Iron Chef:

John Henry Challenge

Brian Chess
Pravir Chandra

Sean Fay
Jacob West

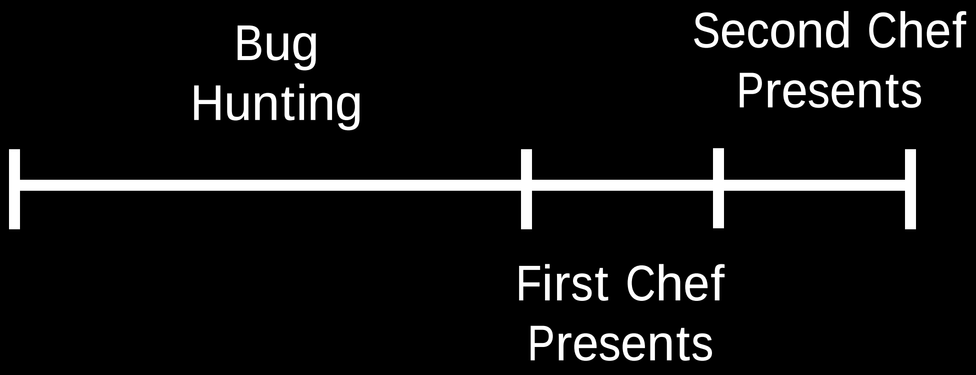
Black Hat
3/27/2008
Amsterdam

Concept

- We love Iron Chef.
- We can’t cook.

Concept

- Compare tools and manual code review in head-to-head “bake off”
- Rules:
  - 45 minutes to find vulnerabilities in the same program
  - Chef with tools can only use tools he has written
  - Secret ingredient: the code!
  - Present results to a panel of celebrity judges
- Judging:
  - Quality of findings
  - Originality
  - Presentation



Bug Hunting

Second Chef Presents

First Chef Presents

Chefs

Name: Pravir Chandra

Specialty: Manual code review

Job: Principle, Cigital

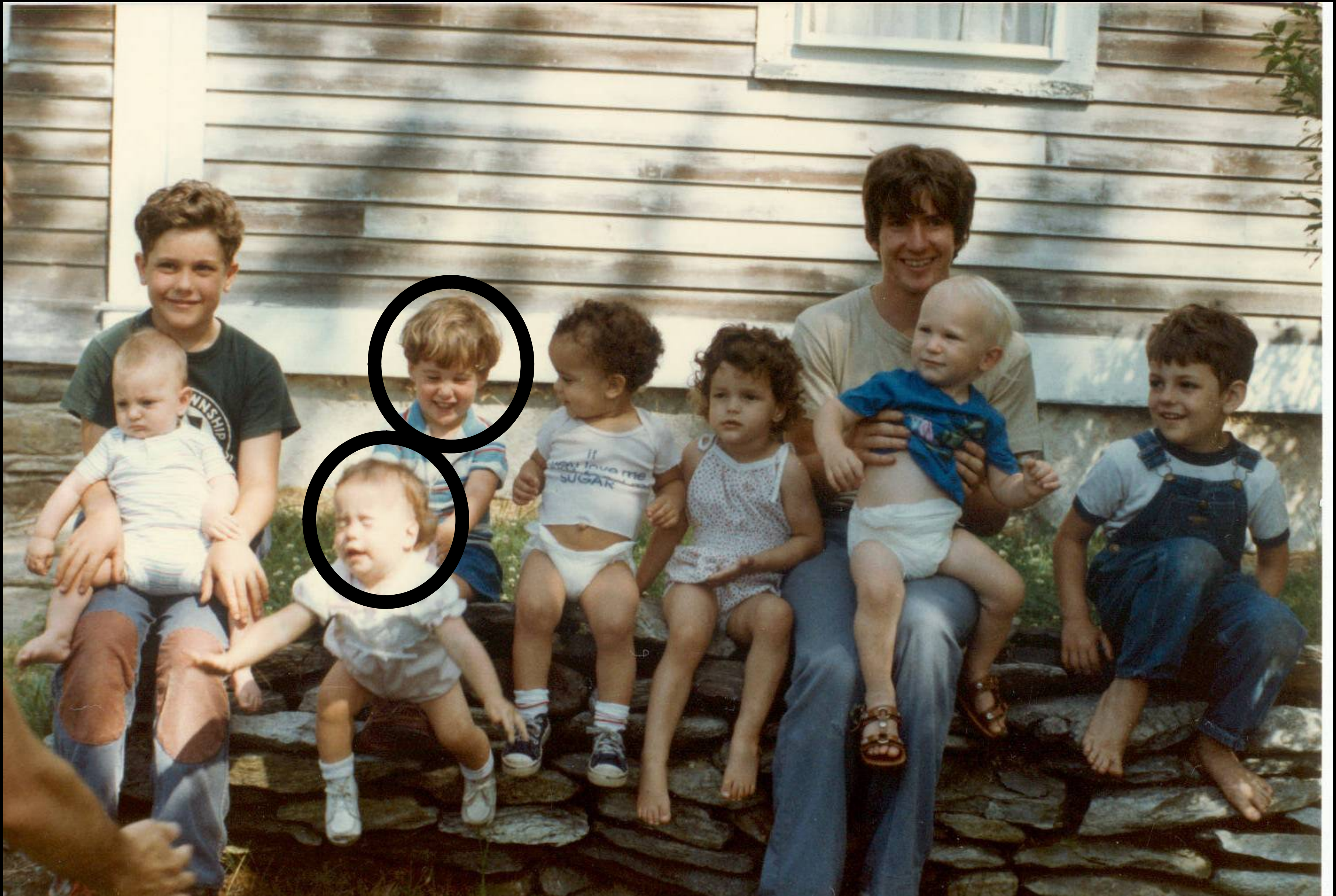
Chefs

Name: Sean Fay

Specialty: Static and runtime analysis

Job: Chief Architect, Fortify Software

Sean Fay



Chefs

Chefs

Chefs

- After judging, you point out bugs these guys missed

Judges

TBA

TBA

TBA

Secret Ingredient

Name:

Version:

Language:

Size:

Home:

Overview:

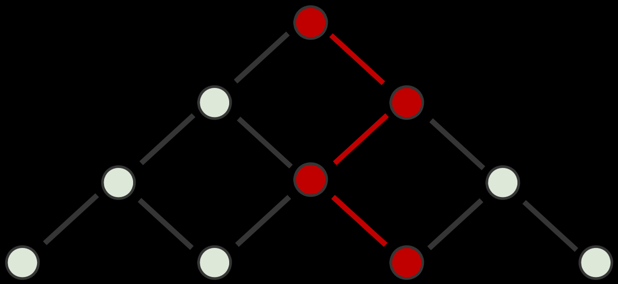
< start >

Runtime Analysis

Black Hat
3/27/2008
Amsterdam

Dynamic Taint Propagation

- Follow untrusted data and identify points where they are misused



Example: SQL Injection

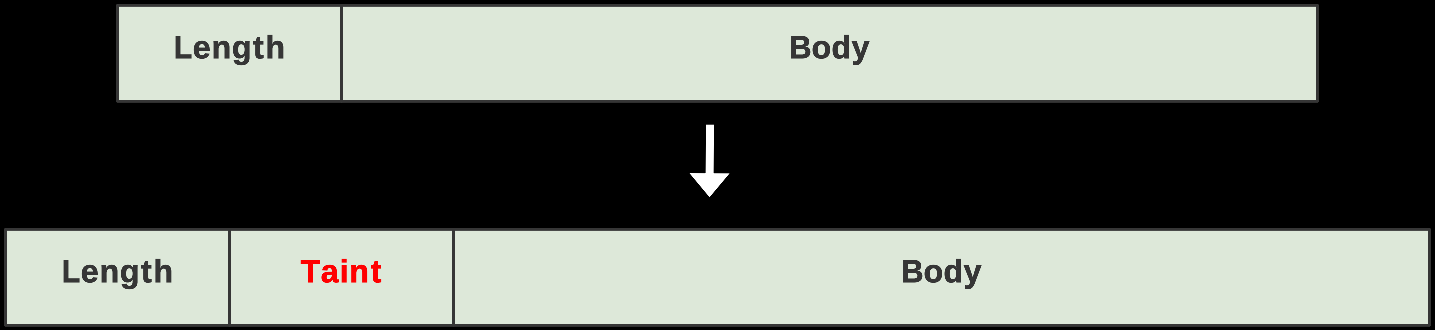
```
...
user = request.getParameter("user");
try {
 sql = "SELECT * FROM users " +
       "WHERE id='" + user + "'";
 stmt.executeQuery(sql);
}
...
```

