Exploring the Relationship Between Web Application Development Tools and Security

Matthew Finifter and David Wagner
University of California, Berkeley
It’s a great time to be a developer!

<table>
<thead>
<tr>
<th>Languages</th>
<th>PHP</th>
<th>Java</th>
<th>Ruby</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl</td>
<td>Perl</td>
<td>Python</td>
<td>Scala</td>
</tr>
<tr>
<td>Haskell</td>
<td>Cold Fusion</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
It’s a great time to be a developer!

<table>
<thead>
<tr>
<th>Languages</th>
<th>Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP</td>
<td>Yii, ASP.NET, Zend, Struts, Django, Snap, GWT,</td>
</tr>
<tr>
<td></td>
<td>RoR, Mason, Sinatra, CakePHP, Fusebox, Catalyst,</td>
</tr>
<tr>
<td></td>
<td>Spring, Grails, Dancer, CodeIgniter, Tapestry,</td>
</tr>
<tr>
<td></td>
<td>Pyjamas, Symfony</td>
</tr>
<tr>
<td>PERL</td>
<td>PYTHON</td>
</tr>
<tr>
<td></td>
<td>Scala</td>
</tr>
<tr>
<td>HASKELL</td>
<td>COLD FUSION</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
It’s a great time to be a developer!

Languages

<table>
<thead>
<tr>
<th>PHP</th>
<th>JAVA</th>
<th>RUBY</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERL</td>
<td>PYTHON</td>
<td>SCALA</td>
</tr>
<tr>
<td>HASKELL</td>
<td>COLD FUSION</td>
<td>…</td>
</tr>
</tbody>
</table>

Frameworks

- Yii, ASP.NET, Zend, Struts, Django, Snap, GWT, RoR, Mason, Sinatra, CakePHP, Fusebox, Catalyst, Spring, Grails, Dancer, CodeIgniter, Tapestry, Pyjamas, Symfony
- Object Relational Model (ORM) Framework
- Templating Language
- Libraries
- Vulnerability Remediation Tools or Services
- Client-side framework
- Meta-framework
- Content Management System (CMS)
Choice is great, but...

- How should a developer or project manager choose?
- Is there any observable difference between different tools we might choose?
- What should you optimize for?
- How will you know you’ve made the right choices?
- We need meaningful comparisons between tools so that developers can make informed decisions.
Talk Outline

• Introduction
• Goals
• Methodology
• Results
• Conclusion and Future Work
Goals

• Encourage future work in this problem space

• Introduce methodology for evaluating differences between tools

• Evaluate security differences between different tools
  • Programming Language
  • Web Application Development Framework
  • Process for Finding Vulnerabilities
Methodology

• Secondary data set from [Prechelt 2010]

• Different groups of developers use different tools to implement the same functionality

• Control for differences in specifications, human variability

• Measure the security of the developed programs
  • Black-box penetration testing (Burp Suite Pro)
  • Manual security review

• Use statistical hypothesis testing to look for associations
Limitations

• Experimental design

• Only one security reviewer (me)

• Application not necessarily representative

• Small sample size

• … and more (see the paper)
Programming Language

• 3 Java teams, 3 Perl teams, 3 PHP teams

• Look for association between programming language and:
  • Total number of vulnerabilities found in the implementation
  • Number of vulnerabilities for each vulnerability class

• Main conclusion: 9 samples is too few to find these associations.
  • Maybe there is no association
  • Maybe we need more data
Results: Total Vulnerabilities

Total Number of Vulnerabilities

- Java 3
- Java 4
- Java 9
- PHP 6
- PHP 7
- PHP 8
- Perl 1
- Perl 2
- Perl 5

Manual
Both
Black-box
Results: Stored XSS

Stored XSS

<table>
<thead>
<tr>
<th></th>
<th>Java 3</th>
<th>Java 4</th>
<th>Java 9</th>
<th>PHP 6</th>
<th>PHP 7</th>
<th>PHP 8</th>
<th>Perl 1</th>
<th>Perl 2</th>
<th>Perl 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>Black-box</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Both</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12
Results: Reflected XSS
Results: SQL Injection
Results: Auth. Bypass

Authentication/Authorization Bypass

- Java 3
- Java 4
- Java 9
- PHP 6
- PHP 7
- PHP 8
- Perl 1
- Perl 2
- Perl 5

- Manual
- Both
- Black-box
Results: “Binary” Vulnerabilities

<table>
<thead>
<tr>
<th></th>
<th>No. Vulnerable Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CSRF</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Session Management</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Password Storage</strong></td>
<td>3</td>
</tr>
</tbody>
</table>

- **Perl**
- **Java**
- **PHP**
Framework Support

• Different frameworks offer different features

• Taxonomy of framework support
  • None
  • Manual
  • Opt-in
  • Opt-out
  • Always on
Framework Support

- Labeled each (team number, vulnerability class) with a framework support level.
- E.g., “team 4 had always-on CSRF protection.”
- This data set allows us to consider association between level of framework support and vulnerabilities.
- In other words, does a higher level of framework support help?
Framework Support

• No associations found for XSS, SQL injection, auth. bypass, or secure password storage.

