OWASP Static Analysis (SA) Track

Session 1: Intro to Static Analysis

Eric Dalci
Cigital
edalci at cigital dot com

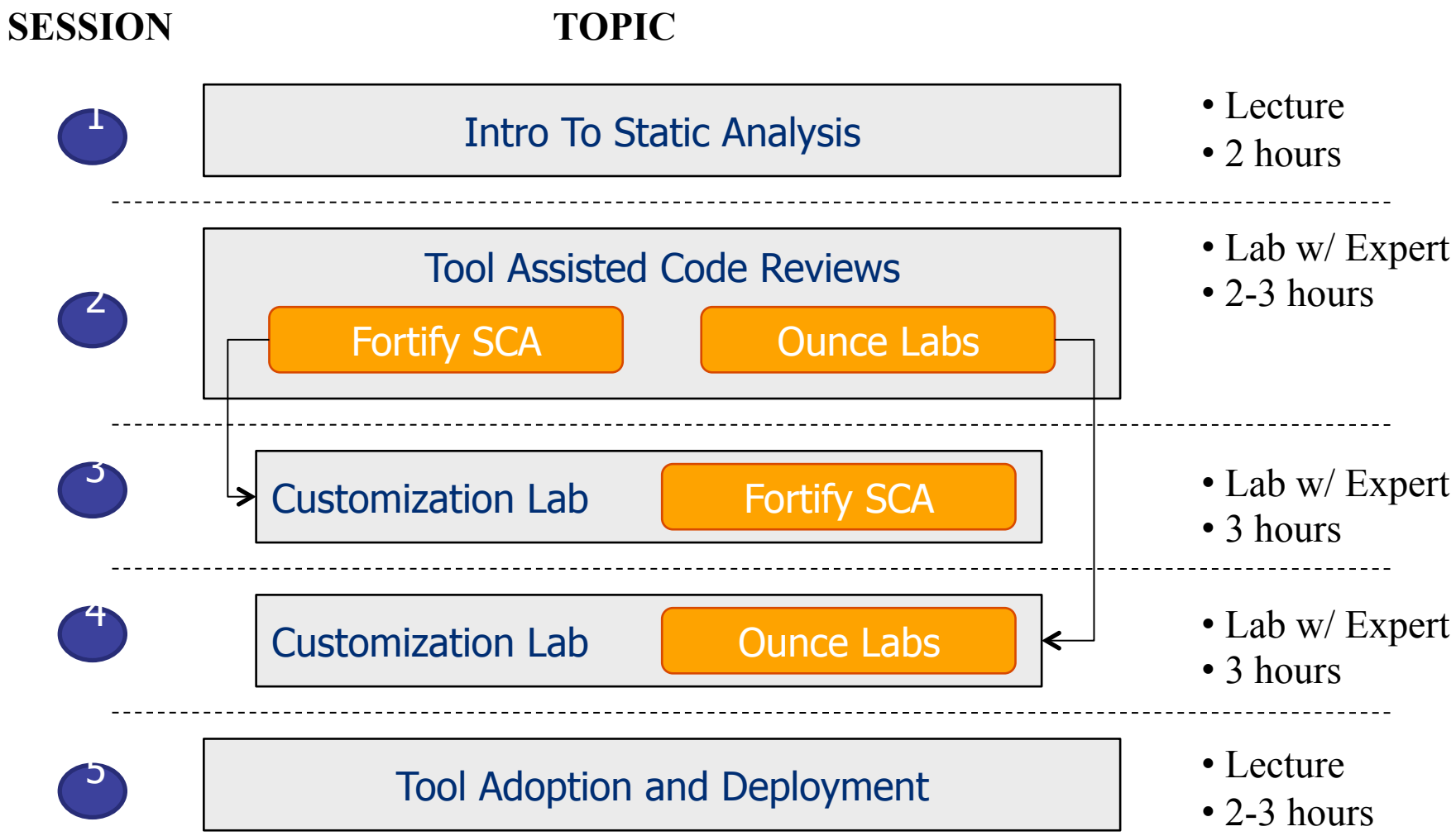
OWASP
5/07/09

The OWASP Foundation
http://www.owasp.org

OWASP SA Track: Goals

- Cover the ins and outs of **Static Analysis**
  - Who, What, When, Where, How, Why
- Provide hands-on experience using commercially available tools
- Provide hands-on tool customization guidance
- Provide guidance on organizational adoption and integration of SA into your SDLC

