ASIDE: IDE Support for Secure Programming

Jing Xie
Advised by: Dr. Bill Chu
Collaborated with: Dr. Heather Lipford
University of North Carolina at Charlotte
Code Level Vulnerability

Client View

Log in Here

Username: [input field]
Password: [input field]

Application

String sql = "SELECT USERNAME, PASSWORD, BALANCE FROM users WHERE TUNEUSER.USERNAME = '' + username + '' AND PASSWORD = '' + password + ''";

Database Server

SELECT USERNAME, PASSWORD, BALANCE FROM users

username = anything
password = ' OR 'a' = 'a

sql = "SELECT USERNAME, PASSWORD, BALANCE FROM users WHERE username = 'anything' AND password = '' OR 'a' = 'a'"
Big picture of application security defense

- Attack detection and prevention
- Vulnerability detection and discovering
- Vulnerable software
Research motivation

Root Cause: developers write insecure code

Tool Support: not only reactive but also excludes developers from the security loop.

Education and training: developers’ cognitive burden

Our goal: developer oriented approach that includes developers into security loop by providing them interactive support in writing secure code in the first place.

Knuth documented 867 errors over a period of 10 years. 368 errors were implementation errors, the rest are requirements / design errors. Mistake of omission is the largest contributor of implementation errors.

Take a look at static code analyzers and dynamic analyzers.
Why do developers make security errors

Developers’ cognitive burden while developing software.

Interview Study:

Reality:
developers are aware of the importance of security for software; but security is just one of many nonfunctional requirements. developers have decent knowledge of developing secure software.

GAP:
there is a disconnect between conceptual understanding of secure programming and actual practices.

Research thesis

**Interactively** identify common secure programming issues using reliable heuristics

Enable developers to **select appropriate actions** while they are in the process of composing the program
- Interactive code refactoring
- Interactive code annotation

**ASIDE:** application security in IDE, Eclipse plugin

Design rationales

- Take full advantage of developer’s application knowledge (e.g. business logic, application context)
- Recognition instead of recalling, a key HCI design principle
- Support best secure software development practice
  - Using trusted library (e.g. OWASP ESAPI)
  - Statistics collection
- Policy driven (adapted to other development environment)

Interactive code refactoring

An analogy: word processor corrects spelling errors.

Target vulnerabilities: due to lack of or insufficient input validation on untrusted inputs

Interactive Code Refactoring

Demo!!
## Code refactoring for input validation

<table>
<thead>
<tr>
<th>Input validation strategy</th>
<th>Advantage(s)</th>
<th>Disadvantage(s)</th>
</tr>
</thead>
</table>
| Right before critical operations (e.g. inserting into database). | Developer knows for sure the type of input. (e.g. first name, password, credit card number, SSN, and etc.) | **Redundant validation**: a variable used in multiple places.  
**Failure to validate**: difficult to foresee all critical operations. |
| As soon as an untrusted input is read into a variable. | Has developer’s attention. Make sure all untrusted inputs are validated.     | Can lead to **false positive**.  
Does not work well with dependency injection design pattern. |
Additional validation features

◊ Semantic validation
  o E.g. once the input is identified as file path, further restrict to a particular file subtree
  o Bounds of integers

◊ For untrusted input of composite type (e.g. `getParameterMap()`)  
  o Perform flow analysis
  o Request for validation as soon as an primitive type (e.g. `java.lang.String`) of data is extracted

Evaluation

How effective is ASIDE at discovering exploitable software vulnerabilities and preventing them?

What constitutes false positives for ASIDE?

How would developers react to this new approach?

Technical Perspective Evaluation:
A comparison study on open source projects source code with Fortify SCA

Human Perspective Evaluation:
User study performed on novice programmers/graduate students & professional software developers

Industry best practice: security audit

65K+ lines of code
Full featured blog server
(1.8M+ hits on google for “powered by Apache Roller”)

a member of SSG at a large financial services company at the time; core Committer of OWASP AppSensor project.

