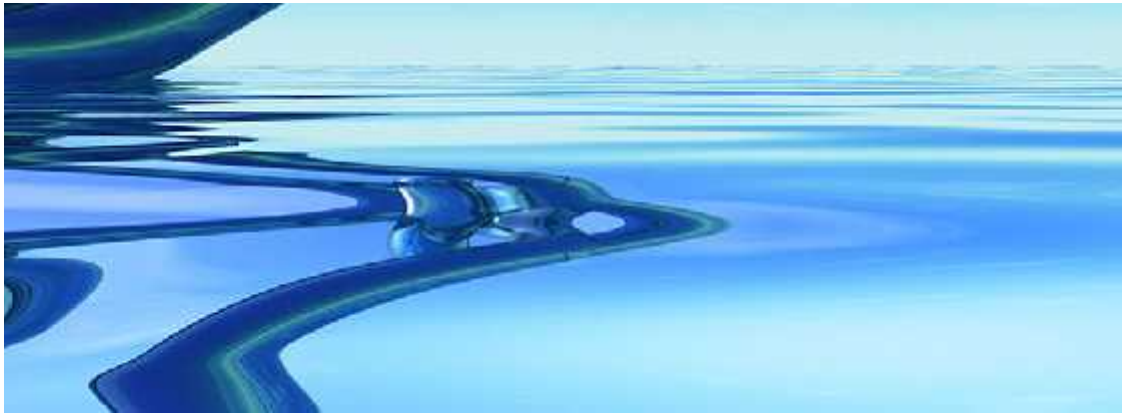
Thema

Securing Applications with Code Review

Dr. Bruce Sams

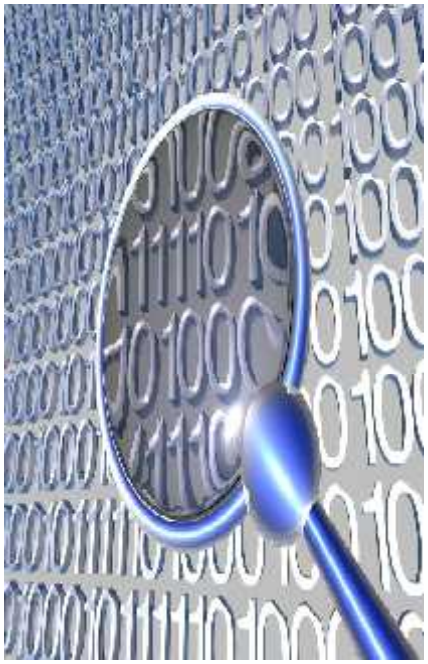
bruce.sams@optimabit.com

OPTiMA..bit
business information technology



Agenda

- Explain what techniques are used in a code review
- Show what kind of problems a review can and cannot identify
- Discuss the practical limits of code review
- Discuss when a review should be performed and by whom