Tracking Taint

1. Associate taint marker with untrusted input as it enters the program
2. Propagate markers when string values are copied or concatenated
3. Report vulnerabilities when tainted strings are passed to sensitive sinks

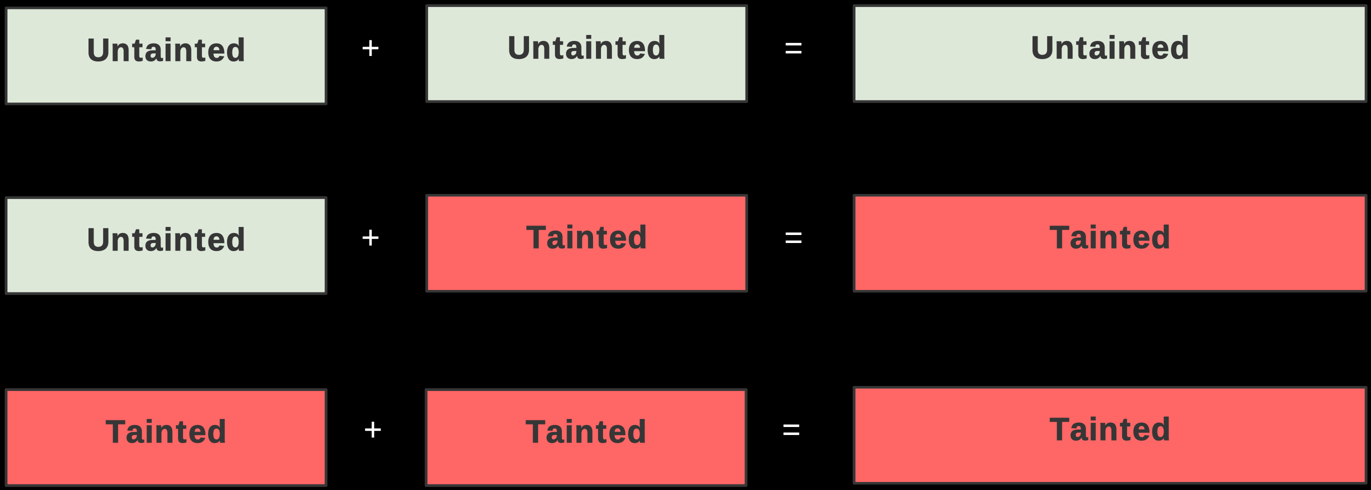
Java: Foundation

- Add taint storage to `java.lang.String`



Java: Foundation

- **`StringBuilder`** and **`StringBuffer`** propagate taint markers appropriately



Untainted + Untainted = Untainted

Untainted + Tainted = Tainted

Tainted + Tainted = Tainted

Java: Sources

- Instrument methods that introduce input to set taint markers, such as:
  - **`HttpServletRequest.getParameter()`**
  - **`PreparedStatement.executeQuery()`**
  - **`FileReader.read()`**
  - **`System.getenv()`**
  - **`...`**

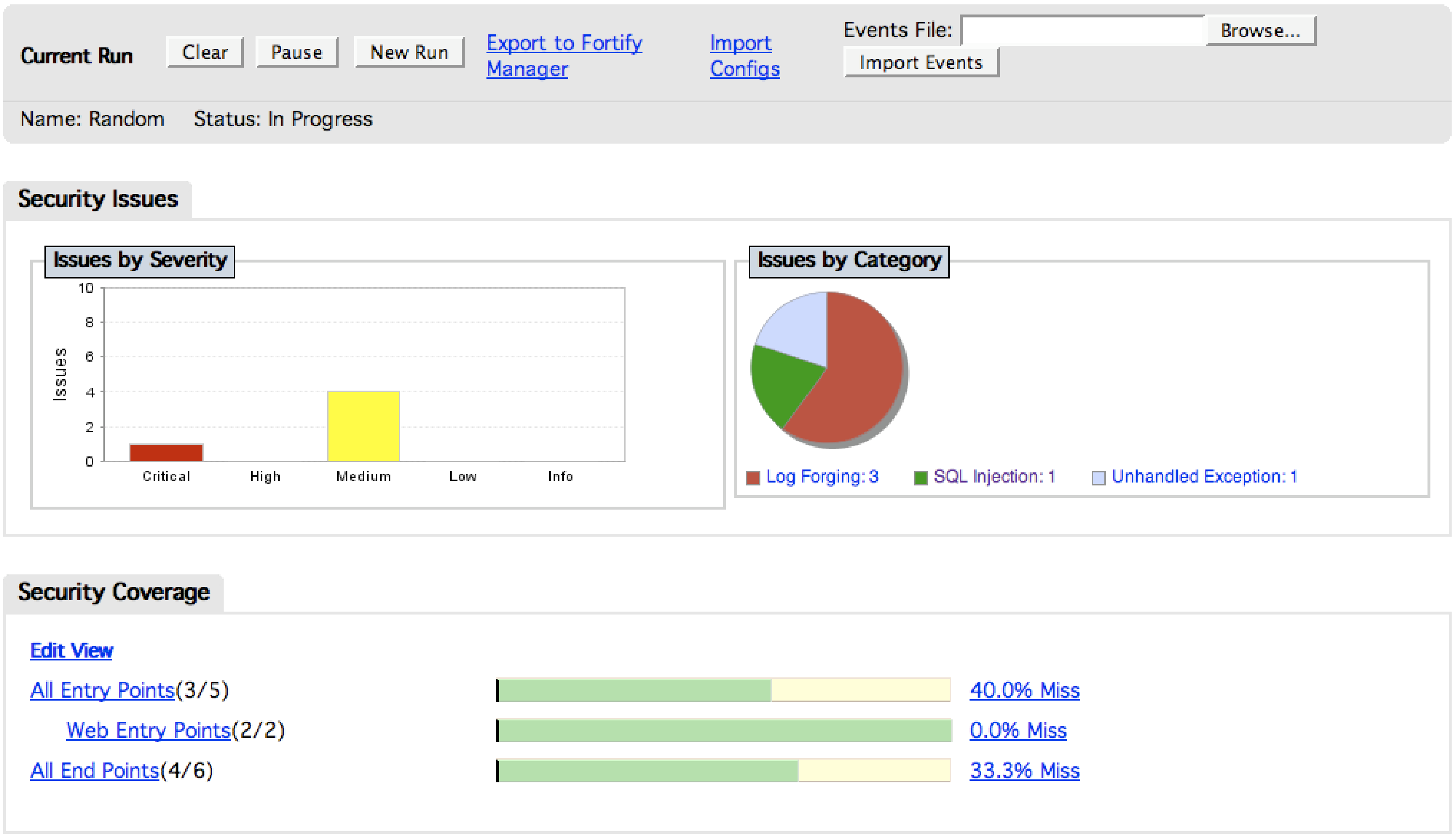
Java: Sinks

- Instrument sensitive methods to check for taint marker before executing, such as:
  - **`Statement.executeQuery()`**
  - **`JspWriter.print()`**
  - **`new File()`**
  - **`Runtime.exec()`**
  - **`...`**

Example: SQL Injection

```
user = request.getParameter("user");
TaintUtil.setTaint(user, 1);
try {
 sql = "SELECT * FROM users " +
       "WHERE id='" + user + "'";
 TaintUtil.setTaint(sql,user.getTaint());
 TaintUtil.checkTaint(sql);
 stmt.executeQuery(sql);
}
```

