Code Analysis for Security Vulnerabilities

Lotfi ben Othmane

SQL Injection Attack

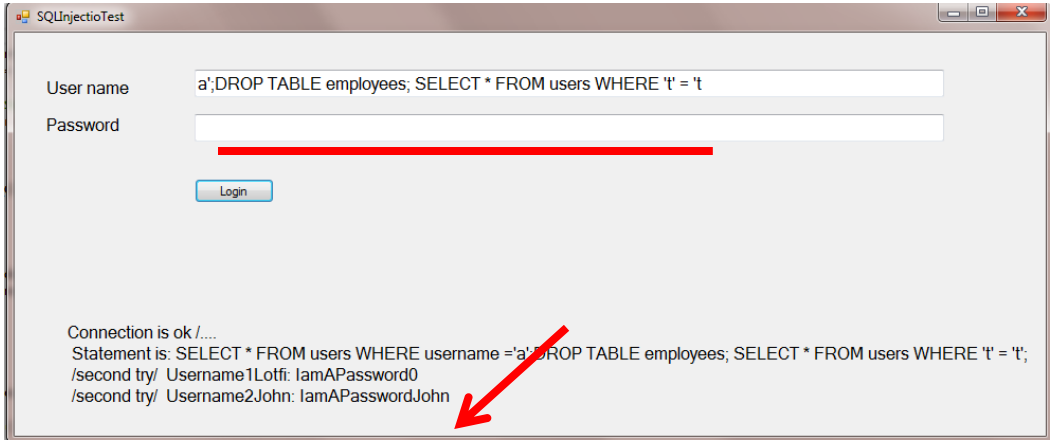
Original query: *SELECT * FROM users*

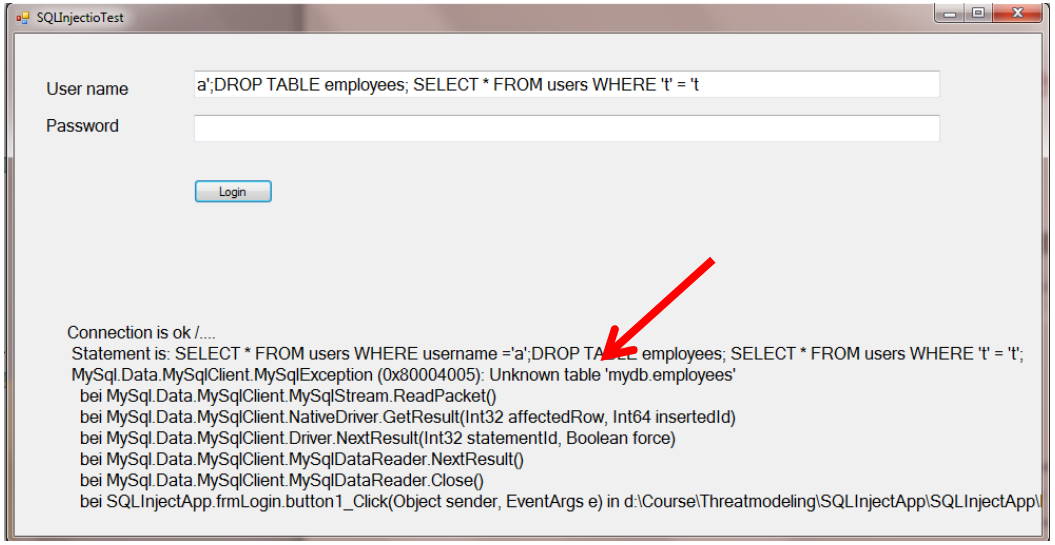
WHERE username =‘

+ variable + "';"

Malicious data: a';DROP TABLE employees; SELECT * FROM users

WHERE 't' = ‘t





Buffer Overflow Vulnerability

```
char buff[10];
int pass = 0;
char secret[10];
strcpy(buff,"Password");
printf("\n Enter your password: ");
gets(secret);
if(strcmp(secret, buff))
        { printf ("\n Wrong Password \n");}
else
        {printf ("\n Correct Password \n");
         pass = 1; }
if(pass)
   printf ("\n Root privileges given to the user \n");
```

How do you know that the code includes a buffer overflow?

Identification of Vulnerabilities

Solution1: Functional testing

```
addr = 0xbffff148 + offset;

ptr = buffer;
addr_ptr = (long*)(ptr);

for (i = 0; i < 10; i++)
      *(addr_ptr++) = addr;

memcpy(buffer + sizeof(buffer) - sizeof(shellcode), shellcode,
sizeof(shellcode));
```

