

# Automated Assessment Tools Theory & Practice

**Barton P. Miller**

Computer Sciences Department  
University of Wisconsin

[bart@cs.wisc.edu](mailto:bart@cs.wisc.edu)

**Elisa Heymann**

Computer Sciences Department  
University of Wisconsin  
Universitat Autònoma de Barcelona

[elisa@cs.wisc.edu](mailto:elisa@cs.wisc.edu)



August 2017

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



# Overview

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

# Injection Attacks

# Objectives

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

# Injection Attacks

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

# SQL Injections

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

# SQL Injections: attacks and mitigations

PERL

- Dynamically generated SQL without validation or quoting is vulnerable

```
$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

```
$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**

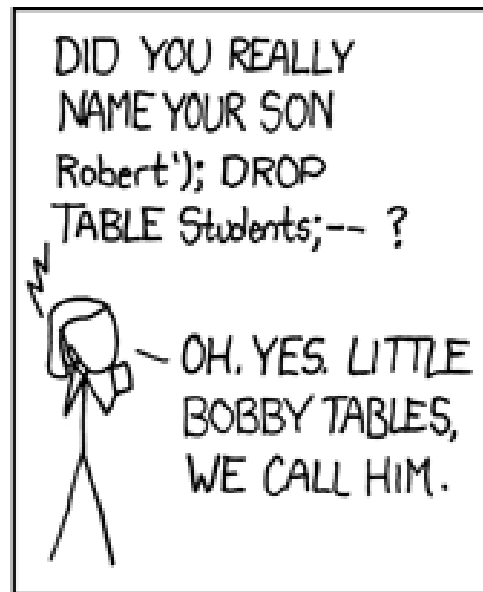
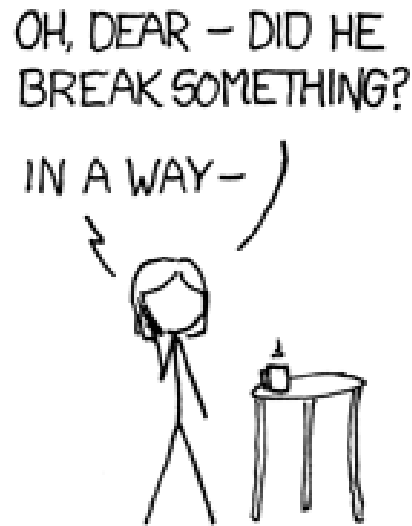
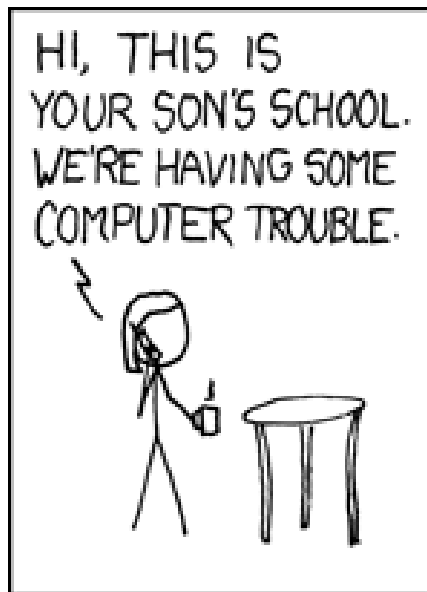
1. User sends malicious data

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



OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.



AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

<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 `csch`
  - 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



PERL

- `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")
`$cmd`
system($cmd)
```

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

- **Safe from shell interpretation**

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

# Perl Command Injection Examples

PERL

- `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



PERL

- 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



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";  
while (($k, $v) = (each %data)) {  
    if ($k eq 'name') {  
        print PH "LogIt.Name('$v')";  
    }  
}
```

Start Python,  
program sent  
on stdin

3. Python source executed – 2<sup>nd</sup> 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 – 2<sup>nd</sup> LogIt is now safe

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



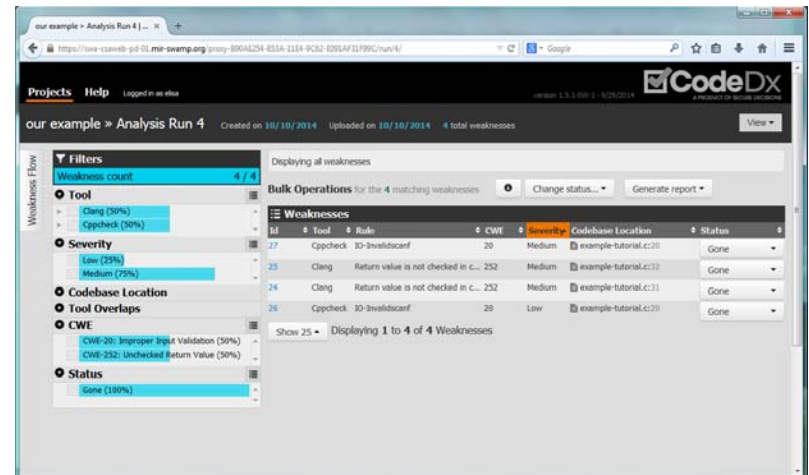
# Introduction to Automated Assessment Tools

# 1. What You Need to Know about How Tools Work

## 2. The Tools And Their Use

# Source Code Analysis Tools

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



# A Bit of History

## Compiler warnings

### Let the Compiler Help

- Turn on compiler warnings and fix problems
- Easy to do on new code
- Time consuming, but useful on old code
- Use lint, multiple compilers
- **-Wall** is not enough!

gcc: **-Wall, -W, -O2, -Werror, -Wshadow, -Wpointer-arith, -Wconversion, -Wcast-qual, -Wwrite-strings, -Wunreachable-code** and many more

- Many useful warning including security related warnings such as format strings and integers

# A Bit of History

- **Lint (1979)**
  - C program checker.
  - Detects suspicious constructs:
    - Variables being used before being set.
    - Division by zero.
    - Conditions that are constant.
    - Calculations whose result is likely to overflow.
- **Current automated assessment tools are a sort of “super-Lint”.**

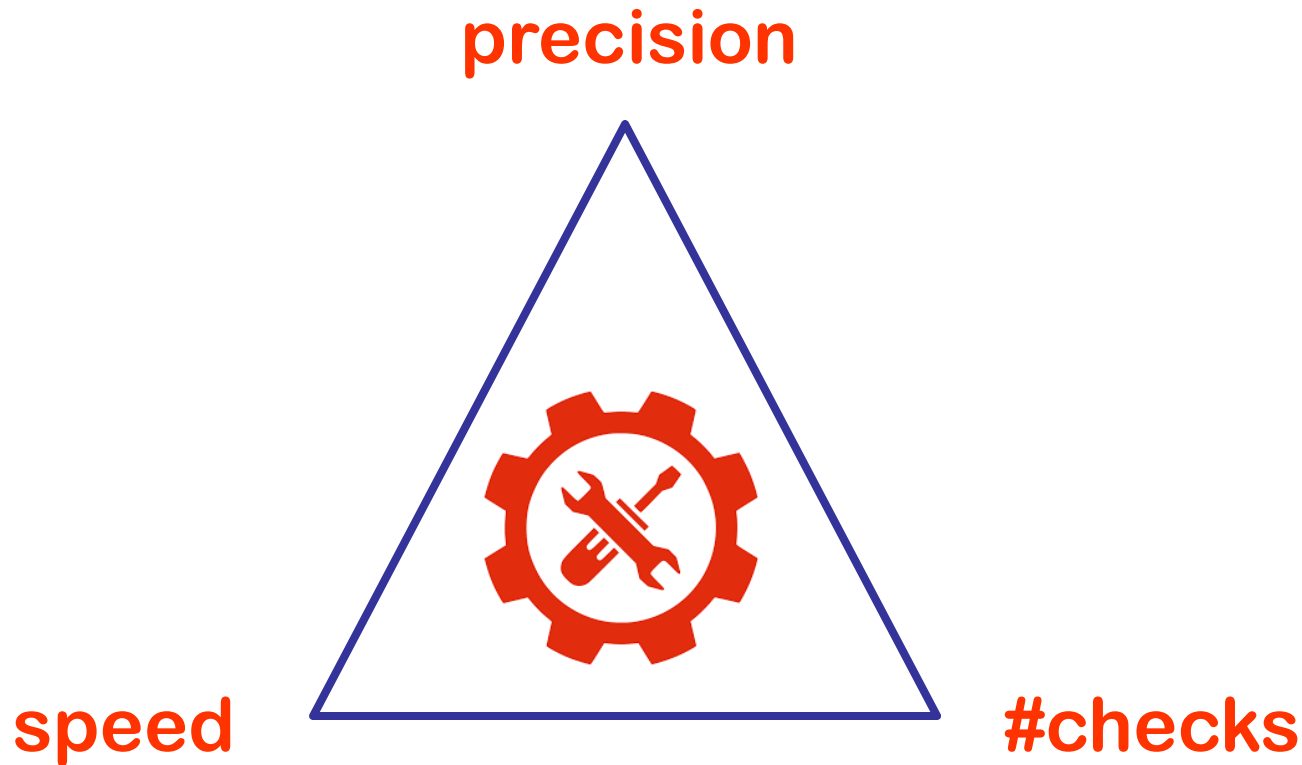
# Source Code Analysis Tools

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

# Source Code Analysis Tools

- Program that parses and then analyses the source code.
- Doesn't know what the program is supposed to do.
- Looks for violations of good programming practices.
- Looks for specific programming errors.
- Works like a compiler
  - Instead of binaries, it produces an intermediate representation

# Source Code Analysis Tools



You can get 2 out of 3



# 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”;  
exec1 (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();  
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

## Control Flow Analysis

Detects control flow dependencies among different instructions.

## Control Flow Graph (CFG)

- Abstract representation of the source code.
- Each node represents a basic block.
- Call or jump targets start a basic block.
- Jumps end a basic block.
- Directed edges represent the control flow.



```

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

    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```

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

```
    int i, j;  
    char *p;
```

```
    i = 0;
```

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

```
    return -1;
```

```
}
```

entry(pat, buf, plen, blen)



```

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;
}

```

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

```
i=0
```

```

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

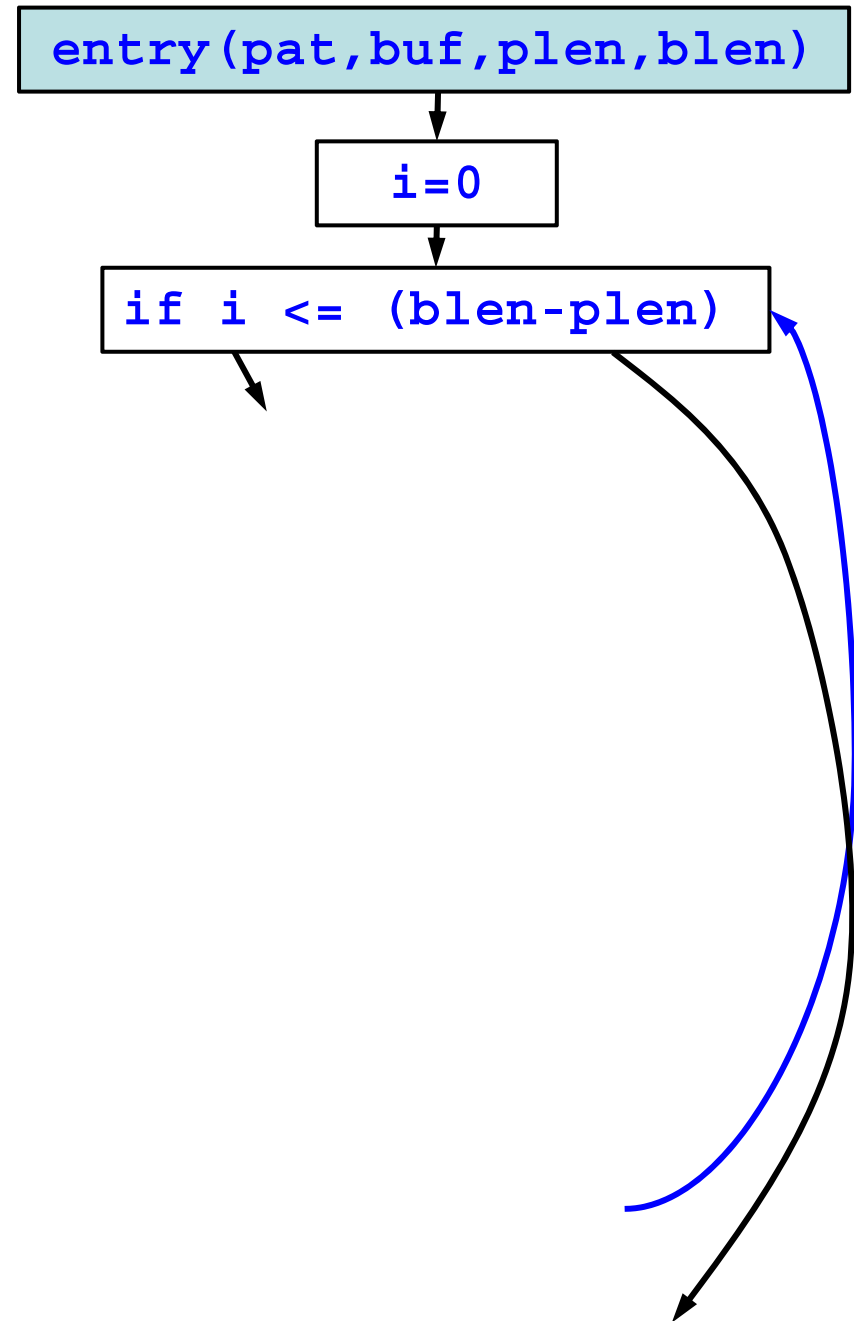
    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```



```

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

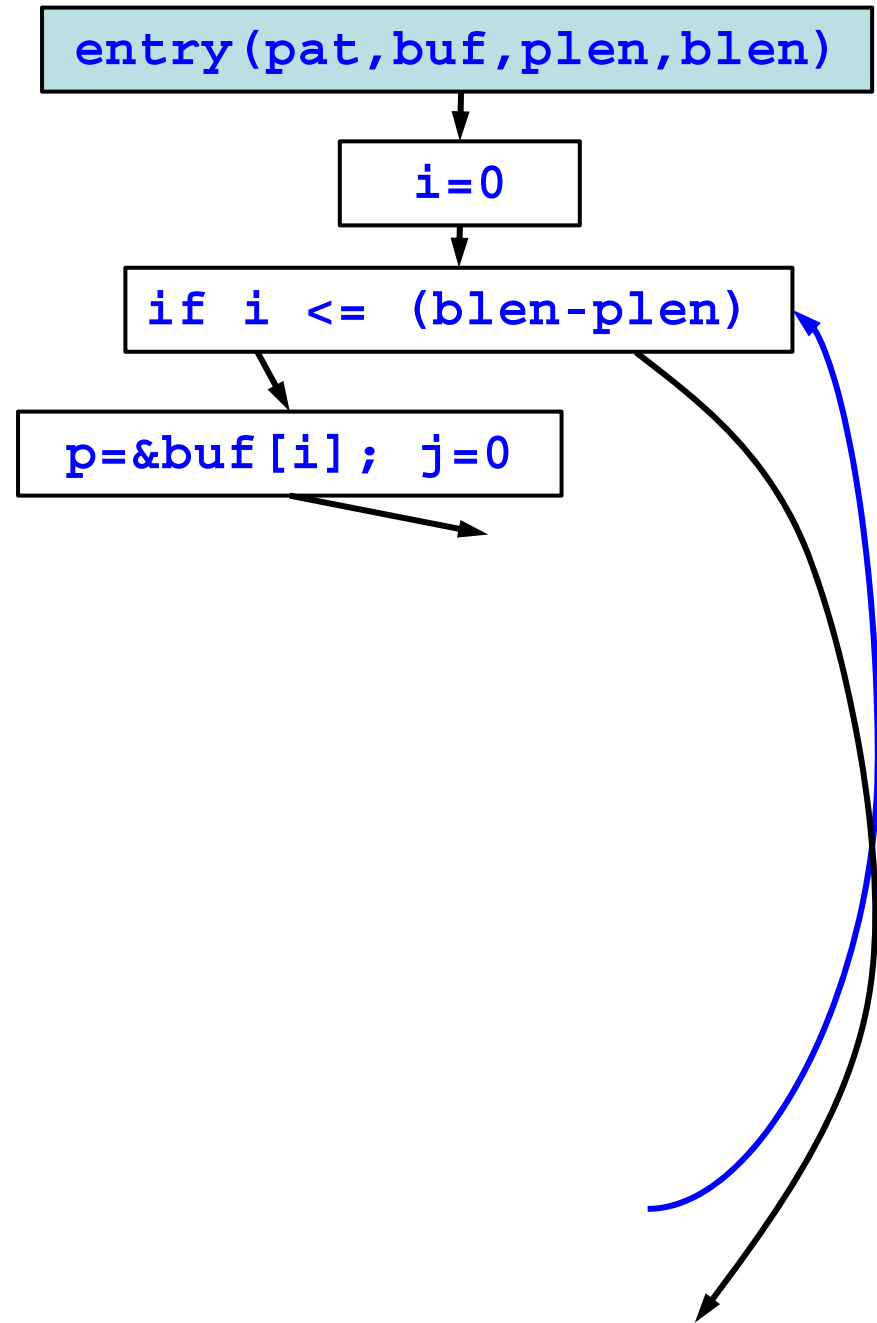
    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```



