**Category: SECURITY**

### Dm: Hardcoded constant database password (DMI\_CONSTANT\_DB\_PASSWORD)

This code creates a database connect using a hardcoded, constant password. Anyone with access to either the source code or the compiled code can easily learn the password.

### Dm: Empty database password (DMI\_EMPTY\_DB\_PASSWORD)

This code creates a database connect using a blank or empty password. This indicates that the database is not protected by a password.

### HRS: HTTP cookie formed from untrusted input – SEVERE (HRS\_REQUEST\_PARAMETER\_TO\_COOKIE)

This code constructs an HTTP Cookie using an untrusted HTTP parameter. If this cookie is added to an HTTP response, it will allow a HTTP response splitting vulnerability. See <http://en.wikipedia.org/wiki/HTTP_response_splitting> for more information.

FindBugs looks only for the most blatant, obvious cases of HTTP response splitting. If FindBugs found any, you almost certainly have more vulnerabilities that FindBugs doesn't report. If you are concerned about HTTP response splitting, you should seriously consider using a commercial static analysis or pen-testing tool.

### HRS: HTTP Response splitting vulnerability – SEVERE (HRS\_REQUEST\_PARAMETER\_TO\_HTTP\_HEADER)

This code directly writes an HTTP parameter to an HTTP header, which allows for a HTTP response splitting vulnerability. See<http://en.wikipedia.org/wiki/HTTP_response_splitting> for more information.

FindBugs looks only for the most blatant, obvious cases of HTTP response splitting. If FindBugs found any, you almost certainly have more vulnerabilities that FindBugs doesn't report. If you are concerned about HTTP response splitting, you should seriously consider using a commercial static analysis or pen-testing tool.

### PT: Absolute path traversal in servlet – SEVERE (PT\_ABSOLUTE\_PATH\_TRAVERSAL)

The software uses an HTTP request parameter to construct a pathname that should be within a restricted directory, but it does not properly neutralize absolute path sequences such as "/abs/path" that can resolve to a location that is outside of that directory. See <http://cwe.mitre.org/data/definitions/36.html> for more information.

FindBugs looks only for the most blatant, obvious cases of absolute path traversal. If FindBugs found any, you almost certainly have more vulnerabilities that FindBugs doesn't report. If you are concerned about absolute path traversal, you should seriously consider using a commercial static analysis or pen-testing tool.

### PT: Relative path traversal in servlet – SEVERE (PT\_RELATIVE\_PATH\_TRAVERSAL)

The software uses an HTTP request parameter to construct a pathname that should be within a restricted directory, but it does not properly neutralize sequences such as ".." that can resolve to a location that is outside of that directory. See <http://cwe.mitre.org/data/definitions/23.html> for more information.

FindBugs looks only for the most blatant, obvious cases of relative path traversal. If FindBugs found any, you almost certainly have more vulnerabilities that FindBugs doesn't report. If you are concerned about relative path traversal, you should seriously consider using a commercial static analysis or pen-testing tool.

### SQL: Nonconstant string passed to execute method on an SQL statement – SEVERE (SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE)

The method invokes the execute method on an SQL statement with a String that seems to be dynamically generated. Consider using a prepared statement instead. It is more efficient and less vulnerable to SQL injection attacks.

### SQL: A prepared statement is generated from a nonconstant String – SEVERE (SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING)

The code creates an SQL prepared statement from a nonconstant String. If unchecked, tainted data from a user is used in building this String, SQL injection could be used to make the prepared statement do something unexpected and undesirable.

### XSS: JSP reflected cross site scripting vulnerability – SEVERE (XSS\_REQUEST\_PARAMETER\_TO\_JSP\_WRITER)

This code directly writes an HTTP parameter to JSP output, which allows for a cross site scripting vulnerability. See <http://en.wikipedia.org/wiki/Cross-site_scripting> for more information.

FindBugs looks only for the most blatant, obvious cases of cross site scripting. If FindBugs found any, you almost certainly have more cross site scripting vulnerabilities that FindBugs doesn't report. If you are concerned about cross site scripting, you should seriously consider using a commercial static analysis or pen-testing tool.

### XSS: Servlet reflected cross site scripting vulnerability in error page - SEVERE (XSS\_REQUEST\_PARAMETER\_TO\_SEND\_ERROR)

This code directly writes an HTTP parameter to a Server error page (using HttpServletResponse.sendError). Echoing this untrusted input allows for a reflected cross site scripting vulnerability. See <http://en.wikipedia.org/wiki/Cross-site_scripting> for more information.

FindBugs looks only for the most blatant, obvious cases of cross site scripting. If FindBugs found any, you almost certainly have more cross site scripting vulnerabilities that FindBugs doesn't report. If you are concerned about cross site scripting, you should seriously consider using a commercial static analysis or pen-testing tool.