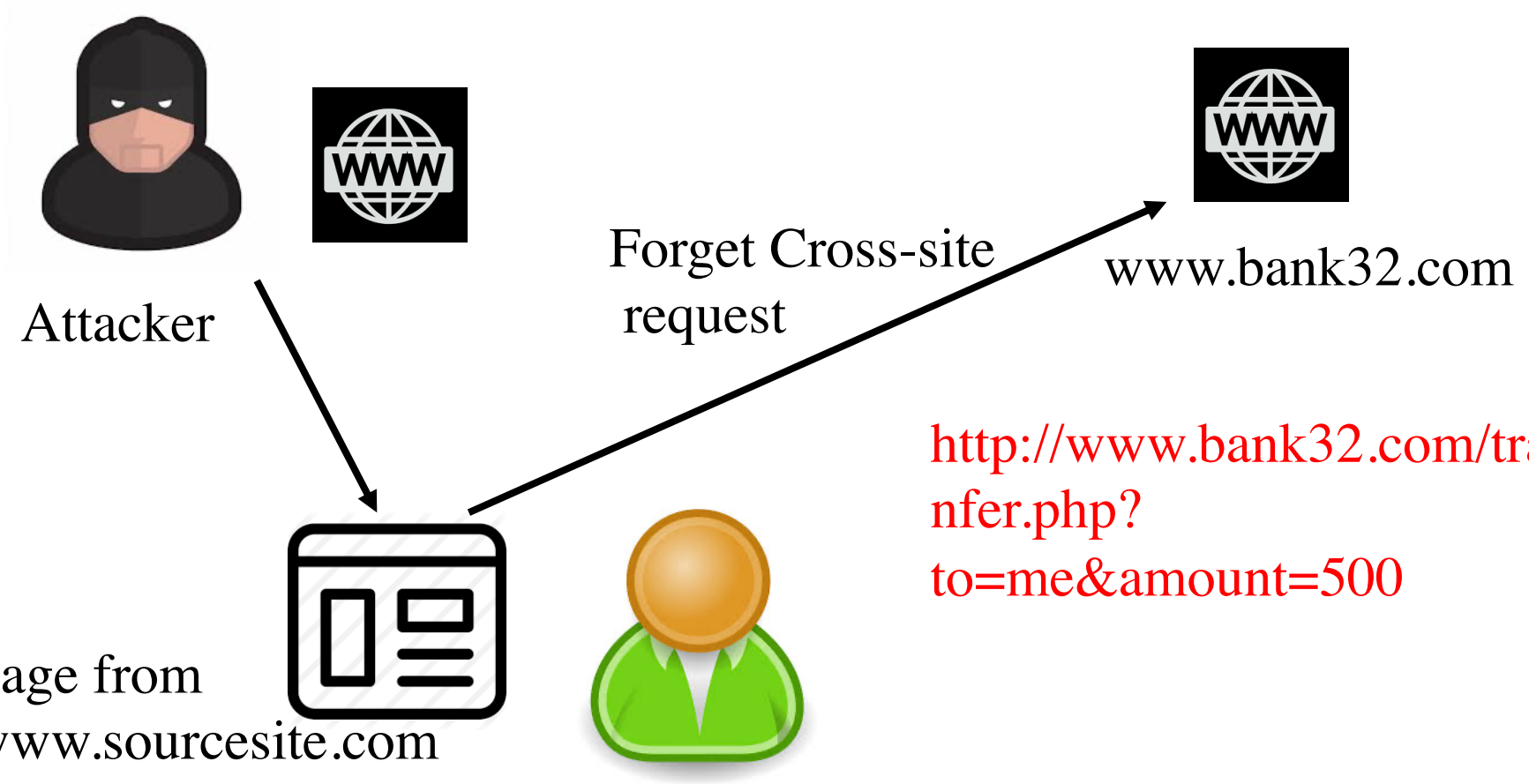
Identification of Vulnerabilities

Security testing: Checking if the program exhibit a malicious behavior.

Approaches

1. Whitebox approach – Access to source code
2. Blackbox approach – No access to the source code

Blackbox Security Testing



Whitebox Security Testing

- Code analysis: Inspect the code to detect possible malicious behavior

- Techniques:
  - Static code analysis – "*Compile-time techniques for predicting safe and computable approximations to the set of behaviors arising dynamically at run-time when executing the program*" Nielsen et al. 2005
  - Dynamic code analysis – "Run-time techniques for predicting safe and computable approximations to the set of behaviors arising dynamically at run-time when executing the program.

Static Code Analysis

- Scans the whole code base
- Identify coding issues: dead code, null pointer dereference

Static Code Analysis

- Static code analysis uses code to verify properties
  - Is a variable a constant => constant propagation
  - Is a code piece reachable -=> reachability analysis
  - Can a pointer variable be null => pointer analysis
  - Is the session closed = > typeset analysis

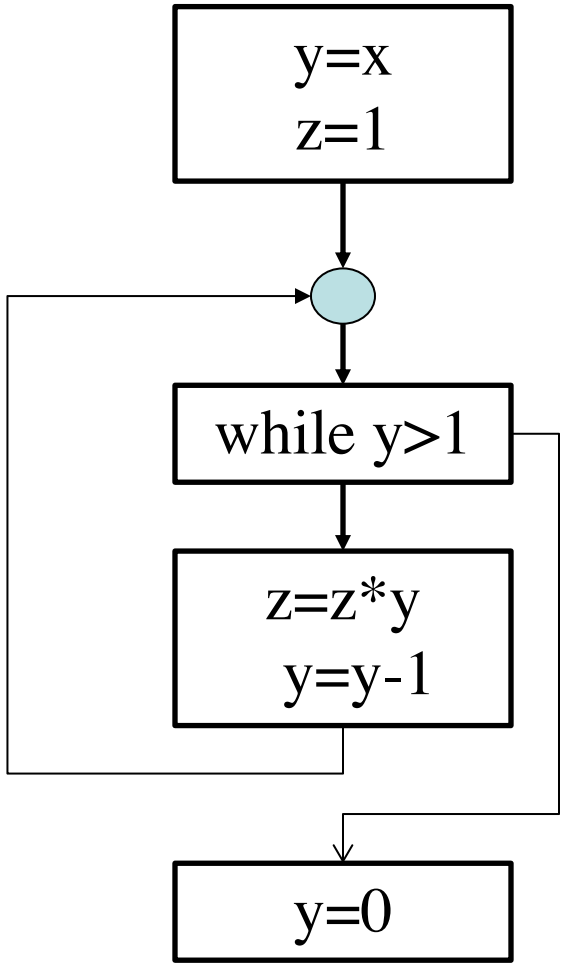
Control-Flow Analysis

- Control-flow analysis determines the order of statements of a program and execution paths
- Control flow graph: A directed graph in which the nodes represent basic blocks and edges represent transfer between the nodes
- Basic bloc: A sequence of statement having one entry point and one exit point
- Path: A sequence of statements of the CFG
- Trace: A set of statements performed during execution