```

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

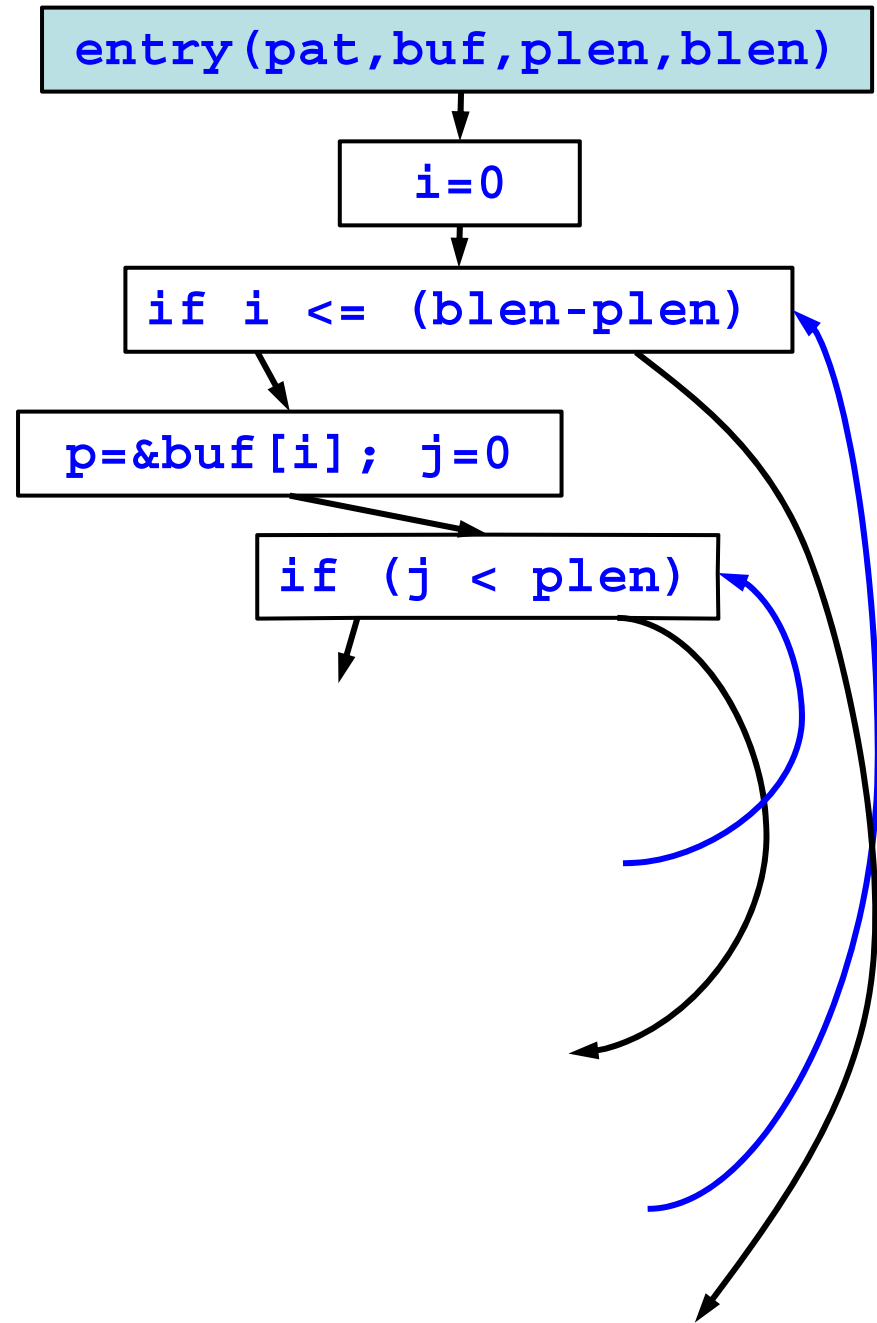
    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```



```

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

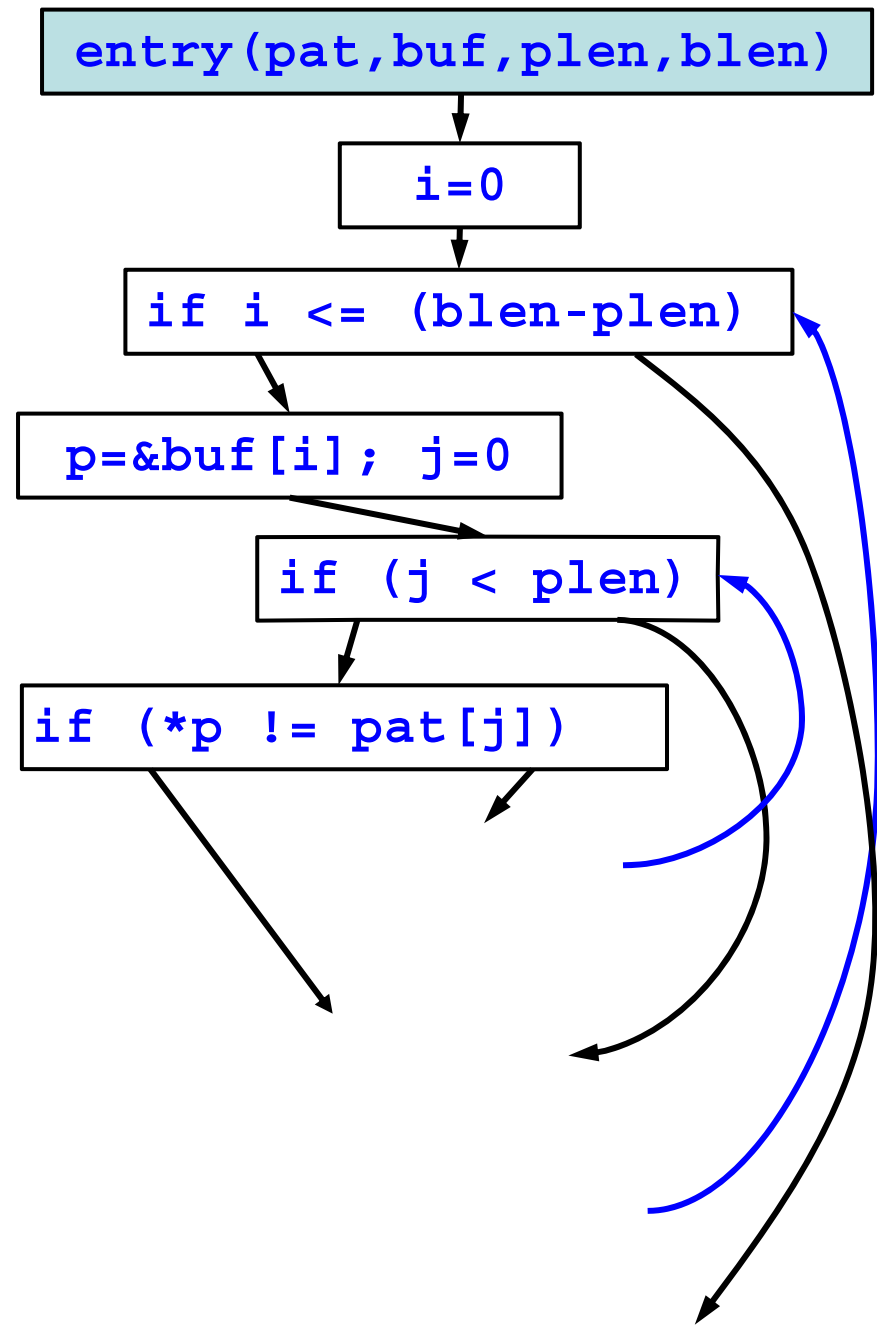
    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```



```

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

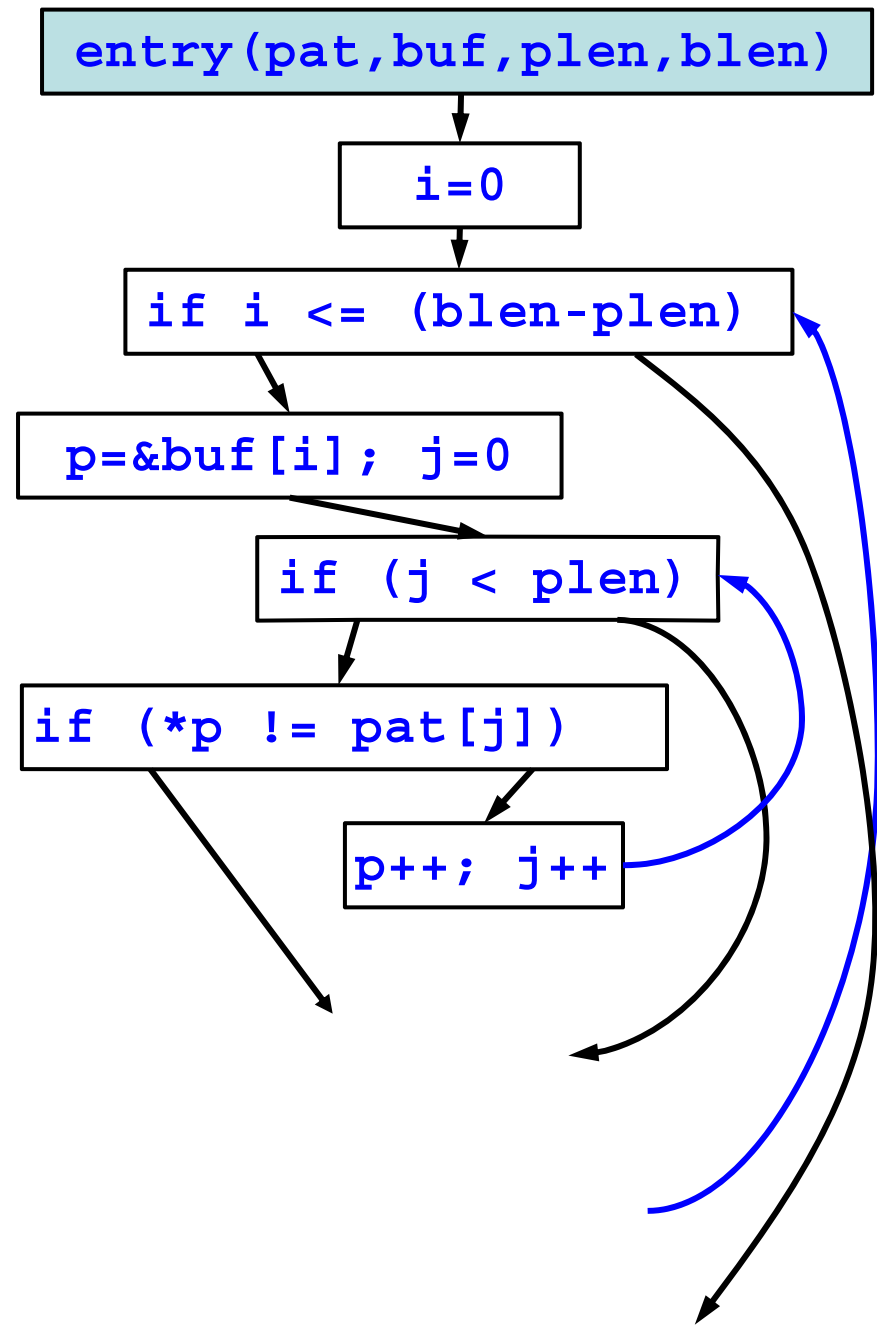
    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```





```

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

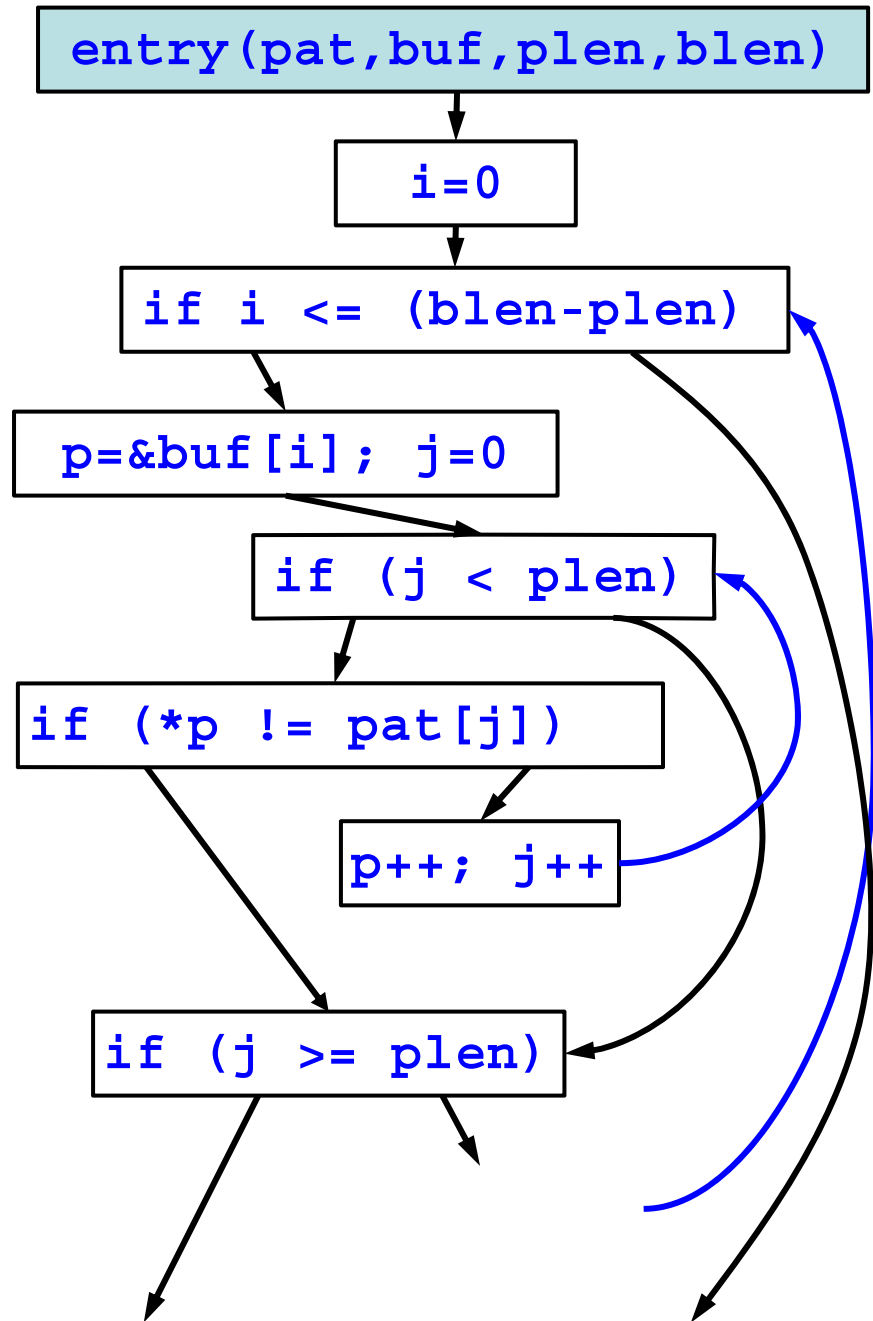
    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```