Tool employed: Fortify SCA latest version at the time.

ASIDE identified 131 of 143 (92%) taint sources

- JSP (not yet implemented)
- Framework binding
- Potentially exploited (94), validate to practice defensive security
- False positives (24)

In proceedings of 27th Annual Computer Security Applications Conference (ACSAC), 2011
Defensive security

An untrusted request URL is directly passed into an `InvalidRequestException` constructor.

False positives

We regard 24 reported taint sources as real false positives, where inputs are used in ways that do not lead to any recognized security vulnerabilities.

```java
// are we doing a preview? or a post?
String method = request.getParameter("method");

boolean preview = (method != null && method.equals("preview")) ? true : false;
```

Figure 5. Untrusted input is used for logic test

```java
if (request.getParameter("excerpts") != null) {
    this.excerpts = Boolean.valueOf(request.getParameter("excerpts"));
}
```

Figure 6. Untrusted input is parsed into harmless Boolean value

Developer behavior study

**Main Goal of the study:** to understand the behavior and reaction of developers to providing awareness and assistance for secure programming.

<table>
<thead>
<tr>
<th></th>
<th>ASIDE CodeGen</th>
<th>ASIDE Explanation</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Novice developers/</td>
<td>9 + 3-hour development + up to 15mins debriefing</td>
<td>9 + 3-hour development + up to 15mins debriefing</td>
<td>Implement certain functionality of a Java Servlet based online stock trading application</td>
</tr>
<tr>
<td>Graduate students</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professional developers</td>
<td>4 + 3-hour development + up to 15mins debriefing</td>
<td>4 + 3-hour development + up to 15mins debriefing</td>
<td></td>
</tr>
</tbody>
</table>
CodeGen:  

```
String t_type = Request.getParameter("trans_action");
String quantity = Request.getParameter("quantity");
String acc.nickname = Request.getParameter("acc.nickname");
String stockname = Request.getParameter("stockname");

int a = 0;
try {
    a = Integer.parseInt(a);
    catch (Exception e) {
        makeTransactionForUser(stockname, "In ");
        return;
    }
    if (a == 0) {

```

Explanation:

**Abstract**

The return value of `getParameter()` at line 83 is vulnerable to be manipulated by malicious users.

**Explanation**

When untrusted input gets into the application without proper validation and is used by critical operations, it may subvert the original semantics of that operation. For example, the following code dynamically constructs and executes a SQL query that searches for items matching a specified name based on the value that is passed by the client. The query restricts the items displayed to those where the owner matches the user name of the currently-authenticated user.

**Remediation Recommendation**

The best practice to avoid introducing the aforementioned vulnerabilities into your code is to validate all the values that are passed into your application. One option is to validate the input against established Regular Expression.
Results from students study

Behavior (development recordings)

Perceptions (interview recordings)

Positive, embracing the warnings

[SE5] Helpful, definitely helpful. It's good at the moment to use it, they come up immediately, it's not like you have to understand them later on. At the moment, if you do something wrong, it shows a warning symbol on the left.

CodeGen users trust that using the tool would make their code more secure.

CodeGen needs more elaboration on why certain code is vulnerable.

Explanation does not provide any concrete actions one can take to mitigate the identified problem.
Results from professionals study

Behavior (development recordings)

- Total number of warnings visible during the study session
- Number of warnings clicked on
- Number of warnings where code was generated

Perceptions (interview recordings)

All participants appreciated the concept of real-time warning of secure programming issues.

[PDE2] If Explanation didn’t give me warnings, I would not pay attention to the vulnerable code.