OWASP SA Track Roadmap



Background

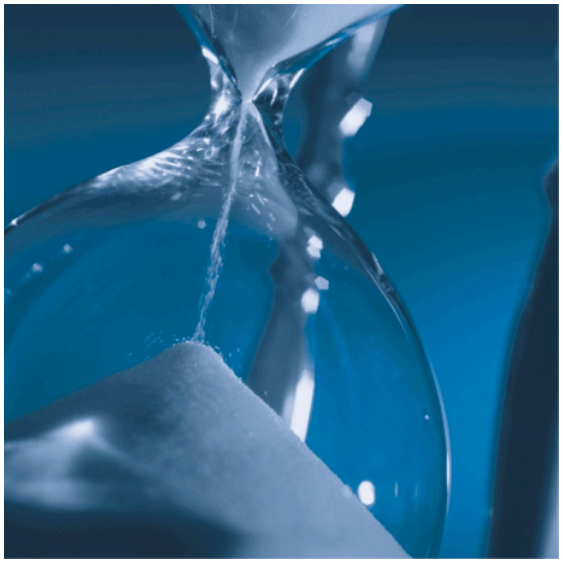
- Work at Cigital Inc.
- And previously at NIST
  - National Institute of Standards and Technology
    - Software Quality Group
    - Software Security Group
- I save the ugly baby…

Objectives

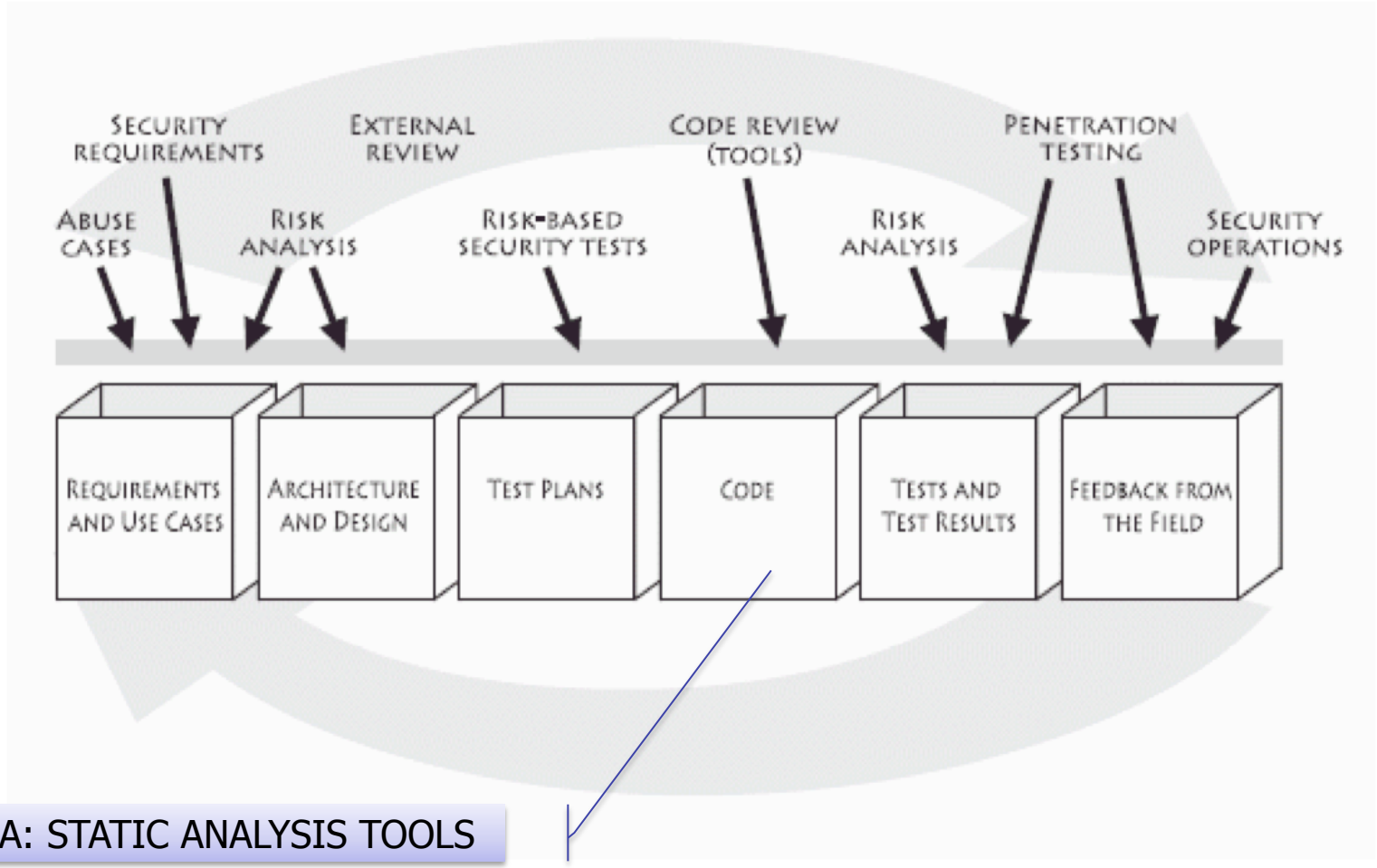
- Understand why you should be using a Static Analysis tool to perform secure code review.
- Know what type of vulnerabilities you can scan for with a Static Analysis tool.
- Know the limits and strengths of Static Analysis Tools