```

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

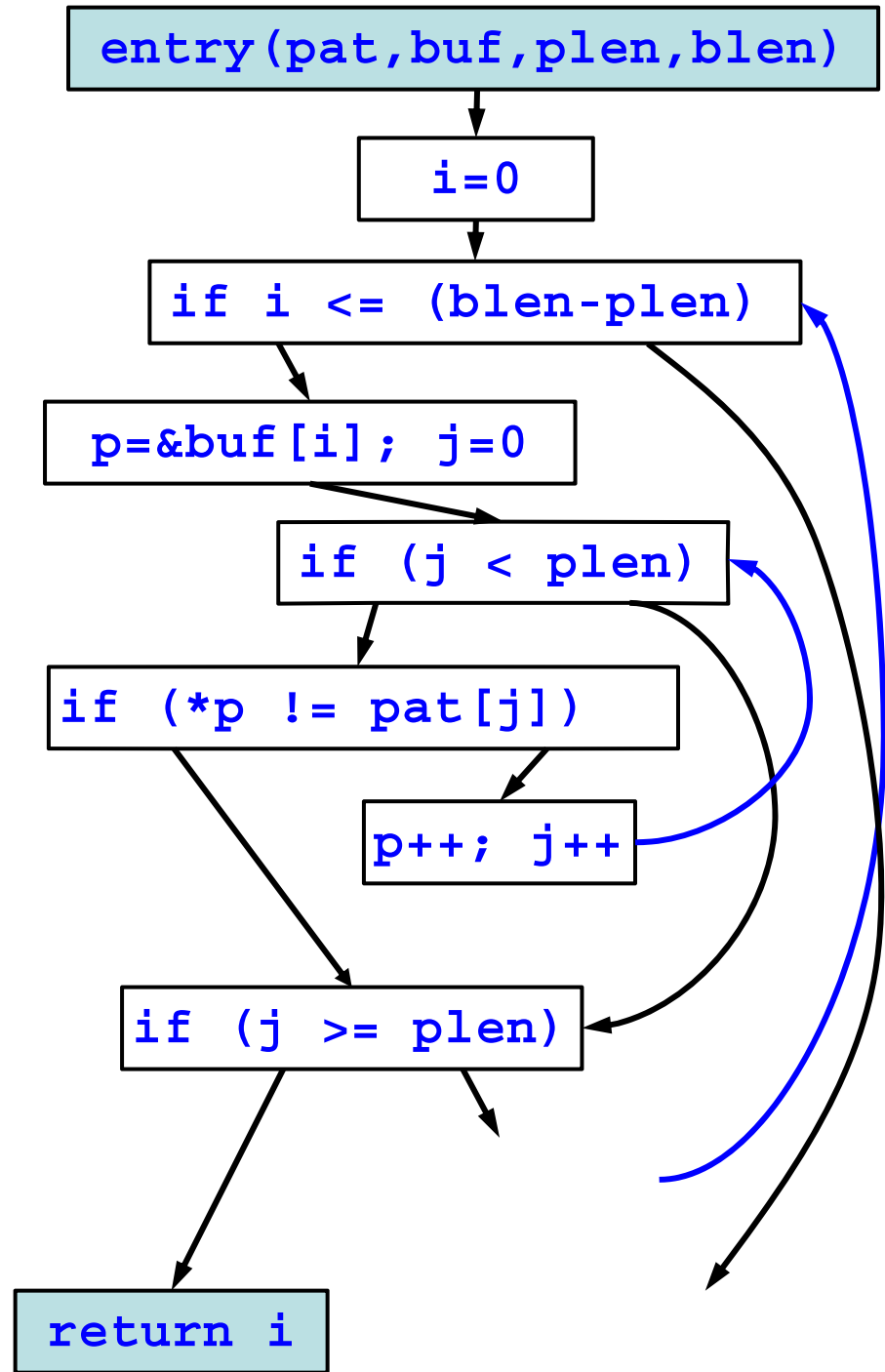
    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```



```

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

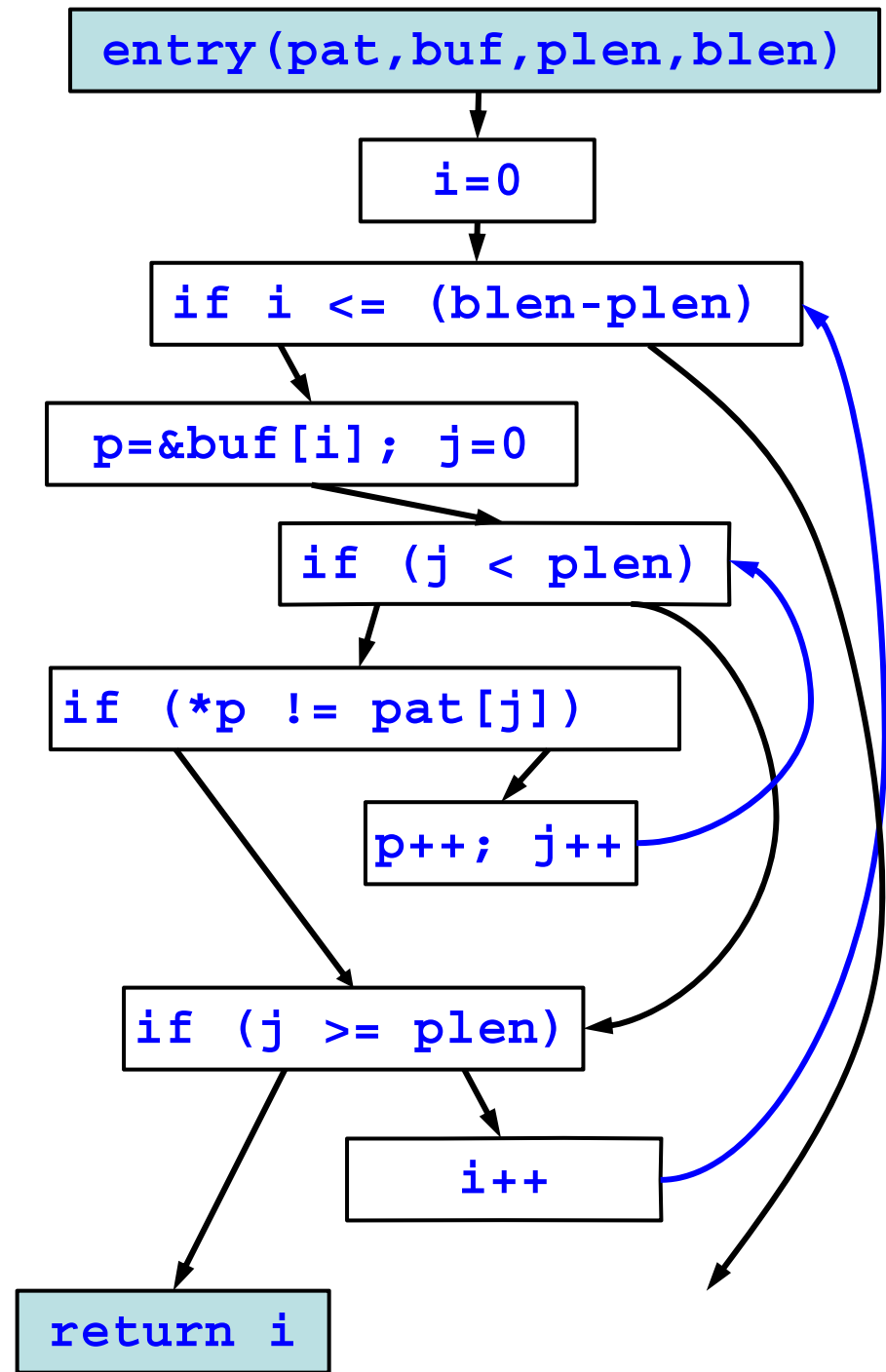
    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```



```

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

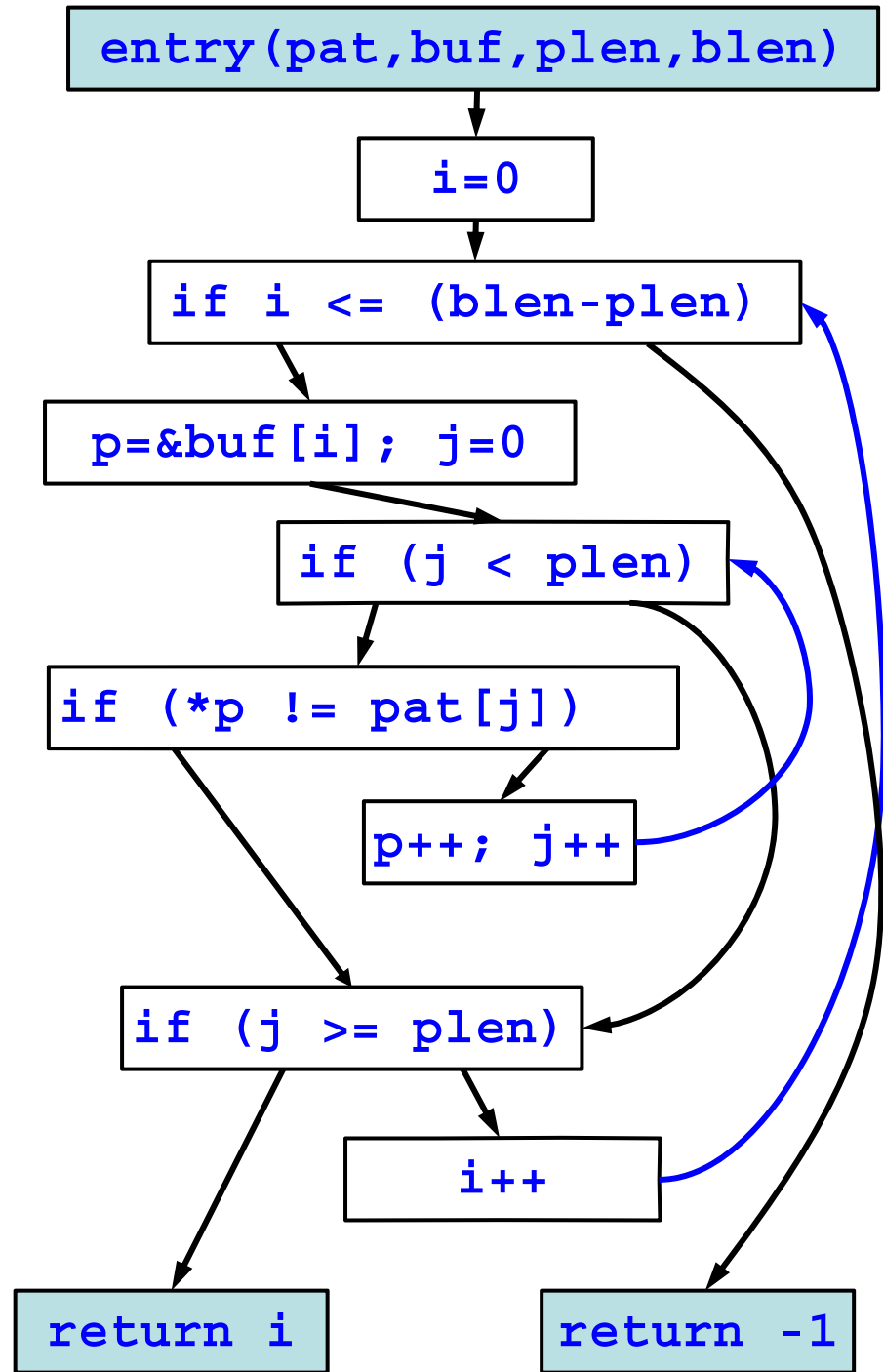
    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```



```

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

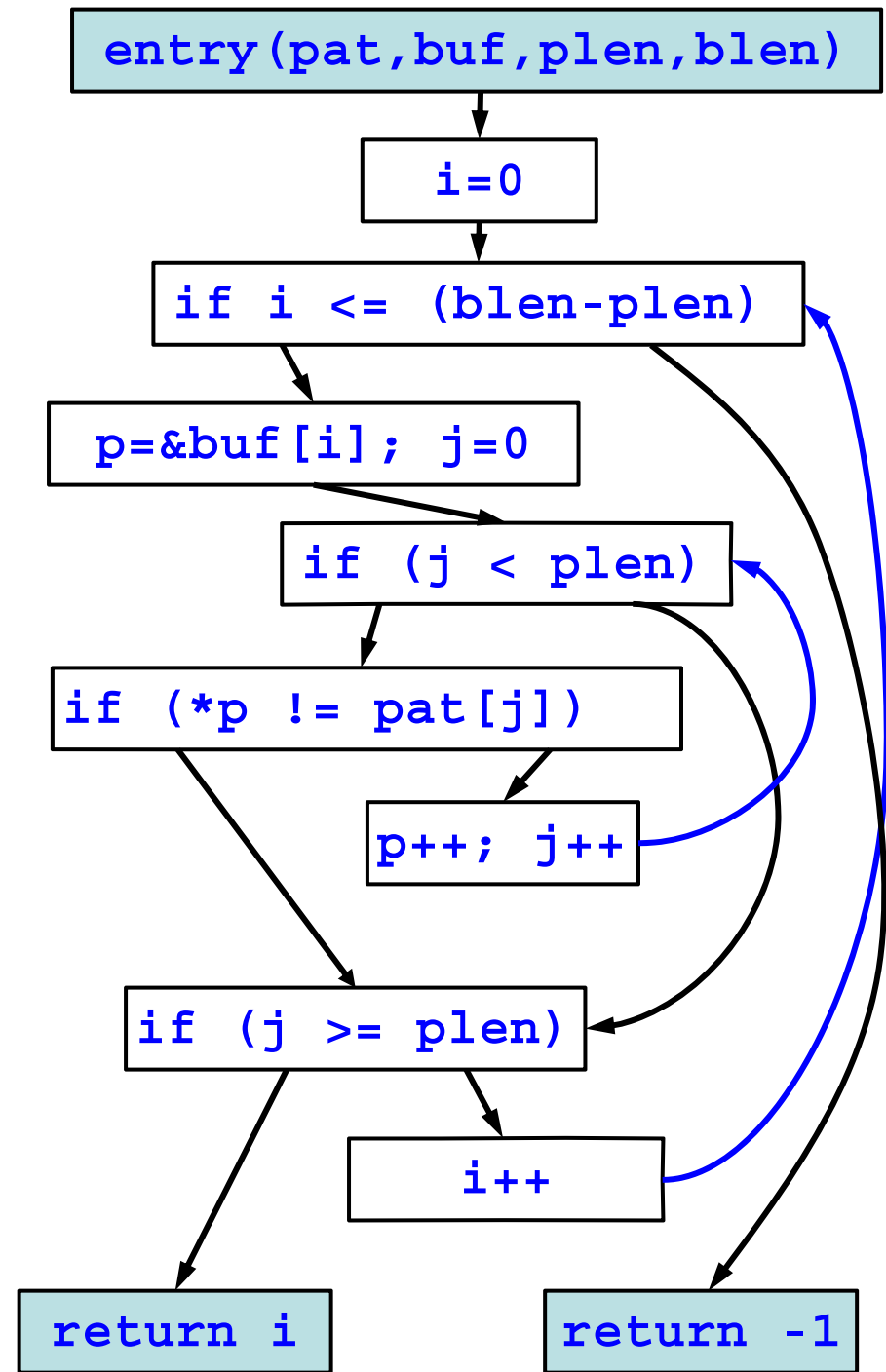
    int i, j;
    char *p;

    i = 0;

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

    return -1;
}

```



# Data Flow Analysis

**Goal:** Is this code safe?

**Subgoal:**

Do we violate the borders of buf and pat?

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

# Data Flow Analysis

- **Simple dependences**

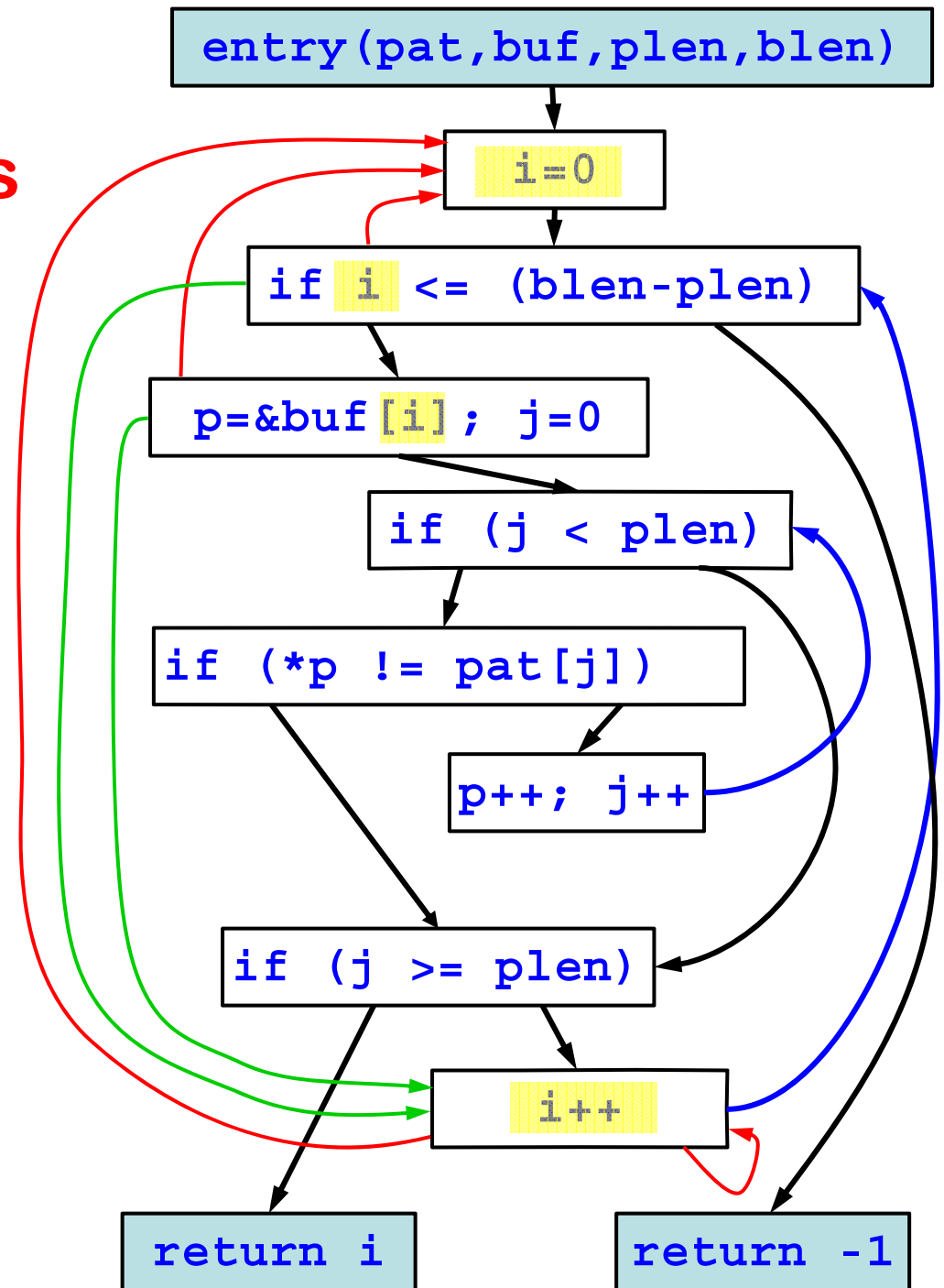
Back edges

Same node edges

- **Loop carried dependences**

- Need to understand the values for `i` to know that references to `buf[i]` are safe.