About

OPTIMAbit

GmbH

OPTIMA..bit
business information technology

Focus

IT Security for applications and infrastrucutres

Customers

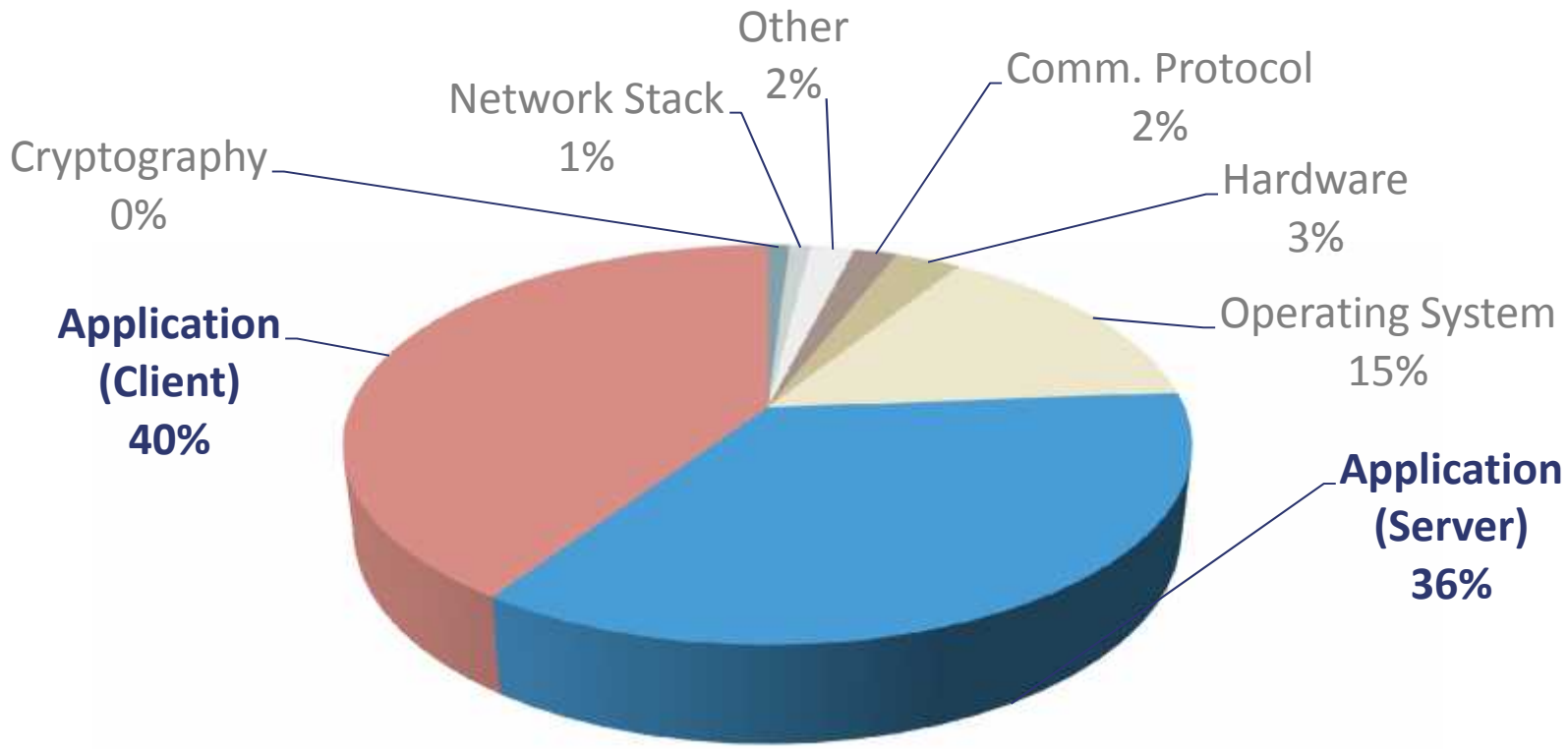
Companies over ca. 500 Employees

Credo

Independent consulting of the highest quality

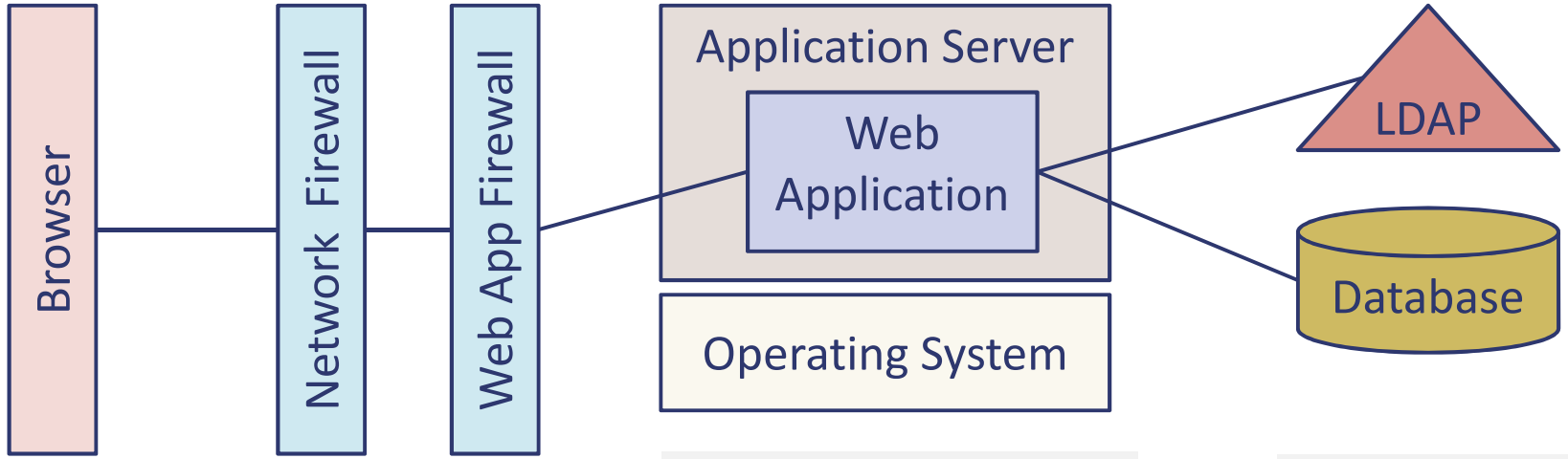
Where do vulnerabilities come from?

