Make My Day –
Just Run A Web Scanner

Countering the faults of typical web scanners through bytecode injection

Toshinari Kureha, Fortify Software
Agenda

- Problems With Black Box Testing
  - Approaches To Finding Security Issues
  - 4 Problems With Black Box Testing
- Solution: WhiteBox Testing With ByteCode Injection
  - The Solution
  - Demo Of Solution
  - Building The Solution
- Q&A
Current Practice
Current Practice

How Do You Find Security Issues?

- Looking at architectural / design documents
- Looking at the source code
  - Static Analysis
- Looking at a running application
  - Dynamic Analysis
Current Practice

- Dynamic Analysis
  - Testing & Analysis Of Running Application
    - Find Input
    - Fuzz Input
    - Analyze Response
  - Commercial Web Scanners
    - Cenzic
    - SPI Dynamics
    - Watchfire
Current Practice

Most People Use Web Scanners Because...

- Easy To Run
- Fast To Run
- “Someone Told Me To”
Dynamic Analysis Demo
Web Scanner Review

- Good
  - Found Real Vulnerabilities
  - Was Easy To Run

- “Did I Do A Good Job?”
Question 1: How Thorough Was My Test?

- Do You Know How Much Of Your Application Was Tested?
Question 1: How Thorough Was My Test?

- How Much Of The Application Do You Think You Tested?
Truth About Thoroughness

- We ran a “Version 7.0 Scanner” on the following:

<table>
<thead>
<tr>
<th>Application</th>
<th>EMMA Code Coverage Tool</th>
<th>Web Source</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HacmeBooks</td>
<td>34% classes</td>
<td>30.5%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12% blocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>14% lines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JCVS Web</td>
<td>45% classes</td>
<td>31.2%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>19% blocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>22% lines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java PetStore 2</td>
<td>70% classes</td>
<td>18%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20% blocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>23% lines</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Web Scanner Review

Good
- Found Real Vulnerabilities
- Was Easy To Run

Bad
- How Thorough Was My Test?
  - No Way To Tell, And Actual Coverage Is Often Low
Question 2: Did I Find All Vulnerabilities?

- 3 Ways To Fail
  - Didn’t Test
  - Tested – But Couldn’t Conclude
  - Can’t Test
Question 2: Did I Find All Vulnerabilities?

1. Didn’t Test
   - If The Web Scanner Didn’t Even Reach That Area, It Cannot Test!

Diagram:
- Tested
- Untested
- Vulnerabilities Found
- Vulnerabilities Not Found

Application
Question 2: Did I Find All Vulnerabilities?

2. Tested, But Couldn’t Conclude

- Certain Classes Of Vulnerabilities Sometimes Can Be Detected Through HTTP Response
  - SQL Injection
  - Command Injection
  - LDAP Injection
public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
    ServletOutputStream out = res.getOutputStream();
    String user = req.getParameter("user");
    if (user != null) {
        try {
            String[] args = { "/bin/sh", "-c", "finger " + user };  
            Process p = Runtime.getRuntime().exec(args);
            BufferedReader fingdata = new BufferedReader(new InputStreamReader(p.getInputStream()));
            String line;
            while ((line = fingdata.readLine()) != null) {
                out.println(line);
                p.waitFor();
            }
        } catch (Exception e) {
            throw new ServletException(e);
        }
    } else {
        out.println("specify a user");
    }
    ...
}
public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
    ServletOutputStream out = res.getOutputStream();
    String user = req.getParameter("user");
    if(user != null) {
        try {
            String[] args = { "/bin/sh", "-c", "sendMail.sh " + user }; 
            Process p = Runtime.getRuntime().exec(args); 
            p.waitFor();
        } catch(Exception e) {
            e.printStackTrace(System.err);
        }
        out.println("Thank you note was sent");
    } else {
        out.println("specify a user");
    }
    ...
}
Question 2: Did I Find All Vulnerabilities?

3. Can’t Test

- Some Vulnerabilities Have No Manifestation In Http Response

I hope they’re not logging my CC# into plaintext log file

“Your order will be processed in 2 days”
Major credit card companies have launched security initiatives focused on retailers. Store owners should not store card information, but Visa and MasterCard have found that many point-of-sale terminals and other transaction software store all the data anyway, sometimes unbeknownst to the retailer.

