Securing Development with PMD

Teaching an Old Dog New Tricks
Integrating Security with Developer Tooling
Key Objectives

- Learn about PMD
- Understand how to extend PMD
- Think about enhancements to similar tools
What Is PMD?

- Open source static analysis tool
- Scans Java source code for potential problems
  - Possible bugs
  - Dead code
  - Suboptimal code
  - Overcomplicated expressions
  - Duplicate code

*Very little related to security!!*
Bug Finders vs Security Static Analysis

- **Bug Finders** (i.e. PMD)
  - Target buggy patterns
  - Minimize false positives even if high false negatives

- **Security Static Analysis**
  - Target insecure patterns
  - Minimize false negatives even if some false positives
  - Context of violation must be investigated
Why Extend Security to PMD?

- Used extensively by Java developers already
- Highly extensible with Rule and Report API
- Strong documentation and support network
- Integrates with many IDEs and build tools
- PMD internals operate similar to commercial tools
How does PMD work?

- Run against source file, directory, or archive
- Builds tree-like structure of source code (AST)
- Performs semantic, basic control & data analysis
- Traverses AST looking for patterns (Rules)
- Generates a report of Rule Violations
What Does AST Look Like?

Source Code

class Example {
    void bar() {
        while (baz)
            buz.doSomething();
    }
}

AST

CompilationUnit
    TypeDeclaration
    ClassDeclaration: (package private)
        UnmodifiedClassDeclaration(Example)
            ClassBody
                ClassBodyDeclaration
                    MethodDeclaration: (package private)
                        ResultType
                            MethodDeclaration(Example)
                                FormalParameters
                                    Block
                                        BlockStatement
                                            Statement
                                                WhileStatement
                                                    Expression
                                                        PrimaryExpression
                                                            PrimaryPrefix
                                                                Name:baz
                                                            Statement
                                                                Expression
                                                                    PrimaryExpression
                                                                        PrimaryPrefix
                                                                            Name:buz.doSomething
                                                                        Arguments
Extending PMD with Custom Rules

- Implement as Xpath expression or Java class
- Wire up rules for use by PMD in ruleset file
- Modify behavior by configuring rule properties
- Group rules into rulesets for enforcement
DEMO

Xpath and Java Rule Writing Demo
Resources to Help Writing Rules

- PMD Website
  - http://pmd.sourceforge.net/xpathruletutorial.html
  - http://pmd.sourceforge.net/howtowritearule.html (Java)

- PMD source code
  - net.sourceforge.pmd.rules.*
  - net.sourceforge.pmd.dfa.DaaRule

- PMD Applied (Centennial Books Nov 2005)

- PMD test cases & framework (wraps JUnit)
  - test.net.sourceforge.pmd.testframework
  - test.net.sourceforge.pmd.*
v1.0 Goals For Custom PMD Security Rules

- Add security without modifying PMD itself
- Write rules that identify “low hanging fruit”
- Perform analysis beyond lexing and pattern match
## Selecting Rules for Implementation

<table>
<thead>
<tr>
<th>GDS Assessment Vulnerability</th>
<th>Customer’s Secure Coding Guideline(s)</th>
<th>Rule Type</th>
<th>OWASP Top 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Injection</td>
<td>2.1 – Commands should not be Constructed through String Concatenation</td>
<td>Data Flow, Structural</td>
<td>A1: Injection</td>
</tr>
</tbody>
</table>
| Cross-Site Scripting (XSS)   | 1.1 – All Input Crossing a Trust Boundary Must be Validated  
1.2 – Data from External Sources must be Properly Encoded or Escaped | Data Flow                     | A2: Cross-Site Scripting (XSS)   |
| Arbitrary File Retrieval     | 1.1 – All Input Crossing a Trust Boundary Must be Validated  
3.2 – Callable Code Must Enforce Authorization Requirements | Data Flow                     | A4: Insecure Direct Object References |
| Use of Cryptographically Insecure Algorithms | 4.1 – Use of Sound Encryption Algorithms  
4.2 – Use of Sound Hashing Algorithms                          | Structural                     | A7: Insecure Cryptographic Storage |
| Arbitrary URL Redirection    | 1.1 – All Input Crossing a Trust Boundary Must be Validated                                            | Data Flow                     | A10: Un-validated Redirects and Forwards |
## Challenges to Writing PMD Security Rules

