Iron Chef: John Henry Challenge

Brian Chess
Pravir Chandra

Black Hat
3/27/2008
Amsterdam

Sean Fay
Jacob West
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
Name: Pravir Chandra
Specialty: Manual code review
Job: Principle, Cigital
Chefs

Name: Sean Fay
Specialty: Static and runtime analysis
Job: Chief Architect, Fortify Software
Chefs
Chefs

• After judging, you point out bugs these guys missed
Judges

TBA

TBA

TBA
Dynamic Taint Propagation

- Follow untrusted data and identify points where they are misused
Example: SQL Injection

```java
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.
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

```java
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

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)  All End Points (4/6)

40.0% Miss  33.3% Miss

Web Entry Points (2/2)  0.0% Miss
## Security Coverage

### Edit View

<table>
<thead>
<tr>
<th>Category</th>
<th>Status</th>
<th>Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Entry Points</td>
<td>(3/5)</td>
<td>40.0% Miss</td>
</tr>
<tr>
<td>Web Entry Points</td>
<td>(2/2)</td>
<td>0.0% Miss</td>
</tr>
<tr>
<td>All End Points</td>
<td>(4/6)</td>
<td>33.3% Miss</td>
</tr>
</tbody>
</table>
### SQL Injection Issue

![SQL Injection Issue](https://via.placeholder.com/150)

#### View/Edit Application View Options

Displaying 1 out of 12 events.

**Group By:**
- Entry Point File

**Expand All Collapse All**

<table>
<thead>
<tr>
<th>Events: 1 total</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Entry Point Type</th>
<th>End Point Type</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Injection</td>
<td>Web</td>
<td>Database</td>
<td>1</td>
</tr>
</tbody>
</table>

**Entry Point File**


<table>
<thead>
<tr>
<th>Entry Point Method</th>
<th>End Point File</th>
<th>URL</th>
<th>Audit Status</th>
<th>Verified Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>String[]</td>
<td>splic.ItemService:201</td>
<td>/splic/listMyItems.do</td>
<td>Under Review</td>
<td>✔️</td>
<td>View</td>
</tr>
</tbody>
</table>
**Source**

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

**URL**: [http://localhost/splc/listMyItems.do](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)`

**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?

<table>
<thead>
<tr>
<th>Severity</th>
<th>Category</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>SQL Injection</td>
<td>/splc/listMyItems.do</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.order.splc.ItemService</td>
<td>196</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>select * from item where item name = 'adam' and ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.Throwable at StackTrace$FirstNested$SecondNested.</td>
</tr>
<tr>
<td>&lt;init&gt;(StackTrace.java:267) at StackTrace$FirstNested.</td>
</tr>
<tr>
<td>&lt;init&gt;(StackTrace.java:256) at StackTrace.</td>
</tr>
<tr>
<td>&lt;init&gt;(StackTrace.java:246) at StackTrace.</td>
</tr>
<tr>
<td>main(StackTrace.java:70)</td>
</tr>
</tbody>
</table>
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)
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 vulnerabilities
• Add taint storage and source information to `java.lang.String` storage
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
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

Source Code → Build Model → Perform Analysis → Present Results

Security Knowledge
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

The diagram illustrates the relationship between analysis scope and execution time for various tools. The x-axis represents the analysis scope ranging from line to program, while the y-axis shows execution time with labels such as "Overnight," "Coffee break," and "Blink of an eye." The tools are plotted on the graph, with "Klocwork," "Ounce," "Coverity," "FindBugs," "MS\analyze," "Flawfinder," and "ITS4\RATS."
Only Two Ways to Go Wrong

- **False positives**
  - Incomplete/inaccurate model
  - Missing rules
  - Conservative analysis

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

The tool that cried “wolf!”
Missing a detail can kill.
Rules: Dataflow

- Specify
  - Security properties
  - Behavior of library code

```java
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

```c
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

```c
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

```c
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
Iron Chef:
John Henry Challenge

Brian Chess
Pravir Chandra

Sean Fay
Jacob West

Black Hat
3/27/2008
Amsterdam