INTERACTIVE DEVELOPER SUPPORT FOR APPLICATION SECURITY

Helping developers to interactively detect, understand, and mitigate security vulnerabilities in their code.

Heather Richter Lipford
University of North Carolina at Charlotte
Software Security

Being resistant to malicious attacks which exploit security bugs in software.

Acunetix Vulnerability Testing Report 2017

POSTED ON JUNE 6, 2017 BY IAN MUSCAT

With Cross-site Scripting (XSS) vulnerabilities found on 50% of sampled targets, this year’s findings continue to reaffirm the widely held understanding that the web application vector is a major, viable and low-barrier-to-entry vector for attackers.

Hackers still exploiting eBay’s stored XSS vulnerabilities in 2017
Application Security Process

Development Team

- Implementation
- Unit Testing
- Code Review
- Fix Bugs

Application Security Team

- Audit
- Static Analysis
- Dynamic Analysis
- Pen testing
- Manual Inspection
- Triage

Bug Tracking

Tyler Thomas, Heather Lipford, and Bill Chu. From the eyes and ears of auditors: An investigation of security during application development. In preparation.
Developers rely on other parties/processes who they believe should handle all security concerns.

Developers believed that security issues do not apply to their particular development context.

A range of factors motivate and constrain attention paid to security concerns.

**Disconnect between general security knowledge and concrete secure programming practices**

Communicating with developers

“I think a lot of what I deal with is not necessarily the technology, it’s the how you communicate it effectively to developers. My boss likes to joke that we are 60 percent psychologists and 40 percent security professionals.”

- **Convince** developers there is a real problem
- **Motivate** stakeholders to fix the vulnerability
- **Explain** how to remediate
- **Train** for the future

**Security champion**
Member of the development team serving as an important liaison and advocate for security
Moving security “to the left”

“I would say that the big challenge there is that security and development have traditionally been disjointed and they have been separate teams. Security is the watchdogs, development does the work and all security has ever done is scan stuff. So write code, ask questions later. And we have to change that.”

- Engage developers with security within their existing tools and processes.
- Reduce the burden of security work by communicating based on their knowledge, and utilizing automated support as much as possible.
- Utilize developer knowledge of the application context to customize security tool support.
Interactions and Touchpoints

**Notification**: concrete, contextual, actionable communication with developers

**Auto-suggestion and automation**: recommending security actions that developers can choose to adopt.

**Annotation**: Gathering security-related information from developers

- Coding
- Unit testing
- Code Review
Touchpoint #1: Coding

Interactive Static Analysis

Increase developers’ awareness of security vulnerabilities, and their secure programming knowledge and behavior.

- Vulnerabilities related to lack of input validation, output encoding, or SQL injection
- Access control vulnerabilities

ASIDE: Application Security in the IDE
Interaction: Notification

Warnings and messages need to be:

- **Understandable** by developers with range of security expertise
- **Consistent** in their structure and presentation
- **Contextual** descriptions based upon the relevant lines of code
- **Actionable** guidance regarding how to trace and resolve the vulnerability
Interaction: Auto-suggestion

- Automatically insert sanitization code based on the type of input or output chosen
- Uses ESAPI validation methods
```java
String accountName = request.getParameter("AccountName");
AccountMapper accounts = getAccounts();

if (((User) request.getSession().getAttribute("USER")).ownAccount(accountName)) {
    accounts.updateAccount(accountName, 0);
} else {
}

String accountName = request.getParameter("AccountName");
AccountMapper accounts = getAccounts();

if (((User) request.getSession().getAttribute("USER")).ownAccount(accountName)) {
    accounts.updateAccount(accountName, 0);
} else {
    
```
Security Performance – Injection, XSS

- 131/143 (92%) taint sources identified
- Remainder due to JSP/frameworks not yet supported
- 118 additional taint sources identified
- 94 potentially exploitable
- 24 false positives

Security Performance – Access Control

- Comparison against known access control vulnerabilities in 6 open source projects
  - 26 known and 20 zero-day vulnerabilities detected

<table>
<thead>
<tr>
<th>Project</th>
<th>Known Vul.</th>
<th>Known Vul. By ASIDE-PHP</th>
<th>0-day Vul. by ASIDE-PHP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moodle 2.1.0</td>
<td>6[I], 7[L]</td>
<td>6[I]</td>
<td>1[I]</td>
</tr>
<tr>
<td>Mybloggie 2.1.3</td>
<td>3[M], 3[UT]</td>
<td>3[M], 3[UT]</td>
<td>15[M]</td>
</tr>
<tr>
<td>SCARF 1.0</td>
<td>1[M], 10[UT]</td>
<td>1[M], 10[UT]</td>
<td></td>
</tr>
<tr>
<td>Bilboblog 0.2.1</td>
<td>1[UT]</td>
<td>1[UT]</td>
<td></td>
</tr>
<tr>
<td>Wheatblog 1.1</td>
<td>1[M]</td>
<td>1[M]</td>
<td></td>
</tr>
<tr>
<td>PhpStat 1.5</td>
<td>1[M]</td>
<td>1[M]</td>
<td>4[M]</td>
</tr>
</tbody>
</table>

