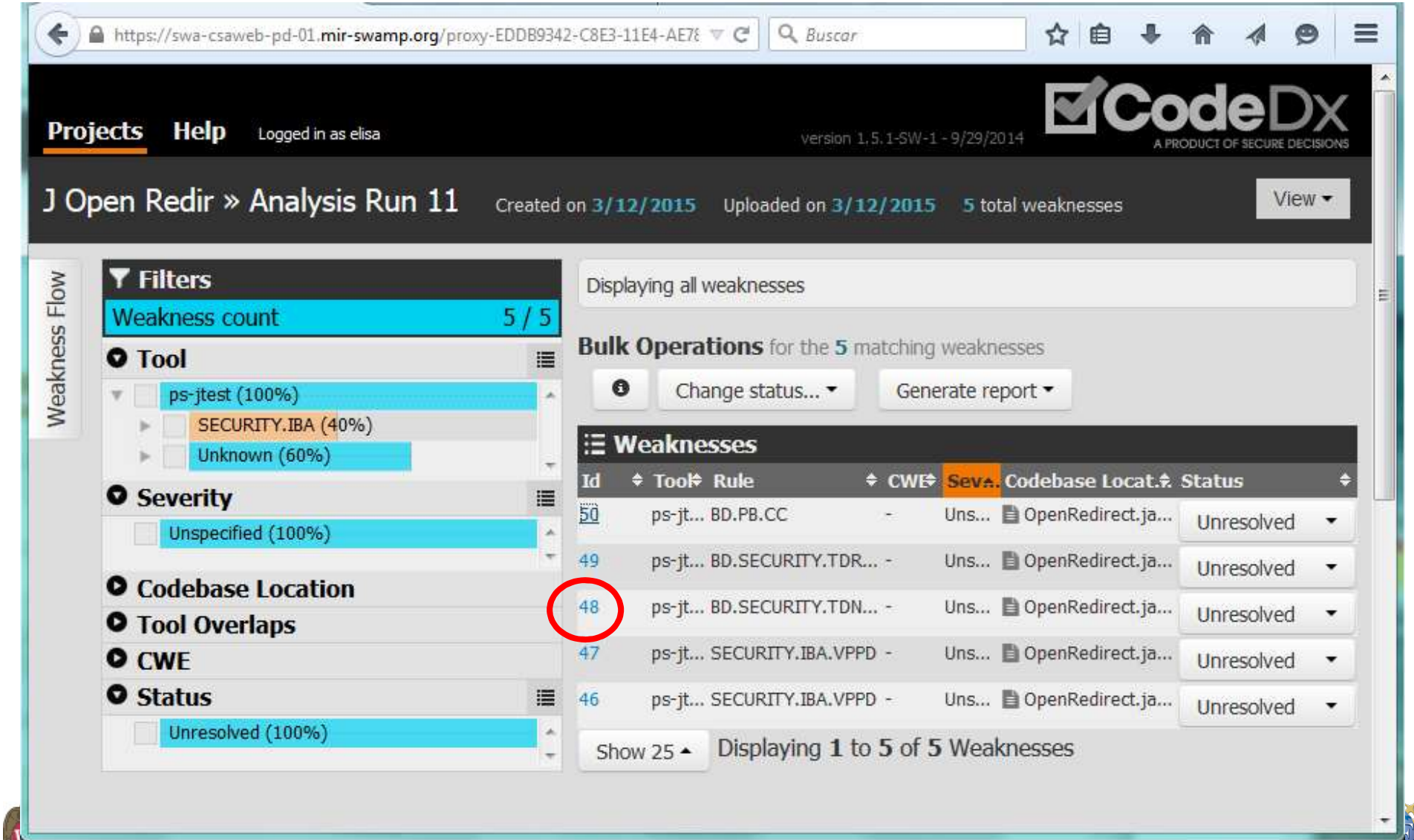
ps-jtest v9.5 Report

Summary

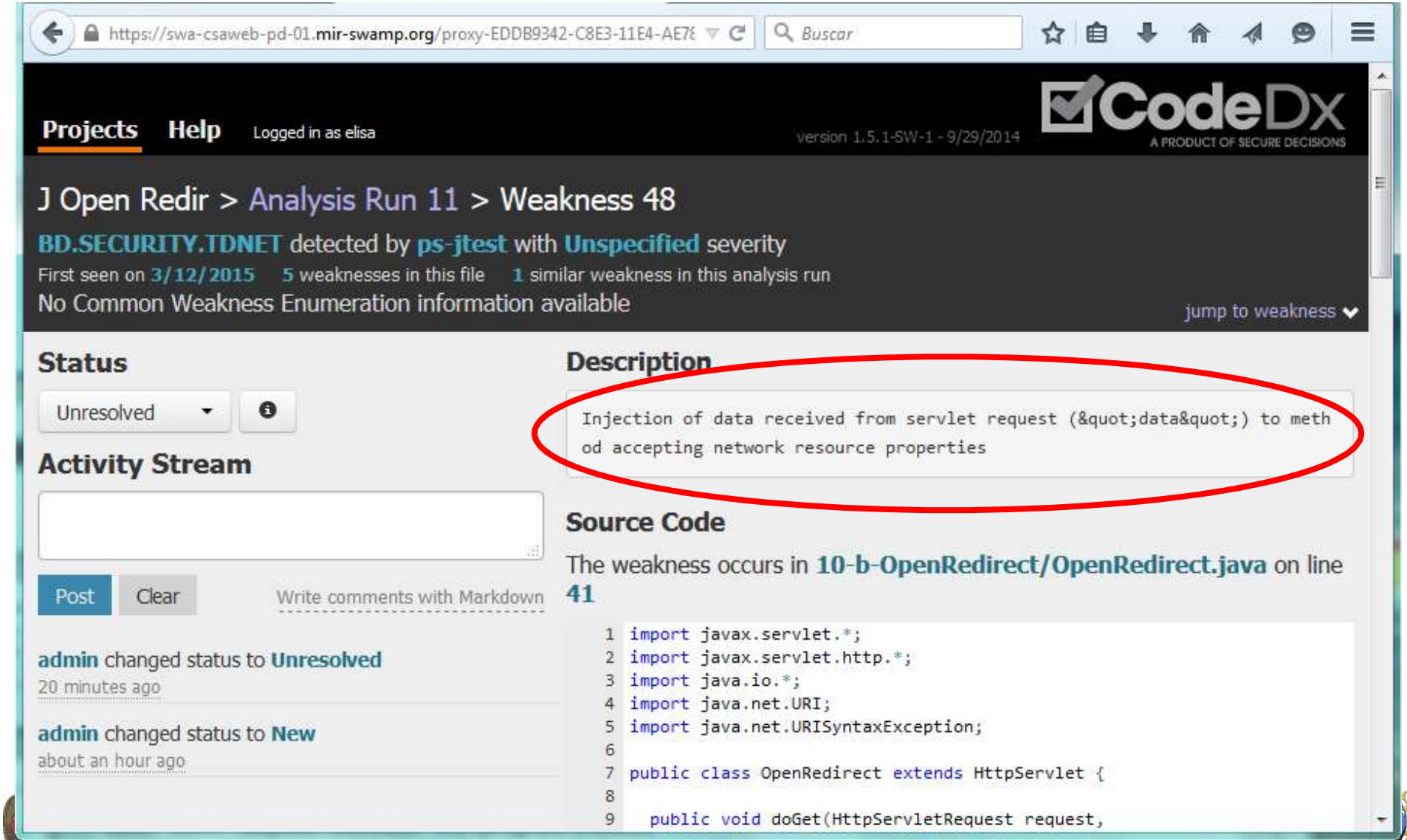
Total
5

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

