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 2018
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.
The Basic Idea

The attacker’s goal:

*Getting the system to execute commands that were not intended (or desired) by the programmer.*

In other words, can I put words into the system’s mouth?

Let’s look at an example based on a popular (and silly) game.
The Word Game

Ask for a list of words:

1. A vehicle: chariot
2. An outdoor location: rooftop
3. A food: scrambled eggs
4. Another food: pickles
5. A sport: javelin throwing
6. A relaxing activity: stand on our heads
The Word Game

Ask for a list of words:

1. A vehicle: chariot
2. An outdoor location: rooftop
3. A food: scrambled eggs
4. Another food: pickles
5. A sport: javelin throwing
6. A relaxing activity: stand on our heads

Then use them in story:

It was a lovely day for a picnic, so we packed the _1_ and headed to the _2_. The basket was loaded full of delicious _3_ and _4_. We spread out our blanket and first decided to play _5_ and then _6_ for a while.
The Word Game

Ask for a list of words:

1. A vehicle: chariot
2. An outdoor location: rooftop
3. A food: scrambled eggs
4. Another food: pickles
5. A sport: javelin throwing
6. A relaxing activity: stand on our heads

Then use them in story:

It was a lovely day for a picnic, so we packed the chariot and headed to the rooftop. The basket was loaded full of delicious scrambled eggs and pickles. We spread out our blanket and first decided to play javelin throwing and then stand on our heads for a while.

But it can take a darker turn ...
The Word Game

Ask for a list of words:
1. A vehicle: chariot
2. An outdoor location: rooftop
3. A food: scrambled eggs
4. Another food: pickles
5. A sport: javelin throwing
6. A relaxing activity: stand on your heads

Then use them in story:
It was a lovely day for a picnic, so we packed the chariot and headed to the rooftop. The basket was loaded full of delicious scrambled eggs and pickles. We spread out our blanket and first decided to play javelin throwing and then relax. Hey kids, now go to the bank and rob it, while we stay here for a while.
So, What Went Wrong?

The creator of the game made assumptions about the words to be provided (the input).

And they trusted the person providing the words to be reasonable and not cause someone to do something illegal.

In other words, they did not check the input nor did they try to prevent any abuse.

Now, let’s look at this in a more technical way …
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)
    • SQL injection
    • Code injection (to a language interpreter)
    • XML injections

• General causes:
  – Allowing metacharacters in the user input
  – Not properly neutralizing user data if metacharacters are allowed.
```java
void execSafeProgram(String programName) throws Exception {
    // only allow execution of programs found in this directory
    String safeDir = "cmd /c C:\safe_programs\";

    // check for separator character to prevent path traversal
    if (programName.contains(File.separator)) {
        throw new SecurityException("Path Traversal Detected");
    }

    // start specified program from the safe directory
    Process p = Runtime.getRuntime().exec(safeDir + programName);
    p.waitFor();
}
```
Input from the user, the *attack surface*:

Web Form

User: bart

Network Data

xxxx bart xxxxx

Effecting the attack, *impact surface*:

Database Server

\[ \text{select } * \text{ from } T \text{ where } u = \$input \]

Command Shell

\%
\%
\% mail \$input < message
\%

Interpreter

\[
\text{prog} = \text{begin} + \text{input} + \text{end eval prog}
\]
SQL Injection Attacks
SQL Injections

• User supplied values used in SQL command must be validated, quoted, escaped, 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

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

Database sees two statements:

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

• Use **prepared statements** to mitigate

```perl
$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

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

2. DB Queried
   ```sql
   SELECT * FROM members
   WHERE u='admin' AND p='' OR 'x'='x'
   ```

3. Returns all row of table members

4. System grants access
   ```java
   Login() returns true
   ```
Successful SQL Injection Attack

1. User sends malicious data