M: Missing check  I: Inconsistent
UT: Untrusted data  L: Logic error

User Behavior

Multiple user studies with advanced students, and two with professionals

The good:

- Raised awareness
- Almost all correct actions
- Liked the quick fixes and help

Needs improvement:

- Vulnerability severity and ranking
- Customize fixes

“Here I would like to see numbers only, but I don't quite see an option for that. I would probably go ahead and activate the letters and numbers quickfix and then modify it so that it's just numbers.”

User Behavior

ASIDE is unobtrusive and informative.

Automated code generation valuable.

Successful identification of access control logic, but difficulty tracing an access control vulnerability.

Needs more examples of exploits and severity of risks.

ASIDE increases awareness of security vulnerabilities, could improve the practice of secure programming.

Touchpoint #2: Unit Testing

Dynamic analysis to detect vulnerabilities not found in static analysis

- Detect Cross-Site Scripting (XSS) vulnerabilities due to improper encoding of untrusted data.
  - Automated construction and evaluation of XSS unit tests.
  - Notification of exact line number of vulnerable code

Minimize false positives by confirming vulnerabilities via execution.

Minimize false negatives by systematically generating attack strings.

Interaction: Automation

- **Unit Test Extraction**
  - Control flow analysis to slice the code into execution paths
  - Taint analysis to determine injection points of untrusted data

- **Attack Generation**
  - Model “context switching” as a context free grammar
  - Generate attack strings using sentences of the grammar

- **Attack Evaluation**
  - Execute attack strings in JWebUnit
  - Change Web page title with line number being tested

<table>
<thead>
<tr>
<th></th>
<th>Vulnerabilities reported</th>
<th>True Positive</th>
<th>False Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZAP</td>
<td>31</td>
<td>9</td>
<td>22</td>
</tr>
<tr>
<td>XSS Unit Testing</td>
<td>17</td>
<td>17</td>
<td>0</td>
</tr>
</tbody>
</table>

Evaluation on iTrust, open source medical records application
Touchpoint #3: Code Review

Security-oriented code review with static analysis

- Interaction between developers and application security experts earlier in the process
- Use static analysis (for example, ASIDE) to focus reviews around detected and mitigated vulnerabilities

- How can we support communication between developers and application security experts?
- How effective is code review for improving developer awareness and practice of application security?
Interaction: Annotation

- Extending Gerrit, an open-source code review tool
  - Reviewers rate vulnerabilities, leave comments
  - Developers ask questions and receive feedback
## Interactions and Touchpoints

<table>
<thead>
<tr>
<th></th>
<th>Coding</th>
<th>Unit testing</th>
<th>Code review</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notification</strong></td>
<td>Vulnerable practices</td>
<td>Identify vulnerable code based on test results</td>
<td>Security decisions, e.g. incorrect remediation</td>
</tr>
<tr>
<td><strong>Auto-suggest</strong></td>
<td>Mitigation controls, e.g. input validation code</td>
<td>Security unit tests for XSS vulnerability detection</td>
<td>Prioritizing issue review</td>
</tr>
<tr>
<td><strong>Annotation</strong></td>
<td>Security decisions, e.g. access control</td>
<td>Seek developer input to help security unit test generation</td>
<td>Comments and rationale of security decisions.</td>
</tr>
</tbody>
</table>
Research Challenges

- Reducing false positives
  - Understanding what developers interpret as a false positive
  - Focus on techniques and tools with low false positives
  - Utilize contextual information to improve accuracy

- Reducing false negatives
  - Characterize false negatives and remaining risk

- Reflect risk assessments
  - Prioritize tool feedback and functionality based upon risk

- Incentives
  - Understand tool features that discourage interaction
  - Understand costs versus benefits of tools and change in process
  - Organizational structures and processes that motivate security work
Acknowledgements

Supported by NSF #1318854, 1044745, 1523041

Collaborators:

- Bill Chu
- Mahmoud Mohammadi, Madiha Tabassum, Tyler Thomas, Jing Xie, and Jun Zhu
- Emerson Murphy-Hill and Justin Smith

http://aside.uncc.edu