Agenda

- Automated "Secure" Code Review
- Exercise – Manual Code Review
- Static Analysis



Catching a bug: Opportunities



Bug vs. Flaw

Implementation bug

50%-50%

Architecture flaw

Source: http://www.flickr.com/photos/sensechange/521943309
Source: http://www.flickr.com/photos/savetheclocktower/172724622

Attacks on the Application Layer

- According to Gartner and CERT, **75 percent** of security breaches occur at the Application layer.
- And from 2005 to 2007 alone, the U.S. Air Force says application hacks have increased from 2 percent to 33 percent of the total number of attempts to break into its systems.

Source: Gartner IT Services Forecast, 2007

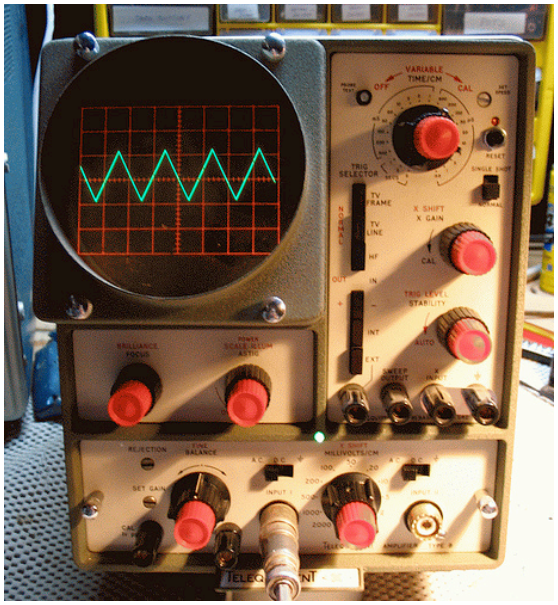
What's in the code?

- Assumptions
  - Ex : "This function call will never fail"
- Function calls
  - Ex : "X calls Y which calls Z which calls System.exit()"
- Settings
  - Ex: "Forward requests from www.blah.com/admin to the servlet userRequest"
- Input data handling
  - Ex: "Hello ${userInput.name} !"
- Error handling
  - Ex: Catch(Exception err) {

    System.out.println("Something bad happened:"+ err.printStackTrace()) }
- Vulnerabilities ?

Type of automated code analysis

- Type checking
- Style checking
- Property Checking
- Program understanding
- Bug finding (Quality)
- **Security Review**