TJX, which operates such discount retail chains as T.J. Maxx and Marshalls in the U.S., released additional details of the breach in a filing with the Securities and Exchange Commission.
Web Scanner Review

- **Good**
  - Found Real Vulnerabilities
  - Was Easy To Run

- **Bad**
  - How Thorough Was My Test?
    - No Way To Tell, And Actual Coverage Is Often Low
  - Did I Find All My Vulnerabilities?
    - Didn’t Test, Tested But Couldn’t Conclude, Can’t Test
Question 3: Are All The Results Reported True?

- No Method Is Perfect
- Under What Circumstances Do Web Scanners Report False Positives?
  - Matching Signature On A Valid Page
  - Matching Behavior On A Valid Page
Question 3: Are All The Results Reported True?

Matching Signature On A Valid Page
Question 3: Are All The Results Reported True?

- Matching Behavior On A Valid Page
  - “To determine if the application is vulnerable to SQL injection, try injecting an extra true condition into the WHERE clause... and if this query also returns the same ..., then the application is susceptible to SQL injection” (from paper on Blind SQL Injection)

- E.g.
    - select ccnum from table where id='5'
  - http://www.server.com/getCC.jsp?id=5' AND '1'='1
    - select ccnum from table where id='5' AND '1'='1'
Question 3: Are All The Results Reported True?

**E.g.**
  - select ccnum from table where id='5'
  - Response:
    - “No match found” (No one with id “5”)
- http://www.server.com/getCC.jsp?id=5' AND '1'='1
  - select ccnum from table where id='5' AND '1'='1'
  - Response
    - “No match found” (No one with id “5’ AND ‘1’=‘1”) 
      - All single quotes were escaped.

**According To The Algorithm ("inject a true clause and look for same response"), This Is SQL Injection Vulnerability!**
Web Scanner Review

- **Good**
  - Found Real Vulnerabilities
  - Was Easy To Run

- **Bad**
  - How Thorough Was My Test?
    - No Way To Tell, And Actual Coverage Is Often Low
  - Did I Find All My Vulnerabilities?
    - Didn’t Test, Tested But Couldn’t Conclude, Can’t Test
  - Are All The Results Reported True?
    - Susceptible To False Signature & Behavior Matching
Question 4: How Do I Fix The Problem?

- Security Issues Must Be Fixed In Source Code
- Information Given
  - URL
  - Parameter
  - General Vulnerability Description
  - HTTP Request/Response
- But Where In My Source Code Should I Look At?
Question 4: How Do I Fix The Problem?

- Incomplete Vulnerability Report -> Bad Fixes

- Report:
  - Injecting “AAAAA…..AAAAA” Caused Application To Crash

- Solution By Developers:
  
  ```java
  ....
  if (input.equals("AAAAA…..AAAAA"))
      return;
  ....
  ```
Web Scanner Review

Good
- Found Real Vulnerabilities
- Was Easy To Run

Bad
- How Thorough Was My Test?
  - No Way To Tell, And Actual Coverage Is Often Low
- Did I Find All My Vulnerabilities?
  - Didn’t Test, Tested But Couldn’t Conclude, Can’t Test
- Are All The Results Reported True?
  - Susceptible To Signature & Behavior Matching
- How Do I Fix The Problem?
  - No Source Code / Root Cause Information
Attacking The Problems

White Box Testing With Bytecode Injection
Agenda

- Problems With Black Box Testing
  - Approaches To Finding Security Issues
  - 4 Problems With Black Box Testing
- Solution: WhiteBox Testing With ByteCode Injection
  - The Solution
  - Demo Of Solution
  - Building The Solution
- Q&A
Review... and Proposal

- Web Scanner
- Application Server
- Database
- File System
- Other Apps

Connection: HTTP

Verify Results

Watch Result
How Will Monitors Solve The Problems?

- How Thorough Was My Test?
- Did I Find All My Vulnerabilities?
- Are All The Results Reported True?
- How Do I Fix The Problem?

- Monitors Inside Will Tell Which Parts Was Hit
- Monitors Inside Detects More Vulnerabilities
- Very Low False Positive By Looking At Source Of Vulnerabilities
- Monitors Inside Can Give Root Cause Information
How To Build The Solution

- How Do You Inject The Monitors Inside The Application?
- Where Do You Inject The Monitors Inside The Application?
- What Should The Monitors Do Inside The Application?
How Do You Inject The Monitors?

- Problem: How Do You Put The Monitors Into The Application?

- Assumption: You Do Not Have Source Code, Only Deployed Java / .NET Application

- Solution: Bytecode Weaving
  - AspectJ for Java
  - AspectDNG for .NET
How Does Bytecode Weaving Work?