Control-Flow Analysis

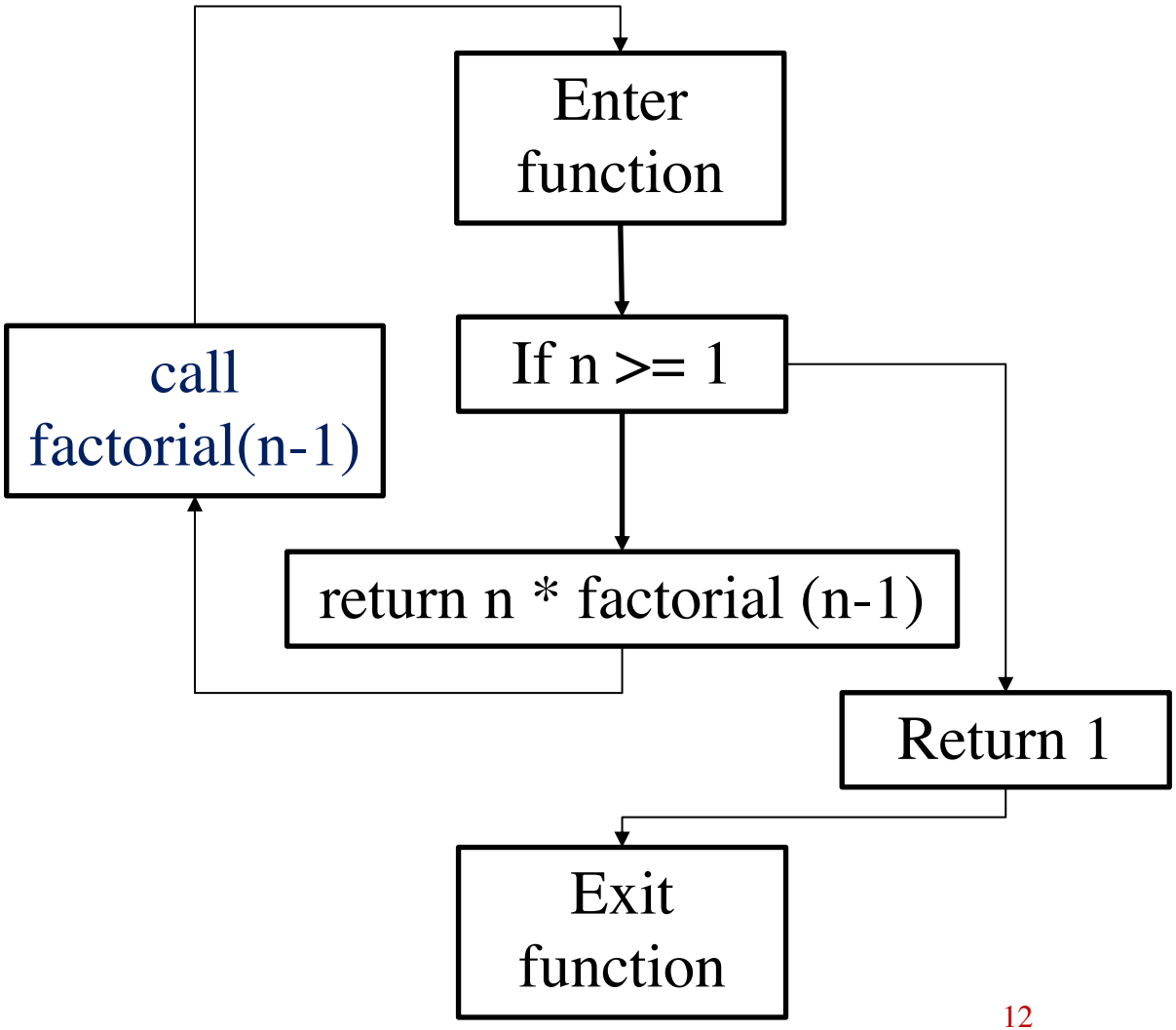
Is statement “y=0” dead code?

```
y=x
z=1
while y>1 do
  z=z*y
  y=y-1
y=0
```



Control-Fow Analysis

```
int factorial (int n) {
if (n>=1)
      return n* factorial(n-1);
else return 1;
 }
```



Data-Fow Analysis

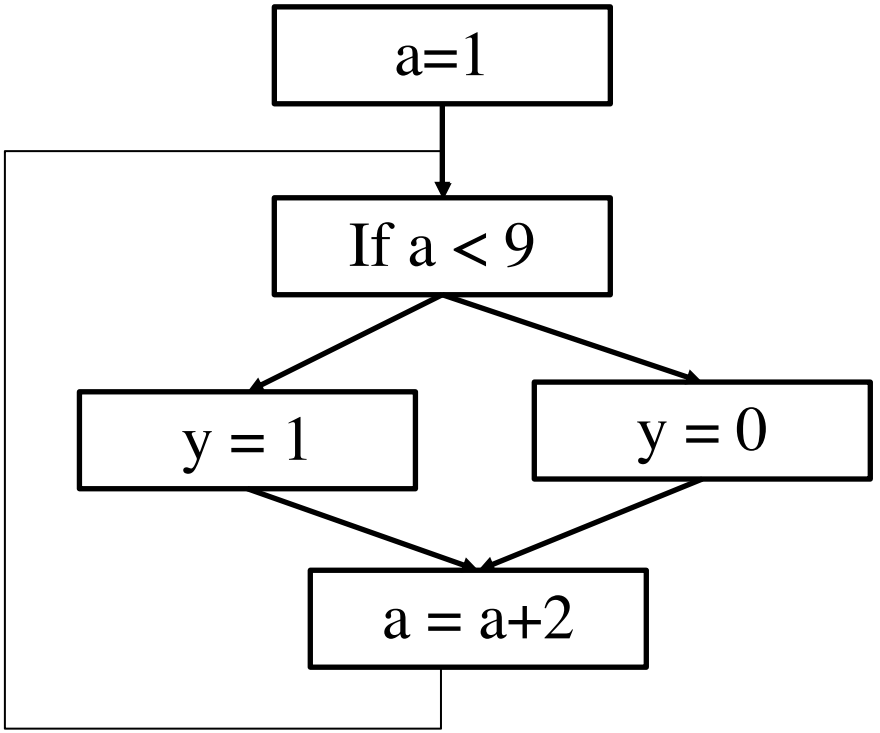
- Data flow analysis allows to analyze how the data (variables) flow across the program.

- There are two analyses
  - Intra-procedure analysis – functions and procedures are not considered
  - Inter-procedure analysis – functions and procedures are considered

Intra-Procedure Analysis

What are the possible values of y?