• Statistically significant associations found for CSRF and session management.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Perl</td>
<td>•</td>
<td>none</td>
<td>opt-in</td>
<td>•</td>
<td>none</td>
<td>opt-in</td>
</tr>
<tr>
<td>2</td>
<td>Perl</td>
<td>•</td>
<td>none</td>
<td>•</td>
<td>none</td>
<td>•</td>
<td>none</td>
</tr>
<tr>
<td>5</td>
<td>Perl</td>
<td>•</td>
<td>none</td>
<td>•</td>
<td>none</td>
<td>•</td>
<td>opt-out</td>
</tr>
<tr>
<td>3</td>
<td>Java</td>
<td>manual</td>
<td>opt-out</td>
<td>•</td>
<td>none</td>
<td>•</td>
<td>none</td>
</tr>
<tr>
<td>4</td>
<td>Java</td>
<td>always on</td>
<td>opt-in</td>
<td>•</td>
<td>opt-in</td>
<td>•</td>
<td>opt-in</td>
</tr>
<tr>
<td>9</td>
<td>Java</td>
<td>•</td>
<td>none</td>
<td>opt-in</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>6</td>
<td>PHP</td>
<td>•</td>
<td>none</td>
<td>opt-out</td>
<td>•</td>
<td>opt-in</td>
<td>none</td>
</tr>
<tr>
<td>7</td>
<td>PHP</td>
<td>•</td>
<td>none</td>
<td>opt-out</td>
<td>•</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>8</td>
<td>PHP</td>
<td>•</td>
<td>none</td>
<td>opt-out</td>
<td>•</td>
<td>opt-in</td>
<td>none</td>
</tr>
</tbody>
</table>

*Table 5: Presence or absence of binary vulnerability classes, and framework support for preventing them.*
Individual Vulnerability Data

- More data to shed light on frameworks

- *How far away from chosen tools to find framework support?*
  - Framework used
  - Newer version of framework used
  - Another framework for language used
  - Some framework for some language
  - No known support

- For both automatic and manual framework support
Individual Vulnerability Data (Manual Support)

Where manual support exists to prevent vulnerabilities

- No known framework
- Some fwk. for some language
- Diff. fwk. for language used
- Newer version of fwk. used
- Framework used

Reflected XSS in JavaScript context
### Individual Vulnerability Data (Automatic Support)

<table>
<thead>
<tr>
<th>Language</th>
<th>Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java3</td>
<td>Authorization bypass</td>
</tr>
<tr>
<td>Java4</td>
<td>Authorization bypass</td>
</tr>
<tr>
<td>Java9</td>
<td>Authorization bypass</td>
</tr>
<tr>
<td>PHP6</td>
<td>Secure password storage</td>
</tr>
<tr>
<td>PHP7</td>
<td>Secure password storage</td>
</tr>
<tr>
<td>PHP8</td>
<td>Secure password storage</td>
</tr>
<tr>
<td>Perl1</td>
<td>Reflected XSS in JavaScript context</td>
</tr>
<tr>
<td>Perl2</td>
<td>Reflected XSS in JavaScript context</td>
</tr>
<tr>
<td>Perl5</td>
<td>Authorization bypass</td>
</tr>
</tbody>
</table>

- No known framework
- Some fwk. for some language
- Diff. fwk. for language used
- Newer version of fwk. used
- Framework used

Where automatic support exists to prevent vulnerabilities
Method of Finding Vulnerabilities

- Automated black-box penetration testing
- Manual source code review
Method of Finding Vulnerabilities

Black-box Manual

20 19 52
Results: Stored XSS

Stored XSS

Java 3  Java 4  Java 9  PHP 6  PHP 7  PHP 8  Perl 1  Perl 2  Perl 5
Manual  Both  Black-box
Results: Reflected XSS
Results: SQL Injection

![SQL Injection Chart]

- **X-axis**: Java 3, Java 4, Java 9, PHP 6, PHP 7, PHP 8, Perl 1, Perl 2, Perl 5
- **Y-axis**: 0, 1, 2, 3
- **Legend**:
  - Black: Manual
  - Gray: Both
  - Light Gray: Black-box
Results: Auth. Bypass

![Graph showing authentication/authorization bypass](image-url)
Results: “Binary” Vulnerabilities

No. Vulnerable Implementations

- CSRF
- Session Management
- Password Storage

Perl  Java  PHP
Related Work

• **BAU ET AL.** *State of the Art: Automated Black-box Web Application Vulnerability Testing.*

• **DOUPÉ ET AL.** *Why Johnny Can’t Pentest: An Analysis of Black-Box Web Vulnerability Scanners.*

• **PRECHELT ET AL.** *Plat_Forms: A Web Development Platform Comparison by an Exploratory Experiment Searching for Emergent Platform Properties.*

• **WAGNER ET AL.** *Comparing Bug Finding Tools with Reviews and Tests.*

• **WALDEN ET AL.** *Java vs. PHP: Security Implications of Language Choice for Web Applications.*

Conclusion

• We should quantify our tools along various dimensions
• This study started (but did not finish!) that task for security
• Language, framework, vulnerability-finding method
Conclusion

- Web security is still hard; each implementation had at least one vulnerability.
- Level of framework support appears to influence security.
- Manual framework support is ineffective.
- Manual code review more effective than black-box testing.
  - But they are complementary.
  - And they perform differently for different vulnerability classes.
Future Work

• Gathering and analyzing larger data sets

• Other dimensions: reliability, performance, maintainability, etc.

• Deeper understanding of why some tools fare better than others

• Not just web applications!
Thank you!

Matthew Finifter

finifter@cs.berkeley.edu