Java

JavaOne℠

Web 2.0 Security Puzzlers: Genuine Vulnerabilities or False Positives

Ray Lai

Intuit

Security and Performance Architect

intuit

- To distinguish what's genuine security vulnerabilities from false positives in Web 2.0 applications

Agenda

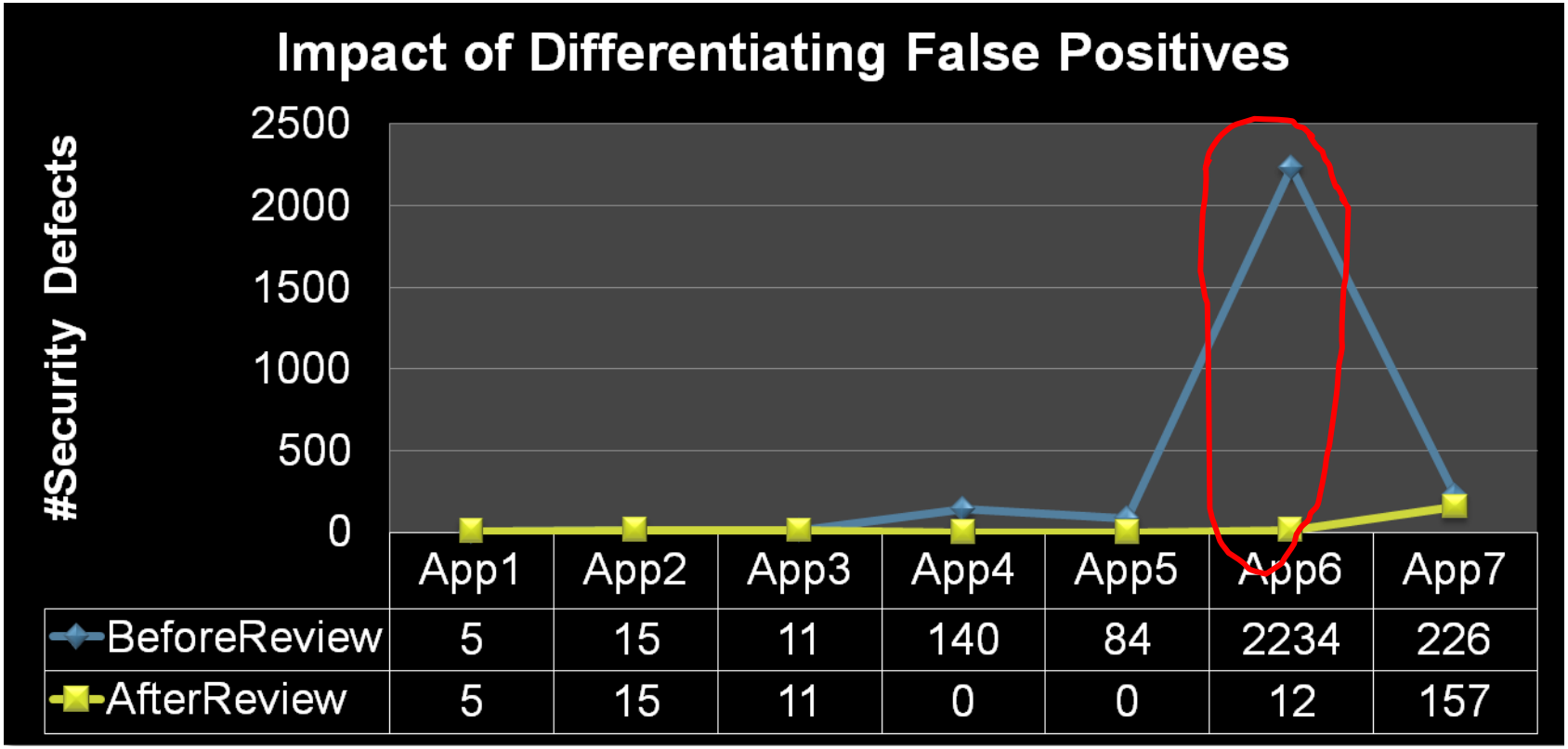
- Security “Context”
- Puzzle 0: Cross-frame Attack
- Puzzle 1: Cross-site Scripting with JavaScript
- Puzzle 2: Form-based Authentication
- Puzzle 3: SQL Injection in EJB3
- Puzzle 4: Hard-coded Passwords
- Puzzle 5: System Information Leakage
- Rule of Thumb