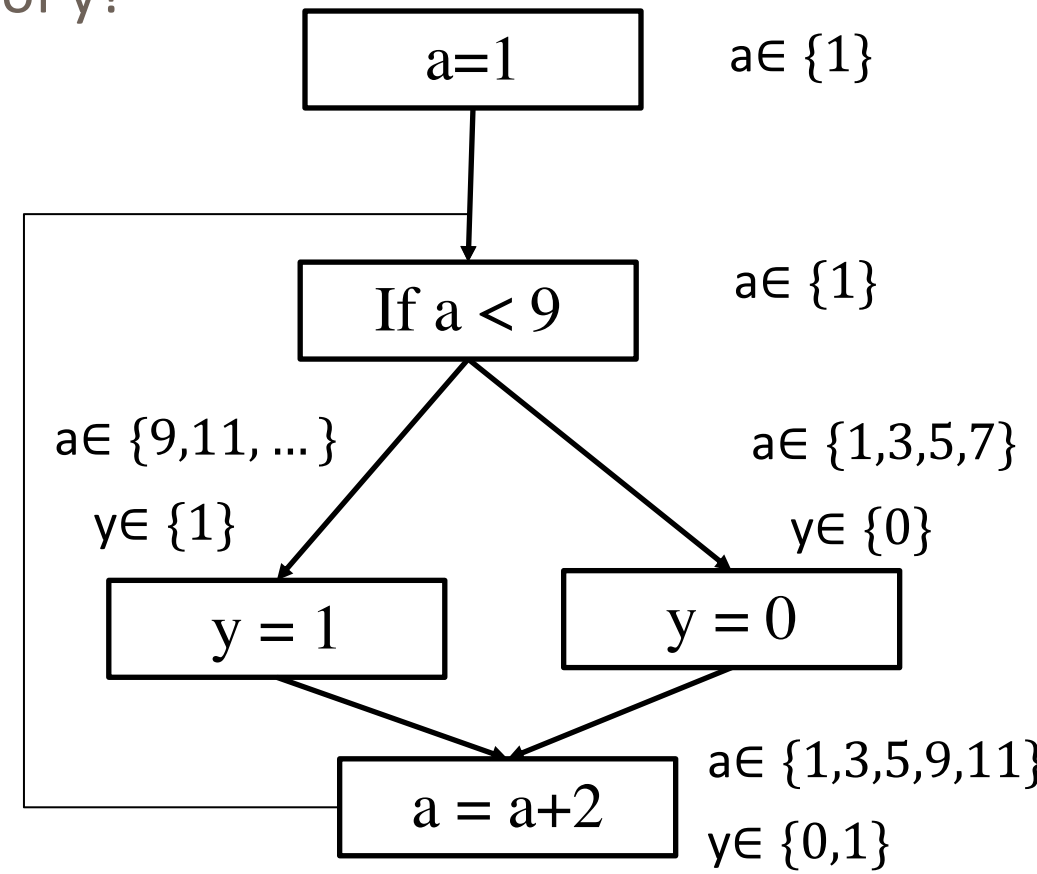
```
a= 1
while (true)
    if (a < 9)
        y := 0;
    else
        y := 1;
    a=a+2;
```



Intra-Procedure Analysis

What are the possible values of y?

```
a= 1
while (true)
    if (a < 9)
        y := 0;
    else
        y := 1;
    a=a+2;
```

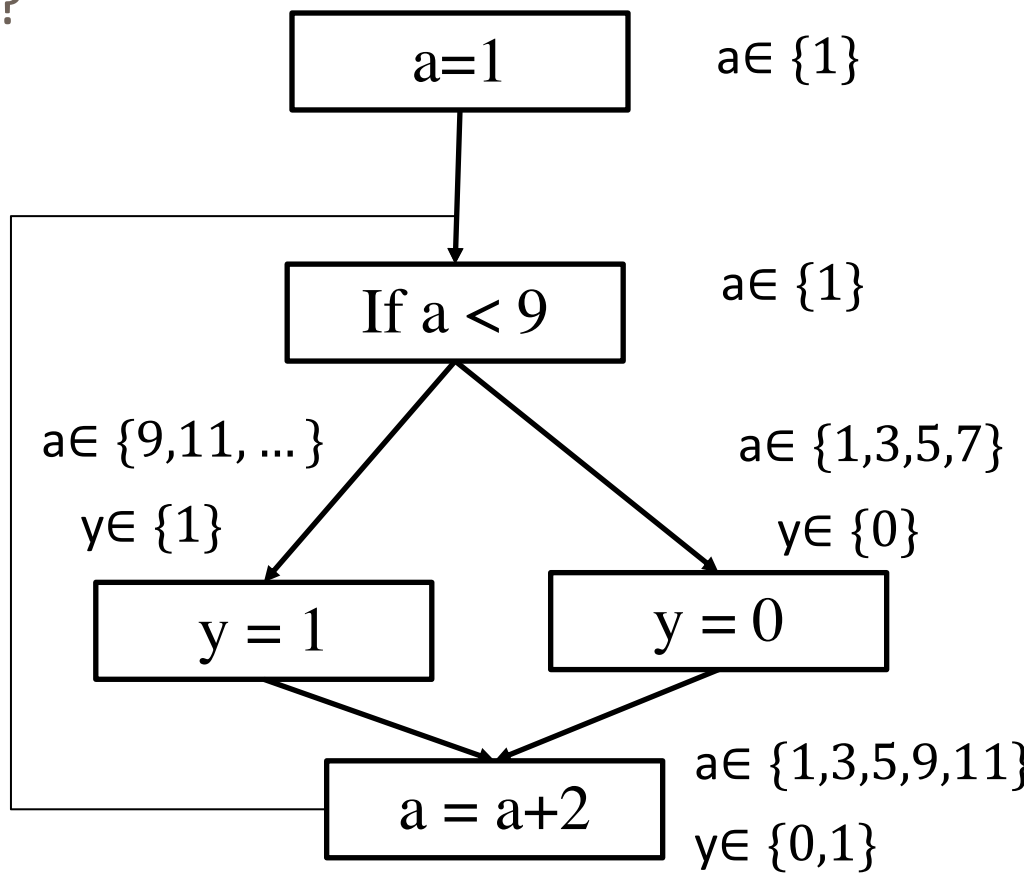


Control and Data Flow Analysis

Is statement “y=1” dead code?