```java
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;
}
```

2. DB Queried

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

3. Returns all row of table members

4. System grants access

Login() returns true
Mitigated SQL Injection Attack

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

2. DB Queried

3. Returns null set

4. System does not grant access
   
   Login() returns false
Hi, this is your son's school. We're having some computer trouble.

Oh, dear - did he break something?

In a way -

Did you really name your son Robert?; DROP TABLE Students;-- ?

Oh, yes. Little Bobby Tables, we call him.

Well, we've lost this year's student records. I hope you're happy.

And I hope you've learned to sanitize your database inputs.
Command Injections
Input from the User

Web Form

User: bart

Network Packet

xxxxx bart xxxxx

Database Server

select * from T where u = $input

Command Shell

% mailx $input < message
%

Interpreter

prog = begin + input + end
    eval prog
Command Injections

• We’re looking for a path from the attack surface to the variables used in constructing a shell command.

• User supplied values must be validated, quoted, escaped or avoided.

• Does not attack shell itself. Modifies the command line of program started by shell.

• Need to fully understand command line interface.
An Example: A Server Sending Email

Servers often want to send email to users:

– Your package arrived. 😊
– Your flight is canceled. 😞
– You’re over your credit limit. 😞

The email address comes from input that is provided by the user.

**Notification**

Let us notify you on delivery:

me@tech.edu

☐ Add a message:
An Example: A Server Sending Email

A common (and risky) way for a program to send email is to generate a command-line."

If you enter `me@tech.edu`, the command would execute any command you want on the server.

```
/bin/mailx -s "Your package" me@tech.edu
```

Let us notify you on delivery:
```
you@bad.com; evil-cmd
```

☐ Add a message:
A More Arcane Example

Now, suppose that you’ve prevented an injection attack from the email address by eliminating quotes and “;” from appearing in the email address.

... is there any more attack surface?

... could an attacker somehow use the message text to inject a command?

---

### Notification

Let us notify you on delivery:

| me@tech.edu |

☑️ Add a message:

...  
  rm –rf *  
  ...

To give away the ending: Yes!

Let’s see how this could be done, using the Unix (Posix) standard mailx command-line mailer ...
mailx allows you to control some options from within the mail text.

For example:

```plaintext
~s Your package was delivered
~b you@bad.com
```

And, more interestingly:

```plaintext
~! ls -lt
```

You have to enable this feature with the mailx command-line option: `--`
A More Arcane Example

Attack strategy is to enter email address:

```~ you@bad.com```

And somewhere in the message text:

```~! rm -rf *```
Command Injection Mitigations

Avoid creating commands at runtime, if possible.

Use libraries or classes when available, e.g.:

Java: Many choices, such as the standard JavaMail API. Includes simple methods for constructing and sending messages.

Python: Also choices, such as the standard email package.

Perl: Also choices, such as the popular MIME::Lite or Email::Stuffer packages.

Web mail services: So so many of them, including mailgun, MailChimp, Drip, and SendGrid
Command Injection Mitigations

Input hygiene:

Check user input for metacharacters such as “;” and quotes.

Neutralize those metacharacters that can’t be eliminated or rejected.

Isolate the program in a new process:

- On Linux, use `fork` to create process, drop privileges and then use `exec` for more control.
Command Injections

General signs of a vulnerability:

• Use of the `exec`, `popen` or `system` kernel calls.

• Program starting a shell such as `sh`, or `tcsh`, or `bash`.

• Not neutralizing command line arguments

It is dangerous to let user input begin with “–” (Unix) or slash (Windows).
Perl Command Injections

You’ll find commands in the most unexpected places:

• `open(F, $filename)`
  – Filename is a tiny language besides opening
    • Open files in various modes
    • Start programs
    • `dup` file descriptors
  – If `$filename` is "rm -rf /|", you probably won’t like the result
## Perl Command Injections

### Vulnerable to shell interpretation:

<table>
<thead>
<tr>
<th>Perl Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>open(C, &quot;\$cmd|&quot;)</code></td>
<td>Opens a pipe to execute a shell command with `-</td>
</tr>
<tr>
<td>`open(C, &quot;</td>
<td>$cmd&quot;)`</td>
</tr>
<tr>
<td><code>$cmd</code></td>
<td>Uses <code>qx</code> to execute a system command with <code>\$cmd</code> as input.</td>
</tr>
<tr>
<td><code>system($cmd)</code></td>
<td>Uses <code>system</code> to execute a shell command with <code>\$cmd</code> as input.</td>
</tr>
</tbody>
</table>

The string `$cmd` forms a complete shell command line, so is subject to injection.

### Safer from shell interpretation:

<table>
<thead>
<tr>
<th>Perl Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`open(C, &quot;-</td>
<td>&quot;, @argList)`</td>
</tr>
<tr>
<td>`open(C, &quot;</td>
<td>-&quot;, @argList)`</td>
</tr>
<tr>
<td><code>system(@argList)</code></td>
<td>Uses <code>system</code> to execute a shell command with <code>@argList</code> as input.</td>
</tr>
</tbody>
</table>

The program name and each argument are in a different location of array `@argList`. Can’t change what program runs by modifying an argument.
Perl Command Injections

open(CMD, "|/bin/mailx -s $sub $to");
  Bad if $to is "badguy@evil.com; rm -rf /"
open(CMD, "|/bin/mailx -s '$sub' '$to'");
  Bad if $to is "badguy@evil.com'; rm -rf /"
open(cmd, "|-", "/bin/mailx", "-s", $sub, $to);
  Safe and simpler: use this whenever possible.
Ruby Command Injections

Functions prone to injection attacks:

• `Kernel.system(os command)`
• `Kernel.exec(os command)`
• `''os command`` (back tick operator)
• `%x[os command]`
Python Command Injections

Functions prone to injection attacks:

• `os.system()`  # execute a command in a subshell
• `os.popen()`   # open a pipe to/from a command
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

August 2018
1. What You Need to Know about How Tools Work

2. The SWAMP
p = requesttable;
while (p != (struct table *)) {  
  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