Several participants failed to generalize the idea from the current prototype implementation to their professional development context.
## Impact on tool design

<table>
<thead>
<tr>
<th>Observation</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>The increased efficiency and reduced cognitive burden do seem to be important for programmers to be willing to take the time to address security vulnerabilities while implementing functionality.</td>
<td>Automated code generation, where possible, is likely to improve software security if it is quick and easy.</td>
</tr>
<tr>
<td>Security and secure programming are concepts with many technical details. Many programmers have little to no background in specific vulnerabilities, tools, and practices.</td>
<td>Tool interactions and explanations need to help people learn and understand how and why to use the tool. This may also be necessary so that professionals trust any advice given and code generated.</td>
</tr>
<tr>
<td>Users do not mind real-time warnings, but do not seem to want them to persist, even if they choose to ignore them.</td>
<td>Tool needs to provide ways to remove warnings from code editing view.</td>
</tr>
<tr>
<td>Even when creating secure code is relatively easy, such as through using CodeGen, users still need to be motivated to make needed changes.</td>
<td>This motivation may depend on organizational factors that encourage use amongst developers, and discourage developers from relying solely on other processes or people to handle all security concerns.</td>
</tr>
</tbody>
</table>

Interactive code annotation

Gold Rush Bank

user (username, role, surname, givenName)
account (accountNumber, nickname, balance)
account_user (accountNumber, username)
transaction (id, accountNumber, date, payee, amount)

Interactive code annotation

**Remind** developers important program constructs for secure coding
- Prevent vulnerable code from being written

Annotate key application logic for: source code review + advance analysis

Different from traditional code annotation

Annotate security relationship between different parts of the system

From a developer perspective: point and click

---

Context for annotation

Where to raise question?

Identifying database access functions (e.g. SQL statements), may be too low level

--- Access routines may be shared in different application threads

Identify “use case”/transaction level routines that lead to accessing protected data

E.g. a statement within a Servlet/Action for Java web applications

What is a valid annotation? Where to annotate?

- A set of logic tests, or assertion (e.g. Spring Security)
- On an execution path from web entry to data access point

There might be an execution path from web entry to data access point without access control check.

*Jing Xie*, Bill Chu, and Heather Lipford. **ASIDE: IDE Support for Web Application Security.** In proceedings of 27\(^{\text{th}}\) Annual Computer Security Applications Conference (ACSAC), 2011
Suppose there are two “use cases” that invoke the same access function

They have different access control checks

Access control check MAY BE needed

Web entry point 1

Access control check

Data access point

Web entry point 2
Walkthrough evaluation

Target Open Source Project
Apache **Roller** (Java): blog server software
**Moodle** (PHP): course management system (CMS)

Statistics (bugtrack & security reports)

<table>
<thead>
<tr>
<th></th>
<th>Fixed issues with detailed information</th>
<th>Code Refactoring</th>
<th>Code Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roller</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Moodle</td>
<td>16</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

*Jing Xie*, Bill Chu, and Heather Lipford. **ASIDE: IDE Support for Web Application Security.** In proceedings of 27th Annual Computer Security Applications Conference (ACSAC), 2011
Broken access control

- Bugtrack ID: ROL-1701 (https://issues.apache.org/jira/browse/ROL-1701)

- Problem description:
  
  roller.weblogger.webservices.adminprotocol.BasicAuthenticator is vulnerable to authentication bypass. If invalid headers are passed to it, an invalid user can gain access to protected resources.

Are web headers valid? 

Yes → Retrieve credentials 

No → Throw an Exception

Are credentials valid? 

Yes → Get all users from DB 

No → Retrieve credentials

Jing Xie, Bill Chu, and Heather Lipford. In proceedings of 27th Annual Computer Security Applications Conference (ACSAC), 2011
protected void doGet(HttpServletRequest req, HttpServletResponse res) {
    try {
        Handler handler = Handler.getHandler(req);
        String userName = handler.getUserName();
        EntrySet c = handler.processGet();
        res.setStatus(HttpServletResponse.SC_OK);
        res.setContentType("application/xml; charset=utf-8");
        String s = c.toString();
        Writer writer = res.getWriter();
        writer.write(s);
        writer.close();
    } catch (HandlerException he) {
        res.sendError(he.getStatus(), he.getMessage());
        he.printStackTrace(res.getWriter());
        logger.error(he);
    }
}
/**
 * This method should be called by extensions of this class within their
 * implementation of authenticate().
 */