Original .class → AspectJ → New Code & Location Spec. → New .class

Similar process for .NET
How Does Bytecode Weaving Work?

List getStuff(String id) {
    List list = new ArrayList();
    try {
        String sql = "select stuff from mytable where id='" + id + "'";
        JDBCstmt.executeQuery(sql);
    } catch (Exception ex) {
        log.log(ex);
    }
    return list;
}

List getStuff(String id) {
    List list = new ArrayList();
    try {
        String sql = "select stuff from mytable where id='" + id + "'";
        MyLibrary.doCheck(sql);
        JDBCstmt.executeQuery(sql);
    } catch (Exception ex) {
        log.log(ex);
    }
    return list;
}

Before "executeQuery()"
Call "MyLibrary.doCheck()"
Bytecode Injection Demo
Applying Byte-Code Injection To Enhance Security Testing

- **How** Do You Inject The Monitors Inside The Application?
- **Where** Do You Inject The Monitors Inside The Application?
- **What** Should The Monitors Do Inside The Application?
Where Do You Inject The Monitors?

- All Web Inputs (My Web Scan Should Hit All Of Them)
  - request.getParameter, form.getBean

- All Inputs (Not All Inputs Are Web)
  - socket.getInputStream.read

- All “Sinks” (All Security Critical Functions)
  - Statement.executeQuery(String)
  - (FileOutputStream|FileWriter).write(byte[])
Applying Byte-Code Injection To Enhance Security Testing

📍 How Do You Inject The Monitors Inside The Application?
📍 Where Do You Inject The Monitors Inside The Application?
📍 What Should The Monitors Do Inside The Application?
What Should The Monitors Do?

- Report Whether The Monitor Was Hit
- Analyze The Content Of the Call For Security Issues
- Report Code-Level Information About Where The Monitor Got Triggered
What Should The Monitors Do?

1) Report whether API was hit or not
2) Analyze The Content Of The API Call
3) Report Code-Level Information

aspect SQLInjection {
    pointcut sqlExec(String sql):call(ResultSet Statement.executeQuery(String)) && args(sql);
    before(String sql) : sqlExec(sql) {
        checkInjection(sql, thisJoinPoint);
    }
    void checkInjection(String sql, JoinPoint thisJoinPoint) {
        System.out.println("HIT:" +
            thisJoinPoint.getSourceLocation().getFileName() +
            thisJoinPoint.getSourceLocation().getLine());
        if (count(sql, ")")%2 == 1) {
            System.out.println("*** SQL Injection detected.  SQL statement being executed as follows: " + sql);
        }
    }
}
Proof Of Concept

- Running The Custom Solution
With Additional Work on UI

- **Web Attack Surface** (28/45)
- **All Attack Surface** (33/81)
- **All Sinks** (28/150)

### Coverage (Edit View)

- Missed 37.8%
- Missed 59.3%
- Missed 81.3%

### Issues by Severity

- Critical
- High
- Medium
- Low
- Info

### Issues by Category

- Cross-Site Scripting: 1
- SQL Injection: 1
- Privacy Violation: 2
- Unhandled Exception: 1
| Entry | Web   | com.order.splc.CheckoutAction | 39  | java.lang.String  
|-------|-------|------------------------------|-----|-------------------
|       |       | com.order.splc.CheckoutForm.getExpirationMon() |     |                   |
| ✔     | ✔     | com.order.splc.CheckoutAction | 38  | java.lang.String  
|       |       | com.order.splc.CheckoutForm.getCvv2() |     |                   |
| ✔     | ✔     | com.order.splc.CheckoutAction | 37  | java.lang.String  
|       |       | com.order.splc.CheckoutForm.getAddr() |     |                   |
| ✔     | ✔     | com.order.splc.CheckoutAction | 36  | java.lang.String  
|       |       | com.order.splc.CheckoutForm.getCcnum() |     |                   |
| ✔     | ✔     | com.order.splc.CheckoutAction | 35  | java.lang.String  
|       |       | com.order.splc.CheckoutForm.getName() |     |                   |
|       |       | com.order.splc.ListHelpAction | 25  | com.order.splc.Help  
|       |       | com.order.splc.AddHelpForm.getBean() |     |                   |
| ●     |       | com.order.splc.ListHelpAction | 25  | com.order.splc.Help  
|       |       | com.order.splc.AddHelpForm.getBean() |     |                   |
| ●     |       | com.order.splc.ListProfilesAction | 34  | com.order.splc.Profiles  
|       |       | com.order.splc.AddProfileForm.getBean() |     |                   |
| ●     |       | com.order.splc.ListItemsAction | 25  | com.order.splc.Item  
|       |       | com.order.splc.AddItemForm.getBean() |     |                   |
With Additional Work on UI