Ca. ¾ of all vulnerabilities arise in applications



Vulnerabilities (overview)

Vulnerabilities exists at all levels



Local data storage
Client Spoofing
JavaScript
Phishing

HTTP/HTTPS „hole“
Insecure zoning

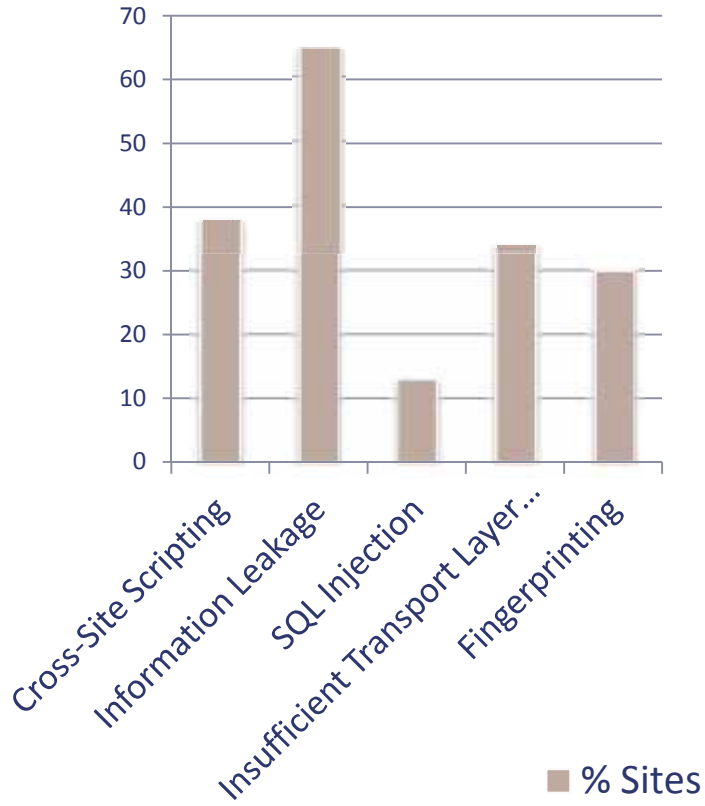
Insecure validation
Insecure sessions
Insecure configuration
Insecure error handling
Insecure persistence

LDAP Injection
SQL Injection
XPATH Injection

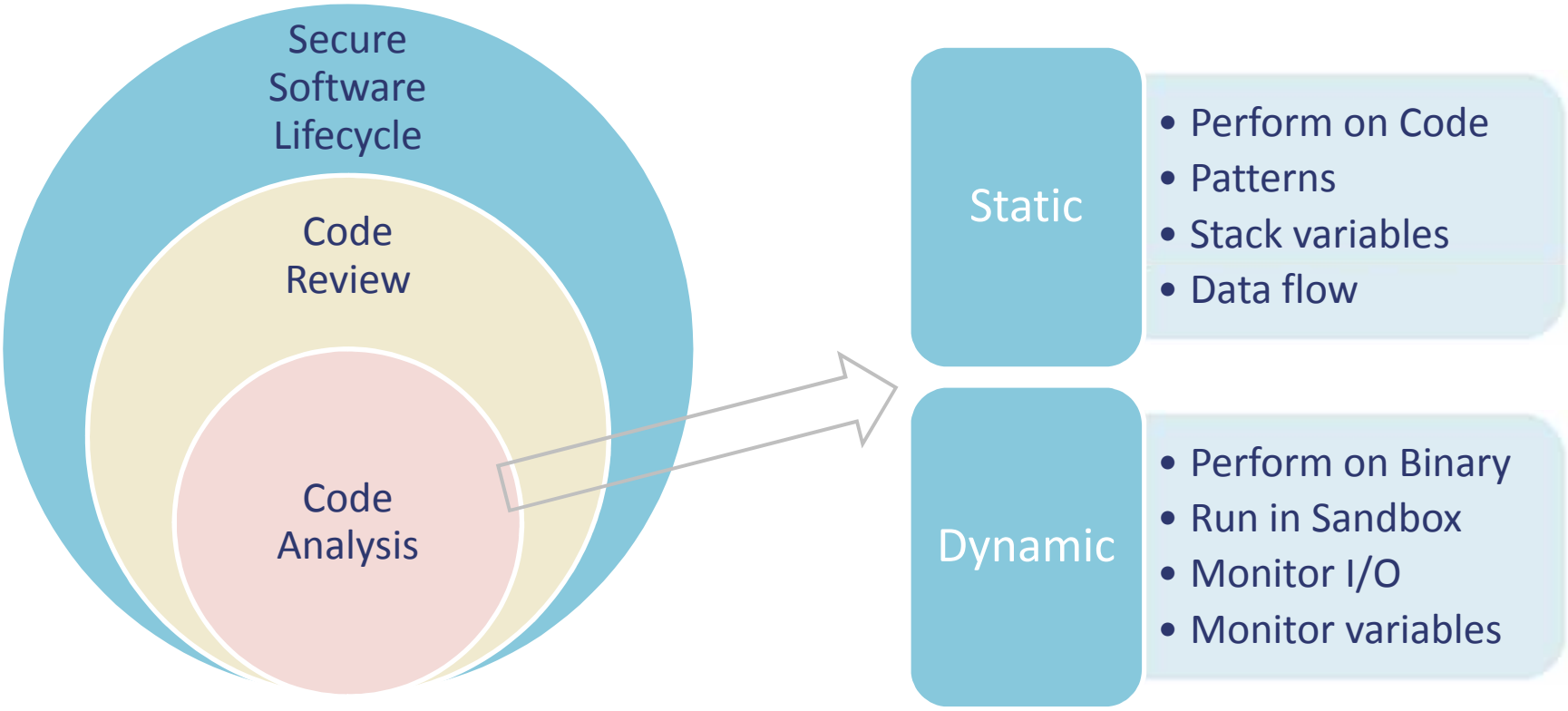
Web Applications are in especially in Danger