<table>
<thead>
<tr>
<th>PMD Analysis Limitations</th>
<th>Impact on Detecting Security Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis limited to single file at a time</td>
<td>Data often passes through multiple files/classes and tiers</td>
</tr>
<tr>
<td>Data Flow Analyzer (DFA) limited to single method (intraprocedural)</td>
<td>Security bugs often result of mixing data and code in wrong context</td>
</tr>
<tr>
<td>DFA tracks local variable declarations and references, but does not evaluate expressions</td>
<td></td>
</tr>
<tr>
<td>Symbols limited to source file, resulting in names and types not fully resolved</td>
<td>Custom code often wraps well-known APIs (Java or Framework)</td>
</tr>
<tr>
<td>Only analyzes JSP files that are XHTML-compliant (i.e. JSP Documents / XML syntax)</td>
<td>Standard JSP syntax more common</td>
</tr>
<tr>
<td></td>
<td>Often severe web application security bugs in presentation layer</td>
</tr>
</tbody>
</table>
Rule Writing Challenges – JSP Files

#1 – Overcome XHTML limitation

- **Solution**: Leverage JSP compiler
- **Result**: Java implementation of JSP logic in `_jspService method`
- **Benefit**:
  - Identify security bugs in any JSP
  - Scope of PMD’s analysis increased
Example of JSP to Java Translation

### JSP Scriptlet Code

```jsp
<%
String a1 = request.getParameter("y1");
String b1 = a1;
%>
<%=b1 %>
```

### Translated Java code equivalent

```java
public void _jspService(HttpServletRequest request,
    HttpServletResponse response)
    throws java.io.IOException,
        ServletException {

    ..snip..
    PageContext pageContext = null;
    ..snip..
    out = pageContext.getOut();
    ..snip..
    String a1 = request.getParameter("y1");
    String b1 = a1;
    out.print(b1);

```
Rule Writing Challenges – Reporting

#2 – Report JSP security violations meaningful to developer

Solution:
- Wrote custom Source Map Format (SMAP) translator (JSR-045)
- Implemented `net.sourceforge.pmd.IRuleViolation`

Result: Report findings in terms of JSP line numbers

Benefit:
- JSP developers remediate bugs in JSP
- Security violations understood by PMD built-in renders
SMAP Example

SMAP index7_jsp.java JSP *S JSP *F + 0 index7.jsp index7.jsp *L 2,10:53,0 12,3:55 14:58,0 15:60 16,3:61,0 *E

Header (SMAP, generated filename, default stratum)

Stratum Section

File Section (contains translated filenames and path)

Line Section (associates line numbers in input source with output source)

End Section
Rule Writing Challenges – DFA w/PMD

#3 – Despite PMD limitations, perform data flow analysis

- **Solution:** Use PMD DFA and Symbol Table

- **Result:**
  - Determine if variable assignments assigned *source*
  - Track those *tainted variables* down each data flow
  - Report security violations if tainted variable passed to *sink*

- **Benefit:** Automated, accurate tracing from source to sink
public void _jspService(HttpServletRequest request,
                HttpServletResponse response)
        throws java.io.IOException,
                ServletException {
        ...
        String a1 = request.getParameter("y1");

        String b1 = a1;
        out.print(b1);

        variable definition
        Name=a1, Type=String

        variable references
        Name=request.getParameter
        Arguments=y1 (Literal)
public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {
    ..snip..
    String a1 = request.getParameter("y1");
    String b1 = a1;
    out.print(b1);

    variable definition
    Name=a1, Type=String (tainted variable)

    variable reference
    Name=request.getParameter
    (method, tainted source)
    Arguments=y1 (Literal)
    Type=javax.servlet.http.HttpServletRequest
public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException
{
    ..snip..

    String a1 = request.getParameter("y1");
    String b1 = a1;
    out.print(b1);

    // variable definition
    Name=b1, Type=String (tainted variable)

    // variable reference
    Name=a1 (tainted variable)
public void _jspService(HttpServletRequest request, HttpServletResponse response) throws java.io.IOException, ServletException {
    ..snip..
    String a1 = request.getParameter("y1");
    String b1 = a1;
    out.print(b1);

    variable references
    Name=out.print
    Arguments=b1 (Name)
    (tainted variable)
public void _jspService(HttpServletRequest request, HttpServletResponse response) throws java.io.IOException, ServletException {
    ..snip..
    String a1 = request.getParameter("y1");
    String b1 = a1;
    out.print(b1);