Coverage (Edit View)

- Web Attack Surface (28/45) Missed 37.8%
- All Attack Surface (33/81) Missed 59.3%
- All Sinks (28/150) Missed 81.3%

Issues by Severity

Issues by Category

- Cross-Site Scripting: 1
- Privacy Violation: 2
- SQL Injection: 1
- Unhandled Exception: 1
## Security Issues Detail

<table>
<thead>
<tr>
<th>Severity</th>
<th>Category</th>
<th>URL Path</th>
<th>File Name</th>
<th>Line Number</th>
<th>Method Name</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>critical</td>
<td>SQL Injection</td>
<td>/splc/listMyItem.do</td>
<td>com.order.sple.ItemService</td>
<td>201</td>
<td>ResultSet java.sql.Statement.executeQuery(String)</td>
<td>View</td>
</tr>
<tr>
<td>medium</td>
<td>Privacy Violation</td>
<td>/splc/listMyItem.do</td>
<td>com.order.sple.ItemService</td>
<td>198</td>
<td>void java.util.logging.Logger.info(String)</td>
<td>View</td>
</tr>
<tr>
<td>medium</td>
<td>Privacy Violation</td>
<td>/splc/finalCheckout.do</td>
<td>com.order.sple.FinalCheckoutAction</td>
<td>47</td>
<td>void java.util.logging.Logger.info(String)</td>
<td>View</td>
</tr>
</tbody>
</table>
## Security Issues Detail – SQL Injection

### Description:
Detected a SQL Injection issue using a comparison between a string literal and another literal (string or number).

### Timestamp:
2007-03-29, 12:45:59:375

### URL:
http://localhost:8380/spc/listItems.do

### Username:
admin

### Session ID:
18A736656EEB350CF019F0E59739E11E

### Referer:
http://localhost:8380/spc/listItemsPage.do

### User Agent:
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; InfoPath.1)

### Trigger:
**Method Argument**
Value:

```sql
select id, account, sku, quantity, price, ccno, description from item where sku = 'blah' or '1'='1'
```

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

### Stack Trace:
```
com.order.spcl.ItemService.getItemList(ItemService.java:201)
  at com.order.spcl.ListItemsAction.execute(ListItemsAction.java)
  at org.apache.struts.action.RequestProcessor.processActionPerform(RequestProcessor.java)
  at org.apache.struts.action.RequestProcessor.process(RequestProcessor.java)
```
## Security Issue Detail – Privacy Violation

<table>
<thead>
<tr>
<th>Category</th>
<th>Privacy Violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subcategory</td>
<td>Credit Card Number</td>
</tr>
<tr>
<td>Description</td>
<td>The application attempted to log a credit card number</td>
</tr>
<tr>
<td>Timestamp</td>
<td>2007-03-28, 18:55:04:718</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://localhost:8380/splc/finalCheckout.do">http://localhost:8380/splc/finalCheckout.do</a></td>
</tr>
<tr>
<td>Username</td>
<td>adam</td>
</tr>
<tr>
<td>Session ID</td>
<td>994C64DA46CC34EFAF9F60B0E197A9CC</td>
</tr>
<tr>
<td>Referer</td>
<td></td>
</tr>
<tr>
<td>User Agent</td>
<td>Mozilla 4.0 (compatible; MSIE 6.0; Windows NT 5.1)</td>
</tr>
<tr>
<td><strong>Trigger</strong></td>
<td><strong>Method Argument</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Value:</strong> User is attempting to checkout using CC number: 5424123412341234</td>
</tr>
<tr>
<td>method</td>
<td>void java.util.logging.Logger.info(String)</td>
</tr>
</tbody>
</table>
Conclusions – Web Scanners

- **Good**
  - Easy To Use
  - Finding Smoking Gun

- **Bad**
  - Lack Of Coverage Information
  - False Negative
  - False Positive
  - Lack Of Code-Level / Root Cause Information
Conclusions – White Box Testing

- Bytecode Injection Require Access To Running Application
- In Exchange …
  - Gain Coverage Information
  - Find More Vulnerabilities, More Accurately
  - Determine Root Cause Information
Conclusions – Use Your Advantage

<table>
<thead>
<tr>
<th></th>
<th>Attacker</th>
<th>Defender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attempts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Knowledge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access To Application</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Thank You

Questions?
- Email: tkureha at fortifysoftware.com