Web applications are the preferred target for cybercriminals

- The number and aggressiveness of attacks is constantly increasing
- The attackers are professionals: the cybermafia or governments.
- Web applications are preferred targets because they often have so many vulnerabilities.



Code Review and Code Analysis



Problem Categories and Strategies

Category	Examples	Difficulty	Strategy
Conventions	naming, formatting	1	Patterns
Structure	cyclomatic complexity, affine/afferent binding, package dependencies, etc.	2	Patterns
Implemen-tation	null pointer, endless loop, unreachable code, dangerous API calls	2 - 4	Patterns Stack Analysis
Security	Authorization, authorization, url encoding, injection, sessions, configuration, intra-procedural calls	4 – 5	Patterns Stack Analysis Data Flow Business Logic

SOME EXAMPLE CODE PROBLEMS

Some Simple Examples

Null Pointer Exception

```
String s = null;

if(s != null || s.length() > 0)  //evaluate s.length()

if(s == null | s.equals(""))     //evaluate s.equals()
```

Little Bug Patterns (lots of them)

```
int x = 1;

int y = x<<32; //bit shift more than 31 places
```

Some Simple Examples

Bad/Incorrect Method Invocation

```
String s = "hello  ";  //extra white space
s.trim();
someMethod(s);  //should use s = s.trim()

s = s.trim();
```

False Positive, False Negative, True Positive?

Many analysis tools report a large number of „false positives“

- False negative (FN): Non-detection of an existing problem
- False positive (FP): Detection of a „problem“ that is not really a problem
- True positive (TP): Detection of a real problem.
- Flagged Non-Conformity (FNC): Indication that a problem might exist at a given code location.

- Most FNCs are the result of turning on too many filters in a tool (e.g. Implement Serializable in Java).
- This can lead to thousands of FNCs:

FNC/TP >= 1000 !

False Positive: Simple Example

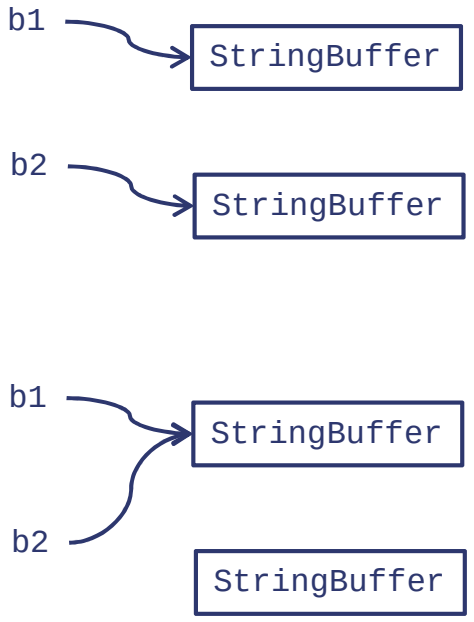
```
int getString(int i) {
   String s = null;
   switch (i) {
      case 1:
         s = "one";
         break;
      case 2:
         s = "two";
         break;
   }
   return s.length(); //NullPointerException?
}
```

False Negative: Pointer Complexity

```
String s1 = <TAINTED DATA>
StringBuffer b1 = new StringBuffer();
StringBuffer b2 = new StringBuffer();

//Do something here  . . .

b1.append(s1);
String s2 = b2.toString();
```

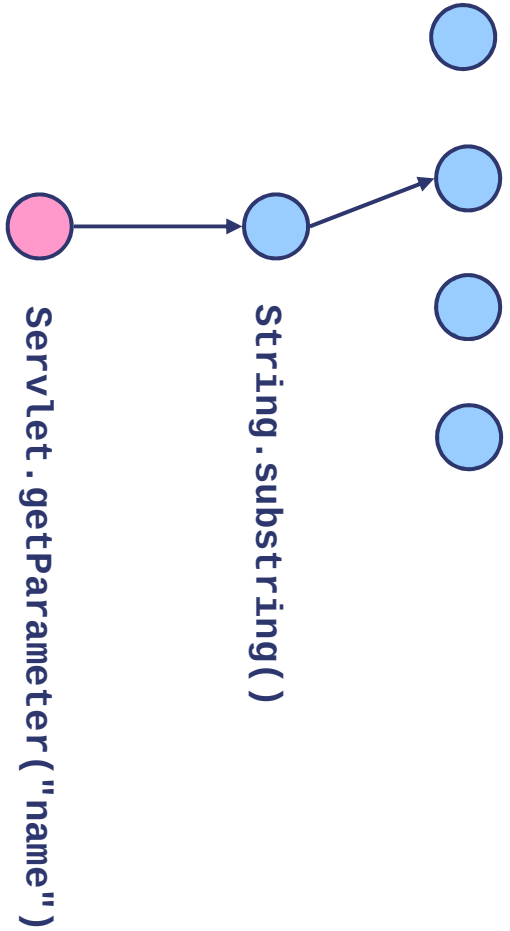


Question: is s2 safe to use?