Results Overview



Name: Random Status: In Progress

Security Issues

Issues by Severity

Issues by Category

🟥 Log Forging: 3 🟩 SQL Injection: 1 □ Unhandled Exception: 1

Security Coverage

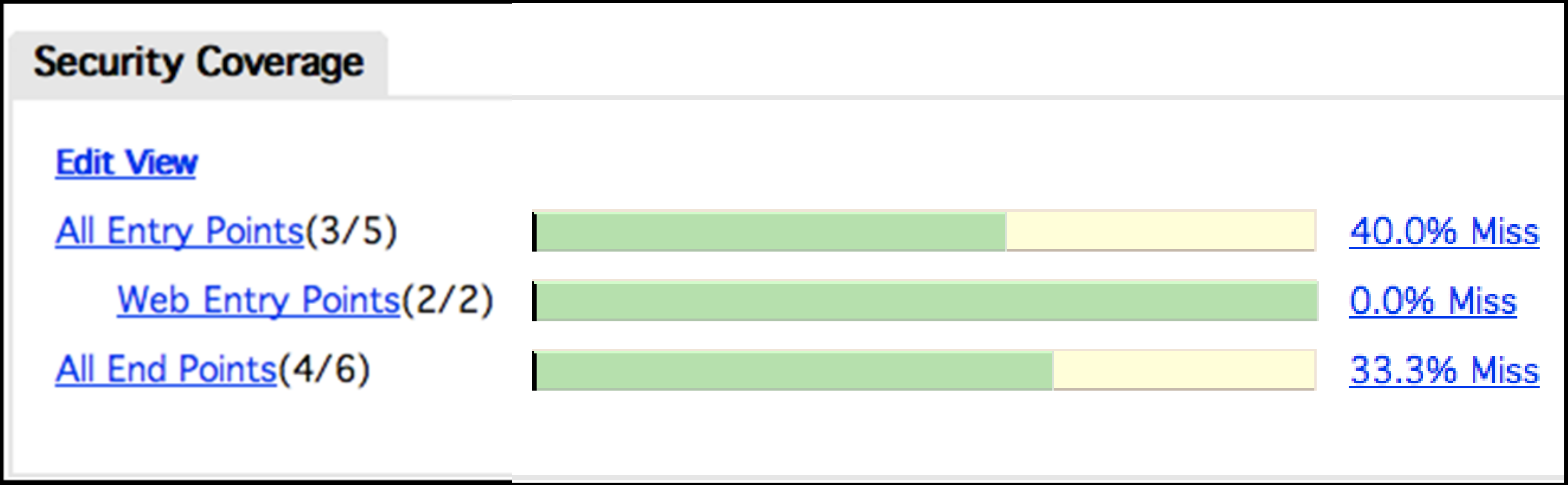
Edit View

All Entry Points(3/5) 40.0% Miss

Web Entry Points(2/2) 0.0% Miss

All End Points(4/6) 33.3% Miss

Security Coverage

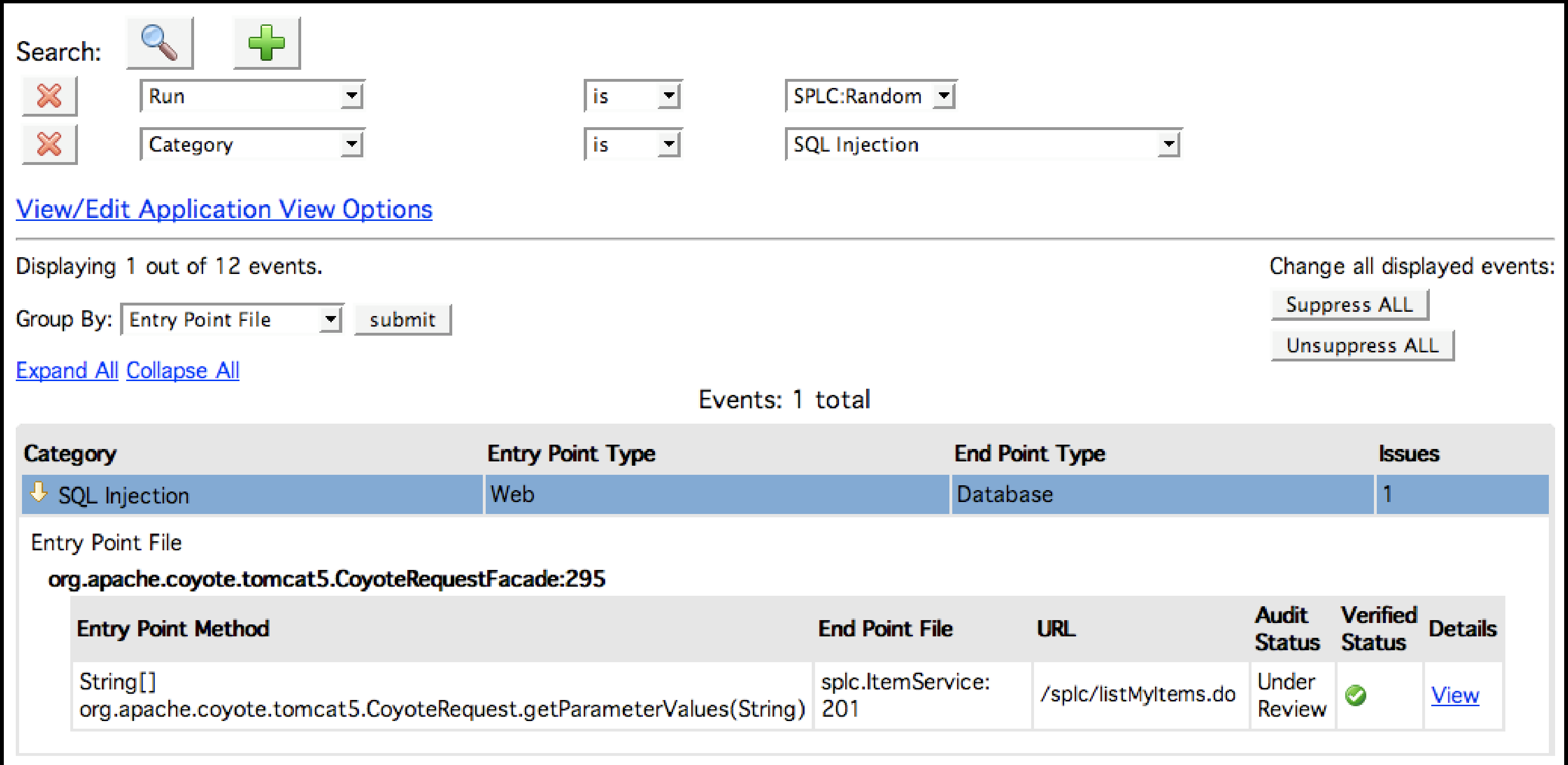


Security Coverage

Edit View

All Entry Points(3/5)	40.0% Miss
Web Entry Points(2/2)	0.0% Miss
All End Points(4/6)	33.3% Miss

SQL Injection Issue



☒	Run	is	SPLC:Random
☒	Category	is	SQL Injection

View/Edit Application View Options

Displaying 1 out of 12 events.

Group By: Entry Point File

Expand All Collapse All

Change all displayed events:

Events: 1 total

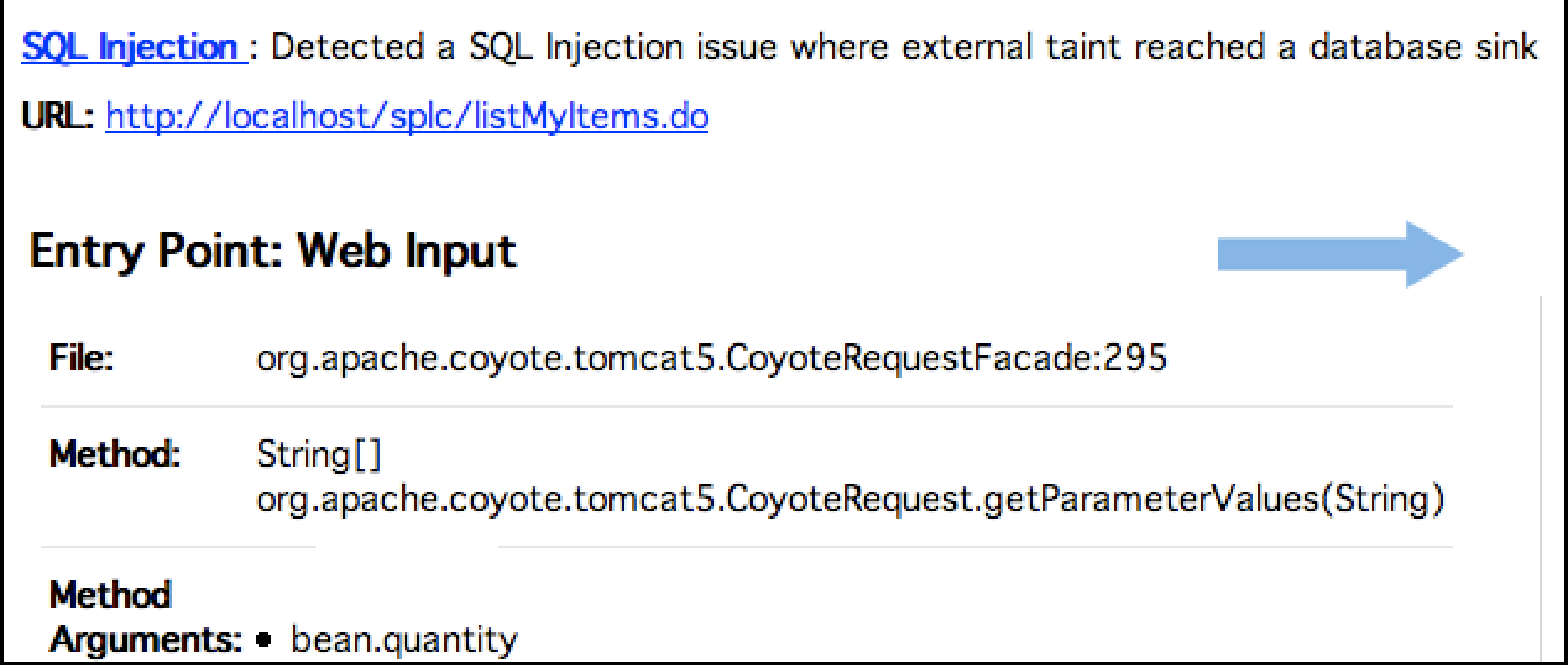
Category	Entry Point Type	End Point Type	Issues
SQL Injection	Web	Database	1