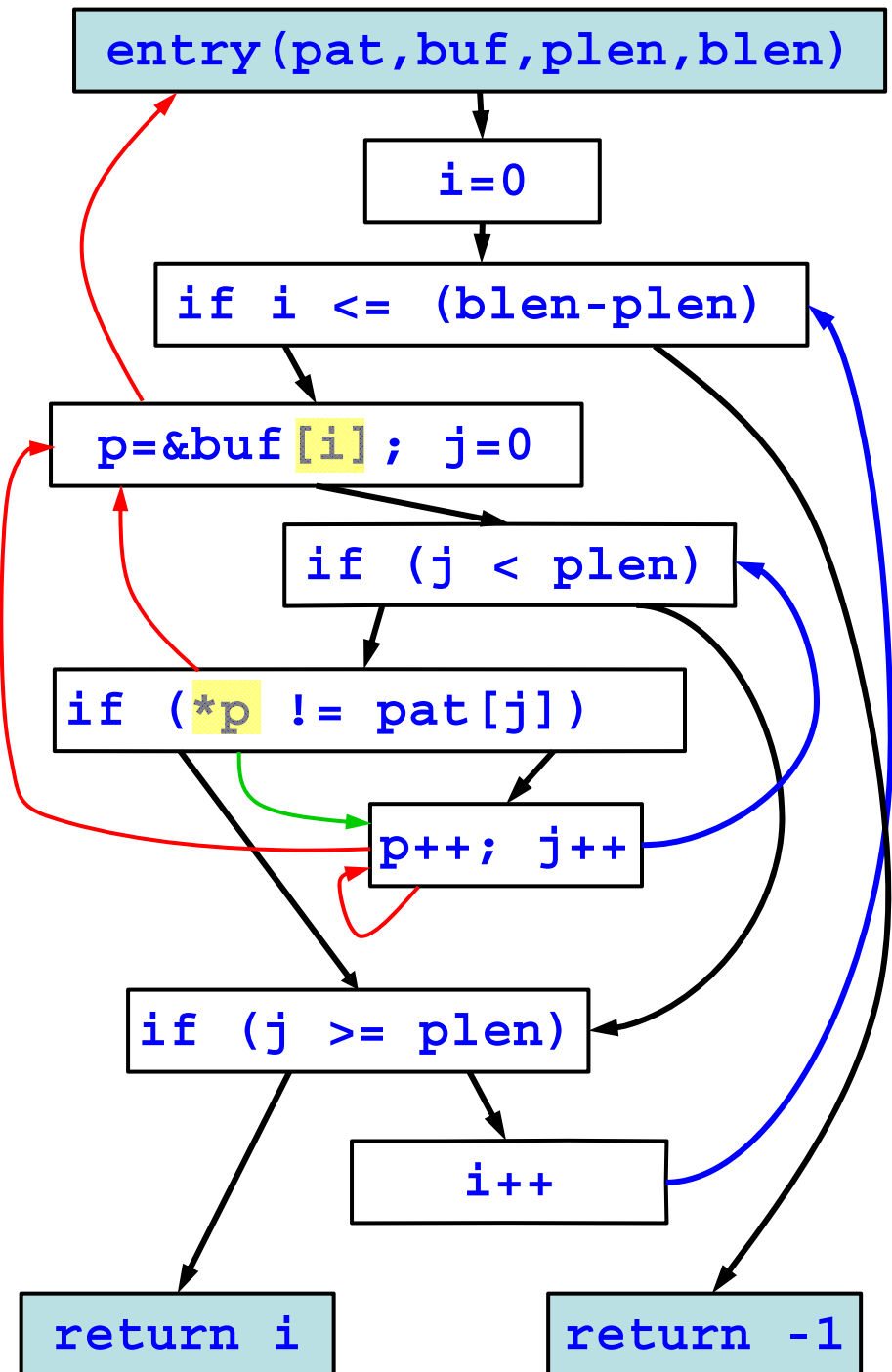
- Same for `j` and `pat[j]`.



# Data Flow Analysis

## • Pointers

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

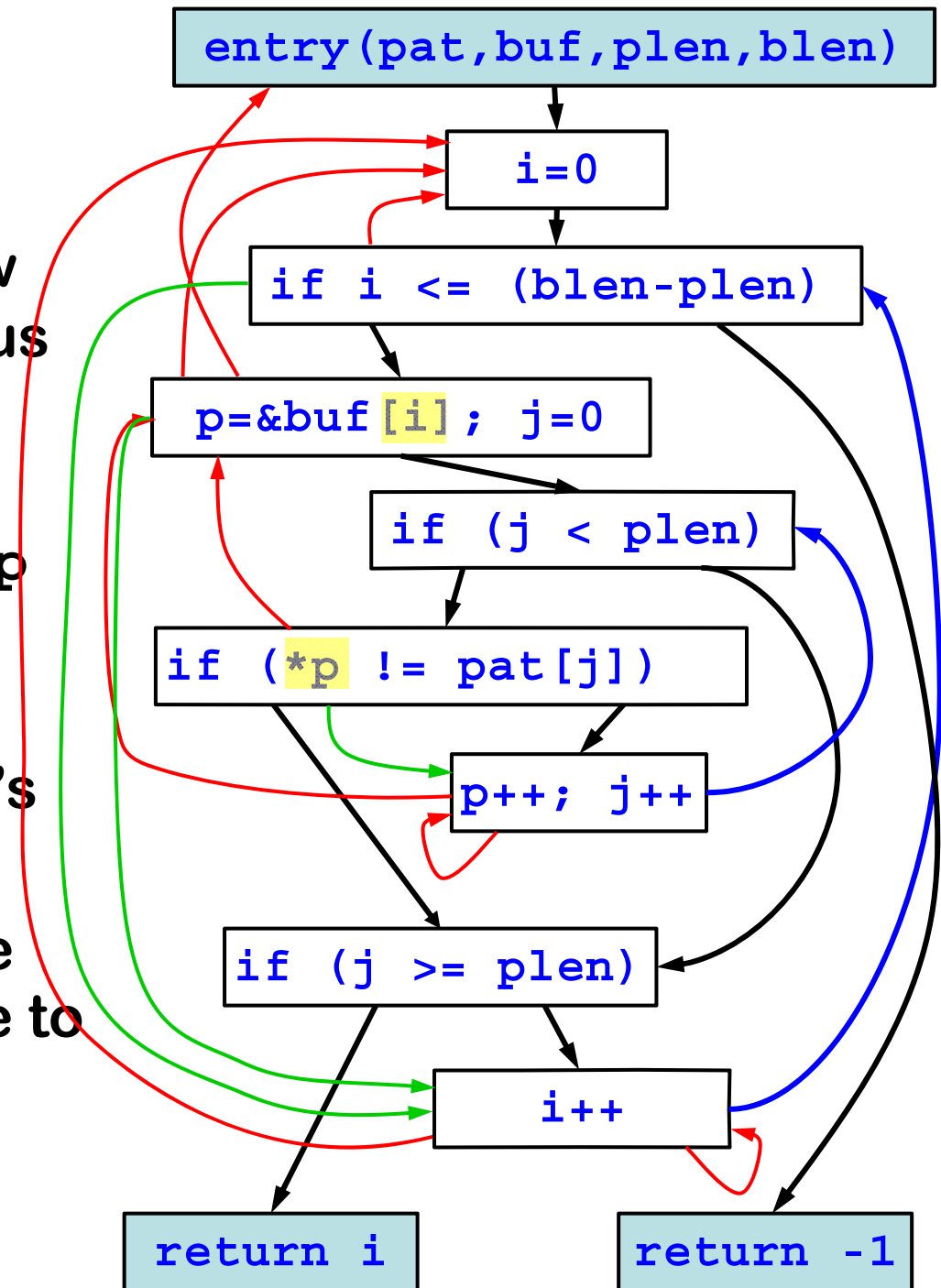




# Data Flow Analysis

## • Pointers

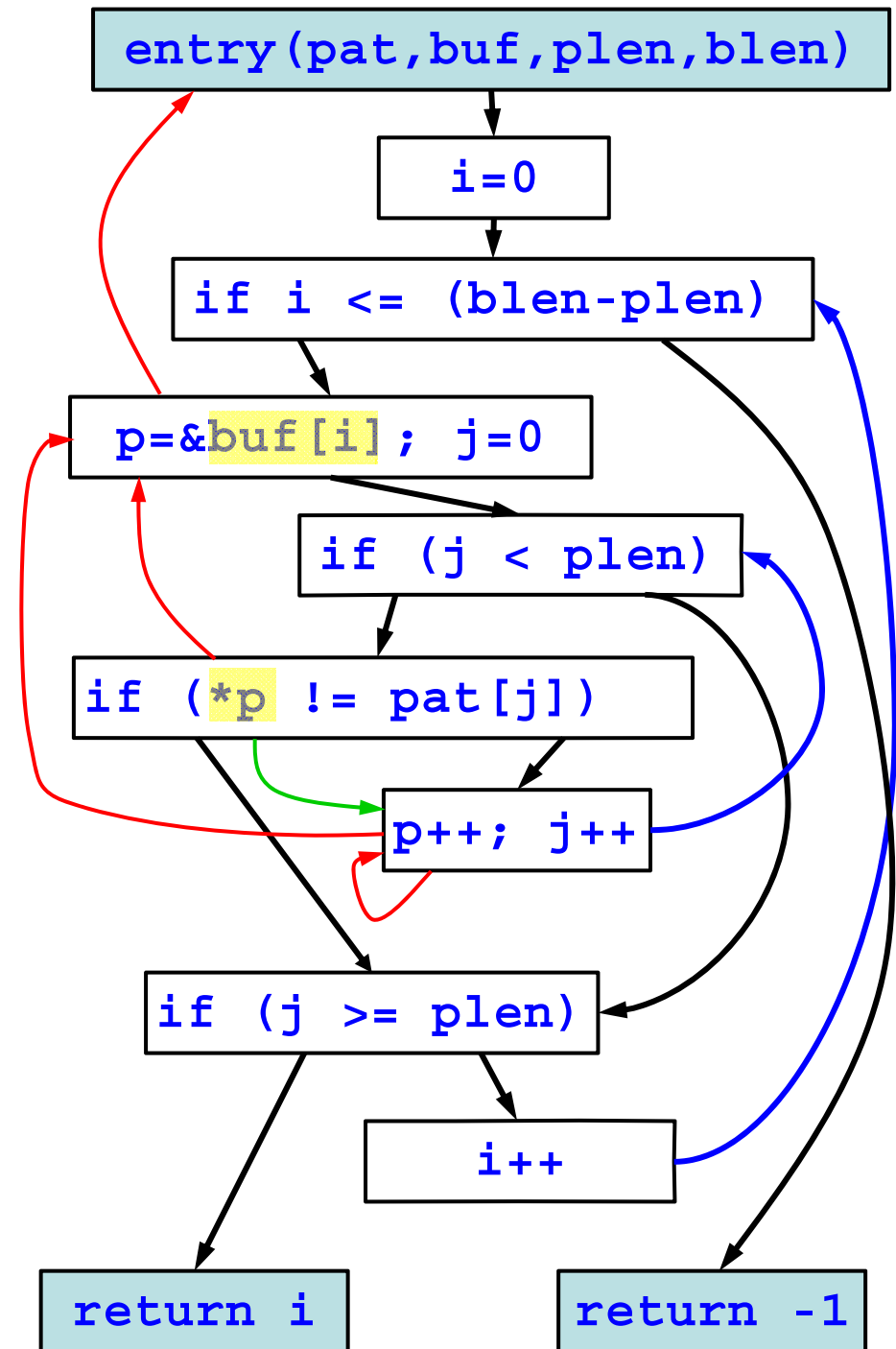
- Similar to the data flow analysis on the previous slide.
- Goal is to answer the question: where does  $p$  point? Are the references safe?
- On what variables is  $p$ 's value based?
- Of course, to calculate  $p$ 's value, we also have to know  $i$ 's value.



# Data Flow Analysis

- Aliases

- Note that there are two completely different ways to name the same memory locations.
- Understand these aliases can be important to understanding how memory is being referenced.



```

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

    int i, j;
    char *p;

    i = 0;

    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;
}

```

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

$i: [0, 0]$

$i: [0, \text{blen} - \text{plen}]$

$i: [0, \text{blen} - \text{plen}]$

$p: \text{buf}[0, \text{blen} - \text{plen}]$

$j: [0, 0]$

$j: [0, \text{plen} - 1]$

$j: [0, \text{plen} - 1]$

$p: [\text{buf}[0, \text{blen} - \text{plen} + \text{plen} - 1]$

$p: [\text{buf}[0, \text{blen} - \text{plen} + \text{plen}]$

$j: [0, \text{plen}]$

$j: [0, \text{blen} - \text{plen}]$

$i: [0, \text{blen} - \text{plen} + 1]$

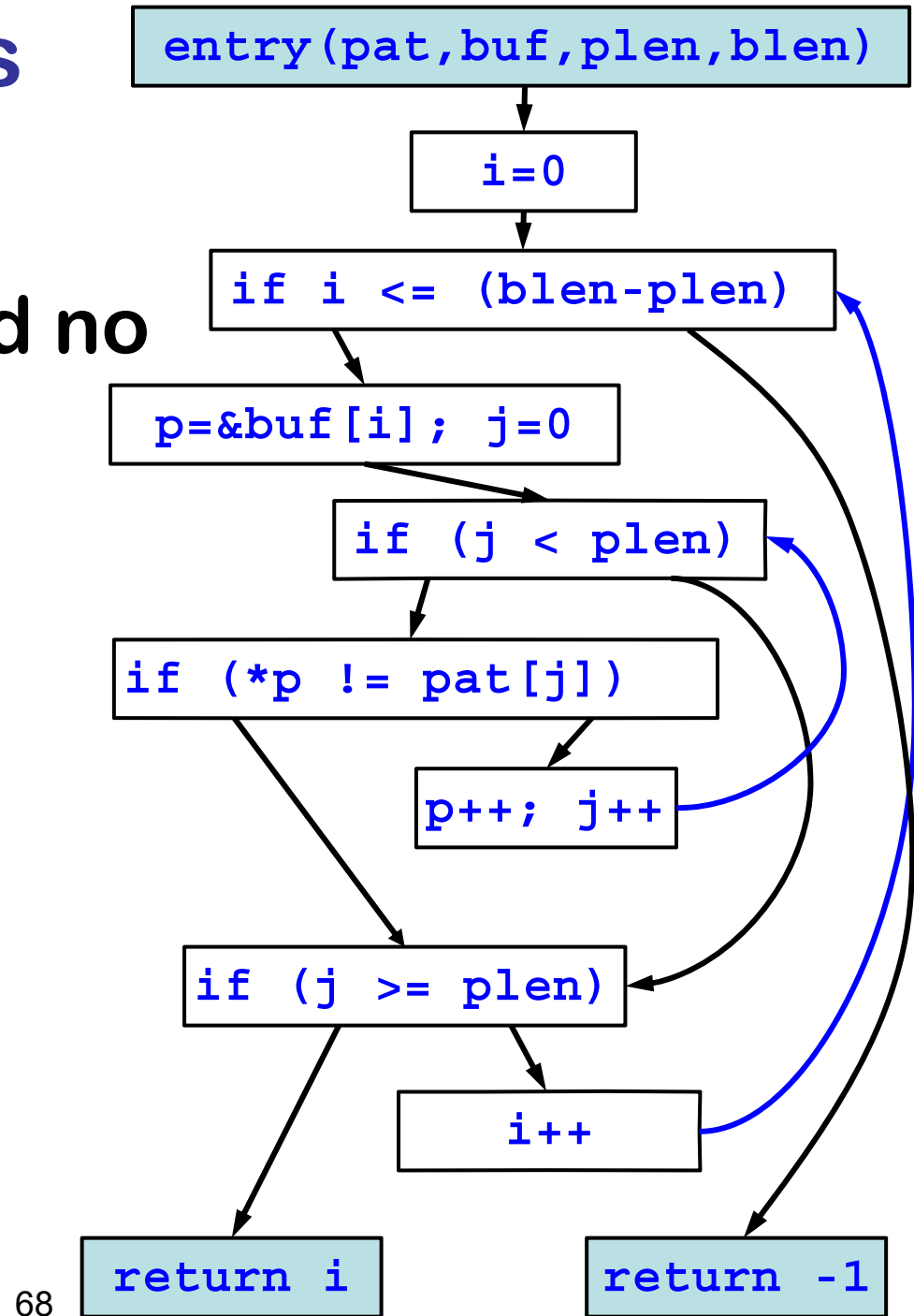


# Semantic Analysis

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.