Security "Context"

- Scenario
  - *"A portal application has ~6,000 security defects identified by a static code analysis tool. Developers disagree with a few issues (including Cross-site Scripting). It took 6 weeks to resolve the gaps."*
    - Recurring Web 2.0 / OWASP* security vulnerabilities
    - Many developers disagree with the triage from static code analysis tools – too much "noise"
    - Similar security vulnerabilities are sometimes considered as "false positives", and sometimes aren't – why?

*Open Web Application Security (OWASP) Top 10 Security Vulnerabilities 2007
http://www.owasp.org/index.php/Top_10_2007

False Positives: Example

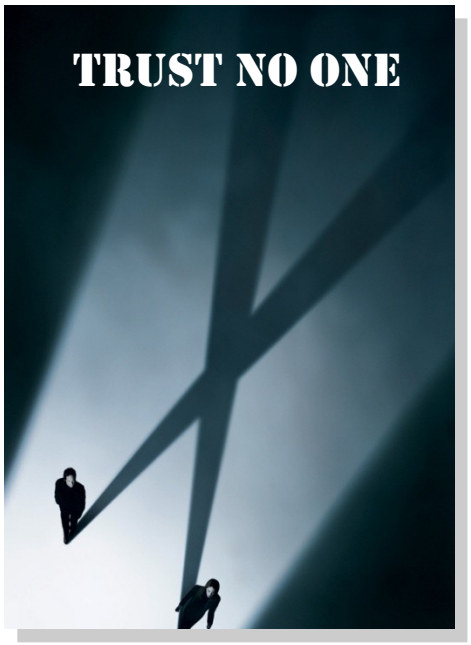


This sampler shows a big difference in the number of security defects for large complex applications after removing false positives

Source: Internal security code audit (Dec 2008)