Tracing Tainted Data

An application's call graph can be large and difficult to model:

- 10 steps with 10 branches gives ~ 10^{10} nodes.
- If each node requires 1KB memory to model, then a full model requires 10^{13} Bytes = 10^4 GB RAM.
- Hard to solve the general problem completely.



Source/Sink Example: SQL Injection

```
String s = request.getParameter("name");
Connection connection = …;
String q = "'SELECT * FROM Users WHERE NAME ='" + s + "'";
connection.executeQuery(q);
```

- Source = Manipulated Information in Request

```
name = xxx' OR 1 = 1;--
```

- Sink = executeQuery

Cross-Site Scripting

The HTML form...

```
<form action="/address">
  Address, please: <input type="text" name="address">
</form>
```

The JSP code...

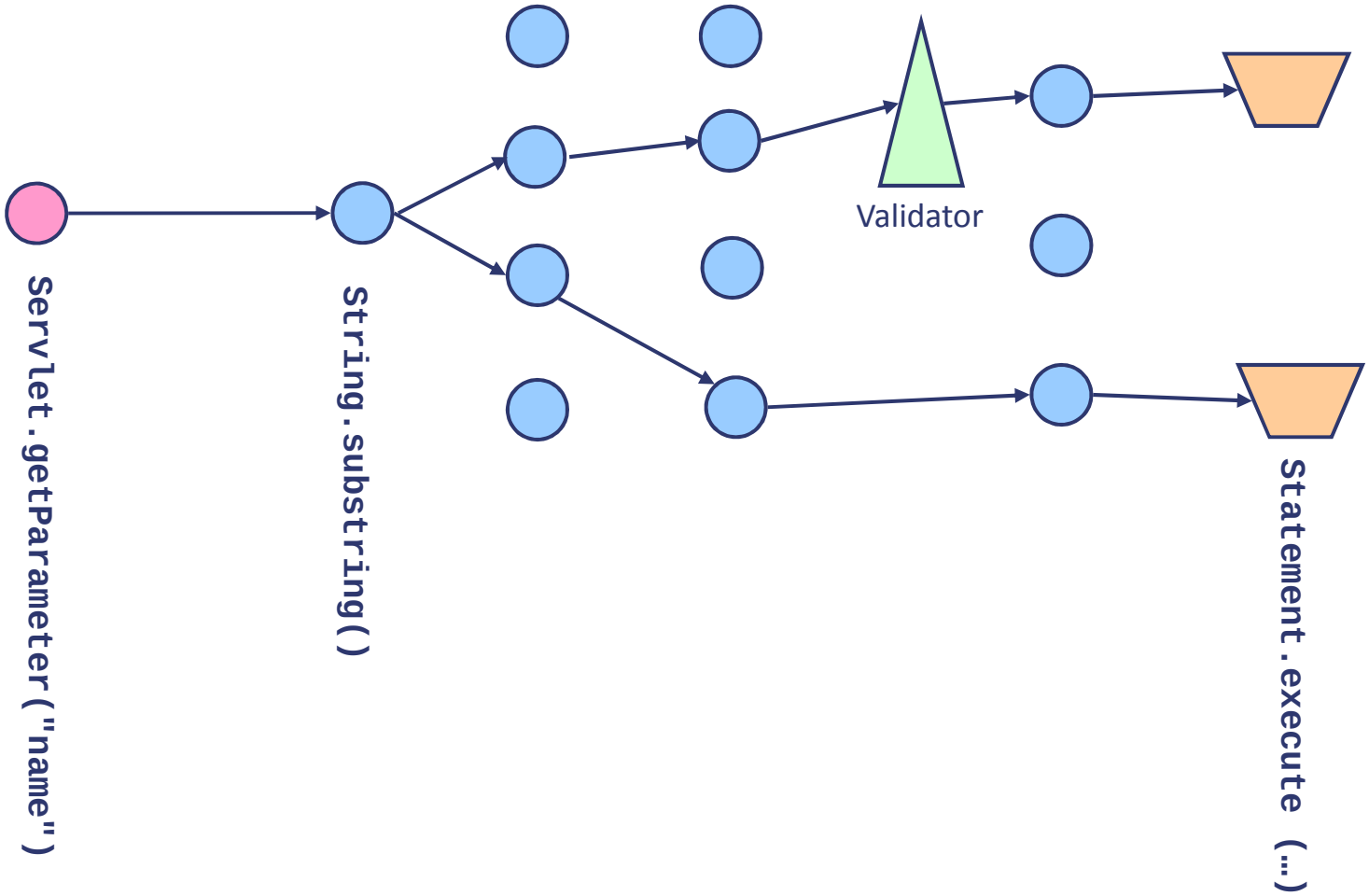
```
Your address: <%= request.getParameter("address") %>
```