### XSS: Servlet reflected cross site scripting vulnerability – SEVERE (XSS\_REQUEST\_PARAMETER\_TO\_SERVLET\_WRITER)

This code directly writes an HTTP parameter to Servlet output, which allows for a reflected cross site scripting vulnerability. See <http://en.wikipedia.org/wiki/Cross-site_scripting>for more information.

FindBugs looks only for the most blatant, obvious cases of cross site scripting. If FindBugs found any, you almost certainly have more cross site scripting vulnerabilities that FindBugs doesn't report. If you are concerned about cross site scripting, you should seriously consider using a commercial static analysis or pen-testing tool.

**Category: MALICIOUS\_CODE**

**DP: Classloaders should only be created inside doPrivileged block (DP\_CREATE\_CLASSLOADER\_INSIDE\_DO\_PRIVILEGED)**

This code creates a classloader, which needs permission if a security manage is installed. If this code might be invoked by code that does not have security permissions, then the classloader creation needs to occur inside a doPrivileged block.

**DP: Method invoked that should be only be invoked inside a adoPrivileged block (DP\_DO\_INSIDE\_DO\_PRIVILEGED)**

This code invokes a method that requires a security permission check. If this code will be granted security permissions, but might be invoked by code that does not have security permissions, then the invocation needs to occur inside a doPrivileged block.

**EI: May expose internal representation by returning reference to mutable object (EI\_EXPOSE\_REP)**

Returning a reference to a mutable object value stored in one of the object's fields exposes the internal representation of the object.  If instances are accessed by untrusted code, and unchecked changes to the mutable object would compromise security or other important properties, you will need to do something different. Returning a new copy of the object is better approach in many situations.

**EI2: May expose internal representation by incorporating reference to mutable object (EI\_EXPOSE\_REP2)**

This code stores a reference to an externally mutable object into the internal representation of the object.  If instances are accessed by untrusted code, and unchecked changes to the mutable object would compromise security or other important properties, you will need to do something different. Storing a copy of the object is better approach in many situations.

**FI: Finalizer should be protected, not public (FI\_PUBLIC\_SHOULD\_BE\_PROTECTED)**

A class's finalize() method should have protected access, not public.

**MS: May expose internal static state by storing a mutable object into a static field (EI\_EXPOSE\_STATIC\_REP2)**

This code stores a reference to an externally mutable object into a static field. If unchecked changes to the mutable object would compromise security or other important properties, you will need to do something different. Storing a copy of the object is better approach in many situations.

**MS: Field isn't final and can't be protected from malicious code (MS\_CANNOT\_BE\_FINAL)**

A mutable static field could be changed by malicious code or by accident from another package. Unfortunately, the way the field is used doesn't allow any easy fix to this problem.

**MS: Public static method may expose internal representation by returning array (MS\_EXPOSE\_REP)**

A public static method returns a reference to an array that is part of the static state of the class. Any code that calls this method can freely modify the underlying array. One fix is to return a copy of the array.

**MS: Field should be both final and package protected (MS\_FINAL\_PKGPROTECT)**

A mutable static field could be changed by malicious code or by accident from another package. The field could be made package protected and/or made final to avoid this vulnerability.

**MS: Field is a mutable array (MS\_MUTABLE\_ARRAY)**

A final static field references an array and can be accessed by malicious code or by accident from another package. This code can freely modify the contents of the array.

**MS: Field is a mutable Hashtable (MS\_MUTABLE\_HASHTABLE)**

A final static field references a Hashtable and can be accessed by malicious code or by accident from another package. This code can freely modify the contents of the Hashtable.

**MS: Field should be moved out of an interface and made package protected (MS\_OOI\_PKGPROTECT)**

A final static field that is defined in an interface references a mutable object such as an array or hashtable. This mutable object could be changed by malicious code or by accident from another package. To solve this, the field needs to be moved to a class and made package protected to avoid this vulnerability.

**MS: Field should be package protected (MS\_PKGPROTECT)**

A mutable static field could be changed by malicious code or by accident. The field could be made package protected to avoid this vulnerability.

**MS: Field isn't final but should be (MS\_SHOULD\_BE\_FINAL)**

This static field public but not final, and could be changed by malicious code or by accident from another package. The field could be made final to avoid this vulnerability.

**MS: Field isn't final but should be refactored to be so (MS\_SHOULD\_BE\_REFACTORED\_TO\_BE\_FINAL)**

This static field public but not final, and could be changed by malicious code or by accident from another package. The field could be made final to avoid this vulnerability. However, the static initializer contains more than one write to the field, so doing so will require some refactoring.