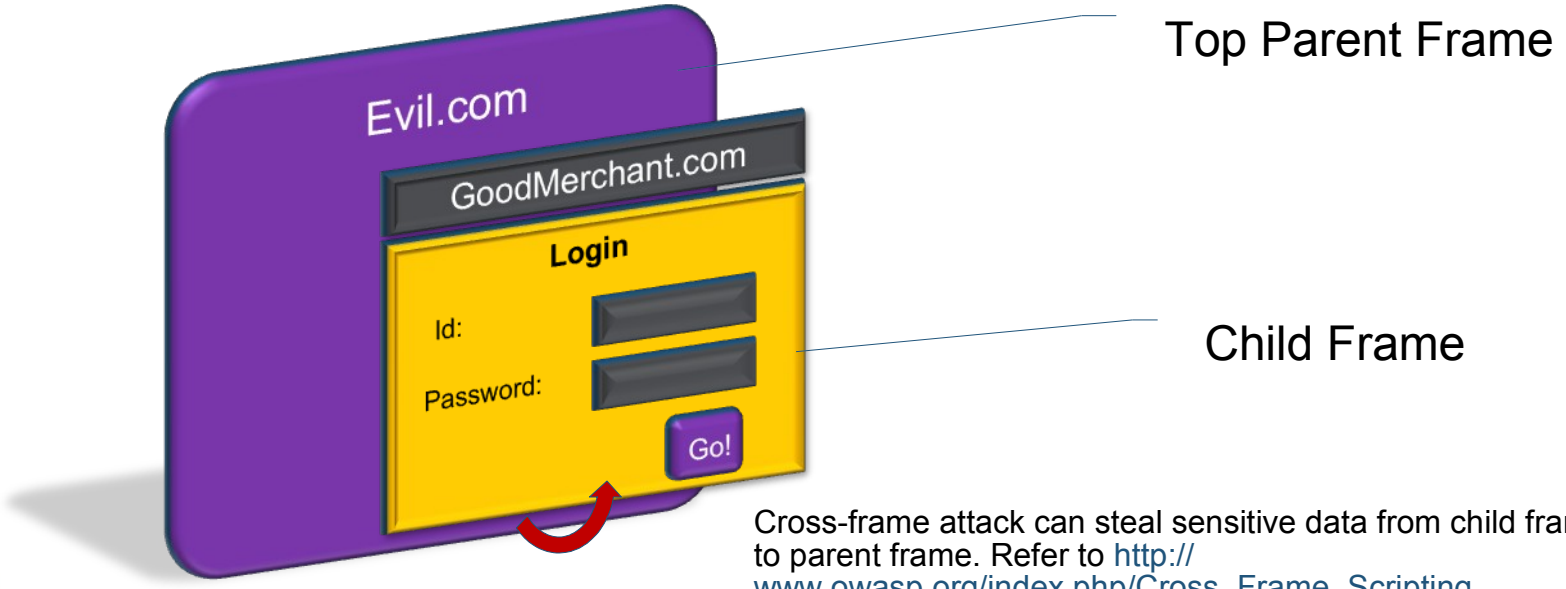
Introducing Security Puzzlers

- Each puzzler depicts a real-life scenario
  - Study the code snippet to understand whether this is a security issue
  - Analyze the security risks and impact
  - Vote whether this is a false positive
  - Rationalize why (or why not)
- Ground rules
  - Don’t make unstated assumptions
  - Time box your vote (decision)
  - Don’t vote “maybe”, “yes except…”



Puzzle 0: Cross-frame Attack

- A security penetration testing vendor A recommends a fix for a *cross-frame attack* in Web pages, but a static code analysis tool vendor B reports an *open re-direct* issue in the security fix. Which is a false positive?



Cross-frame attack can steal sensitive data from child frame to parent frame. Refer to http://www.owasp.org/index.php/Cross_Frame_Scripting

Puzzle 0: Cross-frame Attack (cont'd)

- What's Wrong?

Cross-frame Attack Fix (Vendor A):

```
<SCRIPT LANGUAGE=JAVASCRIPT>
function breakOut() {
  if (self != top)
    window.open("http://www.goodMerchant.com","_top","");
}
</SCRIPT>
```

Both vendors are reputable, and both arguments sound reasonable.

Which one a false positive?

HOT issue – Open Re-direct

"*The file crossFrameDetection.js passes un-validated data to an HTTP redirect function on line x. Allowing un-validated input to control the URL used in a redirect can aid phishing attacks.*" (Vendor B)

Puzzle 0: Cross-frame Attack (cont'd)

Considerations	Assumptions	Reality Check
Vendor's reliability and reputation	Security vendors are always right – they are professional	Interpreting the context of the security defect, and the associated risk is critical Even software vendors have defects
Web browser security	All Web browsers detect and prevents cross-frame attack	Add cross-frame attack prevention

- Resolution
  - Vendor B's triage is false positive

Puzzle 1: Cross-site Scripting (XSS) with JavaScript

- What's Wrong?
  http://192.168.1.104:8080/billPay.jsp?custId=2000

  - Malicious JavaScript can be injected to the REST Web service, e.g. http://192.168.1.104:8080/billPay.jsp?custId=2000;<s alert("You are hacked!");</script>
  - Personal information from local PC can be stolen, or transferred to other malicious Website

Puzzle 1: Cross-site Scripting with JavaScript (cont'd)

Considerations	Assumptions	Reality Check
Input data validation	Default Web pages won't sanitize XSS attack (regex for special characters)	Verify any XSS check on http://192.168.1.104:8080/billPay.jsp?custId=2000
Browser security	IE 6 and Mozilla 3.5.x don't allow cross-frame attack under iFrame	Not every Web browser has the latest patch for cross-frame attack prevention
Un-validated URL	Can the URL be modified or injected?	Is a hard-coded / pre-defined URL validated? Check the domain in each Web page Maintain a white-list or black-list of URLs

Puzzle 1: Cross-site Scripting with JavaScript (cont'd)

- Resolution
  - XSS is a recurring, critical Web 2.0 security vulnerability (not false positive!)
  - Use servlet filter to filter harmful JavaScript, e.g. regex of special escape characters

```
<% String custId = request.getParameter("custId"); %>
```