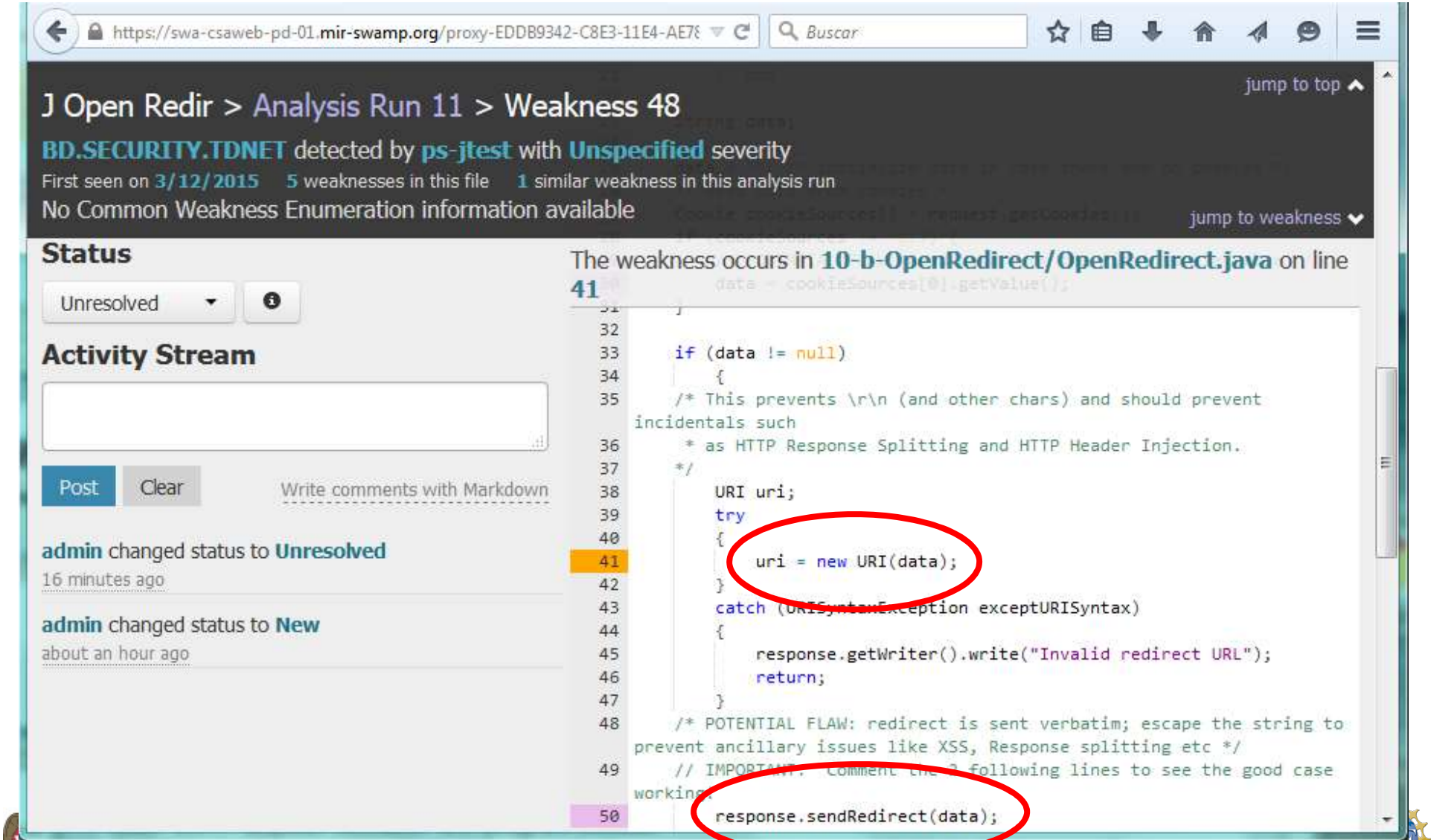
My Results for Java: Jtest – J Open Redir - CodeDx



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



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



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

Questions?

Elisa Heymann Elisa.Heymann@uab.es

Barton P. Miller bart@cs.wisc.edu

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