Automated Code Review

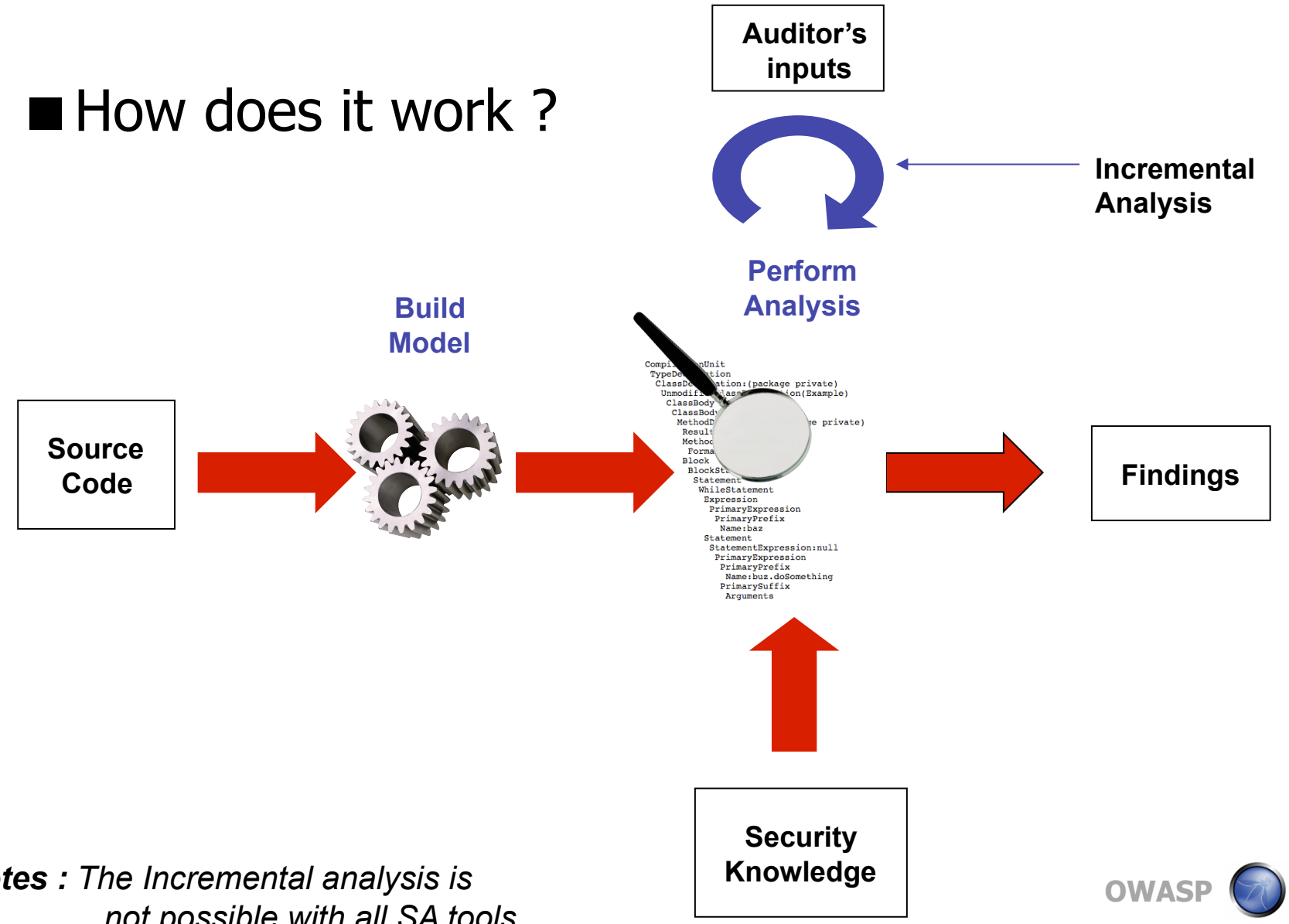
- Automated Code Review (Pros and Cons)
  - [+] Relatively Fast
  - [+] Can be very efficient (high number of findings)
  - [+] Integrated with IDE
    - trace analysis, bug information, etc.
  - [+] Bring Security knowledge to the developers
    - Propose remediation to bugs
  - [+] Consistent

badness-ometer

  - [-] Require human intervention to discriminate false/true positive
  - [-] High level of false positives
  - [-] Many false negatives remain (depending on the tool's coverage)
  - [-] False sense of security

Static Analysis Internals

- How does it work ?



Notes : *The Incremental analysis is not possible with all SA tools*

Code review : Let’s find some bugs

EXERCISES

Exercise: Security Review - 1/5

- What's wrong with this code?

```
#define MAXSIZE    40
int main(int argc, char **argv)
{
     char buf[MAXSIZE];
       if(gets(buf))
         printf("result: %s\n", buf);
       return 0;
}
```

- The problems could be found with
  - Semantic analysis
  - Data flow analysis

Exercise: Security Review - 2/5

- The following XML configuration file setup the session timeout for a web application.
- What's wrong with this setting ?

```
<web-app>
  <session-config>
    <session-timeout>180</session-timeout><!-- time in minutes -->
  </session-config>
</web-app>
```

- This could be discovered with a configuration analysis (Xpath)