Entry Point File

org.apache.coyote.tomcat5.CoyoteRequestFacade:295

Entry Point Method	End Point File	URL	Audit Status	Verified Status	Details
String[] org.apache.coyote.tomcat5.CoyoteRequest.getParameterValues(String)	splc.ItemService: 201	/splc/listMyItems.do	Under Review	✓	View

Source



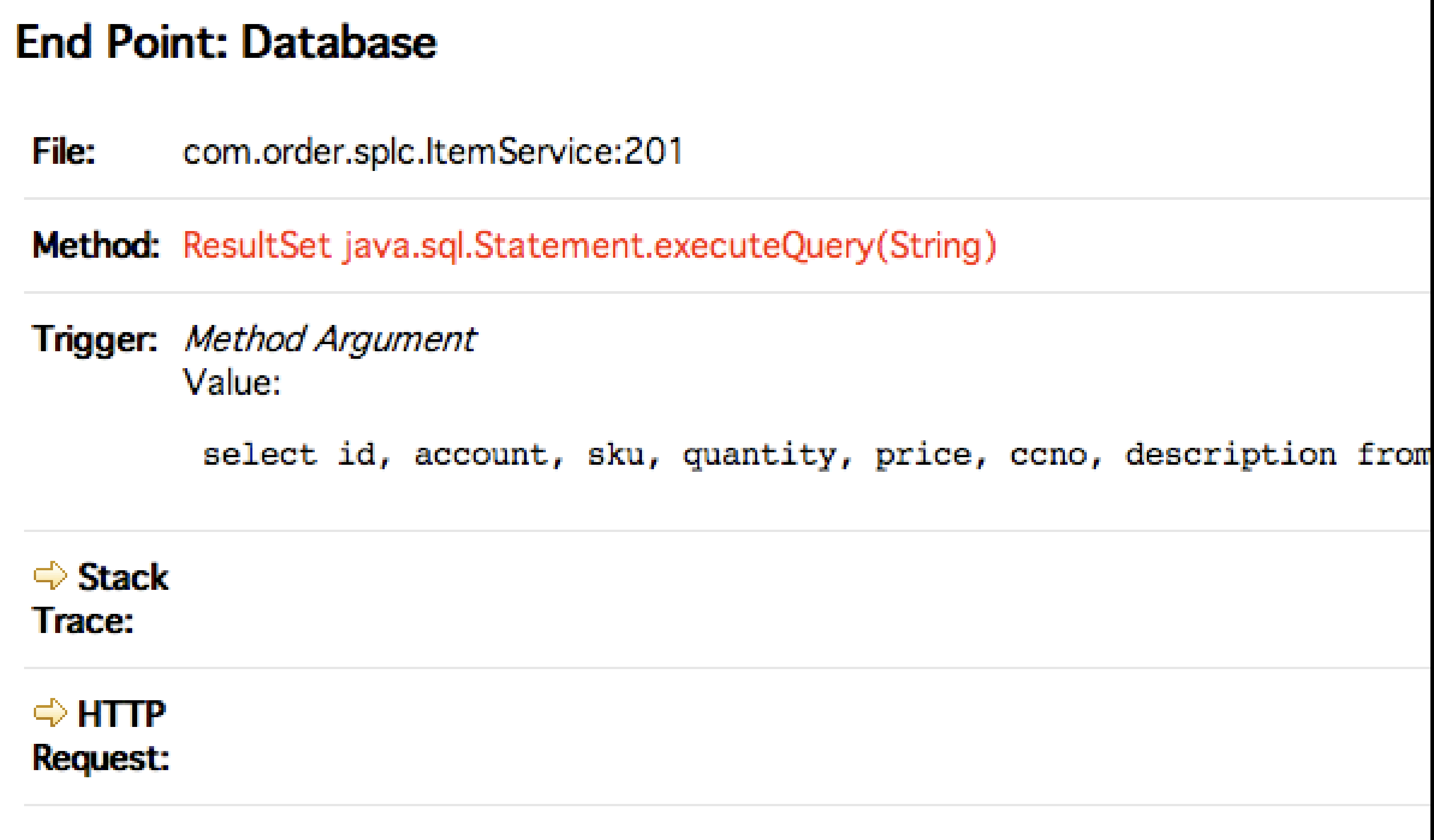
SQL Injection : Detected a SQL Injection issue where external taint reached a database sink

URL: http://localhost/splc/listMyItems.do

Entry Point: Web Input

File:	org.apache.coyote.tomcat5.CoyoteRequestFacade:295
Method:	String[] org.apache.coyote.tomcat5.CoyoteRequest.getParameterValues(String)
Method Arguments:	• bean.quantity

Sink



End Point: Database

File: com.order.splc.ItemService:201

Method: ResultSet java.sql.Statement.executeQuery(String)

Trigger: *Method Argument*
Value:

```
select id, account, sku, quantity, price, ccno, description from
```

⇨ **Stack Trace:**

⇨ **HTTP Request:**

Where is the Problem?

Severity	Category	URL
Critical	SQL Injection	/splc/listMyItems.do

Class	Line
com.order.splc.ItemService	196

Query	Stack Trace
select * from item where item name = ‘adam‘ and ...	java.lang.Throwable at StackTrace$FirstNested$SecondNested. <init>(StackTrace.java:267) at StackTrace$FirstNested. <init>(StackTrace.java:256) at StackTrace. <init>(StackTrace.java:246) at StackTrace. main(StackTrace.java:70)

Instrumentation

- Instrument JRE classes once
- Two ways to instrument program:
  - Compile-time
    - Rewrite the program's class files on disk
  - Runtime
    - Augment class loader to rewrite program

Aspect-Oriented Programming

- Express cross-cutting concerns independently from logic (aspects)
- Open source frameworks
  - AspectJ (Java)
  - AspectDNG (.NET)
- Could build home-brew instrumentation on top of bytecode library (BCEL, ASM)

Example

```
public aspect SQLInjectionCore extends ... {
 //Statement
 pointcut sqlInjectionStatement(String sql):
  (call(ResultSet Statement+.executeQuery(String))
  && args(sql))
  ...
}
```

Instrument Inside or Outside?

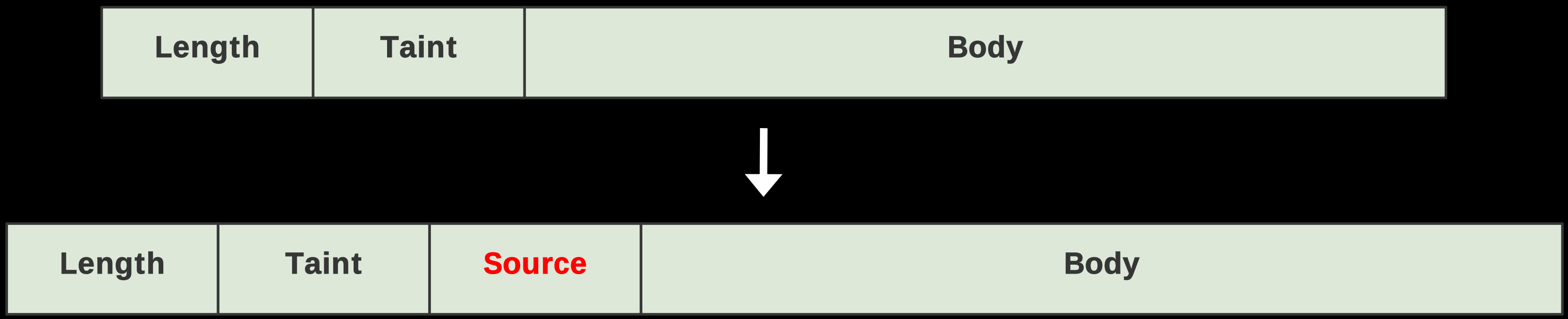
- Inside function body
  - Lower instrumentation cost
- Outside function call
  - Lower runtime cost / better reporting