Does the program terminate?

```
a= 1
while (true)
    if (a < 9)
        y := 0;
    else
        y := 1;
    a=a+2;
```



Pointer Analysis

Goal: Answer questions related to dynamic memory allocations

Types of analysis

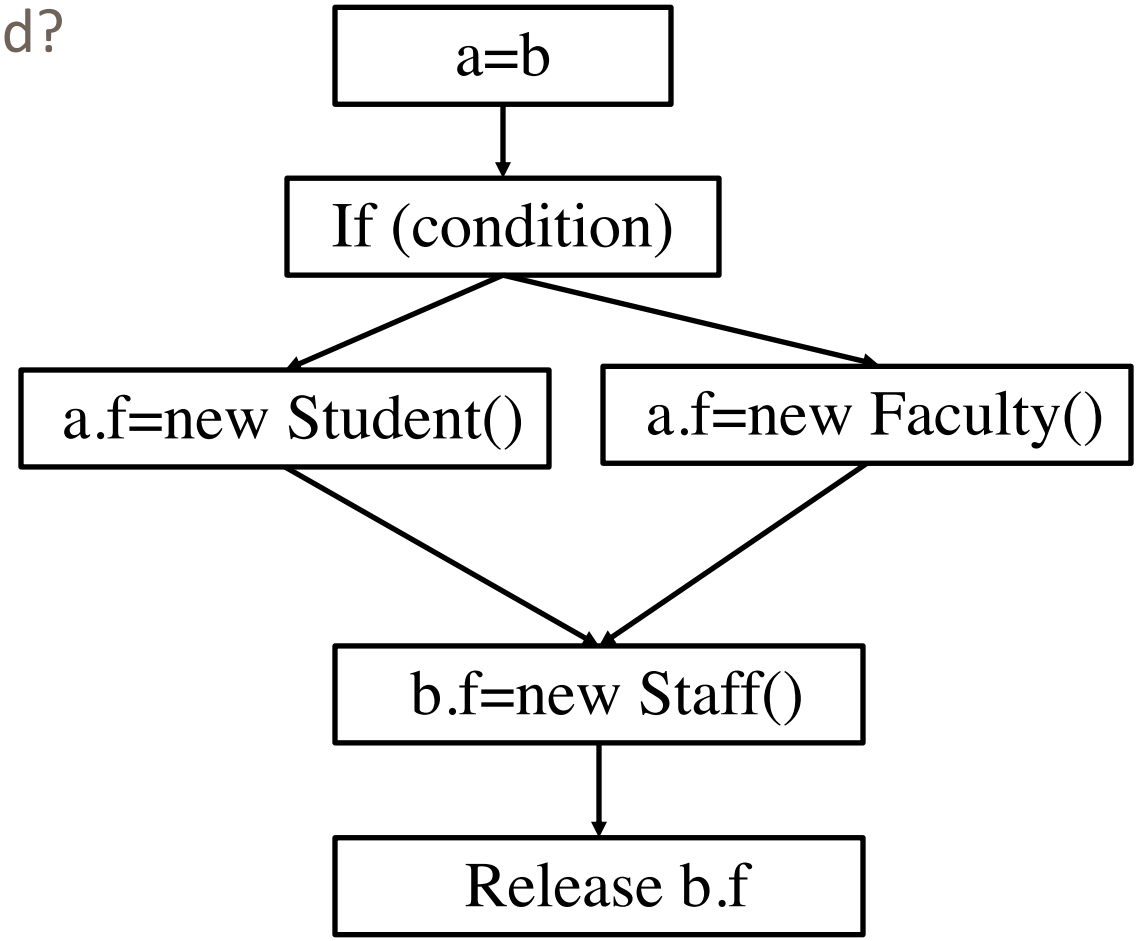
- What memory locations can a pointer refer to?
- When do two pointer expressions refer to the same storage location?

Pointer Analysis

Is Faculty object released?

```
a=b
if (condition)
   a.f = new Student();
else
   a.f=new Faculty()

b.f=new Staff()
```