Exercise: Security Review - 3/5

- What's wrong with this code?

```
char* ptr = (char*)malloc (SIZE);
 if (abrt) {
     free(ptr);
 }
 free(ptr);
```

- This could be found with a control flow analysis

Exercise: Security Review - 4/5

- What's wrong with this code?

```
Public static boolean getUserSSN(String Id)          {
Connection con = null;
Try{
    //… instantiate Connection
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("Select ssn FROM tuserssn
      WHERE id ="+ Id);
    While (rs.next())          { //…Process the query results}
    }
```

- This could be found with data flow analysis

Exercise: Security Review - 5/5

- What's wrong with this code?

```
public class RegisterUser extends HttpServlet
{
    String UserName;
    protected void doPost (HttpServletRequest req, HttpServletResponse
    res)
    {
     UserName = req.getParameter(" UserName ");
    //process UserName
    out.println(UserName + ", thanks for visiting!");
    }
}
```

- This could be found with structural analysis

STATIC ANALYSIS

Code level analysis by Static Analysis tools Examples

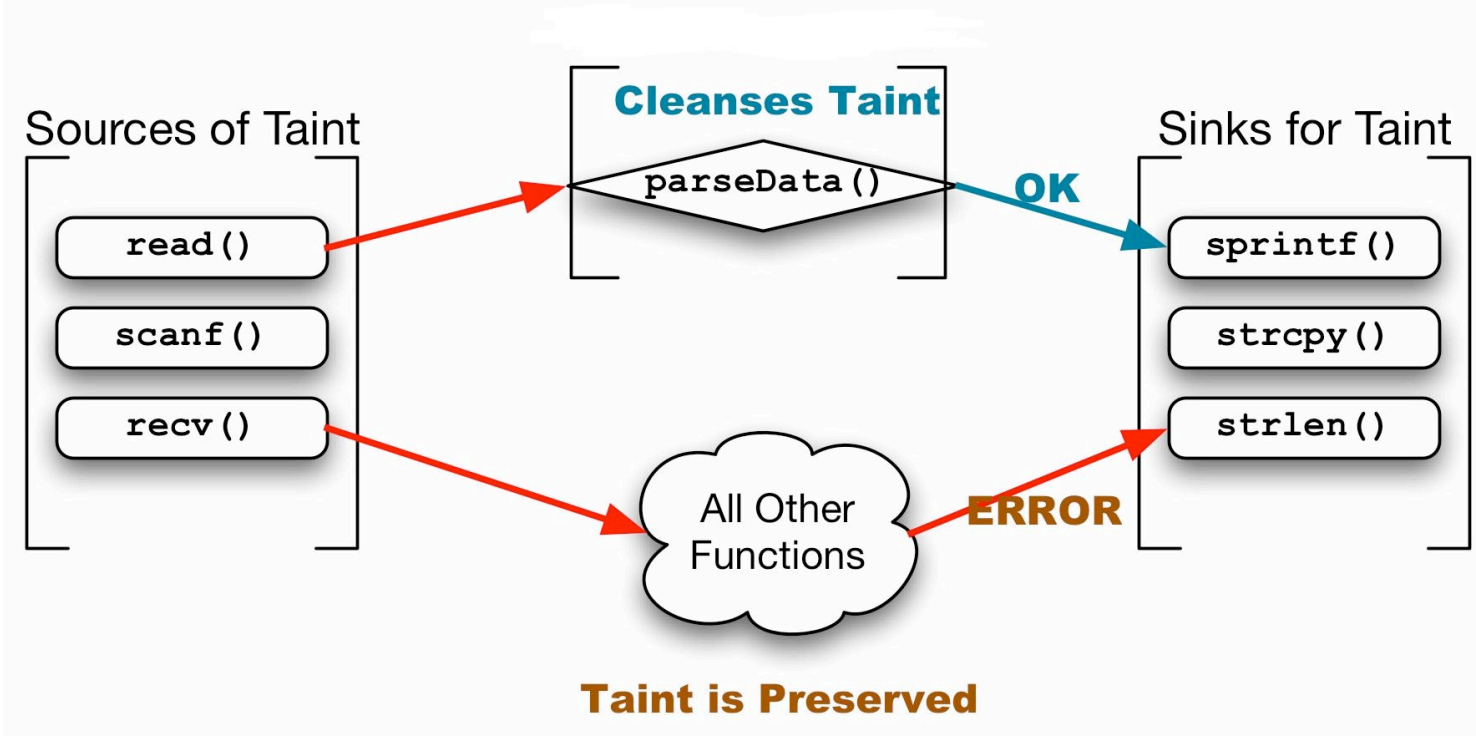
- Data Flow
  - Track user data. Great for spotting SQL injection, XSS, etc.
- Control Flow
  - State machine (Safe State, Error State, etc.)
- Structural
  - Identifies vulnerable code structure
- Semantic Analyzer
  - "Glorified" grep
- Configuration
  - Scan XML and .properties files
- Etc.