# Common Weakness Enumeration (CWE)

- “CWE provides a unified, measurable set of software weaknesses”.
- “Allows a more effective use of software security tools”.
- 719 weaknesses in 244 categories.
- Id, description, consequences, examples, relationship, taxonomy mapping.

<http://cwe.mitre.org/>

1. What You Need to Know about How Tools Work

2. The Tools And Their Use

# Roadmap

- **Motivation**
- **Source code example**
- **Tools for Java applied to the source code**



# What and Why

- Learn about different automated tools for vulnerability assessment.
- Start with small programs with weaknesses.
- Apply different tools to the programs.
- Understand the output, and the strong and weak points of using specific tools.

# How to Describe a Weakness

## Descriptive name of weakness (CWE XX)

An intuitive summary of the weakness.

- **Attack point:** How does the attacker affect the program.
- **Impact point:** Where in the program does the bad thing actually happen.
- **Mitigation:** A version of the program that does not contain the weakness.

(CWEXX\_Long\_Detailed\_File\_Name\_Containing\_The\_Code\_yy.cpp)

# CWE 601: Open Redirect

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

# 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](http://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
```

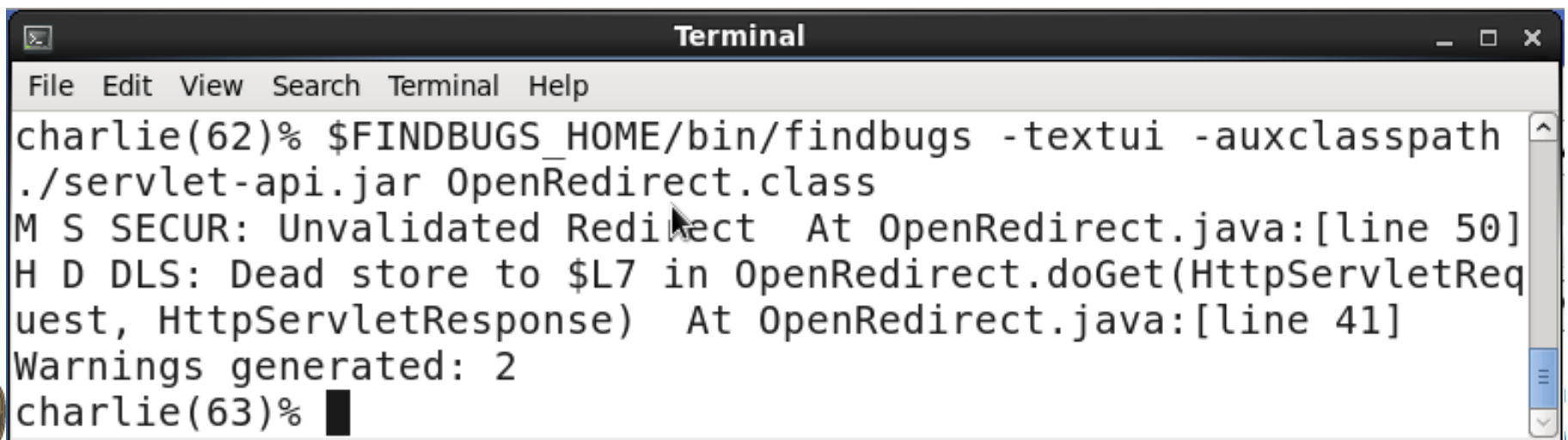


# FindBugs. Open Redirect

- FindBugs

- `$FINDBUGS_HOME/bin/findbugs -textui -auxclasspath ./servlet-api.jar OpenRedirect.class`

- **1 irrelevant warning.**
    - **1 true positive:** It detects the Open Redirect vulnerability.



```
Terminal
File Edit View Search Terminal Help
charlie(62)% $FINDBUGS_HOME/bin/findbugs -textui -auxclasspath
./servlet-api.jar OpenRedirect.class
M S SECUR: Unvalidated Redirect At OpenRedirect.java:[line 50]
H D DLS: Dead store to $L7 in OpenRedirect.doGet(HttpServletRequestReq
uest, HttpServletResponse) At OpenRedirect.java:[line 41]
Warnings generated: 2
charlie(63)%
```



# FindBugs. Open Redirect

The screenshot displays the FindBugs application interface. The top menu bar includes File, Edit, View, Navigation, Designation, and Help. On the left, there is a 'Class name filter' and a 'Filter' button. Below that, the 'Group bugs by' section has buttons for 'Category', 'Bug Kind', and 'Bug Pattern'. A tree view shows a hierarchy of bugs: Bugs (2) > Security (1) > Unvalidated Redirect (1) > Unvalidated Redirect (1) > Unvalidated Redirect (1). The selected bug is 'Unvalidated Redirect'.

The main window shows the source code for 'OpenRedirect.java'. The code is as follows:

```
30     data = cookieSources[0].getValue();
31     }
32
33     if (data != null)
34     {
35         /* This prevents \r\n (and other chars) and should prevent incidentals such
36          * as HTTP Response Splitting and HTTP Header Injection.
37          */
38         URI uri;
39         try
40         {
41             uri = new URI(data);
42         }
43         catch (URISyntaxException exceptURISyntaxException)
44         {
45             response.getWriter().write("Invalid redirect URL");
46             return;
47         }
48         /* POTENTIAL FLAW: redirect is sent verbatim; escape the string to prevent ancillary
49          // IMPORTANT: Comment the 2 following lines to see the good case working!
50         response.sendRedirect(data);
51         return;
52     }
53
```

The bug report details are shown in the bottom-left pane:

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

The bottom-right pane provides a detailed description of the bug:

**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/login?redirect=http://evil.vwebsite.com/fake/login`
2. The user is redirected to a fake login page that looks like a site they trust. (`http://evil.vwebsite.com/fake/login`)
3. The user enters his credentials.
4. The evil site steals the user's credentials and redirects him to the original website.

This attack is plausible because most users don't double check the URL after the redirection. Also, redirection to an authentication page is very common.

# Parasoft Jtest

# Jtest



- 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

The screenshot displays the Parasoft Jtest IDE interface. The main editor window shows the source code for `OpenRedirect.java`. The code includes a comment indicating a potential flaw: `/* POTENTIAL FLAW: Read data from the first cookie value */`. The code snippet is as follows:

```
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);  
    }  
}
```

The IDE's right-hand side contains several panels: **Task List** (empty), **Outline** (showing `doGet(HttpServletRequest, Ht...`), and **Problems** (showing 0 errors, 2 warnings, 0 others). The **Problems** panel lists two warnings:

- 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

The status bar at the bottom of the IDE displays the warning: `SECURITY.IBA.VPPD: 'getCookies()' is a ta...nd should be encapsulated by a validation`.



# Jtest. Open Redirect

The screenshot displays an IDE window titled "Java - OpenRedir/src/OpenRedirect.java - Parasoft Jtest". The main editor shows the following Java code:

```
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);
    }
}
```

The IDE interface includes a menu bar (File, Edit, Source, Refactor, Navigate, Search, Project, Parasoft, Run, Window, Help), a toolbar, and several panels:

- Task List:** Contains a search field and "All" and "Activate..." buttons.
- Outline:** Shows a tree view with "import declarations" and "OpenRedirect" expanded to show "doGet(HttpServletRequest, Http)".
- Problems:** Shows "0 errors, 2 warnings, 0 others".
- Warnings:** Two items are listed:
  - 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

A status bar at the bottom displays the warning: "SECURITY.IBA.VPPD: 'getValue()' is a dang...nd should be encapsulated by a validation".

# Jtest. Open Redirect

The screenshot shows the Parasoft Jtest IDE interface. The main editor displays the following Java code:

```
OpenRedirect.java
{
  /* 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 IDE's right-hand sidebar contains a Task List, a search bar, and an Outline view showing the project structure with 'OpenRedirect' and 'doGet(HttpServletRequest, Http...' visible.

The bottom panel shows the Problems view with the following content:

0 errors, 4 warnings, 0 others

Description

- Warnings (4 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.VRD: No validation check in redirect URL**
- SECURITY.IBA.VRD: No validation check in redirect URL

The status bar at the bottom of the IDE displays the warning: SECURITY.IBA.VRD: No validation check in redirect URL.

# Roadmap

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

<https://www.swampinbox.org/doc/SWAMP-User-Manual.pdf>

# Getting Started with the SWAMP

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

# Register to use the SWAMP

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

Software Assurance Marketpl x +

https://www.mir-swamp.org

MozBackup.Ink Más visitados Programmes & Initiati... Cerca Comenzar con Firefox Iniciar sesión Últimas noticias Tanca la Sessió Nueva pestaña

SWAMP About Contact Resources Policies Help Sign In

**CONTINUOUS ASSURANCE**

**SWAMP**

SOFTWARE ASSURANCE MARKETPLACE

*Do It Early. Do It Often.*

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!

**Sign Up!**

**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 full featured analysis.

# What can I do in the SWAMP?

Software Assurance Marketpl x +

https://www.mir-swamp.org/#home

MozBackup.Ink Más visitados Programmes & Initiati... Cerca Comenzar con Firefox Iniciar sesión Últimas noticias Tanca la Sessió Nueva pestaña

SWAMP About Contact Resources Policies Help elisa Sign Out

You last signed in on 04/04/2017

**CONTINUOUS ASSURANCE**

**SWAMP**  
SOFTWARE ASSURANCE MARKETPLACE

*Do It Early. Do It Often.*

- Packages**  
Upload your code and manage your software packages. (12)
- Assessments**  
Perform assessments on packages using code analysis tools. (36)
- Results**  
View the status and results of completed assessments. (36)
- Runs**  
View assessments scheduled to run at regular intervals. (0)
- Projects**  
Create projects to share results with other users. (2)
- Events**  
View events associated with your projects & account. (9)

Copyright © 2012-2017 Software Assurance Marketplace, Morgridge Institute for Research

# Create a Project

The screenshot shows the SWAMP website interface. The browser address bar displays <https://www.mir-swamp.org/#projects>. The navigation menu includes 'SWAMP', 'About', 'Contact', 'Resources', 'Policies', and 'Help'. The user 'elisa' is logged in, with a 'Sign Out' button. The main content area is titled 'Projects' and includes a breadcrumb 'Home / Projects'. A paragraph explains that projects are used to share assessment results. A red circle highlights the '+ Add New Project' button. Below this, there is a section 'Projects I Own' with a table listing two projects: 'Tutorial Java' and 'Tools tutorial'. The 'Projects I Joined' section shows 'No projects.'

Project	Description	Date Added
Tutorial Java	Tutorial Java	11/13/2014 15:59
Tools tutorial	Tools tutorial	10/09/2014 16:33



# Create a Project

The screenshot shows a web browser window with the URL <https://www.mir-swamp.org/#projects/add>. The page title is "Add New Project" and the breadcrumb trail is "Home / Projects / + Add New Project". The form contains the following fields:

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

A red circle highlights the "Save Project" button at the bottom left of the form. A note at the bottom right states "\*Fields are required".



# Packages

SWAMP [About](#) [Contact](#) [Resources](#) [Policies](#) [Help](#) elisa [Sign Out](#)

## Packages

Home / Packages

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

Filters: any project </> any type all items

[+ Add New Package](#)

Package	Description	Type	Versions
buffer overflow		C/C++	• 1.0
command injection		C/C++	• 1.0
hard coded password		C/C++	• 1.0
info exposure		C/C++	• 1.0

# Upload your Software Package

The screenshot shows a web browser window displaying the 'Add New Package' page on the SWAMP website. The browser's address bar shows the URL <https://www.mir-swamp.org/#packages/add>. The page header includes the SWAMP logo and navigation links: About, Contact, Resources, Policies, and Help. A user named 'elisa' is logged in, with a 'Sign Out' button. The main content area is titled 'Add New Package' and features a sidebar with icons for various actions. The form includes the following fields and options:

- Name \***: OpenRedirect2017
- Description**: (Empty text area)
- File source**:
  - Local file system**: The package source code is located on your local hard drive.
  - Remote Git repository**: The package source code is located on a remote Git server.
- File \***: 10-b-OpenRedirect.tar (with a file selection icon)
- Version \***: 1.0
- Version notes**: (Empty text area)

On the right side of the form, there are two expandable sections: 'PACKAGE INFO' and 'PACKAGE VERSION INFO'.

# Upload your Software Package

The screenshot shows a web browser window with the URL `https://www.mir-swamp.org/#packages/add`. The page title is "Software Assurance Marketpl". The browser's address bar shows the URL and a search bar with the text "Buscar". The page has a navigation menu with links for "About", "Contact", "Resources", "Policies", and "Help". The user is logged in as "elica" and has a "Sign Out" button.

The main content area has tabs for "Details", "Source", "Build", and "Sharing". A notice box states: "Notice: This appears to be a Java bytecode package. You can set the language type if this is not correct." Below this, there are form fields for "Package path" (containing "10-open-redirect/") and "Language" (set to "Java"). A "Show File Types" button is also present.

Under the "Java type" section, there are three radio button options:

- Java source  
The package contains uncompiled Java code in its original source code format (.java files).
- Java bytecode  
The package contains Java code which has been compiled (.class, .jar, or .apk files).
- Android APK  
The package contains compiled Java code for the Android platform.

Below this, there is a section for "Java version" which is currently collapsed.

# Upload your Software Package

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

Software Assurance Marketpl x +

https://www.mir-swamp.org/#packages/add 80% Buscar

MozBackup.Ink Más visitados Programmes & Initiati... Cerca Comenzar con Firefox Iniciar sesión Últimas noticias Tanca la Sessió Nueva pestaña

SWAMP About Contact Resources Policies Help elisa Sign Out

## Add New Package

Home / Packages / Add New Package

Details Source Build Sharing

The following parameters are used to configure the build script which is used to build the package.

Class path \* . Add JAVA BYTECODE INFO

Advanced settings

\*Fields are required

Package dependencies

No dependencies have been defined.

Add New Dependency

\*Fields are required

# Upload your Software Package

The screenshot shows a web browser window displaying the 'Add New Package' page on the SWAMP website. The browser's address bar shows the URL <https://www.mir-swamp.org/#packages/add>. The page features a navigation menu with links for 'About', 'Contact', 'Resources', 'Policies', and 'Help'. A user profile for 'elisa' is visible in the top right corner with a 'Sign Out' button. The main content area is titled 'Add New Package' and includes a breadcrumb trail: 'Home / Packages / Add New Package'. Below the title, there are tabs for 'Details', 'Source', 'Build', and 'Sharing'. A message states: 'This package version is shared with members of the following projects:'. A table lists the shared projects:

Project	Description
<input checked="" type="checkbox"/> Tutorial Java	Tutorial Java
<input type="checkbox"/> Tools tutorial	Tools tutorial

At the bottom of the page, there are three buttons: 'Save New Package' (circled in red), 'Prev', and 'Cancel'. The footer contains the copyright notice: 'Copyright © 2012-2017 Software Assurance Marketplace, Morigridge Institute for Research' and the SWAMP logo.

# Upload your Software Package

The screenshot shows a web browser window displaying the SWAMP (Software Assurance Marketplace) interface. The browser's address bar shows the URL: <https://www.mir-swamp.org/#packages/f5a26d3a-8df7-49e4-8f3f-3edc>. The page title is "OpenRedirect2017 Package". The user is logged in as "elisa" and has a "Sign Out" button. The page features a sidebar with various icons, the top of which is circled in red. The main content area displays the package details for "OpenRedirect2017", including its name, language (Java 7 Source Code), creation date (05/02/2017), last modified date (05/02/2017), external URL (none), and description (none). There are also buttons for "Assessments 2", "Results 2", and "Runs 0". At the bottom, there is a "Versions" section with an "Add New Version" button.

Name	OpenRedirect2017	<a href="#">Edit</a>
Language	Java 7 Source Code	
Creation date	05/02/2017	
Last modified date	05/02/2017	
External URL	none	
Description	none	

# Run your Assessments

Software Assurance Marketpl x +

https://www.mir-swamp.org/#assessments

MozBackup.Ink Más visitados Programmes & Initiati... Cerca Comenzar con Firefox Iniciar sesión Últimas noticias Tanca la Sessió Nueva pestaña

SWAMP About Contact Resources Policies Help elicia Sign Out

## Assessments

Home / Assessments

Results 36 Runs 0

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.