Secure using input validation or output encoding

```
validateInput(request.getParameter(”address”));

response.write(URLEncoder.encode(XXX,String encoding));
```

Tracing Tainted Information



Further Issues and Complications

Maps and containers are generally difficult to handle.

```
String name = <TAINTED DATA>
map.put("USER", name);
map.put("USER", someOtherString);
map.get("USER");  // tainted?
```

Pointer uncertainties add to the reflection problem (e.g., suppose a dynamic className as in previous example comes from a Map)

Dynamic Class Loading

Dynamic class loading and reflection complicates knowing which classes will be instantiated at runtime.

```
String myClass = <TAINTED DATA>
Class class = Class.forName(myClass);
Object o = class.newInstance();
```

Pointer uncertainties add to the reflection problem (e.g., suppose the className String comes from a Map)

False Negative: Authentication Logic

```
String userType = <TAINTED DATA>
if (userType.equals("NormalUser")){
   setUserPermisions("Normal_Permissions");
} else { //user must be admin
   setUserPermissions("Admin_Permissions");
}
```

How can a tool understand the business logic?

How can a tool interpret the Permission Strings?

Easter Egg

```
if (Date == Easter) {
  System.exit();
}
```

EXTENSIONS FOR SECURITY ANALYSIS

New FindBugs detectors to improve review

Focus on security issues

- **31 bug patterns from "CERT Oracle Secure Coding Standard for Java"**
  - MSC01-J. Do not use insecure or weak cryptographic algorithms
  - MET04-J. Ensure that constructors do not call overridable methods
  - SER10-J. Do not serialize direct handles to system resources
  - . . .
- **5 further common bug patterns**
  - SQL Injection
  - HTTP Response Splitting
  - Command Injection
  - . . .

SQL Injection :: Java-Code

```
public void inject(Statement s, String input) throws
  SQLException {

  s.execute("SELECT * FROM products WHERE id = " + input);

}
```

SQL Injection :: Java-Bytecode

Java

```
public void inject(java.sql.Statement, java.lang.String)   throws
  java.sql.SQLException;
```

Bytecode:

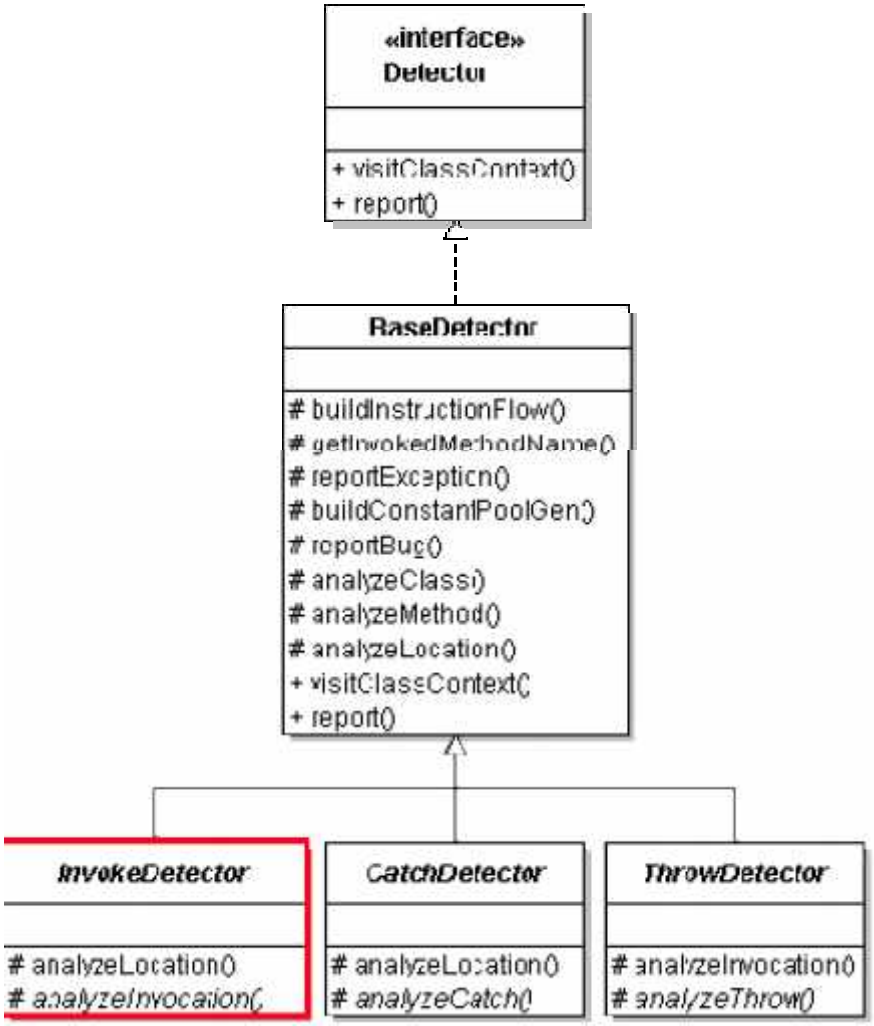
```
0: aload_1
1: new        #23; //class java/lang/StringBuilder
4: dup
5: ldc        #25; //"SELECT * FROM produkte WHERE id ="
7: invokespecial     #27; //Method
10:  aload_2
11:  invokevirtual  #30; //Method
14:  invokevirtual  #34; //Method
17:  invokeinterface        #38,  2;    //
  java/sql/Statement.execute:(Ljava/lang/String;)Z
22:  pop
23:  return
```