Data Flow Analysis

- Data flow analysis tracks data from its source to its consumption site.
- For a web application, data flow analysis is probably the most relevant as it is able to follow untrusted user input.
- Data originates from Source type of function
- Data is being consumed (e.g. interpreted) in Sink type of function.
- Entry points are directly accepting user controlled data (i.e. Inbound taint)
- Data flow analysis uses taint propagation techniques.

The Data-Flow Model

- Taint can have different origin (user input, property files, database, etc.)
- Tainted Data flows between Sources and Sinks.

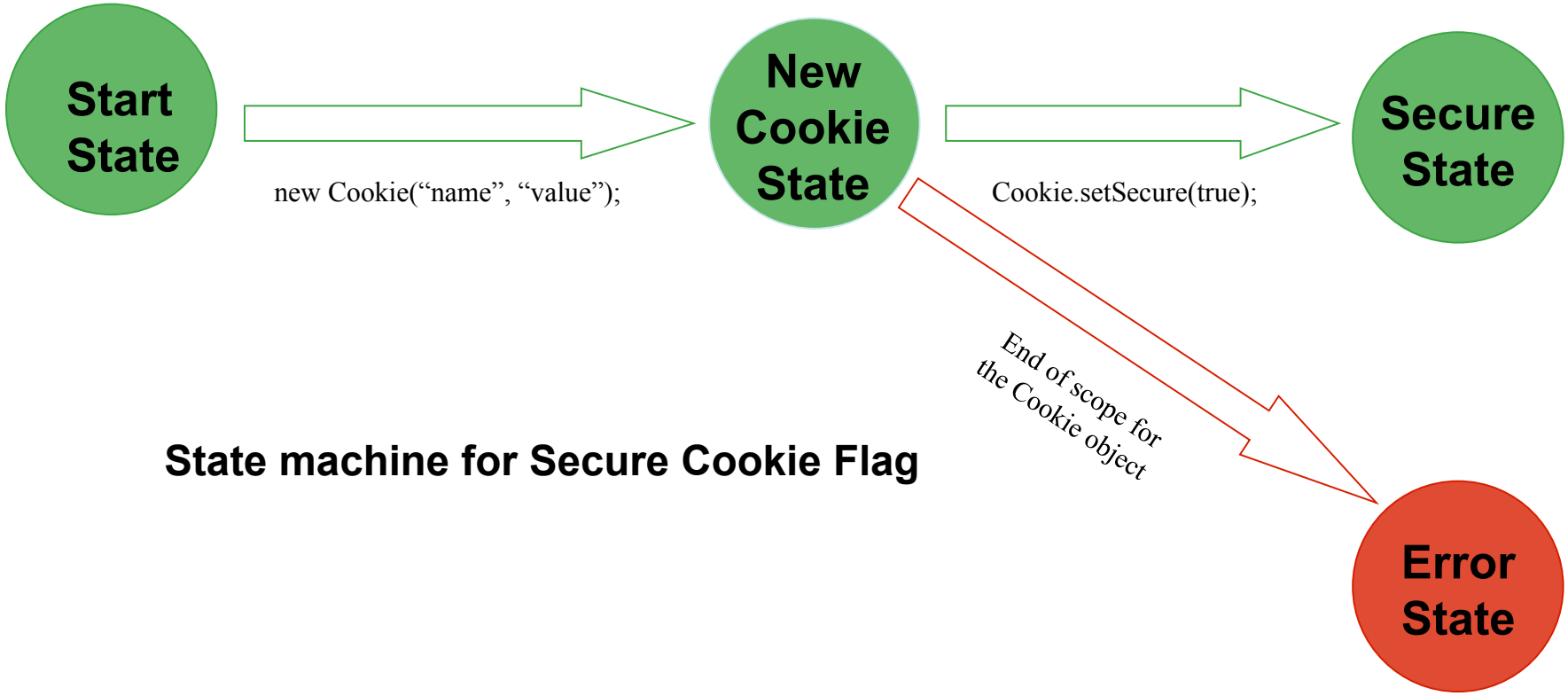


Control flow

- The control flow analysis is the analysis of state and transition. We can represent a control flow using a state machine.
- Control flow analysis is good for finding race condition type of problem where sequence of calls matters.
- Examples:
  - Open and close a resource
  - Validate and invalidate a session ID

Control Flow Model : A State Machine

- Rule : Call "setSecure" if new Cookie is created.