Positive Filtering

```
String custId =
request.getParameter("custId");
String pattern = "^\\d+$";
if (!custId.matches(pattern))…
```

Negative Filtering

```
String pattern="[<>{}\\[\\];\\&]";
String cleanCustId =
custId.replaceAll(pattern," ");
…
```

Puzzle 1a: Cross-site Scripting with JavaScript

- Any issue with a re-directed error.jsp when a Java exception is thrown

Web.xml:

```
<error-page>
   <exception-type>java.lang.Throwable
   </exception-type>
<location>/error.jsp</location>
</error-page>
```

The error.jsp is vulnerable to XSS. Since the origin URL is XSS-sanitized, we don't need to fix this error.jsp.

Is this a false positive?

Discovery	****
Ambiguity	****
Impact	***
Verifiable	*****

Puzzle 1a: Cross-site Scripting with JavaScript (cont'd)

- Code Snippet

error.jsp:

```
<form
action="<%=request.getSession().getAttribute("orig_url")
%>" method="post" target="_self" id="submit_form">
    <%
        StringBuilder params = new StringBuilder();
        Enumeration<String> xss =
            request.getParameterNames();
        …
    %>
    <%=params.toString()%>
    <input value="Retry" type="submit">
</form>
```

*__orig_url__ refers to the URL in the previous example

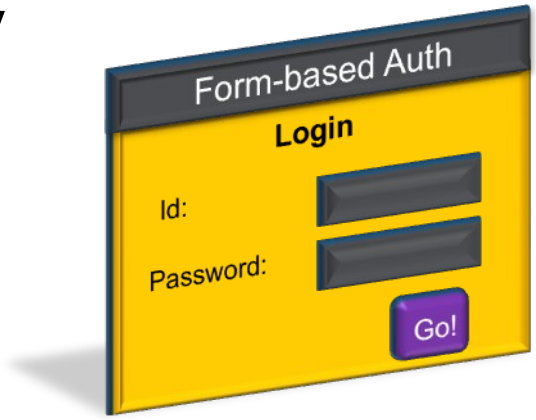
Puzzle 1a: Cross-site Scripting with JavaScript (cont'd)

Considerations	Assumptions	Reality Check
Security risk analysis	No user will go directly to http://192.168.1.104:8080/error.jsp	Perform security risk analysis (e.g. DREAD) to assess the probability and impact
URL re-direction	Error.jsp definition in web.xml will always capture all Java run-time errors	Error handling for java.lang.Throwable in web.xml does not necessarily handle all HTTP GET errors

- Resolution
  - False positive

Puzzle 2: Form-based Authentication

- Cross-site Request Forgery (CSRF) - What's Wrong?
  - CSRF is an attack which forces users to execute unwanted actions on a Web application in which they are currently authenticated
    - Example: users link a malicious URL link via email
  - A static code analysis tool complains my login form having CSRF vulnerability



Puzzle 2: Form-based Authentication (cont'd)

- Code Snippet

login.jsp

```
<form method="post" action='j_security_check' >
   <input type="text" id="j_username" value=""
name="j_username" class="input">
   <br>Password
<input type="password" value="" name="j_password"
class="input">
   <br>
   <input type="image" src="images/signin.gif" alt="Sign
In" width="135" height="22" border="0" value="Go!">
   ...
</form>
```

Form-based authentication is a standard Web security. Does it have CSRF issue?

Is this a false positive?

Discovery	****
Ambiguity	*
Impact	***
Verifiable	***

Puzzle 2: Form-based Authentication (cont'd)

Considerations	Assumptions	Reality Check
Common security standard	Form-based authentication is secure	Use of nonce / random request identifier will address cookie or weak session management issue in CSRF attack
Root cause of CSRF	Poor implementation for the form-based authentication	The issue is not about the form
Servlet filter	Web containers should have built-in CSRF protection	Verify CSRF support in Web or app server

Puzzle 2: Form-based Authentication (cont'd)