protected void verifyUser(String userName, String password) throws HandlerException {
    User ud = getUserData(userName);
    String realpassword = ud.getPassword();

    boolean encrypted = Boolean.valueOf(WebloggerConfig.getProperty("passws.encryption.enabled"));
    if (encrypted) {
        password = Utilities.encodePassword(password, WebloggerConfig.getProperty("passws.encryption.algorithm"));
    }

    if (!userName.trim().equals(ud.getUserName())) {
        throw new UnauthorizedException("ERROR: User is not authorized: " + userName);
    }

    if (!password.trim().equals(realpassword)) {
        throw new UnauthorizedException("ERROR: User is not authorized: " + userName);
    }

    if (!ud.hasRole("admin")) {
        throw new UnauthorizedException("ERROR: User must have the admin role to use the RAP endpoint: " + userName);
    }

    if (!ud.isEnabled().booleanValue()) {
        throw new UnauthorizedException("ERROR: User is disabled: " + userName);
    }
}
Cross-site request forgery

2/7 are CSRF vulnerabilities

Moodle has developed a pattern to prevent CSRF

But it was missed in at least these two cases by developers

Change an existing user profile

Client form submission

Update database content

---

Jing Xie, Bill Chu, and Heather Lipford. **ASIDE: IDE Support for Web Application Security.**
In proceedings of 27th Annual Computer Security Applications Conference (ACSAC), 2011
**ASIDE Solution**

**Heuristic:** Whenever a form submission/web request contains operation to update (add, delete, modify) database entries, the form submission needs to be checked for CSRF.

Raise question at **Line 72**
You may want an admin privilege check

adminLogin (request, response)

Add JSP web request

Index JSP web request

Admin privilege check

## Summary of benefits

<table>
<thead>
<tr>
<th>Technique</th>
<th>Code refactoring</th>
<th>Code annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Audience</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Students</strong></td>
<td>Shape awareness, reminder of secure coding best practices, aid in grading</td>
<td>Shape awareness, reminder of secure coding best practices, aid in grading</td>
</tr>
<tr>
<td><strong>Professional developers</strong></td>
<td>Reminder of secure coding best practices, take care of “grunt work”</td>
<td>Reminder of secure coding best practice, advanced analysis</td>
</tr>
<tr>
<td><strong>Enterprise</strong></td>
<td>Encourage secure coding, policies, practice and standards, collect SSDLC statistics</td>
<td>Collect SSDLC statistics, aid in code review</td>
</tr>
</tbody>
</table>
Work in Progress

December 16\textsuperscript{th} 2011 – March 15\textsuperscript{th} 2012: prototype ASIDE that has interactive code annotation implemented.

March 16\textsuperscript{th} 2012 – April 15\textsuperscript{th} 2012: evaluate CodeAnnotate via projects as well as iterate the design and implementation.

March 16\textsuperscript{th} 2012 – May 14\textsuperscript{th} 2012: conduct a user study to evaluate ASIDE in terms of developer behavior impact of interactive code annotation.

May 15\textsuperscript{th} 2012 – Sep 24\textsuperscript{th} 2012: analyze data from both project evaluation and user study, and write up a conference paper for CHI 2013.
Contributions summary

★ **Provide** an in-depth understanding of why software programmers make security errors during programming.

★ **Devise** a novel approach that reminds programmers potential insecure code and provides them secure programming support during program construction.

★ **Develop** interactive code refactoring and code annotation to assist programmers in producing code with less common code vulnerabilities.

★ **Implement** prototype software and conduct extensive open source projects study to evaluate the effectiveness of the techniques.

★ **Conduct** comprehensive user studies to evaluate the design of the implemented prototype as well as gain insights on how developers perceive this new approach.
THANK YOU ALL!

From: Jing Xie

University of North Carolina at Charlotte

Check ASIDE out at here: https://www.owasp.org/index.php/OWASP_ASIDE_Project