State machine for Secure Cookie Flag

Structural analysis

- Structural analysis refers to a particular code construct
- The structural analysis can involve relation between Classes (e.g. inheritance, Class type, etc.)
- Language specific code construct could be analyzed for vulnerabilities or quality issues.
- For instance in Java, code construct such as try/ catch blocks, member field assignment, method with specific signature, return statements, etc. would be recognized.

Semantic Analysis

- The Semantic of code relate to the meaning of a particular code within its context.
  - Ex: The Class *Animal.Insect.Bug* is different from the Class *Software.Security.Bug*
- The ancestor of semantic analysis is grep
- Example:
  - In C code, a semantic analysis would find all instance of “gets()”

Configuration

- A configuration analysis applies to XML or property files.
- Typically properties are set in deployment descriptor.
- It is possible to overload/create a property at runtime, but then we will need other type of analysis to handle that code.

Exercise : What type of analysis would you apply ?

- Check that the code always call *"produce()"* before *"consume()"*
- Check that there are no clear text password encoding in property files
- Check that no User controlled data ends up in the variable *"command": Runtime.getRuntime().exec(command)*
- Check that "*unsafeEncrypt()"* never get used.
- Check that all the finally blocks have the necessary clean up code *"buffer.flush()".*

OWASP top 10 & possible corresponding analysis

1.	Cross Site Scripting (XSS)	1. Data Flow Analysis
2.	Injection Flaws	2. Data Flow Analysis
3.	Malicious File Execution	3. Data Flow Analysis
4.	Insecure Direct Object Reference	4. Data Flow Analysis
5.	Cross Site Request Forgery (CSRF)	5. NA
6.	Information Leakage and Improper Error Handling	6. All 5 analysis
7.	Broken Authentication and Session Management	7. Control Flow
8.	Insecure Cryptographic Storage	8. Structural, Semantic
9.	Insecure Communications	9. Structural, Semantic
10.	Failure to Restrict URL Access	10. Configuration

SA tools' Dirty little secret

- Without special engineering, SA tools **can't follow the flow of control or data when it's not explicit in the code.**
- For Web 2.0 and mashup don't even ask the news is even worse.

Resources: Spring Framework's vulnerability
www.springsource.com/securityadvisory

More examples of SA tools' limits

- Ignore what you do well and their impact to the rest of the findings
  - Ex. .NET Request Validation is turned on, but the tool ignore it and report injection type of problems.
- No bridge between **declarative and programmatic security**
  - Ex: XML, <Property secure="true"/>
  - Ex: Code, Property.set("secure","false")

And more…

- We talked about what you can't cheaply detect…
  - 'business logic' problems
  - **Flaws**
  - Just because it was detected, doesn't mean it's **exploitable** (or discoverable, externally)

Tool coverage

	Visible in the Code	Visible in the design
Generic defects	SA tools' sweet spot. Tools' built-in rules should find those issues. *Ex: Buffer Overflow*	Most likely found through Architecture analysis Ex: The program sends credentials in clear text
Context-Specific defects	The tools needs to be customized to understand context specific functions and rules. *Ex: Processing of Trade order*	Require understanding of general security principles and context specific knowledge Ex: *Trading data not sanitized properly for Personal information and visible to third party*

How to improve a tool's results?

- Customize (Rules, Engines, Filters, etc.)
- Extend the tool's coverage: Write custom rules
  - Access the engine API
  - Use given rule grammar to write new rules
- Feed information to the model (dynamic model change)
  - Example: defining validation functions

Future evolution

- SA Tool should help code **understanding**
- SA Tools should help manual code review (Hybrid code review). They should point to interesting part of the code (e.g. "**Point of Interests**")
- Rule **extension** should be easier
- Code **visualization** should help architecture review
- Querying the SA Model should almost be like natural languages (maybe like a search engine….Google you code !?)

Q/A

Thank you !