- Resolution
  - Use dynamically generated form with nonce/random request identifier
  - Use CSRF servlet filter

```
JSP:
<%= helper.renderFormWithNonce() %>


Output:
<form method="POST" action="/signup" >
  User id: <input type="text" name="username">
  Password: <input type="password" name="j_password">
    <input type="submit" name="action" value="Signup">
</form>
    <input type="hidden" name="j_username"
value="12bc34de56fg78">
```

Puzzle 3: SQL Injection in EJB3

- What's Wrong?
  - EJB3 entity manager (via annotation) has SQL codes generated to handle Create-Read-Update-Delete (CRUD) operations
  - Static code analysis tool complains standard EJB3 JPA code snippet having SQL Injection
    - CRUD has insert or update statements which can be injected or modified in the persistence layer
    - Does it mean that use of EJB3 with JPA exposes an attack surface to SQL injection?

Puzzle 3: SQL Injection in EJB3 (cont'd)

- Code Snippet

```
public List<E> findByQuery(String queryString, Map<String,
Object> parameters) {
    Query query = getEntityManager().createQuery(queryString);
    return findByQuery(query, parameters);
}
```

findByQuery() is a generated method, and queryString is a string variable that can be modifiable or injected. EJB3 is widely used and EJB security is fairly robust.

Is this a false positive?

Discovery	*****
Ambiguity	*
Impact	*
Verifiable	***

Puzzle 3: SQL Injection in EJB3 (cont'd)

Considerations	Assumptions	Reality Check
EJB security	EJB security is robust. There should not be any SQL injection issue	Verify if static code analysis tool or penetration testing tool will complain SQL injection vulnerability
SQL injection	queryString is used internally	The findByQuery() defined in the entity manager is public

Puzzle 3: SQL Injection in EJB3 (cont'd)

- Resolution
  - Use parameterized SQL statement

Example:

```
String userName = …;
String itemName = …;
String myQuery =
        "SELECT * FROM employee WHERE staffId=:staffId AND
owner=:owner";
Query stmt = entityManager.createQuery(myQuery);
stmt.setParameter("staffId", staffId);
stmt.setParameter("owner", userName);
List employees = stmt.getResultList();
...
```

Puzzle 4: Hard-coded Password

- What's Wrong?
  - EJB3 or Hibernate specifies username and password for the data source
  - Static code analysis tool complains password is hard-coded in the configuration file
    - EJB3 or Hibernate should be safe since Java EE is safe, and many developers are using them

Puzzle 4: Hard-coded Password (cont'd)

- Code Snippet

persistence.xml

```
<properties>
   <property name="hibernate.connection.driver_class"
value="oracle.jdbc.OracleDriver"/>
   <property name="hibernate.dialect"
value="org.hibernate.dialect.Oracle9Dialect"/>
   <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.NoCacheProvider"/>
   <property name="hibernate.connection.password"
value="myId"/>
   <property name="hibernate.connection.username"
value="myPassword"/>
…
</properties>
```

Discovery	*****
Ambiguity	*
Impact	*
Verifiable	***

Puzzle 4: Hard-coded Password (cont'd)

Considerations	Assumptions	Reality Check
File system security	OS file system security should prevent hackers from accessing the file persistence.xml	Many default file system implementation makes password file or persistence.xml world-readable
Firewall security	Persistence.xml is stored in a secure server behind the firewall	From CSI statistics, large amount of security attacks are from internal users (behind the firewall)
Container security	Java EE container security protects all files and objects	Verify if the scope of Java EE security covers the issue

Puzzle 4: Hard-coded Password (cont'd)

- Resolution
  - Custom encrypted data source provider
  - Hash your password

Example:

```
<persistence-unit name="default" transaction-
type="JTA">
<provider>
oracle.toplink.essentials.PersistenceProvider
</provider>
<jta-data-source>jdbc/MyDataSource
</jta-data-source>
<properties>
 <property name="toplink.logging.level"
value="INFO"/>
</properties>
</persistence-unit>
```


http://download-uk.oracle.com/docs/cd/B31017_01/web.1013/b28221/cfgdepds005.htm

Puzzle 5: System Information Leakage

- Scenario
  - Many developers use `printStackTrace()` in error handling, but this usually discloses system classes and path that may be used for future exploit
    - Setting up an error page in web.xml can re-direct all exceptions to a pre-defined, pre-formatted Web page. Thus, system information won't be disclosed