Types of Taint

- Track distinct sources of untrusted input
  - Report XSS on data from the Web or database, but not from the file system
- Distinguish between different sources when reporting vulnerabilities
  - Prioritize remotely exploitable vulnerabilites

Java: Foundation – Round 2

- Add taint storage and source information to **`java.lang.String`** storage



Length | Taint | Body

Length | Taint | Source | Body

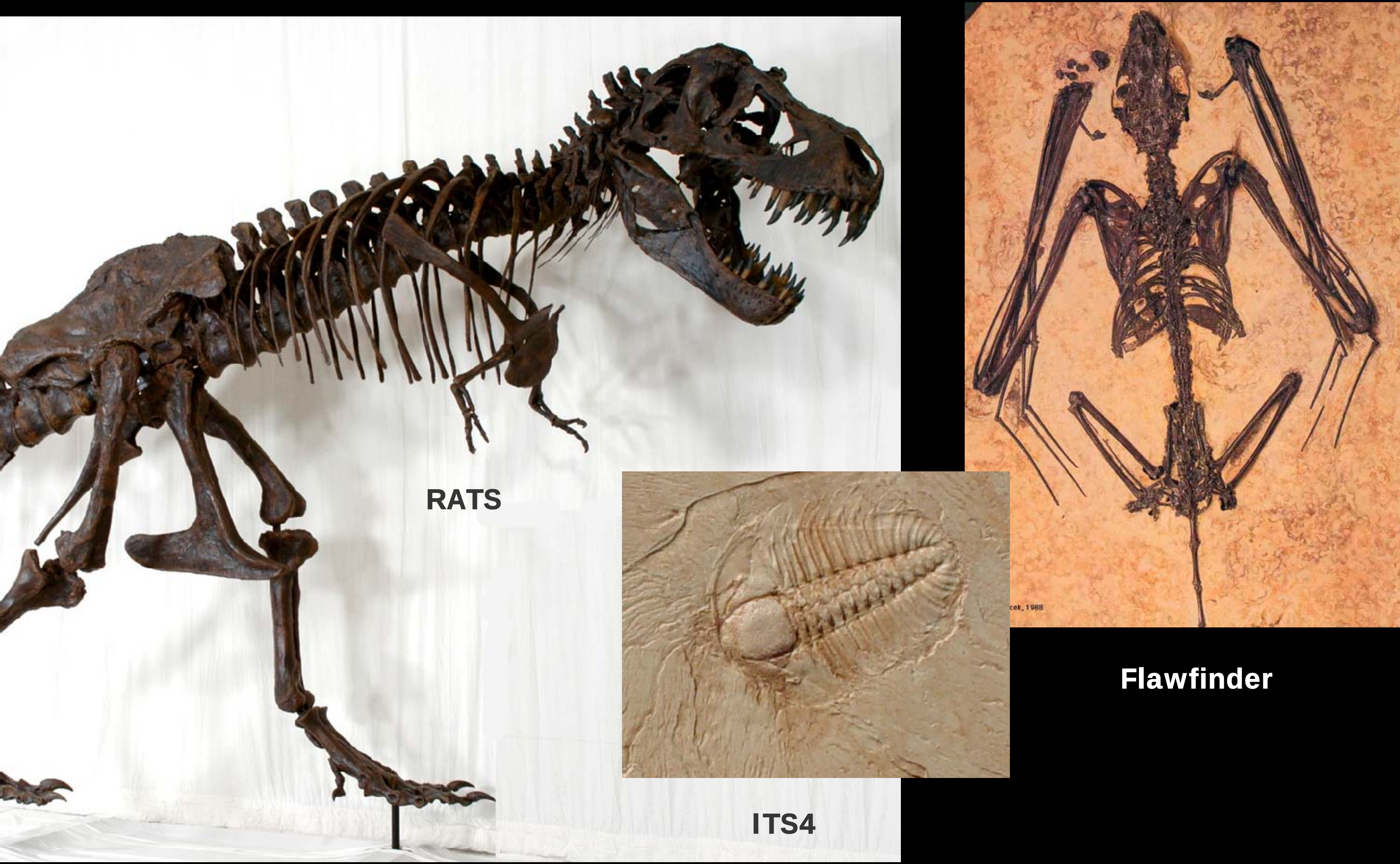
Writing Rules

- Identifying the right methods is critical
  - Missing just one source or sink can be fatal
- Leverage experience from static analysis
  - Knowledge of security-relevant APIs

Static Analysis

Black Hat
3/27/2008
Amsterdam

Prehistoric static analysis tools



RATS

ITS4

Flawfinder

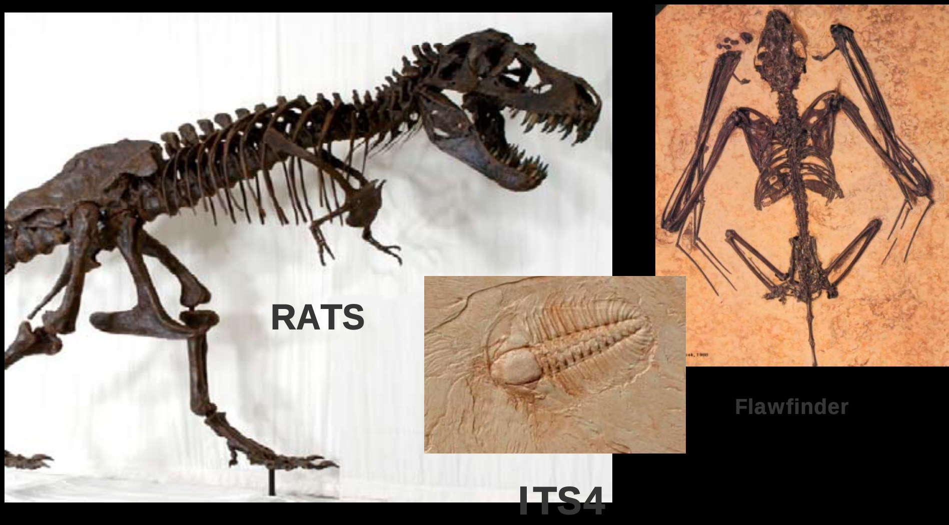
Prehistoric static analysis tools

(+) Good

- Help security experts audit code
- Repository for known-bad coding practices

(-) Bad

- NOT BUG FINDERS
- Not helpful without security expertise

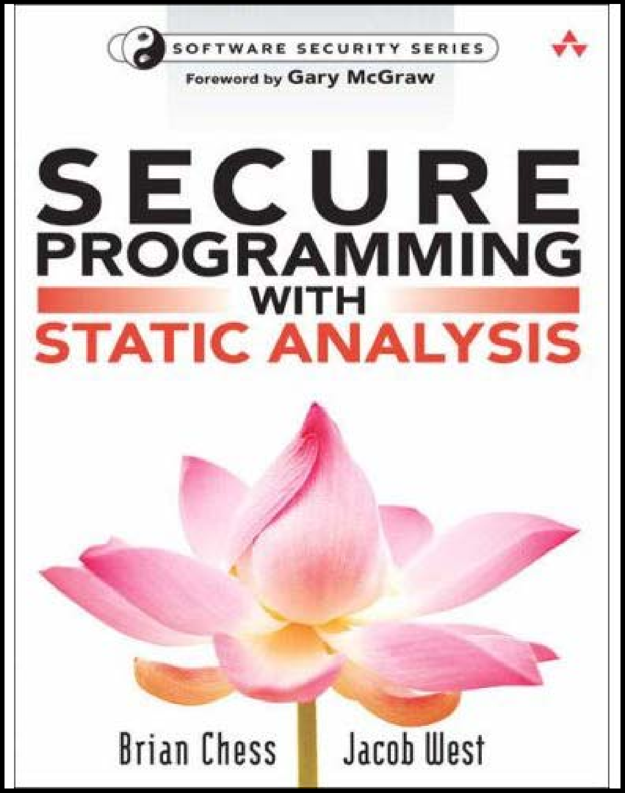


Advanced Static Analysis Tools: Prioritization

```
int main(int argc, char* argv[]) {
    char buf1[1024];
    char buf2[1024];
    char* shortString = "a short string";
    strcpy(buf1, shortString); /* eh. */
    strcpy(buf2, argv[0]);     /* !!! */
     ...
}
```