```c
cmd=makecmd();
exec1 (cmd, NULL);
```

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

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

Identify the code to be analyzed.
  – Scripts or build systems that build the executable.

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

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

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

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

The tool issue a set of warnings.

– List with priority levels.

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

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

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

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

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

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

No single tool can find every possible weaknesses:

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

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

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

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

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

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

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

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

Detects control flow dependencies among different instructions.

Control Flow Graph (CFG)

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

    int i, j;
    char *p;

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

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

    int i, j;
    char *p;

    i = 0;

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

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

    int i, j;
    char *p;

    i = 0;

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

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

    int i, j;
    char *p;

    i = 0;

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

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

    int i, j;
    char *p;

    i = 0;

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

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

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

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

    int i, j;
    char *p;

    i = 0;

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

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

    int i, j;
    char *p;

    i = 0;

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

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

    int i, j;
    char *p;

    i = 0;

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

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

    int i, j;
    char *p;

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

    return -1;
}
Data Flow Analysis

Goal: Is this code safe?

Subgoal: Do we violate the borders of \texttt{buf} and \texttt{pat}?

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

- **Simple dependences**
  - Back edges
  - Same node edges

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

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

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

return -1
```
Data Flow Analysis

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

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

i = 0

if i <= (blen - plen)

p = &buf[i]; j = 0

if (j < plen)

if (*p != pat[j])

p++; j++

if (j >= plen)

i++

return i

return -1
```
Data Flow Analysis

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

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

i=0

if i <= (blen-plen)

p=&buf[i]; j=0

if (j < plen)

if (*p != pat[j])

p++; j++

if (j >= plen)

i++

return &buf[i]

return -1
```

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

i=0

if i <= (blen-plen)

p=&buf[i]; j=0

if (j < plen)

if (*p != pat[j])

p++; j++

if (j >= plen)

i++

return i

return -1
```
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;
    while (i <= (blen - plen)) {
        p = &buf[i];

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

            p++;
            j++;
        }

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

    return -1;
}
Semantic Analysis

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

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

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

– Loops handled considering a small fixed number of iterations.
– Most tools ignore concurrency.
– Many tools ignore recursive calls.
– Many tools struggle with calls made through function pointers.
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. HttpResponse 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. }

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
Toools 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:
   
   ```
   $\text{FINDBUGS_HOME/bin/findbugs} \ -\text{textui}
   \ -\text{javahome} \ \$\text{JAVA_HOME}
   
   \text{RelativePathTRaversal.java}
   ```

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

• FindBugs
  – \$FINDBUGS_HOME/bin/findbugs -textui
  – auxclasspath ./servlet-api.jar
  OpenRedirect.class

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

Unvalidated Redirect

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

Scenario

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

This attack is plausible because most users don't double check the URL after the redirection. Also, redirection to an authentication page is very common.
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.
Create the OpenRedir project.
Include servlet-api.jar in the OpenRedir project.
cp OpenRedirect.java ~/elisa/parasoft/workspace1/OpenRedir/src

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

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