Filters any project any package any tool any platform all items x

Run Assessments + Run New Assessment

Package	Tool	Platform	Results
<input type="checkbox"/> <input type="checkbox"/> buffer overflow latest	Parasoft C/C++test latest	Red Hat Enterprise Linux 6 64-bit latest	1
<input type="checkbox"/>	cppcheck latest		1
<input type="checkbox"/>	Clang Static Analyzer latest		1
<input type="checkbox"/> <input type="checkbox"/> command injection latest	Parasoft C/C++test latest	Red Hat Enterprise Linux 6 64-bit latest	1
<input type="checkbox"/>	cppcheck latest		1

# Run your Assessments

The screenshot shows a web browser window with the URL <https://www.mir-swamp.org/#assessments/run>. The page title is "Run New Assessment". The user is logged in as "elisa" and has a "Sign Out" button. The main content area contains two sections for configuration:

- Package:** "Select a package to assess:" with a dropdown menu showing "OpenRedirect2017". "Select a version:" with a dropdown menu showing "Latest".
- Tool:** "Select a tool to use:" with a dropdown menu showing "Findbugs". "Select a version:" with a dropdown menu showing "Latest".

At the bottom, there are three buttons: "Save and Run" (highlighted with a red circle), "Save", and "Cancel". A left sidebar contains several navigation icons.



# My Assessments

SWAMP [About](#) [Contact](#) [Resources](#) [Policies](#) [Help](#) elisa [Sign Out](#)

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.

**Filters** any project any package any tool any platform any date 50 items

**Viewer**  Code Dx  Threadfix  Native

**Notice:** Click the view assessment results button to view the selected results using the selected viewer.

Auto refresh

[View Assessment Results](#)

Package	Tool	Platform	Date	Status	Results
OpenRedirect2017 1.0	Parasoft Jtest 9.6.0	Ubuntu Linux 16.04 LTS 64-bit Xenial Xerus	05/02/2017 22:00	finished	5
OpenRedirect2017 1.0	Findbugs 3.0.1	Ubuntu Linux 16.04 LTS 64-bit Xenial Xerus	05/02/2017 21:08	finished	3

# View your Results. FindBugs - Native

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

Software Assurance Marketpl x OpenRedirect2017 > Analysis x Weakness 34 Details | Code D x findbugs3.0.1 Report x +

https://www.mir-swamp.org/#results/756b3185-2f6b-11e7-9165-001a4a814413/view

findbugs v3.0.1 Report

### Summary

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

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

# View your Results. FindBugs - CodeDX

The screenshot shows the CodeDX FindBugs web interface. The browser address bar displays the URL: `https://swa-csaweb-pd-01.mir-swamp.org/proxy-B69B51BA-2F6B-11E7-88...`. The page title is "OpenRedirect2017 > Analysis Run 4". The interface includes a sidebar with "Filters" and "Weakness Flow" sections. The "Filters" section shows "Weakness count" as 1 / 1, "Tool" as FindBugs (100%) and Security (100%), "Severity" as Medium (100%), and "Status" as Unresolved (100%). The main content area displays "Bulk Operations" for the 1 matching weakness, with buttons for "Change status..." and "Generate report". Below this is a "Weaknesses" table with columns for Tool, Rule, CWE, Codebase Location, and Status. The table contains one entry: FindBugs... Non-validated redirect (CWE 601) at OpenRedirect.java:50, with a status of Unresolved. The number "34" is circled in red in the table's left column. The interface also shows "Displaying all weaknesses" and "Displaying 1 to 1 of 1 Weakness".

# View your Results. FindBugs - CodeDX

The screenshot shows a web browser displaying the FindBugs interface. The browser's address bar shows a URL starting with 'https://swa-csaweb-pd-01.mir-swamp.org/proxy-...'. The page title is 'Projects' and the version is 'v1.8.3 SW 11/12/2015'. The main content area displays a vulnerability report for 'OpenRedirect2017 > Analysis Run 2 > Weakness 34'. The vulnerability is identified as 'Non-validated redirect' detected by 'FindBugs' with the type '[UNVALIDATED\_REDIRECT]'. It was first seen on '5/2/2017', has '1 weakness in this file' and '1 similar weakness in this analysis run', and is of 'Medium severity'. The CWE is '601 - URL Redirection to Untrusted Site ('Open Redirect')' with links to 'CWEVis' and 'MITRE'. A 'jump to weakness' link is also present.

**Status**

New

**Activity Stream**

Post Clear Write comments with Markdown

Status set to **New** during Analysis Run 2 by **admin**  
17 minutes ago

**Description**

The following redirection could be use by an attacker to redirect users to a fishing website.

Unvalidated requests are a vulnerability that facilitate phishing attacks.

**Scenario**

- A user is requested to visit the malicious url:  
`website.com/login?redirect=http://evil.vwebsite.com/fake/login`
- The user is redirected to a fake login. ( `evil.vwebsite.com/fake/login` )
- The user enters his credentials.
- He is redirected to the original website.

This attack is plausible because most users don't double check the URL after redirection. In this example, redirection after authentication is common.

**Counter measures**

- White list URLs (if possible).
- Validate that the beginning of the URL is part of white list.

# View your Results. FindBugs - CodeDX

OpenRedirect2017 > Analysis Run 2 > Weakness 34 **Non-validated redirect** detected by **FindBugs** [UNVALIDATED\_REDIRECT] [jump to top](#)

First seen on **5/2/2017** 1 weakness in this file 1 similar weakness in this analysis run **Medium** severity

CWE **601** - URL Redirection to Untrusted Site ('Open Redirect') [ [CWEVis](#) | [MITRE](#) ] [URL is part of white list](#) [jump to weakness](#)

### Status

New

### Activity Stream

[Write comments with Markdown](#)

Status set to **New** during Analysis Run 2 by **admin**  
19 minutes ago

### Source Code

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

```
1 import javax.servlet.*;
2 import javax.servlet.http.*;
3 import java.io.*;
4 import java.net.URI;
5 import java.net.URISyntaxException;
6
7 public class OpenRedirect extends HttpServlet {
8
9     public void doGet(HttpServletRequest request,
10                       HttpServletResponse response)
11         throws ServletException, IOException {
12
13 response.setContentType("text/html");
14     PrintWriter returnHTML = response.getWriter();
15
16     returnHTML.println("<html><head><title>");
17 returnHTML.println("OpenRedirect");
18 returnHTML.println("</title></head><body>");
19 returnHTML.println("<h2>Elisa: Bad case</h2>");
20
21     // bad
22
23     String data;
```



# View your Results. FindBugs - CodeDX

OpenRedirect2017 > Analysis Run 2 > Weakness 34 **Non-validated redirect** detected by **FindBugs** [UNVALIDATED\_REDIRECT] [jump to top](#)

First seen on **5/2/2017** 1 weakness in this file 1 similar weakness in this analysis run **Medium** severity [View Details](#)

CWE **601** - URL Redirection to Untrusted Site ('Open Redirect') [ [CWEVis](#) | [MITRE](#) ] [jump to weakness](#)

### Status

New [i](#)

### Activity Stream

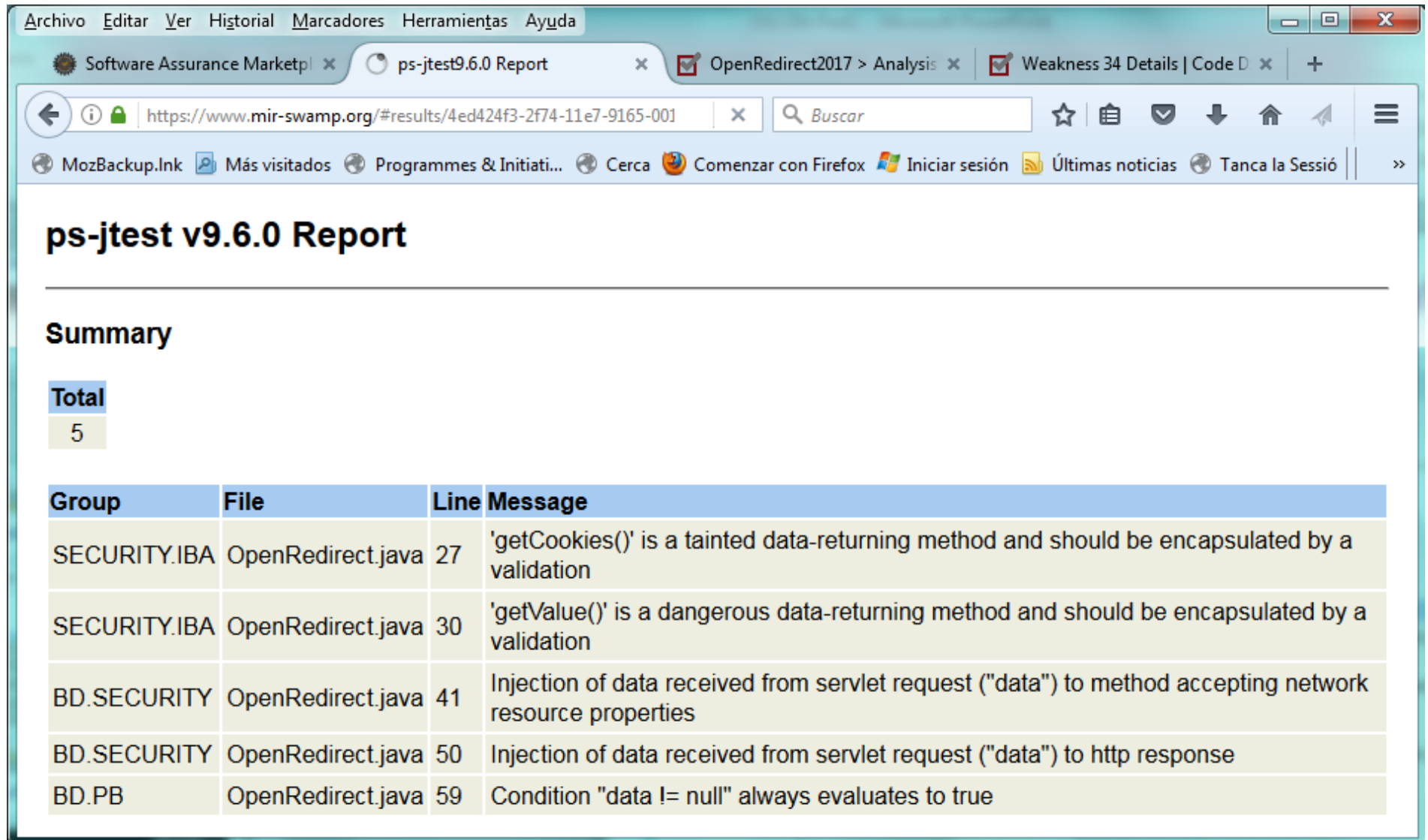
[Post](#) [Clear](#) Write comments with Markdown

Status set to **New** during Analysis Run 2 by **admin**  
21 minutes ago

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

```
25 data = ""; /* initialize data in case there are no cookies */
26 /* Read data from cookies */
27 Cookie cookieSources[] = request.getCookies();
28 if (cookieSources != null) {
29     /* POTENTIAL FLAW: Read data from the first cookie value */
30     data = cookieSources[0].getValue();
31 }
32
33 if (data != null)
34 {
35     /* This prevents \r\n (and other chars) and should prevent incidentals such
36     * as HTTP Response Splitting and HTTP Header Injection.
37     */
38     URI uri;
39     try
40     {
41         uri = new URI(data);
42     }
43     catch (URISyntaxException exceptURISyntax)
44     {
45         response.getWriter().write("Invalid redirect URL");
46         return;
47     }
48     /* POTENTIAL FLAW: redirect is sent verbatim; escape the string to prevent
49     ancillary issues like XSS, Response splitting etc */
50     // IMPORTANT: Comment the following lines to see the good case working!
51     response.sendRedirect(data);
52 }
```

# View your Results. JTest - Native



**ps-jttest v9.6.0 Report**

---

**Summary**

**Total**  
5

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

# View your Results. JTest - CodeDX

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

Software Assurance Marketpl x OpenRedirect2017 > Analysis x OpenRedirect2017 > Analysis x Weakness 34 Details | Code D x +

https://swa-csaweb-pd-01.mir-swamp.org/proxy-B69B51BA-2F6B-1

MozBackup.Ink Más visitados Programmes & Initiati... Cerca Comenzar con Firefox Iniciar sesión Últimas noticias Tanca la Sessió

Projects v1.8.3 SW 11/12/2015 CodeDX

OpenRedirect2017 » Analysis Run 3 Created on 5/2/2017 Uploaded on 5/2/2017 5 total weaknesses View

Weakness Flow

**Filters**

Weakness count 5 / 5

**Tool**

- Jtest (100%)
- BugDetective (License Required) (60%)
- Security (40%)

**Severity**

- High (100%)

**Codebase Location**

**Tool Overlaps**

**CWE**

**Status**

- New (100%)

Displaying all weaknesses

**Bulk Operations** for the 5 matching weaknesses

**Weaknesses**

Id	Tool	Rule	CWE	Codebase Location	Status
39	Jtest	Avoid conditions tha...	-	OpenRedirect.java:59	New
38	Jtest	Protect against HTT...	79	OpenRedirect.java:50	New
37	Jtest	Protect against net...	601	OpenRedirect.java:41	New
36	Jtest	Encapsulate all dan...	79	OpenRedirect.java:30	New
35	Jtest	Encapsulate all dan...	79	OpenRedirect.java:27	New

Show 25 ▲ Displaying 1 to 5 of 5 Weaknesses



# View your Results. JTest - CodeDX

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

Software Assurance Ma x OpenRedirect2017 > Ar x Weakness 38 Details | C x OpenRedirect2017 > Ar x Weakness 34 Details | C x +

https://swa-csaweb-pd-01.mir-swamp.org/proxy-B69B51BA-2F6B-: Buscar

MozBackup.Ink Más visitados Programmes & Initiati... Cerca Comenzar con Firefox Iniciar sesión Últimas noticias Tanca la Sessió

Projects v1.8.3 SW 11/12/2015 CodeDX

## OpenRedirect2017 > Analysis Run 3 > Weakness 38

**Protect against HTTP response splitting** detected by **Jtest** [BD.SECURITY.TDRESP]

First seen on **5/2/2017** 5 weaknesses in this file 1 similar weakness in this analysis run **High** severity

**CWE 79 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')** [CWEVis | MITRE] [jump to weakness](#)

Status	Description
New	Injection of data received from servlet request ("data") to http response

### Activity Stream

Post Clear Write comments with Markdown

Status set to **New** during Analysis Run 3 by **admin** 4 minutes ago

This rule detects cases of probable HTTP response splitting vulnerabilities.