FindBugs detector hierarchy

FindBugs

+ Easily extensible
+ OpenSource

– Only Java
– Weak Taint Tracking



SQL Injection: Findbugs Detector

```
if (fullQualifiedMethodName.equals("java.sql.Statement.execute")) {

ConstantFrame frame =
  getContext().getConstantDataflow(method).getFactAtLocation(locatio
  n);

ConstantPoolGen cpg = buildConstantPoolGen(method);

int numArguments = frame.getNumArguments(instruction, cpg);

Constant value = frame.getStackValue(numArguments - 1);
   if (!value.isConstant()) {
      // Found possible SQL Injection
   }
}
```

WebGoat with OPTIMAbit Detectors

New detectors find more bugs, but also more false positives

- **3 Installations:**
  - FindBugs without Plugins
  - FindBugs + fb-contrib
  - FindBugs + OPTIMAbit detectors
- **OPTIMAbit detectors:**
  - XPath Injection
  - Command Injection
  - Insecure Cryptography

Installation	Problem Types Detected	True Positives	False Positives
No Plugins	3	15	0
fb-contrib	4	20	0
OPTIMAbit Detectors	8	117	16

SQL Injection: 14

HTTP Response Splitting: 2

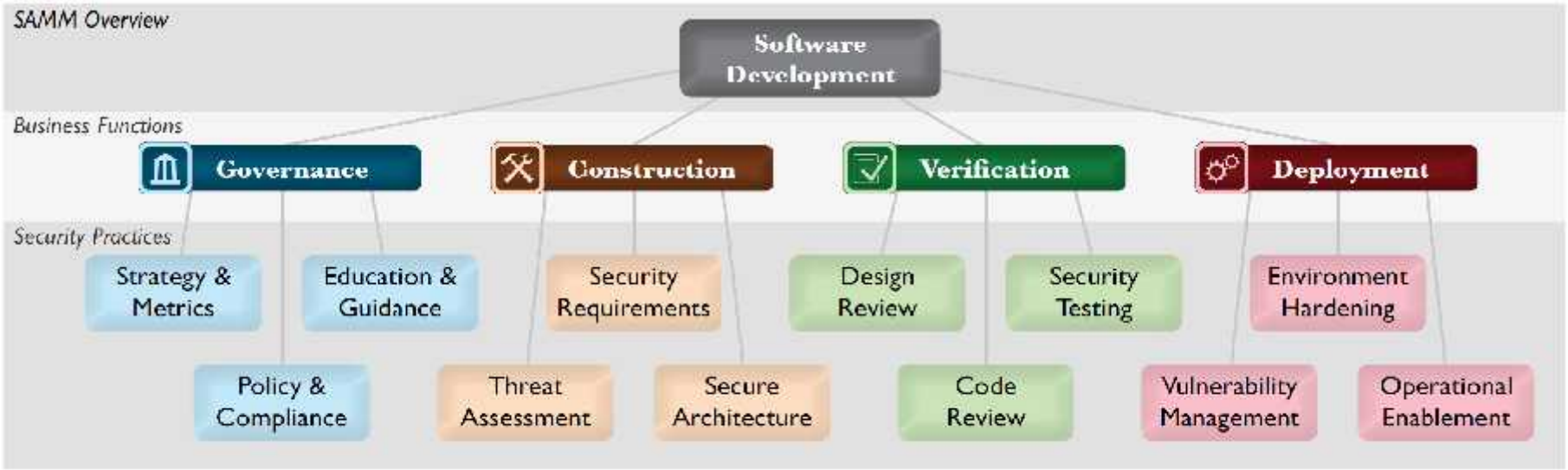
Practical issues in selecting a tool

Topic	Comment
Support for multiple Languages	•Java/C#: very good •Objective C, JavaScript: weak •COBOL, PL1, XSLT: effectively nonexistent
Analyze source and/or binaries	•Binaries need compileable project •Compiled active pages can be linked to code.
IDE Integration Build Management	•How easy is it to integrate a process into development?
Framework Support	•Data flow analysis requires knowing „Sources“ and „Sinks“. •Input methods for frameworks. •Validators for frameworks