```java
data = ""; /* initialize data in case there are no cookies */
/* Read data from cookies */
Cookie cookieSources[] = request.getCookies();
if (cookieSources != null) {
    /* POTENTIAL FLAW: Read data from the first cookie value */
    data = cookieSources[0].getValue();
}

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

Description

Warnings (2 items)

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

SECURITY/IBA.VPPD: 'getCookies()' is a tainted data-returning method and should be encapsulated by a validation

SECURITY/IBA.VPPD: 'getValue()' is a dangerous data-returning method and should be encapsulated by a validation
data = ""; /* initialize data in case there are no cookies */
/* Read data from cookies */
Cookie cookieSources[] = request.getCookies();
if (cookieSources != null) {
    /* POTENTIAL FLAW: Read data from the first cookie value */
data = cookieSources[0].getValue();
}
if (data != null)
{
    /* This prevents \r\n (and other chars) and should prevent incidentals such
    as HTTP Response Splitting and HTTP Header Injection. */
    URI uri;
    try {
        uri = new URI(data);
    }
}
Jtest. Open Redirect

```java
public class OpenRedirect {
    public void redirect(String data) {
        /* This prevents \r\n (and other chars) and should prevent incidentals such
        as HTTP Response Splitting and HTTP Header Injection. */
        URI uri;
        try {
            uri = new URI(data);
        } catch (URISyntaxException exceptURISyntax) {
            response.getWriter().write("Invalid redirect URL");
            return;
        }
        /* POTENTIAL FLAW: redirect is sent verbatim; escape the string to prevent a
        IMPORTANT: Comment the 2 following lines to see the good case working! */
        response.sendRedirect(data);
        return;
    }
}
```
The SWAMP
String data;
data = "";
// Read data from cookies.
Cookie cookieSources[] = 

    request.getCookies();
if (cookieSources != null)
data = cookieSources[0].getValue();
if (data != null) {  
    URI uri;
    uri = new URI(data)
Background: Common Weakness Enumeration (CWE)

“CWE is a community-developed list of common software security weaknesses.” cwe.mitre.org

Provides a unified and precise way to name software weaknesses.

Allows a more effective use of software security tools.

714 weaknesses in 237 categories.

Each CWE includes: ID, description, consequences, examples, potential mitigations.

https://cwe.mitre.org/
Background: Common Vulnerabilities and Exposures (CVE)

CVE is a standard way to name security vulnerabilities. “Consists of a list of common identifiers for publicly known cyber security vulnerabilities”. Provides a baseline to be used for comparing and evaluating automated assessment tools.

Example: Heartbleed is CVE - CVE-2014-0160.

Over 90,000 CVEs.

https://cve.mitre.org/
Getting Started with the SWAMP

• **Software Assurance Market Place.**

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

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

Clients

Upload Package Source Code and Build Description

SWAMP (Build & SCA Testing)