This rule triggers when tainted data is passed to the following methods:

- javax.servlet.http.HttpServletResponse
- \* void sendRedirect(...) methods
- \* void addCookie(...) methods
- \* void addIntHeader(...) methods
- \* void addDateHeader(...) methods
- \* void setHeader(...) methods

# View your Results. JTest - CodeDX

OpenRedirect2017 > Analysis Run 3 > Weakness 38

Protect against HTTP response splitting detected by Jtest [BD.SECURITY.TDRESP]  
First seen on 5/2/2017 5 weaknesses in this file 1 similar weakness in this analysis run High severity  
CWE 79 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') [CWEVis | MITRE]

Status: New

Activity Stream: Status set to New during Analysis Run 3 by admin 7 minutes ago

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

```
25 data = ""; /* initialize data in case there are no cookies */
26 /* Read data from cookies */
27 Cookie cookieSources[] = request.getCookies();
28 if (cookieSources != null) {
29     /* POTENTIAL FLAW: Read data from the first cookie value */
30     data = cookieSources[0].getValue();
31 }
32
33 if (data != null)
34 {
35     /* This prevents \r\n (and other chars) and should prevent
36     incidentals such
37     * as HTTP Response Splitting and HTTP Header Injection.
38     */
39     URI uri;
40     try
41     {
42         uri = new URI(data);
43     }
44     catch (URISyntaxException exceptURISyntax)
45     {
46         response.getWriter().write("Invalid redirect URL");
47         return;
48     }
49     /* POTENTIAL FLAW: redirect is sent verbatim; escape the string to
50     prevent ancillary issues like XSS, Response splitting etc */
51     // IMPORTANT: Comment on 2 following lines to see the good case
52     // working!
53     response.sendRedirect(data);
```

# Questions?

**Elisa Heymann**

**Barton P. Miller**

[Elisa.Heymann@uab.es](mailto:Elisa.Heymann@uab.es)

[bart@cs.wisc.edu](mailto:bart@cs.wisc.edu)

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



# Java Hands-on Exercise

**Barton P. Miller**

Computer Sciences Department  
University of Wisconsin

[bart@cs.wisc.edu](mailto:bart@cs.wisc.edu)

**Elisa Heymann**

Computer Sciences Department  
University of Wisconsin  
Universitat Autònoma de Barcelona

[elisa@cs.wisc.edu](mailto:elisa@cs.wisc.edu)



August 2017

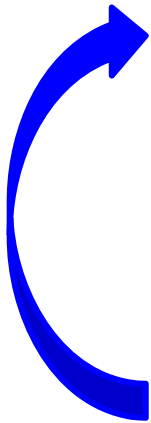


# Goals of the Assignment

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

# Steps we'll go through

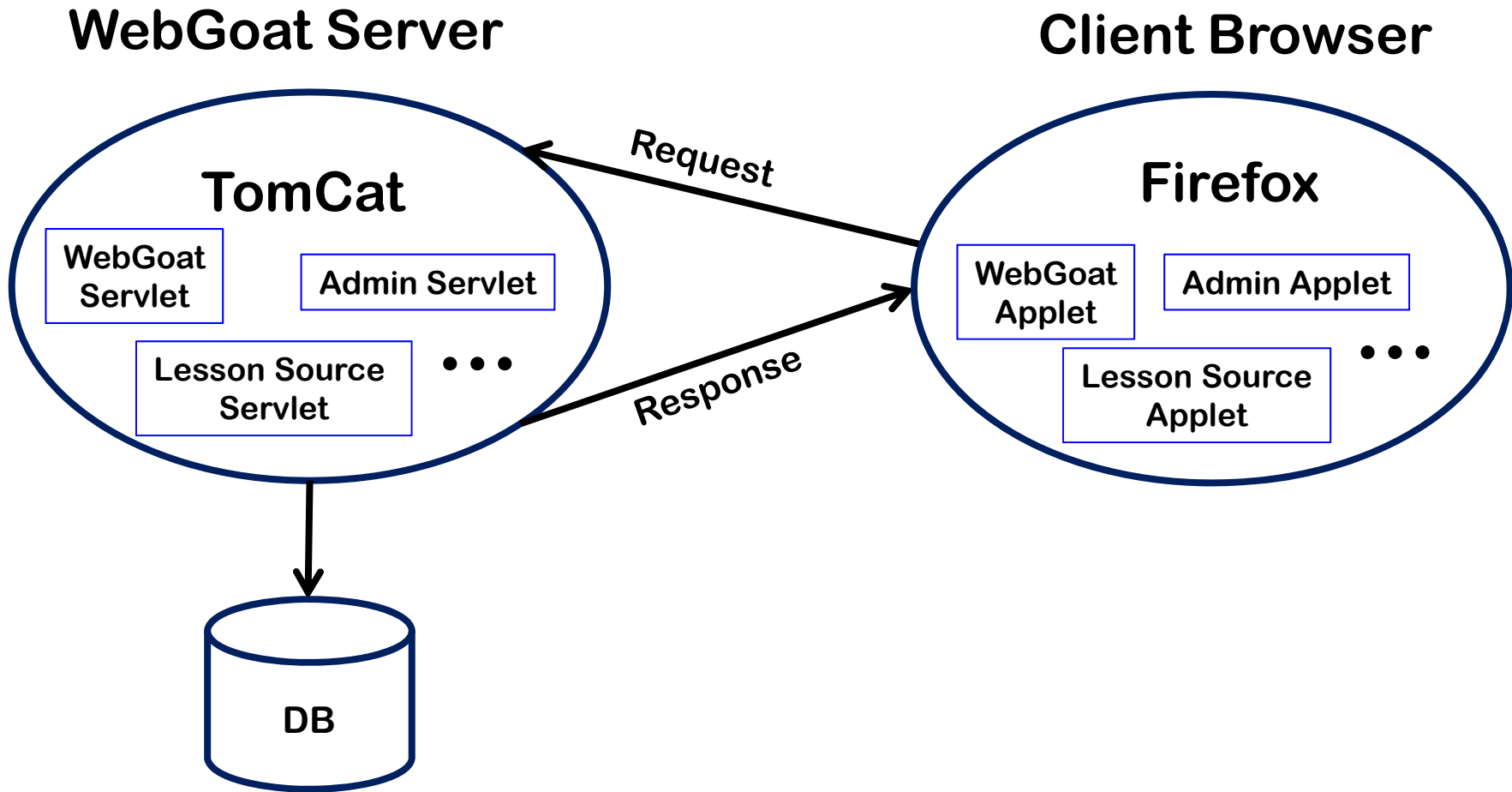
1. Introduction to a sample application and use it.
2. Upload the software package to the SWAMP.
3. Run assessments with the SWAMP tools.
4. Understand the output.
5. Modify the source code to fix a problem.



# WebGoat

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

# WebGoat Basic Architecture





# Run WebGoat

**On your console run the WebGoat server:**

```
cd ~/WebGoat/WebGoat-7.1
```

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

(you can run the [~/server.sh](#) script)

**On your web browser run the client code:**

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

Go to Injection Flaws-> String SQL injection

# Assess WebGoat in the SWAMP

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

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

**On your web browser go to:**

<https://www.mir-swamp.org/>

# Assess WebGoat in the SWAMP

When creating the MyWebGoat package in the SWAMP:

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

When seeing the results:

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

# Edit & Compile WebGoat Lessons

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

```
vi SqlStringInjection.java
```

```
cd ~/WebGoat/WebGoat-Lessons-develop
```

```
mvn clean package
```

```
cp target/plugins/*.jar ../WebGoat-7.1/webgoat-  
container/src/main/webapp/plugin_lessons/
```

(you can run the [~/compile.sh](#) script)

# Edit & Compile WebGoat Lessons

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

# WebGoat Exercise 2. Code Injection

No need to run the assessment again.

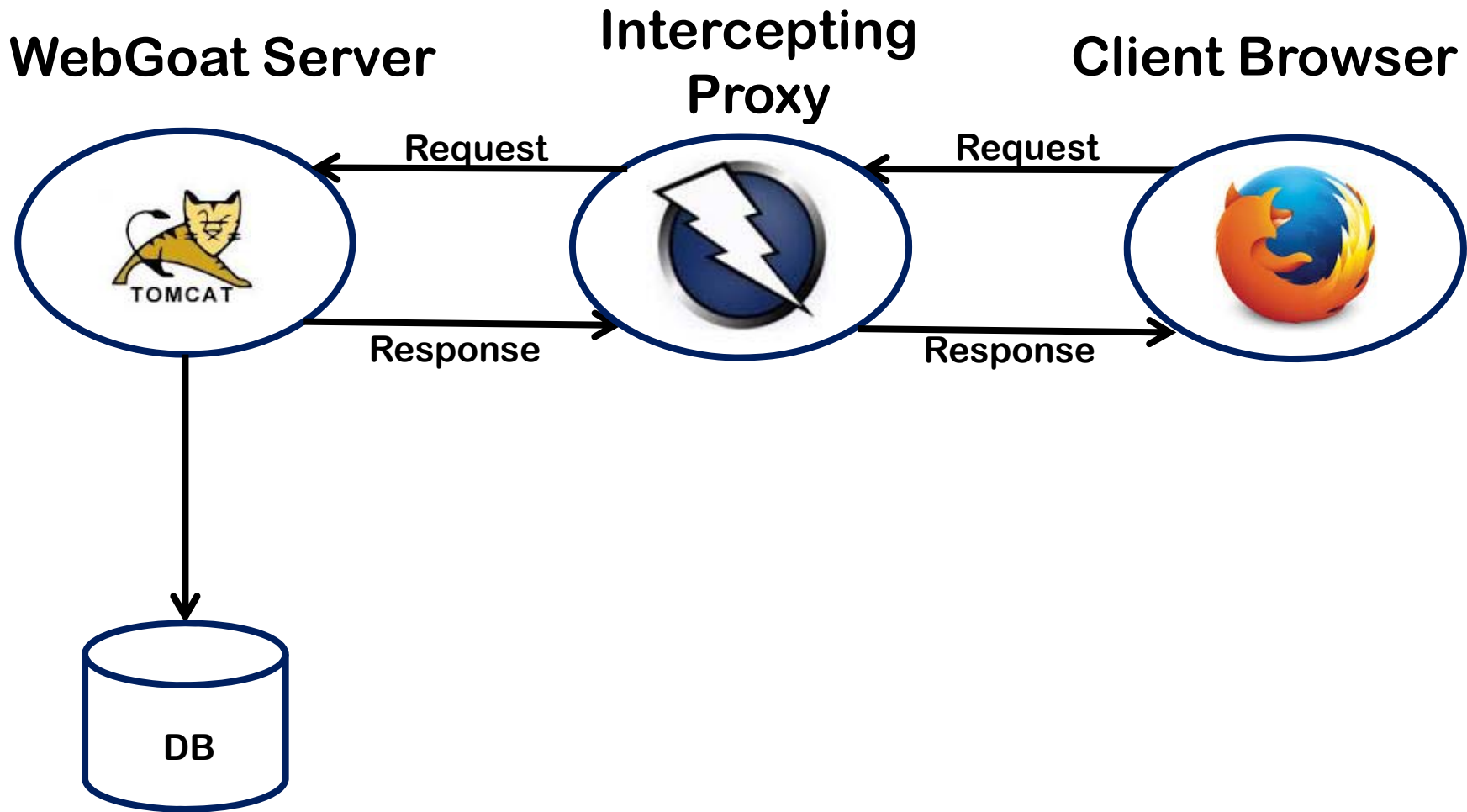
When seeing the results:

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

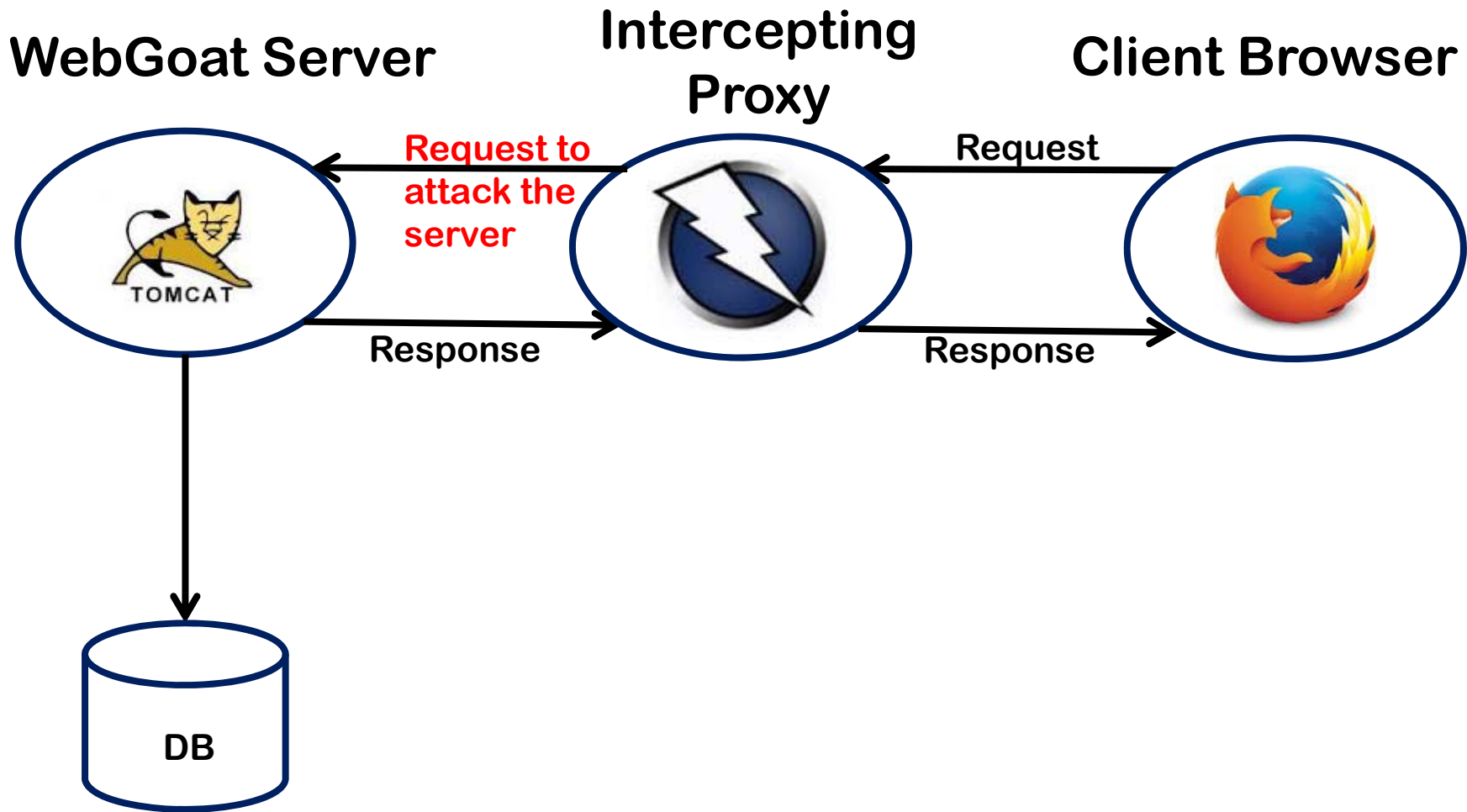
Options to attack the server:

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

# WebGoat Exercise 2. Code Injection



# WebGoat Exercise 2. Code Injection





# WebGoat Exercise 2

**On your console start OWASP ZAP, the intercepting proxy:**

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

**On your another terminal run the WebGoat server:**

```
cd ~/WebGoat/WebGoat-7.1  
java -jar webgoat-standalone/target/webgoat-standalone-7.1-exec.jar --port 8888  
(you can run the ~/server.sh script)
```

**On your web browser run the WebGoat client code:**

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

Go to Injection Flaws-> Command injection

# WebGoat Exercise 2

## Agenda:

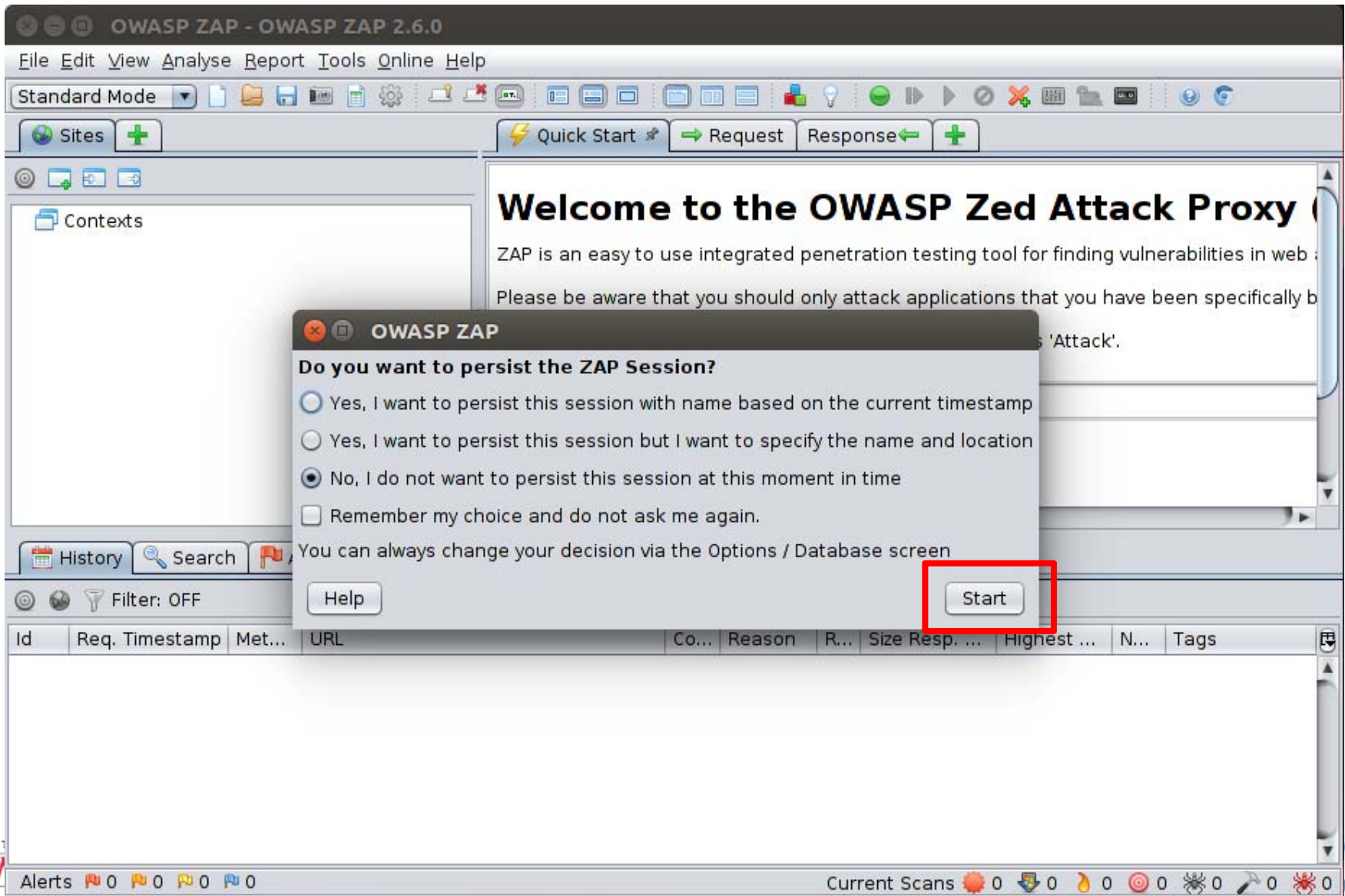
Try the lesson: Select the “lesson plan to view” and click “View”.