Static Analysis Is Good For Security

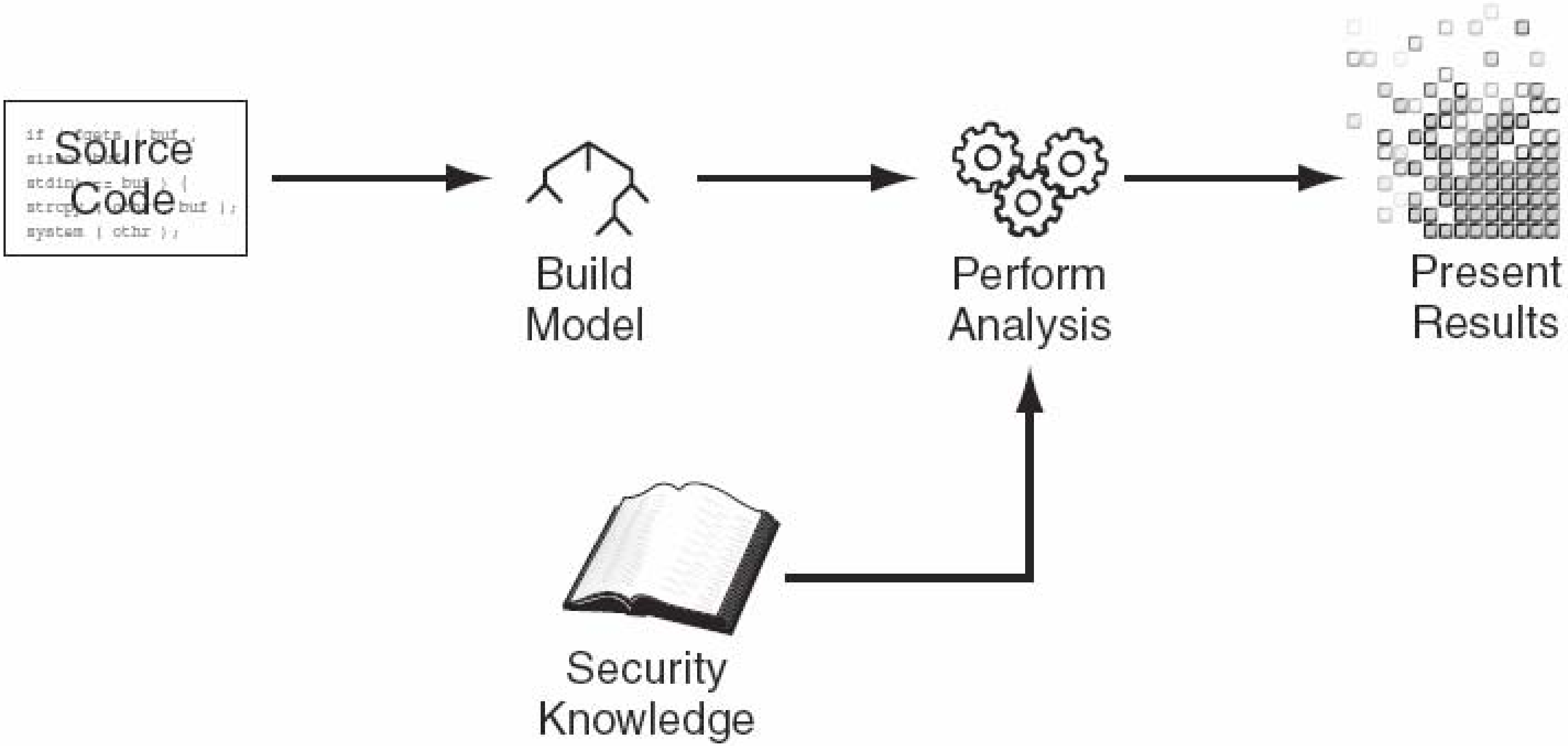
- Fast compared to manual review
- Fast compared to testing
- Complete, consistent coverage
- Brings security knowledge with it
- Makes security review process easier for non-experts
- Useful for all kinds of code, not just Web applications



What You Won’t Find

- Architecture errors
  - Microscope vs. telescope
- Bugs you’re not looking for
  - Bug categories must be predefined
- System administration mistakes
- User mistakes

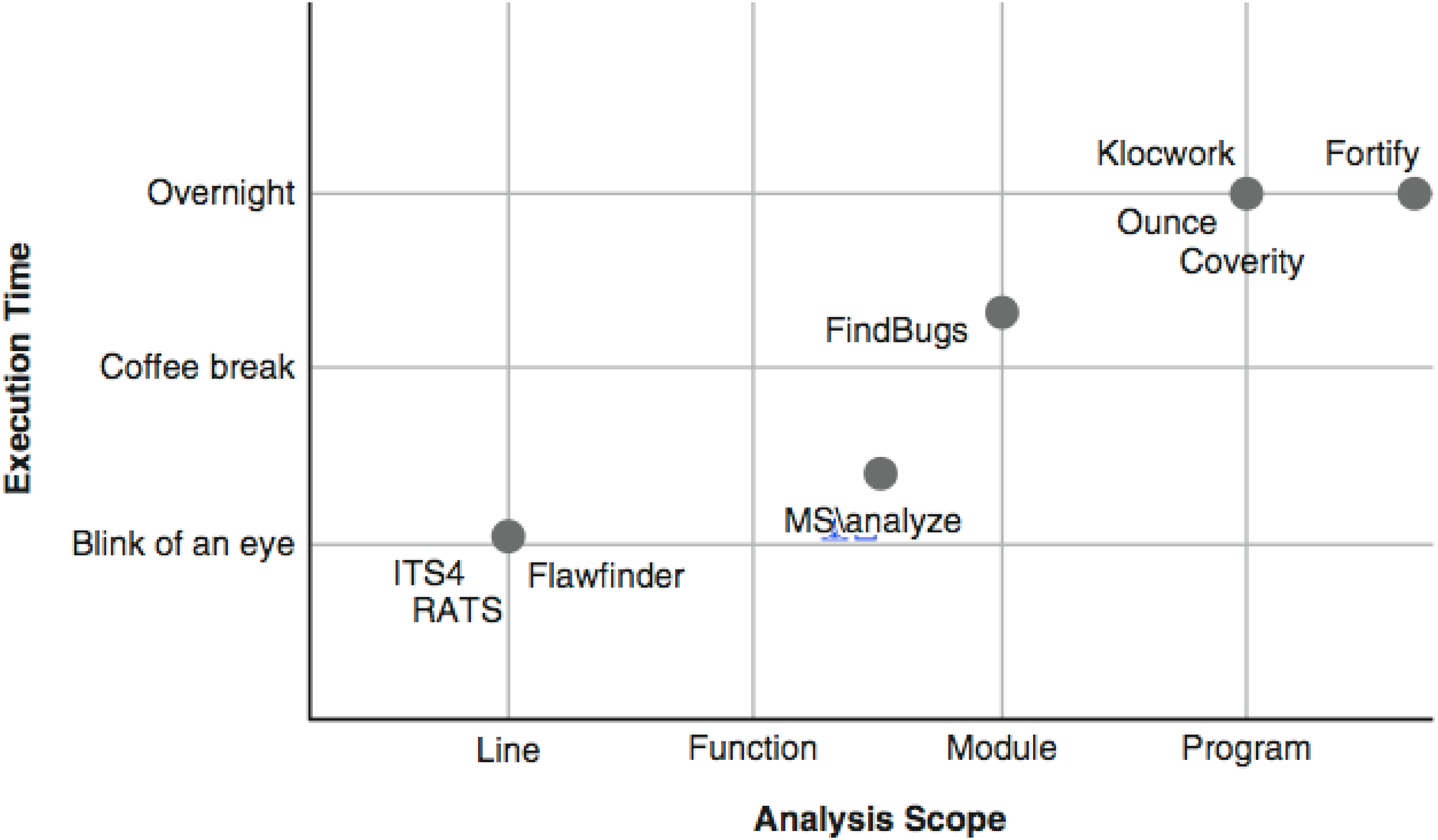
Under the Hood



Building a Model

- Front end looks a lot like a compiler
  - Language support
  - One language/compiler is straightforward
  - Lots of combinations is harder
- Could analyze compiled code...
  - Everybody has the binary
  - No need to guess how the compiler works
  - No need for rules
- ...but
  - Decompilation can be difficult
  - Loss of context hurts. A lot.
  - Remediation requires mapping back to source anyway

Capacity: Scope vs. Performance



Only Two Ways to Go Wrong

- False positives
  - Incomplete/inaccurate model
  - Missing rules
  - Conservative analysis
- False negatives
  - Incomplete/inaccurate model
  - Missing rules
  - “Forgiving” analysis

