Securing Development with PMD
Teaching an Old Dog New Tricks

Justin Clarke, Gotham Digital Science

justin@gdssecurity.com
justin.clarke@owasp.org
www.linkedin.com/in/connectjunkie

HAGEN©2007
Well, this trick has been around for five years now, so we might be able to learn it...

Integrating Security with Developer Tooling

pmd
DON'T SHOOT THE MESSENGER

C:\

eclipse

HUDSON

< APACHE ANT >

Bugzilla

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

$\left.\begin{array}{l}\\ \\ \\ \\ \\ \end{array}\right\}$ ***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

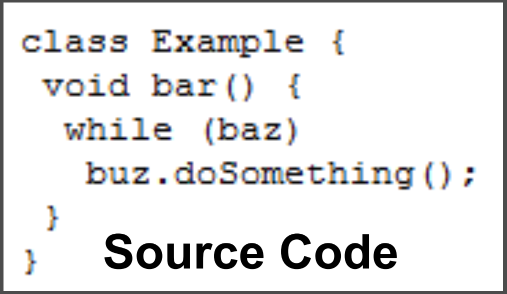
Why Extend Security to PMD?

- 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?

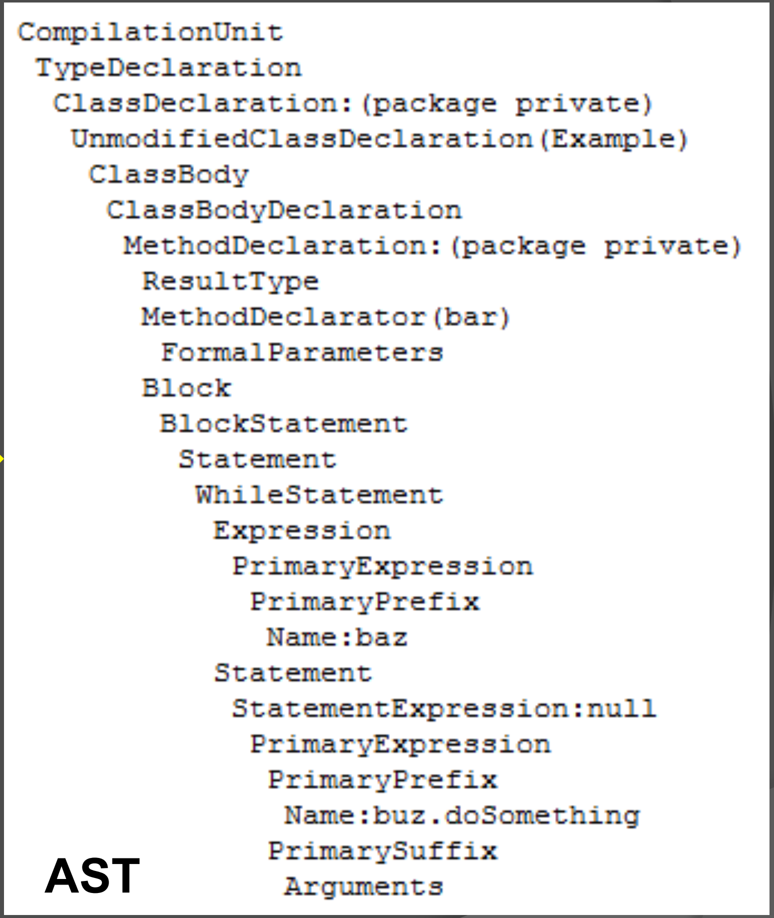
- Run against source file, directory, or archive
- Builds tree-like structure of source code (AST)
- Performs semantic, basic control & data analysis
- Traverses AST looking for patterns (Rules)
- Generates a report of Rule Violations

What Does AST Look Like?



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

Source Code



```
CompilationUnit
 TypeDeclaration
  ClassDeclaration:(package private)
   UnmodifiedClassDeclaration(Example)
    ClassBody
     ClassBodyDeclaration
      MethodDeclaration:(package private)
       ResultType
       MethodDeclarator(bar)
        FormalParameters
       Block
        BlockStatement
         Statement
          WhileStatement
           Expression
            PrimaryExpression
             PrimaryPrefix
              Name:baz
           Statement
            StatementExpression:null
             PrimaryExpression
              PrimaryPrefix
               Name:buz.doSomething
              PrimarySuffix
               Arguments
```

AST

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 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

GDS Assessment Vulnerability	Customer's Secure Coding Guideline(s)	Rule Type	OWASP Top 10
SQL Injection	2.1 – Commands should not be Constructed through String Concatenation	Data Flow, Structural	A1: Injection
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

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

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

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

JSP Scriptlet Code

```
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 );
```

Translated Java code equivalent

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

PMD Data Flow Analysis

```
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

DataFlowNodes

variable references
Name=request.getParameter
Arguments=y1 (Literal)

PMD Data Flow Analysis

```
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)

DataFlowNode

PMD Data Flow Analysis

```
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

variable reference
Name=request.getParameter
Arguments=y1 (Literal)

PMD Data Flow Analysis Extended (XSS)

```
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

PMD Data Flow Analysis Extended (XSS)

```
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)***

PMD Data Flow Analysis Extended (XSS)

```
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)

PMD Data Flow Analysis Extended (XSS)

```
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?

DEMO

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

```
<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 name="**/*.java" /></fileset>
 </pmd>
</target>
```

PMD ANT Task Example – Rules Config

```
<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 name="**/*.java" /></fileset>
 </pmd>
</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 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

- http://www.nysforum.org/committees/security/051409_pdfs/A%20CISO%27S%20Guide%20to%20Application%20Security.pdf
- http://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html
- https://www.owasp.org/
- pmd.sourceforge.net
- http://tomcopeland.blogs.com/
- PMD Applied (Centennial Books Nov 2005)
- Secure Programming with Static Analysis (Addison-Wesley Professional July 2007)