CODE REVIEW AS A PROCESS

Secure SDLC via OpenSAMM

12 Security Practices



- **Idea: Perform SDLC Benchmark on your company and compare with others (anonymously)**

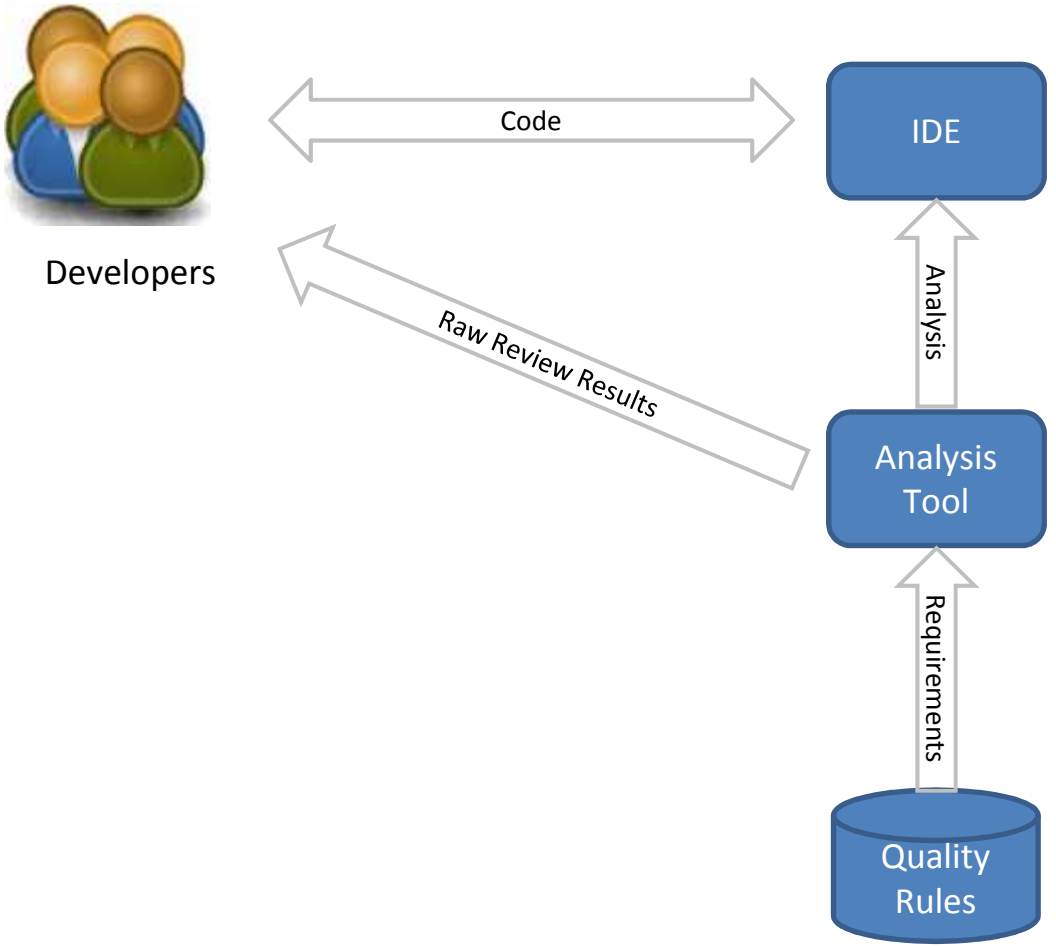
Scenario: Local Review (Worst Case)

- Developers are overloaded with tool output

- No central learning or feedback

- No independent review

- Unstructured



Scenario: Expert Review Team

+ Developers see no false positives

+ Central learning & feedback

+Independent review

+ Structured

CONCLUSIONS

Static Analysis: Advantages/Disadvantages

Advantages

- Scales well for large projects
- Many tools available
- Repeatable results
- Very good for finding simple patterns

Disadvantages

- Hard to find complex problems such as authentication, authorization and access control.
- High numbers of false positives.
- Frequently can't find configuration issues, since they are not represented in the code.

Conclusions

- Static analysis is a powerful method for examining code that can detect many problems that human reviewers would not find.
- Static analysis is becoming a standard part of the SDLC.
- The quality of the results depends strongly on the tools, code type and reviewer.
- Setting up a code review system at the enterprise level requires substantial effort and expertise.

Vielen Dank

für Ihre Aufmerksamkeit

Dr. Bruce Sams bruce.sams@optimabit.com

OPTIMA..bit
business information technology