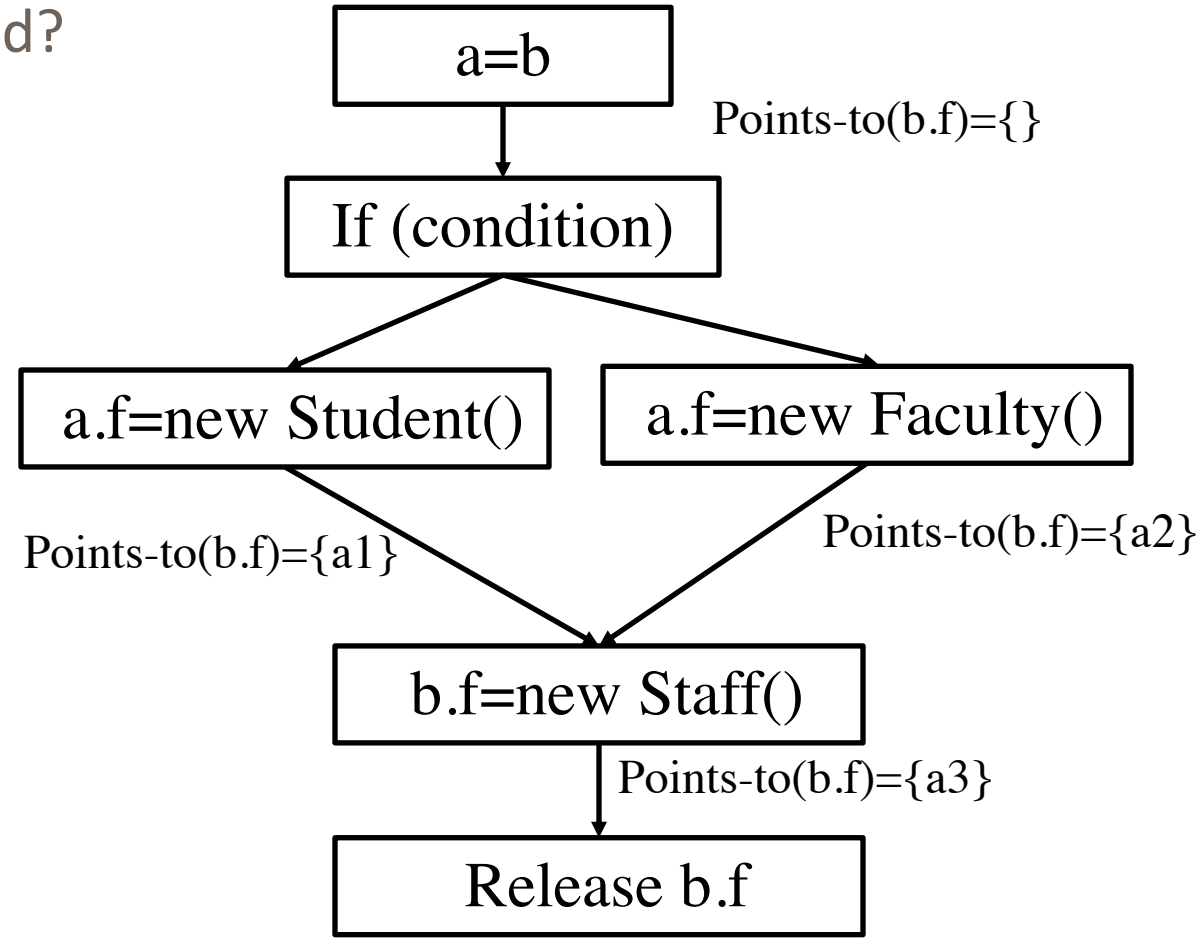


Pointer Analysis

Is Faculty object released?

```
a=b
if (condition)
   a.f = new Student();
else
   a.f=new Faculty()

b.f=new Staff()
```



Exercise - Pointer Analysis

```
char buff[10];
int pass = 0;
char secret[10];
strcpy(buff,"Password");
printf("\n Enter your password: ");
gets(secret);
if(strcmp(secret, buff))
        { printf ("\n Wrong Password \n");}
else
        {printf ("\n Correct Password \n");
         pass = 1; }
if(pass)
   printf ("\n Root privileges given to the user \n");
```

Write a small pseudocode to detect the buffer overflow using dataflow and control flow techniques

Taint Analysis

Taint analysis tracks information flow between sources and sinks

Uses of taint analysis

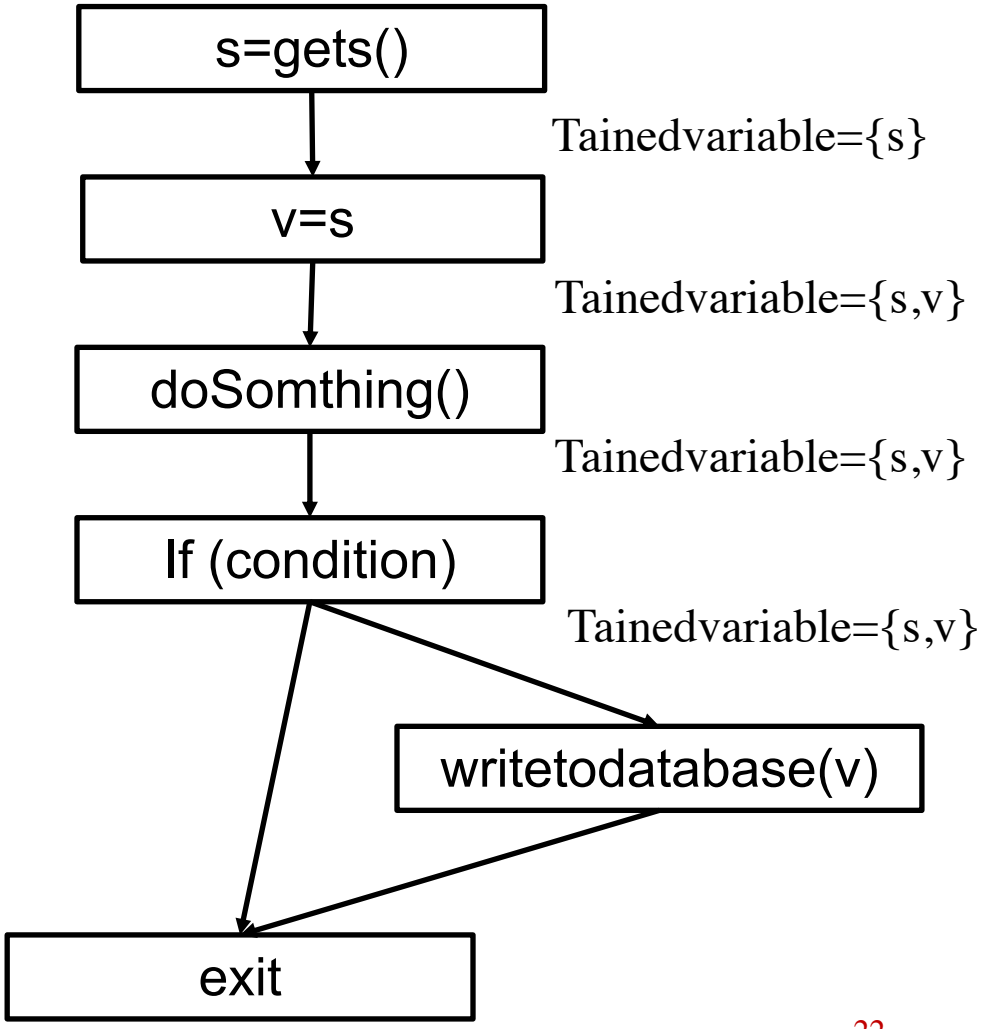
1. Check input sanitization
2. Information leakage
3. Vulnerabilities like SQL injection

It allows to identify the paths of interest and reduces the verification process.