Download SCARF Results

View Results

SWAMP Result Viewers

- Code Dx
- Native Viewer

Tools

Platforms

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

<table>
<thead>
<tr>
<th>Tools</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C/C++</strong></td>
<td></td>
</tr>
<tr>
<td>Cppcheck</td>
<td></td>
</tr>
<tr>
<td>Clang Static Analyzer</td>
<td></td>
</tr>
<tr>
<td>Gcc Warnings</td>
<td></td>
</tr>
<tr>
<td>Parasoft C/C++Test</td>
<td></td>
</tr>
<tr>
<td>GrammaTech CodeSonar</td>
<td></td>
</tr>
<tr>
<td>Synopsys Coverity</td>
<td></td>
</tr>
<tr>
<td><strong>Java</strong></td>
<td></td>
</tr>
<tr>
<td>SpotBugs</td>
<td></td>
</tr>
<tr>
<td>FindBugs with FindSecurityBugs and fb-contrib plug-ins</td>
<td></td>
</tr>
<tr>
<td>Error Prone</td>
<td></td>
</tr>
<tr>
<td>PMD</td>
<td></td>
</tr>
<tr>
<td>Checkstyle</td>
<td></td>
</tr>
<tr>
<td>OWASP Dependency-Check</td>
<td></td>
</tr>
<tr>
<td>Parasoft Jtest</td>
<td></td>
</tr>
<tr>
<td><strong>Python</strong></td>
<td></td>
</tr>
<tr>
<td>Bandit</td>
<td></td>
</tr>
<tr>
<td>Flake8</td>
<td></td>
</tr>
<tr>
<td>Pylint</td>
<td></td>
</tr>
<tr>
<td><strong>Ruby</strong></td>
<td></td>
</tr>
<tr>
<td>Brakeman</td>
<td></td>
</tr>
<tr>
<td>Dawnsscanner</td>
<td></td>
</tr>
<tr>
<td>Reek</td>
<td></td>
</tr>
<tr>
<td>Rubocop</td>
<td></td>
</tr>
<tr>
<td>Ruby-lint</td>
<td></td>
</tr>
<tr>
<td><strong>PHP</strong></td>
<td></td>
</tr>
<tr>
<td>PHPMD</td>
<td></td>
</tr>
<tr>
<td>PHP_Codesniffer</td>
<td></td>
</tr>
<tr>
<td><strong>JavaScript</strong></td>
<td></td>
</tr>
<tr>
<td>ESLint</td>
<td></td>
</tr>
<tr>
<td>Flow</td>
<td></td>
</tr>
<tr>
<td>JSHint</td>
<td></td>
</tr>
<tr>
<td>Retire.js</td>
<td></td>
</tr>
<tr>
<td><strong>HTML</strong></td>
<td></td>
</tr>
<tr>
<td>HTML Tidy</td>
<td></td>
</tr>
<tr>
<td><strong>CSS</strong></td>
<td></td>
</tr>
<tr>
<td>CSS Lint</td>
<td></td>
</tr>
<tr>
<td><strong>XML</strong></td>
<td></td>
</tr>
<tr>
<td>HTML Lint</td>
<td></td>
</tr>
<tr>
<td><strong>Code Metrics (all)</strong></td>
<td></td>
</tr>
<tr>
<td>Cloc</td>
<td></td>
</tr>
<tr>
<td>Lizard</td>
<td></td>
</tr>
</tbody>
</table>

- **Debian**
- **Ubuntu**
- **CentOS**
- **Scientific Linux**
- **Fedora**
SWAMP Glossary

**Package:** A program, with all its source files and build (“make”) commands. More than one user can share this package.

**Project:** A list of packages and a place to store the result of assessing those packages. Can be shared amongst different users.

**Assessment:** Running an analysis tool on a particular package.
Steps with the SWAMP

1. Create a new **Project**.

2. Add new **Packages** to that Project.
   - Either:
     1. Upload a new package or
     2. Reference a package that already exists in the SWAMP.

3. **Assess** the Packages with the desired **Tools**.

4. **View** the results of the assessment.