Web.xml:

```
<error-page>
   <exception-type>java.lang.Throwable
   </exception-type>
<location>/error.jsp</location>
</error-page>
```

Puzzle 5: System Information Leakage (cont'd)

- What's Wrong?

```
Stack trace (sometimes with sensitive data) output to screen:
…
<ErrorCode>1016</ErrorCode>
        <ErrorDetails>com.goodMerchant.xxx.handshake.shared.BHException: An
error occured while validating the request message: Inbound message parsing
error at
com.goodMerchant.xxx.handshake.shared.SharedExceptionHandler.handleException
(SharedExceptionHandler.java:64) at
com.goodMerchant.xxx.common.messaging.adapter.servlet.HttpAdapterServlet.doP
ost(HttpAdapterServlet.java:72)...
Caused by: com.goodMerchant.xxx.common.messaging.exception.CommonException:
Inbound message parsing error
...
Caused by: org.apache.xmlbeans.XmlException: error: The document is not a
CustomerEntitlementRoot@http://www.goodMerchant.com/CustomerEntitlement:
document element mismatch got GatewayRequest...
```

If you send a HTTP GET or POST message, a system stack trace will be output to the standard output (e.g. user screen) even though error page is set up in web.xml. What's the issue with system stack trace output to screen? Is this a false positive?

Puzzle 5: System Information Leakage (cont'd)

Considerations	Assumptions	Reality Check
Sensitivity	System stack trace is for debugging. Not a security issue	Sometimes debug message contains sensitive business or personal data
Error handling – re-direction in web.xml	If Java Throwable is defined in the error page in web.xml, all errors will be intercepted and only a pre-defined Web page will be displayed to user screen	HTTP GET and POST may generate errors

Puzzle 5: System Information Leakage (cont'd)

- Resolution
  - Use secure logger – set up access control to the server.log
  - Sanitize or mask sensitive data from the logger, e.g. don't output password in the logger

Example:

```
catch (SecurityException ex) {
   Exception exceptionX =
MyExceptionHandler.handle(ex);
   logger.log(Level.INFO,"Exception Occurred",
exceptionX);
}
```

Rule of Thumb

Puzzle	Genuine Defect	False Positive
XSS	If the HTTP request header does not sanitize XSS	If the external input is sanitized
CSRF – form-based authentication	Weak cookie or session management exploited by CSRF	Form-based authentication is the root cause of CSRF attack
SQL Injection – EJB3	Vanilla EJB3 entity manager interface exposed to public for SQL injection attack	Private interface with queryString sanitized
Hard-coded password	Persistence.xml publicly accessible	Persistence.xml protected (e.g. OS level file permission)
Denial of Service	Unreleased resource in public interface – invoke numerous times cause system crash	Singleton pattern for resource – resource handler may be kept open
System Information Leakage	Error page defined in web.xml will re-direct Throwable, but it may not capture all Java throwables	Re-direct printStackTrace() to a private, protected system logger and mask sensitive data

References

- OWASP top 10 security vulnerabilities (2007)
  - http://www.owasp.org/index.php/Top_10_2007
- XSS
  - Good list of regex for XSS detection http://www.securityfocus.com/infocus/1768
- Hard-coded password (persistence.xml)
  - Custom provider http://download-uk.oracle.com/docs/cd/B31017_01/web.1013/l
  - Encrypt password http://www.nabble.com/Password-Security-Encryption-for-RCF

References

- System Information Leakage
  - https://www.securecoding.cert.org/confluence/display/java/FIC
  - http://www.owasp.org/index.php/Error_Handling
- CSRF
  - http://www.owasp.org/index.php/Testing_for_CSRF_(OW
  - http://lwn.net/Articles/254709/
  - http://blog.codeville.net/2008/09/01/prevent-cross-site-rec
  - http://www.projectzero.org/sMash/1.1.x/docs/zero.devgui

Thank You

Ray Lai
rayymlai@gmail.com

Author of J2EE Platform Web Services
Co-author of Core Security Patterns, Java EE
.NET Interoperability