Taint Analysis

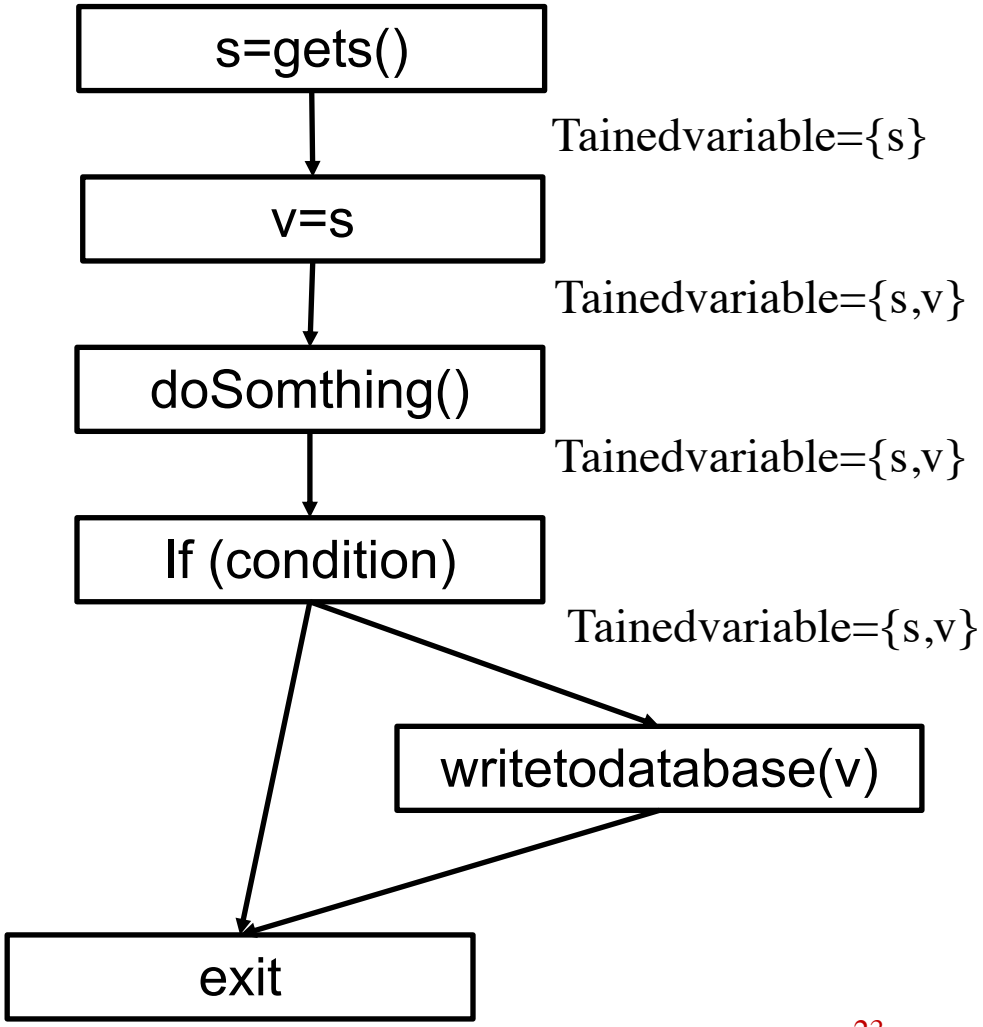
Is there a danger from a potential SQL injection?

```
s=gets()
v=s
doSomthing()
If (condition)
    writetodatabase(v)
```



Taint Analysis

Our sink is writetodatabse()



Taint Analysis

Factors that impact the precision and performance of static code analysis

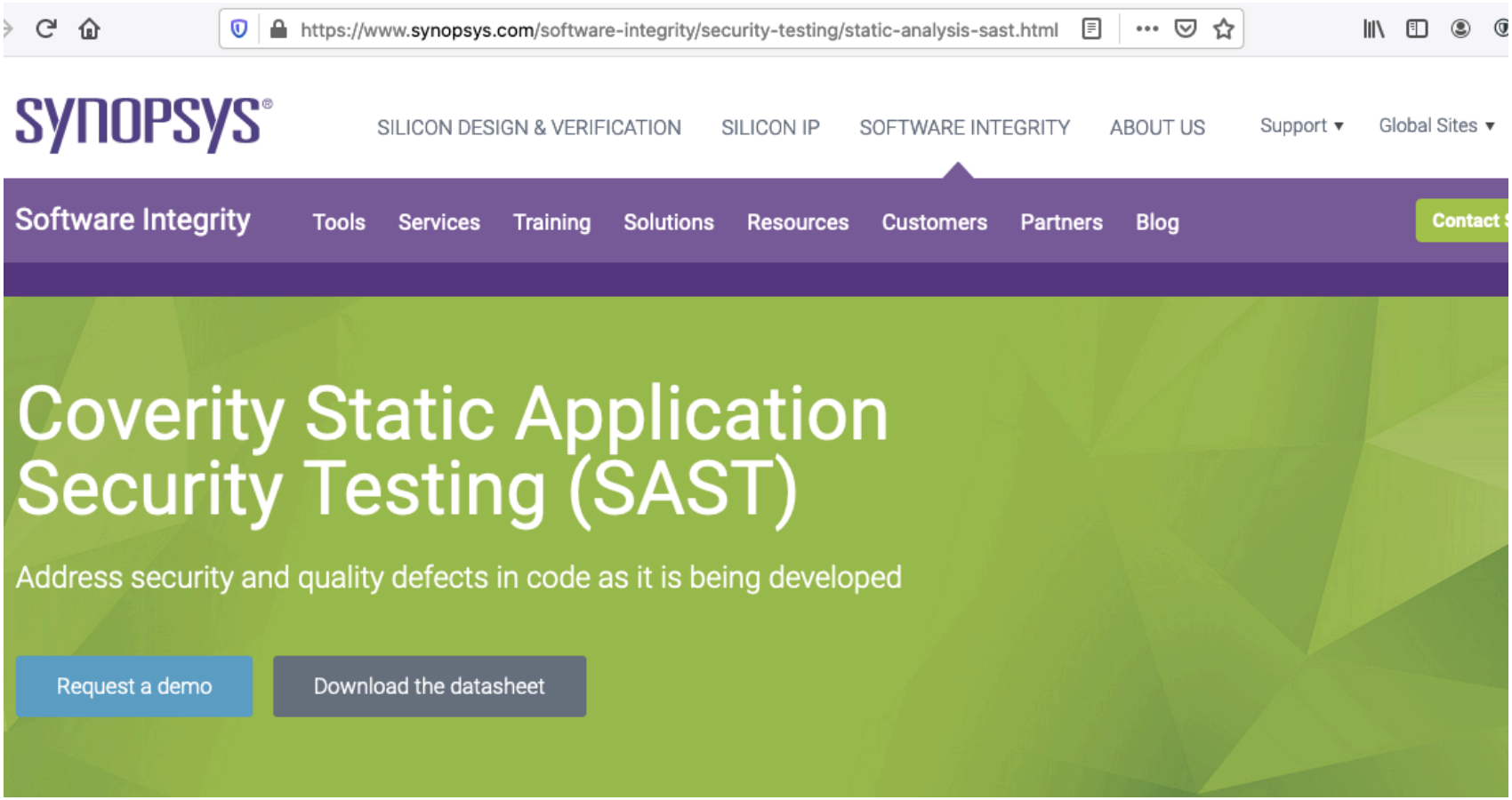
1. Flow-sensitivity
2. Field sensitivity
3. Pointer analysis