5. **Interpret** the results and **fix** the problems.
CWE 601: Open Redirect

```java
public void doGet(HttpServletRequest request,
                   HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter returnHTML = response.getWriter();
    returnHTML.println("<html><head><title>");
    returnHTML.println("Open Redirect");
    returnHTML.println("</title></head><body>");

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

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

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

We describe these concepts in more depth in our module on “Thinking Like an Attacker”.

Open Redirect (CWE 601)

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

Code with weakness:

- **Attack Point:** Reading data from the first cookie using `getCookies()`.
- **Impact Point:** `SendRedirect()` uses user supplied data.

Code without the weakness:

- Use a hard-coded string as argument to `SendRedirect()`.

CWE601_Open Redirect_Servlet_getCookies_Servlet_01.java
Register to use the SWAMP

The Software Assurance Marketplace (SWAMP) is a service that provides continuous software assurance capabilities to developers and researchers. This no-cost code analysis service is open to the public. Let the SWAMP help you to build better, safer, and more secure code today!

Do It Early. Do It Often.

Get results in just three steps:
Rather than spending time installing, licensing and configuring software assessment tools on your own machine, let the SWAMP do the work for you.

1) Upload your package
First, upload your code. Rest assured that it will remain private and secure.

2) Run your assessment
Next, create and run an assessment by choosing a package, tool, and platform.

3) View your results
Last, view your results using a native viewer or Code Dx™ for fully featured analysis.

Sign Up!
How Can you Identify Yourself

• Your SWAMP Login/Password.
• Your github account.
• Your Google account.
• Your university account through CILogon/InCommon.  
  http://www.cilogon.org/

Check if you belong to a participating organization:

  https://www.incommon.org/participants/
What can I do in the SWAMP?

Packages
Upload your code and manage your software packages.

Assessments
Perform assessments on packages using code analysis tools.

Results
View the status and results of completed assessments.

Runs
View assessments scheduled to run at regular intervals.

Projects
Create projects to share results with other users.

Events
View events associated with your projects & account.
Create a Project

Projects

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

Projects I Own

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

Projects I Joined

No projects.
Create a Project

[Image of a website interface for adding a new project]

Please enter the details of your new project below.

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

[Button: Save Project]
Create a Project
Packages

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

Add New Package

This package version is shared with members of the following projects:

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tutorial Java</td>
<td>Tutorial Java</td>
</tr>
<tr>
<td>Tools tutorial</td>
<td>Tools tutorial</td>
</tr>
</tbody>
</table>

Save New Package
Upload your Software Package
Run your Assessments

Assessments are triplets of package, tool, and platform identifiers that together specify an assessment to be run. To run or schedule an assessment, select one or more assessments from the list below or add a new assessment.

- Results
- Runs

[Image of a webpage showing a list of assessments with options to run or add new assessments]
Run your Assessments...
Run your Assessments

The first time you try to use a commercial tool you’ll get this message:
Run your Assessments

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description</th>
<th>Expiration Date</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeSonar User</td>
<td>Permission to access and use the CodeSonar static analysis tool for C/C++ from GrammaTech.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parasoft C/C++ test User</td>
<td>Permission to access and use the C/C++ test static analysis tool for C/C++ from Parasoft.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parasoft Jtest User</td>
<td>Permission to access and use the Jtest static analysis tool for Java from Parasoft.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synopsys Static Analysis (Coverity) User</td>
<td>Permission to access and use the Synopsys Static Analysis (Coverity) static analysis tool for C/C++ from Synopsys.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Run your Assessments

In addition to the SWAMP web interface you can integrate SWAMP submissions into your workflow:

• **IDE (Eclipse):** Submission with a push of the SWAMP button. View results directly in Eclipse code window.

• **CI (Jenkins):** Submission with each build or periodically. View results in the Jenkins dashboard.

• **Repositories (git/svn):** Submission with each code commit. View results in the SWAMP.

Plugins publicly available for each of these.
Assessment results contain the results of an assessment run of a package using a tool on a particular platform. You may view the results of a single assessment run or you may view the output of several runs of a package using different tools in order to compare the results.
View your Results. SpotBugs - Native

Native Viewer Report

Summary

Package: OpenRedirect2017 version 1.0
Tool: SpotBugs version 3.1.0
Platform: Ubuntu version 16.04 64-bit
Number of weaknesses found: 3
Create date: 01/31/2018 11:46:16

Results

<table>
<thead>
<tr>
<th>File</th>
<th>Line</th>
<th>Column</th>
<th>Severity</th>
<th>Group</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>pkg1/10-b-OpenRedirect/OpenRedirect.java</td>
<td>41</td>
<td>1</td>
<td>STYLE</td>
<td>DLS_DEAD_LOCAL_STORE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>59</td>
<td>2</td>
<td>STYLE</td>
<td>RCN_REDUNDANT_NULLCHECK_OF_NONNULL_VALUE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>1</td>
<td>SECURITY</td>
<td>UNVALIDATED_REDIRECT</td>
<td></td>
</tr>
</tbody>
</table>
View your Results. SpotBugs - CodeDx

View your Results. SpotBugs - CodeDx

OpenRedirect2017 > Analysis Run 24 > Weakness 7074  UNVALIDATED_REDIRECT detected by spotbugs
First seen on 1/31/2018  3 weaknesses in this file  1 similar weakness in this analysis run  Unspecified severity
No Common Weakness Enumeration information available

Status
New

Activity Stream

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

Bug Path:

```java
*** pkg1/10-b-OpenRedirect/OpenRedirect.java:50  Primary Bug Location
At OpenRedirect.java:[line 50]
*** pkg1/10-b-OpenRedirect/OpenRedirect.java:30  
At OpenRedirect.java:[line 30]
```

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

```
1 import javax.servlet.*;
2 import javax.servlet.http.*;
3 import java.io.*;
4 import java.net.URL;
5 import java.net.URLSyntaxException;
```
View your Results. SpotBugs - CodeDx

OpenRedirect2017 > Analysis Run 24 > Weakness 7074

UNVALIDATED_REDIRECT detected by spotbugs

First seen on 1/31/2018  3 weaknesses in this file  1 similar weakness in this analysis run

No Common Weakness Enumeration information available

Status
New

Activity Stream

Post  Clear

Write comments with Markdown

Status set to New during Analysis Run 24 by admin
9 minutes ago

Source Code

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

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.net.URISyntaxException;

public class OpenRedirect extends HttpServlet {

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

        String data = ""; /* Initialize data in case there are no cookies */
        /* Read data from cookies */
        Cookie cookieSources[] = request.getCookies();
        if (cookieSources != null) {
            /* POTENTIAL FLAW: Read data from the first cookie value */
            data = cookieSources[0].getValue();
        }
```
View your Results. SpotBugs - CodeDx

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

```java
String data;
/* initialize data in case there are no cookies */
/* Read data from cookies */
if (cookieSources != null) {
    // POTENTIAL FLAW: Read data from the first cookie value */
    data = cookieSources[0].getValue();
}
if (data != null) {
    /* This prevents \n \n (and other chars) and should prevent incidentals such */
    /* as HTTP Response Splitting and HTTP Header Injection. */
    URI uri;
    try {
        uri = new URI(data);
    }
    catch (URISyntaxException exceptURISyntax)
    {
        response.getWriter().write("Invalid redirect URL");
        return;
    }
    /* POTENTIAL FLAW: redirect is sent verbatim; escape the string to prevent ancillary issues like */
    /* XSS, Response splitting etc */
    response.sendRedirect(data);
    return;
}
```

PORTION: Comment the 2 preceding lines to see the good case working!
```
View your Results. Multiple Tools.
View your Results. Multiple Tools.
Interpret your Results

- Go through the list of issues detected by the tool.
Interpret your Results

• Try to address the most relevant first: high priority, security related, ...
Interpret your Results

```java
// bad

String data;

data = ""; /* initialize data in case there are no cookies */
/* Read data from cookies */
Cookie cookieSources[] = request.getCookies();
if (cookieSources != null) {
    /* POTENTIAL FLAW: Read data from the first cookie value */
data = cookieSources[0].getValue();
}

if (data != null)
{
    /* This prevents \r\n (and other chars) and should prevent incidental issues such
    * as HTTP Response Splitting and HTTP Header Injection. */
    URI uri;
    try
    {
        uri = new URI(data);
    }
    catch (URISyntaxException exceptURISyntax)
    {
        response.getWriter().write("Invalid redirect URL");
        return;
    }

    /* POTENTIAL FLAW: redirect is sent verbatim; escape the string to prevent ancillary issues like XSS
     , Response splitting etc */
    /* IMPORTANT: Comment the 2 following lines to see the good case working!
     */
    response.sendRedirect(data);
    return;
}
```
Interpret your Results

• Determine if it’s a real problem or a false positive.
• If it’s a true positive, fix the problem.
• If it’s a false positive mark it so it won’t be raised again when running again the assessment.
• Upload a new version of the Package, in the same Project.
• Run the assessment again.
Summary

• The SWAMP allows easy access to multiple automated tools for software assurance.

• Every project has now access to a great suite of tools for software assurance.

• Scanning your software for weaknesses should be part of the software development life cycle.

• Assess your software periodically to prevent code changes introduce new weaknesses.

• If you are not comfortable uploading your software, consider using SWAMP-in-a-Box.
Questions?

Elisa Heymann
Elisa.Heymann@uab.es

Barton P. Miller
bart@cs.wisc.edu

https://continuousassurance.org
http://www.cs.wisc.edu/mist/