Rules: Dataflow

- Specify
    - Security properties
    - Behavior of library code

```
buff = getInputFromNetwork();
copyBuffer(newBuff, buff);
exec(newBuff);
```

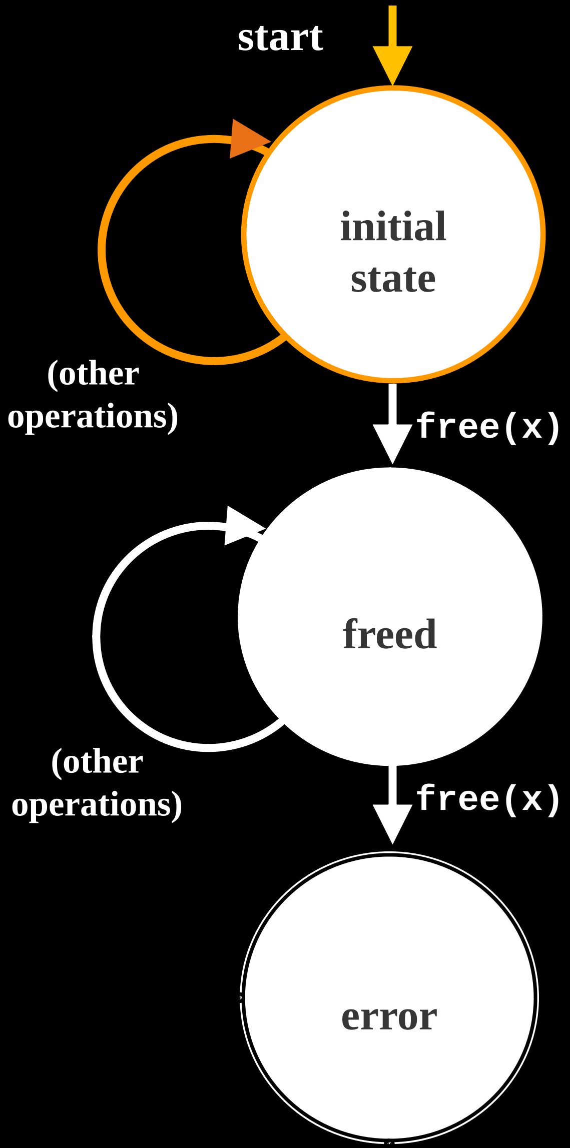
- Three rules to detect the command injection vulnerability

```
1) getInputFromNetwork() postcondition:
      return value is tainted
2) copyBuffer(arg1, arg2) postcondition:
      arg1 array values set to arg2 array values
3) exec(arg) precondition:
      arg must not be tainted
```

Rules: Control Flow

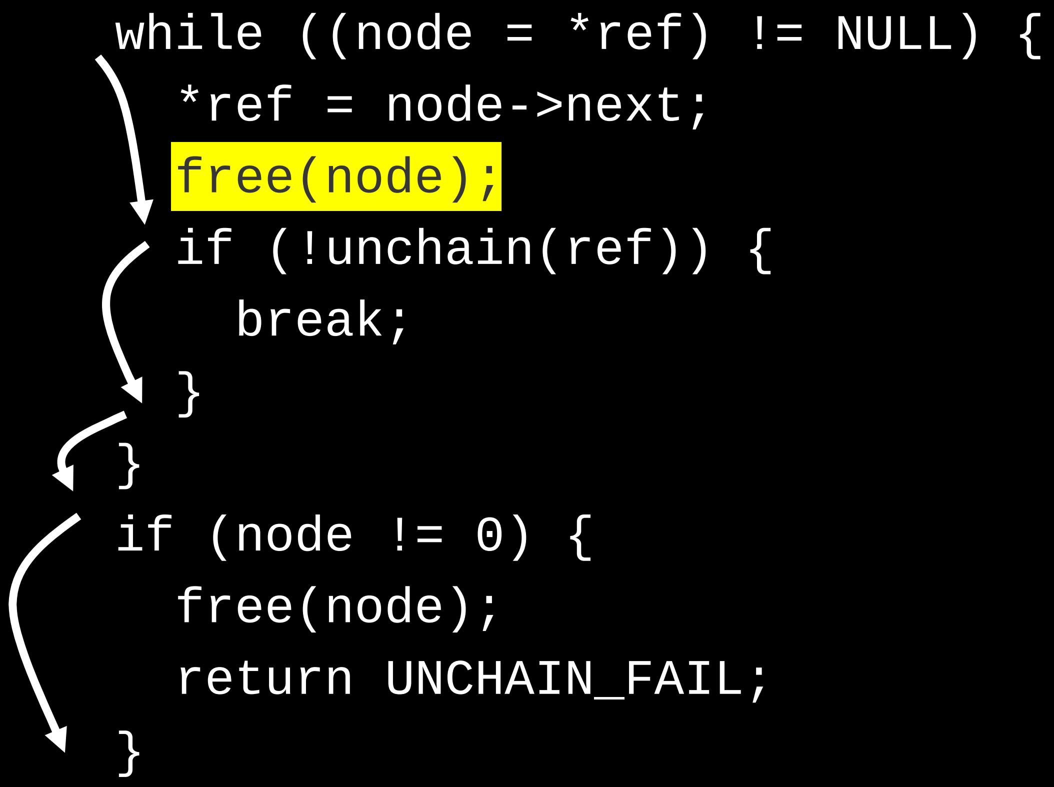
- Look for dangerous sequences
- Example: Double-free vulnerability

```
while ((node = *ref) != NULL) {
  *ref = node->next;
  free(node);
  if (!unchain(ref)) {
    break;
  }
}
if (node != 0) {
  free(node);
  return UNCHAIN_FAIL;
}
```