Code Analysis in Practice

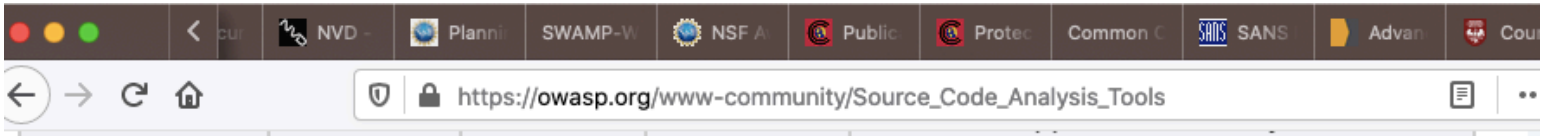
Use taint analysis to detect SQl injection

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    boolean success = false;
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String query = "select * from tbluser where username='" + username + "' and password =
'" + password + "'";
            conn = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/user", "root",
"root");
        stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        if (rs.next()) {
            // Login Successful if match is found
            success = true;
        }
```

Code Analysis Tools



Code Analysis Tools



				/cwe#SupportedSecurityStandards).
Checkmarx Static Code Analysis		Open Source or Free		Android, Apex, ASP.NET, C\#, C++, Go, Groovy, HTML5, Java, JavaScript, JSP, .NET, Objective-C, Perl, PHP, PL/SQL, Python, Ruby, Scala, Swift, TypeScript, VB.NET, Visual Basic 6, Windows Phone
Codacy		Commercial		Offers security patterns for languages such as Python, Ruby, Scala, Java, JavaScript and more. Integrates with tools such as Brakeman, Bandit, FindBugs, and others. (free for open source projects)
CodeScan Cloud		Commercial		A Salesforce focused, SaaS code quality tool leveraging SonarQube's OWASP security hotspots to give security visibility on Apex, Visualforce, and Lightning

Performance of Code Analysis

- Sound – If *X* is true than *X* is found to be true
  - No false negatives
  - Detect all errors and miss no errors

- Complete – If *X* is found to be true than *X* is true
    - No false alarms

Performance of Code Analysis

Terminology is in terms of finding errors

- TP True Positive: found a real errors
- FP False Positive: false alarm
- TN True Negative: no error, no alarm—OK
- FN False Negative: missed errors

Recall: TP/(TP+FN) measures how (un)sound

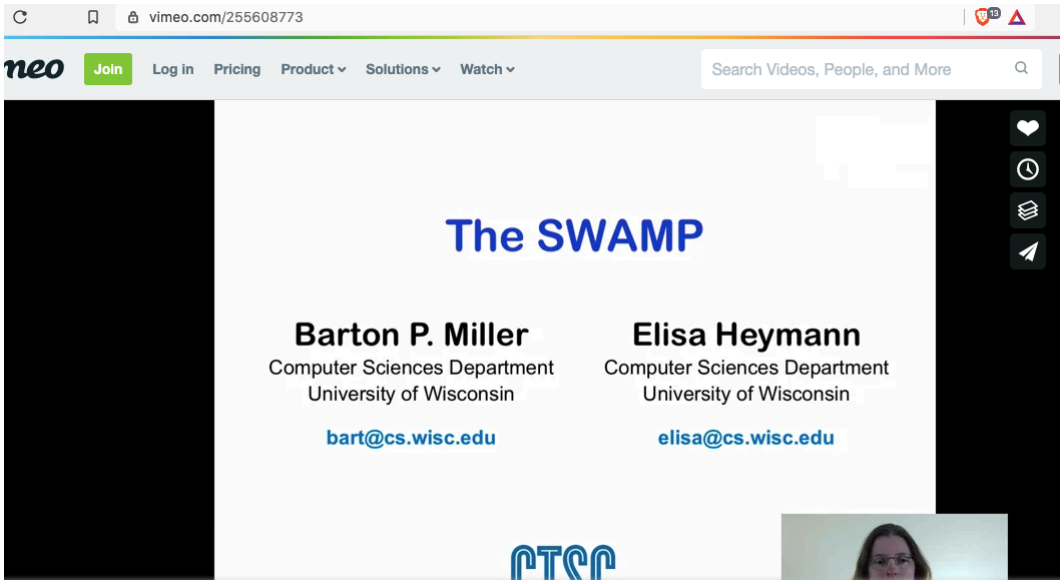
Precision: TP/(TP+FP) measures how (in)complete

Code Analysis Platform

Platform: https://iastate.cosalab.org/

Introduction: https://vimeo.com/255608773

Course material at wisc: https://research.cs.wisc.edu/mist/SoftwareSecurityCourse/



Thank you

Any Question?