    XSS Vulnerability

    variable references
    Name=out.print
    (method, XSS sink)
    Type=javax.servlet.jsp.JspWriter
    Arguments=b1 (Name)
    (tainted variable)
DFA Security Rule Usage Notes

- Violations need to be manually investigated for proper escaping/validation

- Configurable sources and sinks via properties
  - URL Redirection
    - `javax.servlet.http.HttpServletResponse.sendRedirect`
  - SQL Injection
    - `java.sql.execute`

- Effective source/sink same method / “reflected” variants
PMD Structural Rule Example – SQLi

- DFA susceptible to false negatives
  - Data traverse multiple files between source and sink

- Supplement with structural rule
  - Investigates AST AdditiveExpression nodes
  - Performs following analysis
    - Is string a SQL command?
    - Is concatenated data of type String?
    - Is concatenated data a method argument?
Using PMD Security Rules
Basic Usage Steps

- **Configure** CLASSPATH
  - Add `pmd-gds-1.0.jar`
  - Add jars/classes used when building (for type resolution)

- Configure PMD to use
  `/rulesets/GDS/SecureCodingRuleset.xml`

- Run PMD and audit results
PMD ANT Task Example - CLASSPATH

```xml
<path id="pmd.classpath">
    <fileset dir="${pmd.dir.home}\lib">
        <include name="pmd-${pmd.version}.jar" />
        ..snip..
    </fileset>
    <pathelement location="lib\${gds.jar}" />
    <pathelement location="${app1.src}\build\classes"/>
    <fileset dir="C:\tomcat\apache-tomcat-6.0.29\lib">
        <include name="servlet-api.jar" />
    </fileset>
</path>

<target name="pmd" description="Runs PMD">
    <taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask" classpathref="pmd.classpath" />
    <pmd rulesetfiles="rulesets/GDS/SecureCodingRuleset.xml" shortFilenames="false"
         formatter type="text" toConsole="true" />
    <fileset dir="${app1.src}" include="**/*.java" />
</target>
```
PMD ANT Task Example – Rules Config

```xml
<path id="pmd.classpath">
  <fileset dir="${pmd.dir.home}\lib">
    <include name="pmd-${pmd.version}.jar" />
    ..snip..
  </fileset>
  <pathelement location="lib\${gds.jar}" />
  <pathelement location="${app1.src}\build\classes"/>
  <fileset dir="C:\tomcat\apache-tomcat-6.0.29\lib">
    <include name="servlet-api.jar" />
  </fileset>
</path>

<target name="pmd" description="Runs PMD">
  <taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask"
    classpathref="pmd.classpath" />
  <pmd rulesetfiles="rulesets/GDS/SecureCodingRuleset.xml" shortFilenames="false"
    formatter type="text" toConsole="true" />
  <fileset dir="${app1.src}" include="**/*.java" /></target>
```
Configuring JSP to Java Translation

- Add JSP compiler task to build tool (build.xml)

- **Configure** `smapSuppressed` to *false* and `smapDump` to *true*

```
<jasper2 validateXml="false" uriroot="C:\Code\web.war"
   webXmlFragment="${jspBuildDir}/WEB-INF/generated_web.xml" outputDir="${jspBuildDir}/WEB-INF/src"
   smapSuppressed="false" smapDumped="true"/>
```

- Add extra clean task to remove `.smap` files before production deployment
Custom Rules with PMD Eclipse Plug-in

- Plug-in only supports xpath rules out of the box

- Put custom rules on plug-in `CLASSPATH`
  - Requires modification of PMD Eclipse plugin jars
  - Add rules to PMD Eclipse plugin source and compile
  - Wrap PMD Eclipse plugin with custom plugin
Current and Future Development

- Publish version 1.0 of Secure Coding Ruleset @ https://github.com/GDSSecurity
- Integrate NIST Juliet Test cases
- Contribute to PMD project (need to pass tests first!)
- Extend rules beyond Java with PMD 5
- Write PMD 5.0 Rules
- Enhance PMD feature set
Conclusion

- Learned about PMD and extensibility
- Discussed approach for rule writing & deployment
- Use, add and improve SecureCodingRuleset on GitHub
- Look for other developer tools where it would be practical to add security
References

- https://www.owasp.org/
- pmd.sourceforge.net
- http://tomcopelandland.blogs.com/
- PMD Applied (Centennial Books Nov 2005)
- Secure Programming with Static Analysis (Addison-Wesley Professional July 2007)