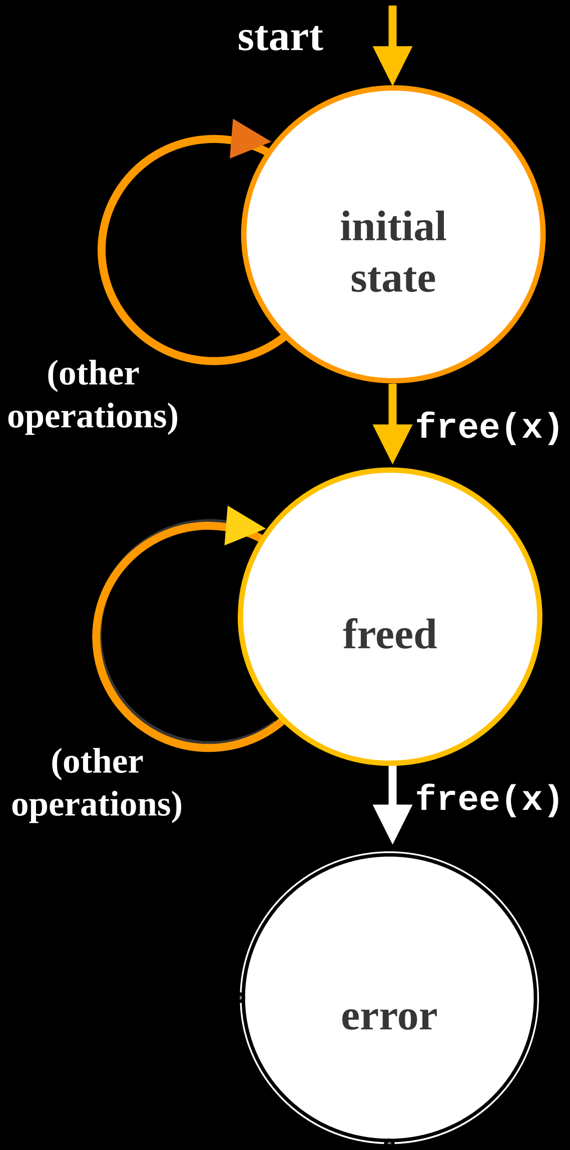


Rules: Control Flow

- Look for dangerous sequences
- Example: Double-free vulnerability



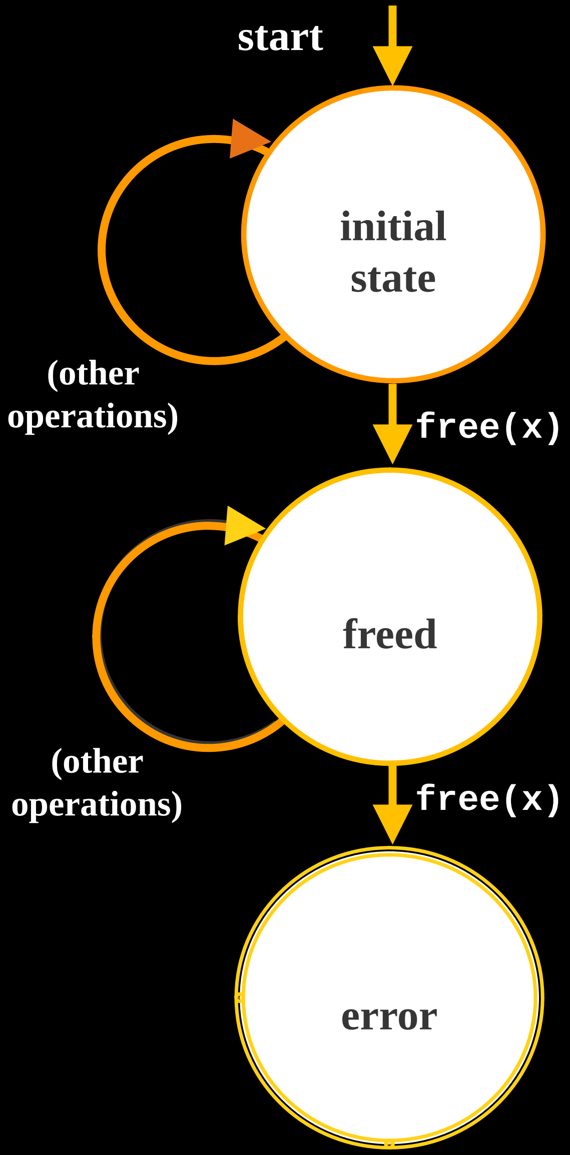
```
while ((node = *ref) != NULL) {
  *ref = node->next;
  free(node);
  if (!unchain(ref)) {
    break;
  }
}
if (node != 0) {
  free(node);
  return UNCHAIN_FAIL;
}
```



Rules: Control Flow

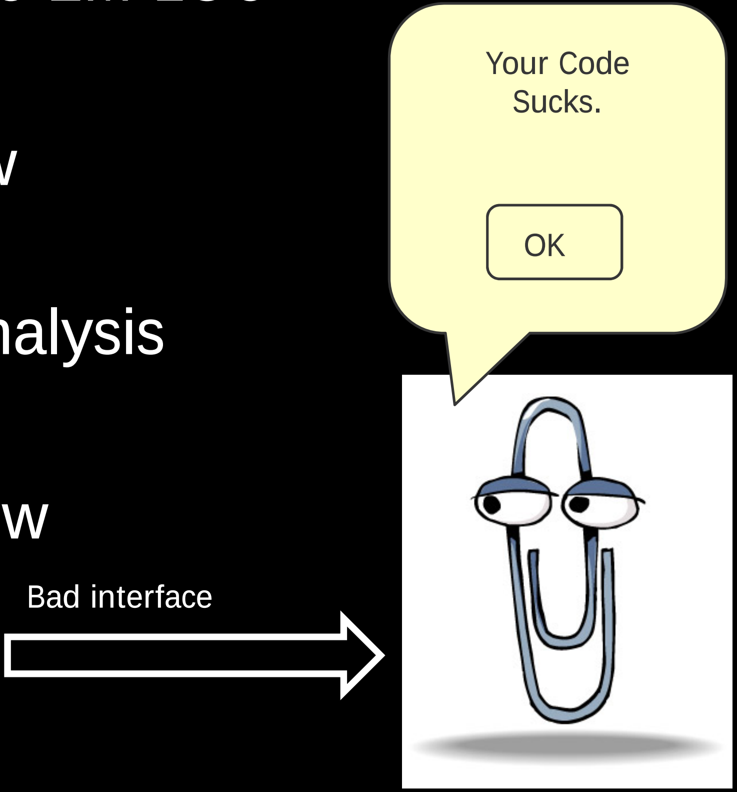
- Look for dangerous sequences
- Example: Double-free vulnerability

```
while ((node = *ref) != NULL) {
  *ref = node->next;
  free(node);
  if (!unchain(ref)) {
    break;
  }
}
if (node != 0) {
  free(node);
  return UNCHAIN_FAIL;
}
```

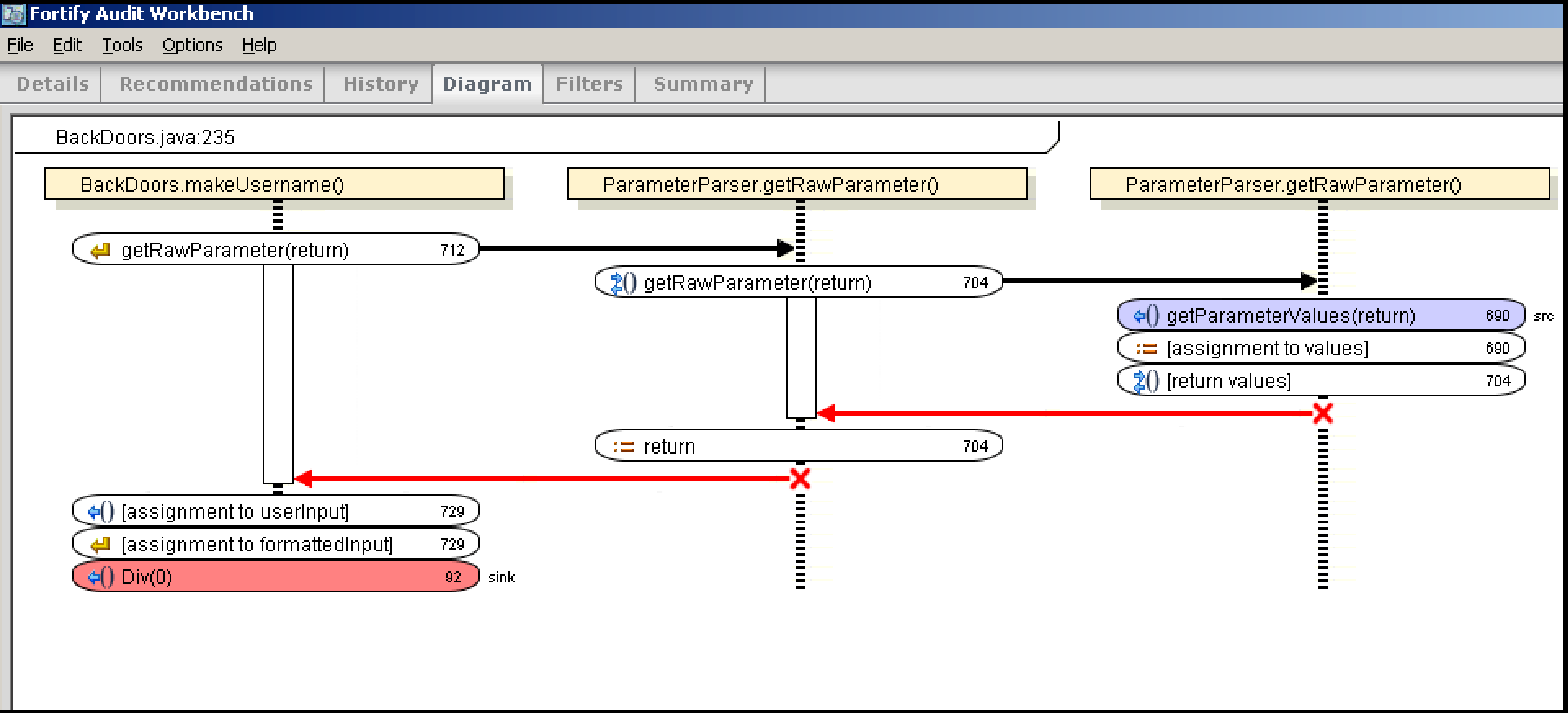


Displaying Results

- Must convince programmer that there’s a bug in the code
- Different interfaces for different scenarios:
  - Security auditor parachutes in to 2M LOC
  - Programmer reviews own code
  - Programmers share code review responsibilities
- Interface is just as important as analysis
- Don’t show same bad result twice
- Try this at home: Java Open Review http://opensource.fortify.com



Interface


Fortify Audit Workbench
File Edit Tools Options Help
Details
Recommendations
History
Diagram
Filters
Summary
BackDoors.java:235
BackDoors.makeUsername()
ParameterParser.getRawParameter()
ParameterParser.getRawParameter()
getRawParameter(return) 712
getRawParameter(return) 704
getParameterValues(return) 690
src
[assignment to values] 690
[return values] 704
return 704
[assignment to userInput] 729
[assignment to formattedInput] 729
Div(0) 92
sink

Iron Chef:
John Henry Challenge

Brian Chess
Pravir Chandra

Sean Fay
Jacob West

Black Hat
3/27/2008
Amsterdam