Modify that request to attack the server:

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

# WebGoat Exercise 2

./zap.sh

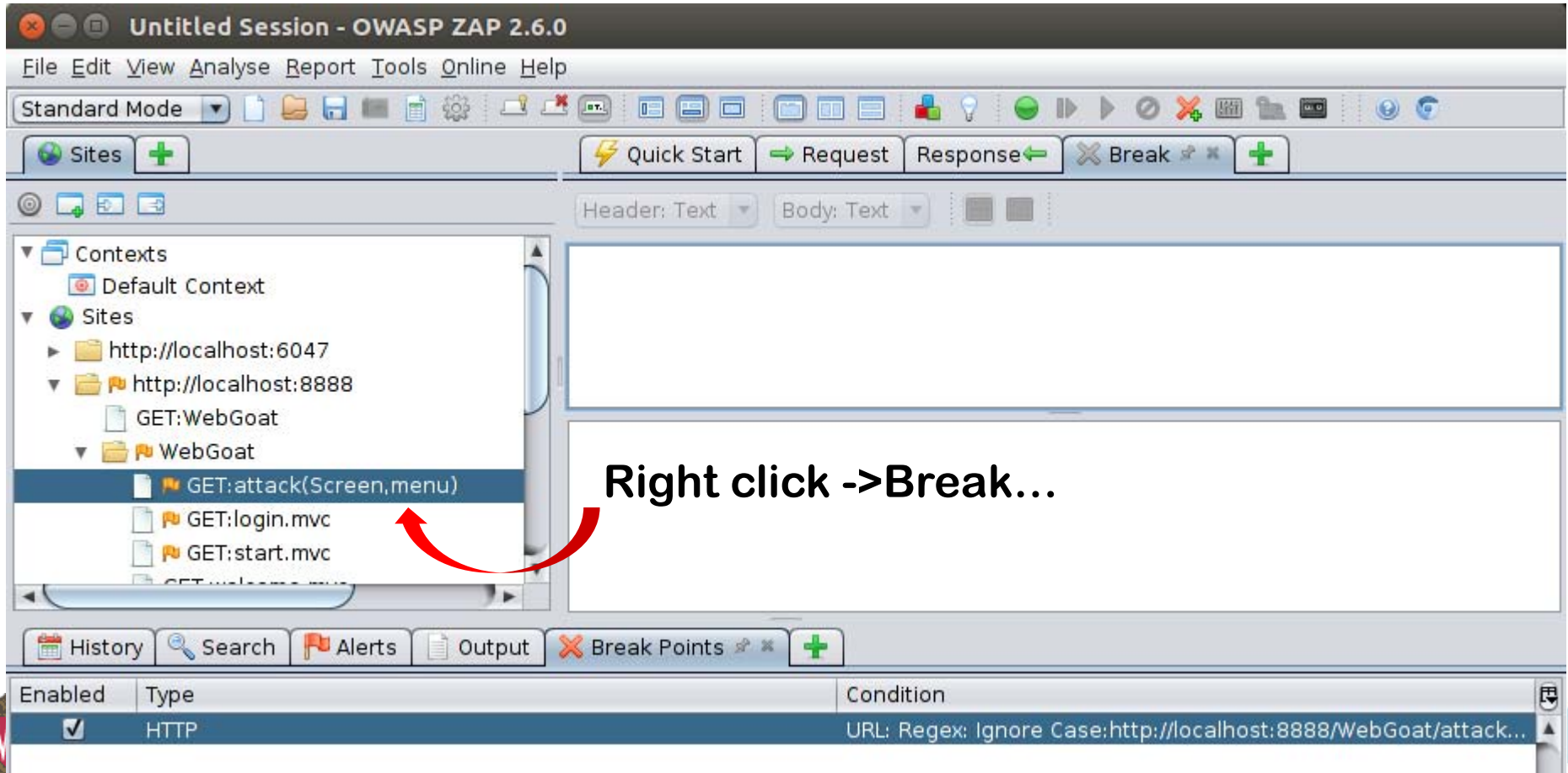


# WebGoat Exercise 2

In your browser go to localhost:8888/WebGoat

Go to Injection Flaws-> Command injection, choose a lesson plan and click on “View”.

Define a breakpoint in ZAP:



The screenshot shows the OWASP ZAP 2.6.0 interface. The left sidebar displays a tree view of the site structure under 'http://localhost:8888'. The 'WebGoat' folder is expanded, and the 'GET:attack(Screen,menu)' endpoint is selected. A red arrow points from this endpoint to the 'Break Points' tab at the bottom. The 'Break Points' tab is active, showing a table with one breakpoint defined:

Enabled	Type	Condition
<input checked="" type="checkbox"/>	HTTP	URL: Regex: Ignore Case:http://localhost:8888/WebGoat/attack...

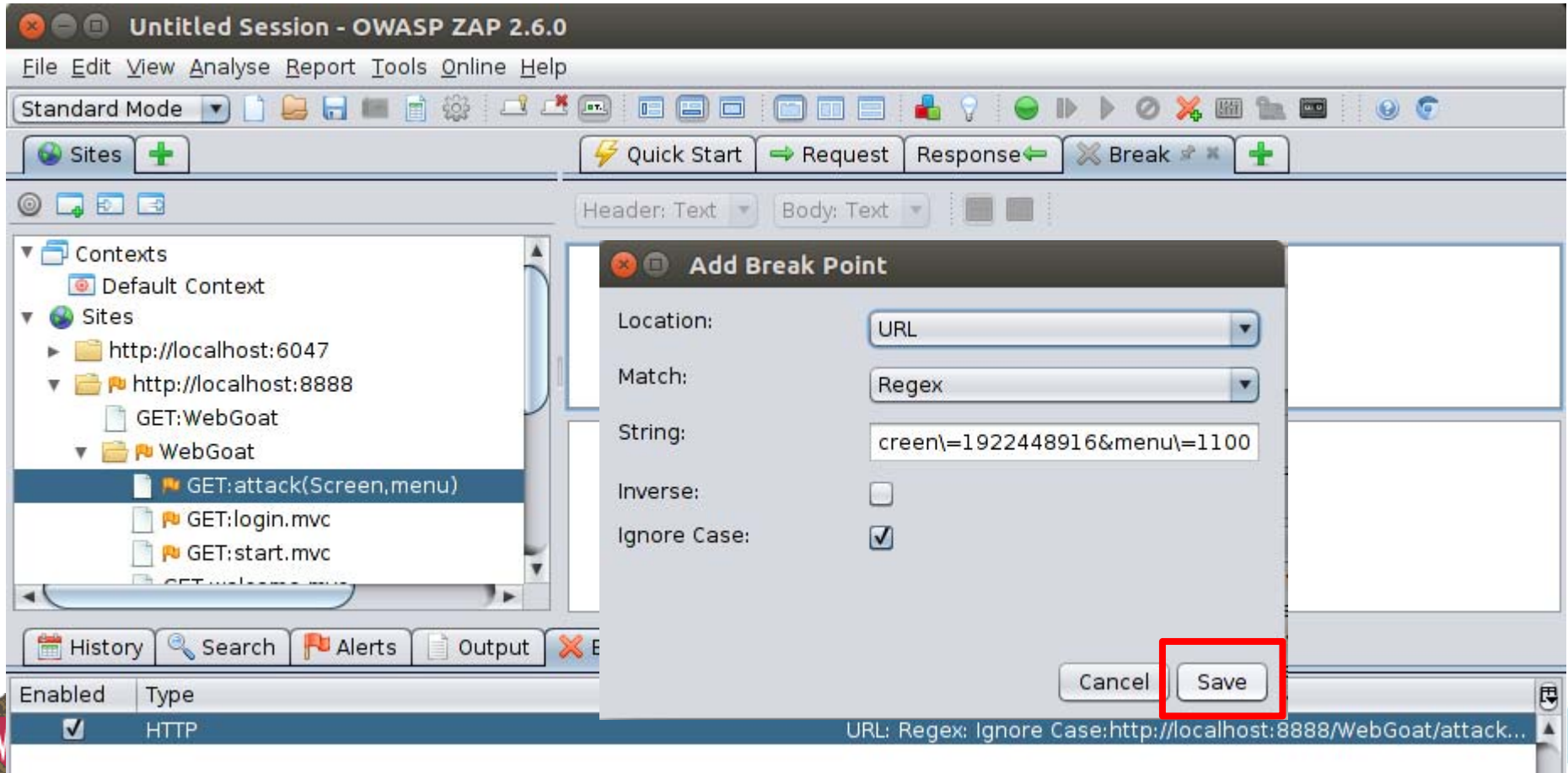
Text overlay: Right click -> Break...

# WebGoat Exercise 2

In your browser go to localhost:8888/WebGoat

Go to Injection Flaws-> Command injection, choose a lesson plan and click on “View”.

Define a breakpoint in ZAP:





# WebGoat Exercise 2

**Modify the request to inject an OS command:**

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

Click here to continue until the next break:  
One click: Response  
Click again: Will run until the next break  
Modify your request here

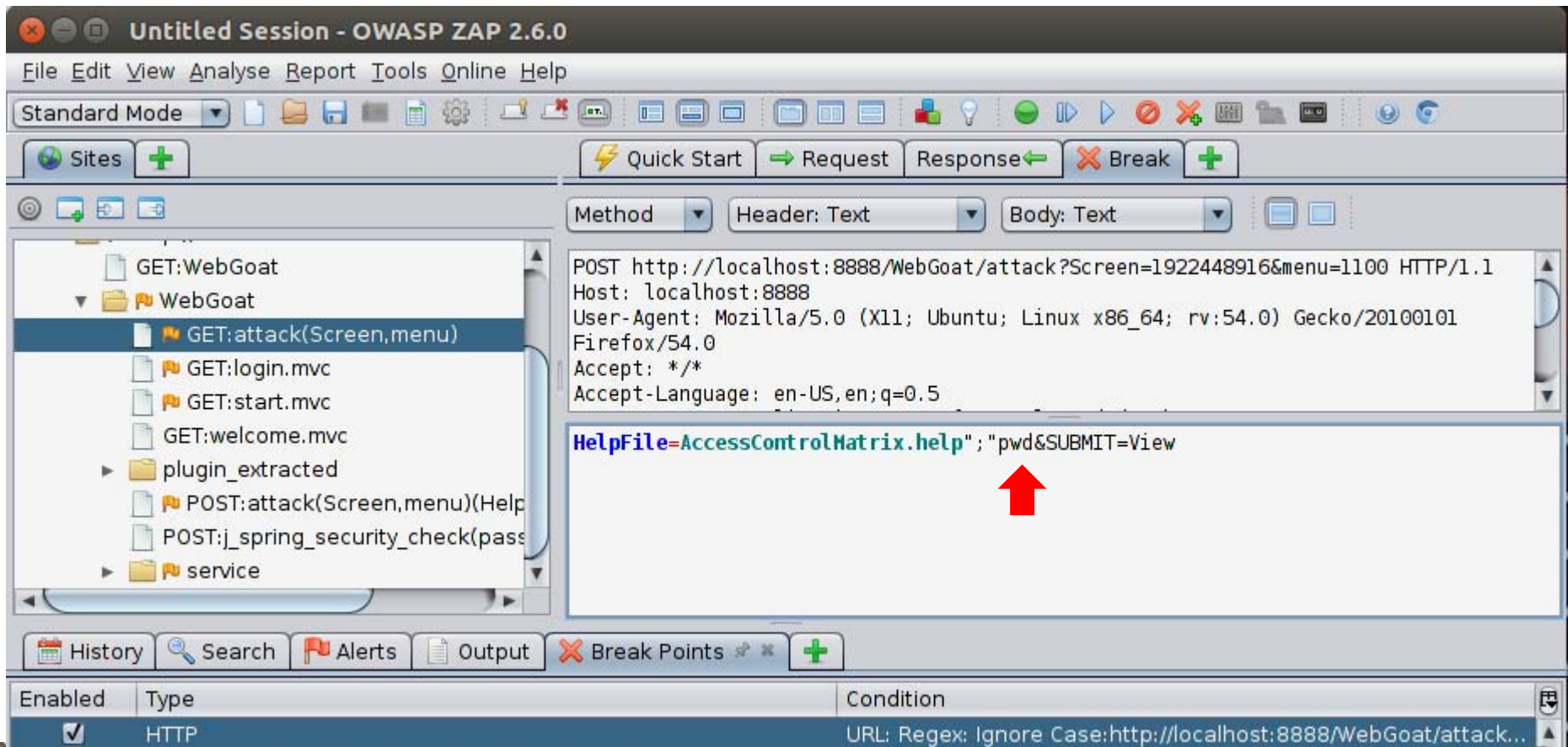
Method: POST  
Header: Text  
POST http://localhost:8888/...  
Host: localhost:8888  
User-Agent: Mozilla/5.0 (X...  
Firefox/54.0  
Accept: \*/\*  
Accept-Language: en-US, en;...  
HelpFile=AccessControlMatr...

Enabled	Type	Condition
<input checked="" type="checkbox"/>	HTTP	URL: Regex: Ignore Case:http://localhost:8888/WebGoat/attack...

# WebGoat Exercise 2

Modify the request to inject an OS command:

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



The screenshot shows the OWASP ZAP 2.6.0 interface. The 'Request' tab is active, displaying a POST request to `http://localhost:8888/WebGoat/attack?Screen=1922448916&menu=1100`. The request headers are visible, including `Host: localhost:8888`, `User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:54.0) Gecko/20100101 Firefox/54.0`, `Accept: */*`, and `Accept-Language: en-US,en;q=0.5`. The request body contains the modified payload: `HelpFile=AccessControlMatrix.help";"pwd&SUBMIT=View`. A red arrow points to the end of the payload. The left sidebar shows a tree view of the application structure, with `GET:attack(Screen,menu)` selected. The bottom status bar shows a table with columns 'Enabled', 'Type', and 'Condition', with a single entry for 'HTTP'.

Enabled	Type	Condition
<input checked="" type="checkbox"/>	HTTP	URL: Regex: Ignore Case:http://localhost:8888/WebGoat/attack...



# WebGoat Exercise 2

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

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

```
HelpFile=AccessMatrix.help";""ls -l"&SUBMIT=View
```

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

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



# WebGoat Exercise 2

The screenshot shows the WebGoat application interface. The browser address bar displays `localhost:8888/WebGoat/start.mvc#attack/1922448916/4488`. The left sidebar contains a navigation menu with categories like 'Database Backdoors', 'Blind Numeric SQL Injection', 'Denial of Service', etc. The main content area shows the current lesson plan for 'AccessControlMatrix.help'. The URL in the address bar is highlighted with a red box. The lesson plan content includes a title, a concept to teach, and a general goal. The output of the command `cat "/afs/cs.wisc.edu/u/e` is displayed, showing a directory listing of the `/afs` directory. This output is also highlighted with a red box.

localhost:8888/WebGoat/start.mvc#attack/1922448916/4488

You are currently viewing: **AccessControlMatrix.help";""cd /;ls -la"**

Select the lesson plan to view:

View

ExecResults for '['/bin/sh, -c, cat "/afs/cs.wisc.edu/u/e  
Output...

**Lesson Plan Title:** Using an Access Control Matrix

**Concept / Topic To Teach:**

In a role-based access control scheme, a role represents

**General Goal(s):**

Each user is a member of a role that is allowed to acces

```
total 124
drwxr-xr-x  25 root root  4096 Jun 30 14:20 .
drwxr-xr-x  25 root root  4096 Jun 30 14:20 ..
drwxr-xr-x 399 root root 16384 Dec 31  1969 afs
drwxr-xr-x   2 root root  4096 Jun 30 14:38 bin
drwxr-xr-x   4 root root  4096 Jun 30 14:43 boot
-rw-r--r--   1 root root     0 Jul 11 16:00 clearAFScach
lrwxrwxrwx   1 root root    23 Jun 30 14:19 common -> /a
drwxr-xr-x  19 root root  4240 Jul 11 16:00 dev
```

# Edit & Compile WebGoat Lessons

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

```
vi CommandInjection.java
```

```
cd ~/WebGoat/WebGoat-Lessons-develop
```

```
mvn clean package
```

```
cp target/plugins/*.jar ../WebGoat-7.1/webgoat-  
container/src/main/webapp/plugin_lessons/
```

(you can run the [~/compile.sh](#) script)

# Edit & Compile WebGoat Lessons

In `CommandInjection.java`:

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

```
cd ~/WebGoat/WebGoat-Lessons-develop
```

```
mvn clean package
```

```
cp target/plugins/*.jar ../WebGoat-7.1/webgoat-  
container/src/main/webapp/plugin_lessons/
```