Fortify Security Report

Jul 13, 2016

zacharylewis

Executive Summary

Issues Overview

On Jun 21, 2013, a source code review was performed over the WebGoat5.0 code base. 187 files, 9,564 LOC (Executable) were scanned and reviewed for defects that could lead to potential security vulnerabilities. A total of 2320 reviewed findings were uncovered during the analysis.

Issues by Category

Cross-Site Scripting: Persistent	611
System Information Leak	284
Cross-Site Scripting: Reflected	270
Poor Error Handling: Overly Broad Catch	149
Poor Logging Practice: Use of a System Output Stream	137
Access Control: Database	124
Privacy Violation	101
Dangerous File Inclusion	93
Trust Boundary Violation	85
SQL Injection	78
Poor Error Handling: Empty Catch Block	32
Password Management: Password in Comment	28
Cross-Site Request Forgery	27
Log Forging	25
Header Manipulation	20
Password Management: Hardcoded Password	20
Code Correctness: Erroneous Class Compare	19
Unsafe Reflection	17
Hidden Field	15
Poor Error Handling: Overly Broad Throws	15
Unreleased Resource: Streams	15
Privacy Violation: Heap Inspection	12
Unreleased Resource: Database	12
Null Dereference	8
Denial of Service	7
Axis 2 Misconfiguration: Debug Information	6
J2EE Bad Practices: Threads	6
Poor Style: Non-final Public Static Field	6
Command Injection	5
J2EE Bad Practices: getConnection()	5
Path Manipulation	5
Code Correctness: Erroneous String Compare	4
Code Correctness: Regular Expressions Denial of Service	4
Cookie Security: Cookie not Sent Over SSL	4
J2EE Bad Practices: Leftover Debug Code	4
Missing Check against Null	4
Poor Style: Redundant Initialization	4
Weak Encryption	4
Code Correctness: Multiple Stream Commits	3
J2EE Bad Practices: Non-Serializable Object Stored in Session	3
J2EE Misconfiguration: Missing Error Handling	3
Redundant Null Check	3
System Information Leak: HTML Comment in JSP	3
System Information Leak: Incomplete Servlet Error Handling	3
Dead Code: Unused Method	2
J2EE Misconfiguration: Missing Data Transport Constraint	2
Missing Check for Null Parameter	2
Object Model Violation: Just one of equals() and hashCode() Defined	2

Password Management: Empty Password	2
Password Management: Null Password	2
Race Condition: Static Database Connection	2
Resource Injection	2
Unchecked Return Value	2
Unreleased Resource: Sockets	2
Weak Cryptographic Hash	2
Cross-Site Scripting: Poor Validation	1
Denial of Service: Parse Double	1
Dynamic Code Evaluation: Code Injection	1
File Disclosure: J2EE	1
Insecure Randomness	1
J2EE Bad Practices: Sockets	1
J2EE Misconfiguration: Excessive Servlet Mappings	1
J2EE Misconfiguration: Excessive Session Timeout	1
J2EE Misconfiguration: Missing Servlet Mapping	1
Password Management: Password in Configuration File	1
Poor Error Handling: Throw Inside Finally	1
Poor Style: Confusing Naming	1
Poor Style: Value Never Read	1
Race Condition: Singleton Member Field	1
XPath Injection	1

Recommendations and Conclusions

The Issues Category section provides Fortify recommendations for addressing issues at a generic level. The recommendations for specific fixes can be extrapolated from those generic recommendations by the development group.

Project Summary

Code Base Summary

Code location: C:/Program Files/HP_Fortify/HP_Fortify_SCA_and_Apps_3.90/Samples/advanced/webgoat/WebGoat5.0
Number of Files: 187
Lines of Code: 9564
Build Label: <No Build Label>

Scan Information

Scan time: 02:26
SCA Engine version: 5.16.0.0042
Machine Name: fortify
Username running scan: fortify

Results Certification

Results Certification Valid

Details:

Results Signature:

SCA Analysis Results has Valid signature

Rules Signature:

There were no custom rules used in this scan

Attack Surface

Attack Surface:
Command Line Arguments:
org.owasp.webgoat.lessons.Encoding.main
org.owasp.webgoat.session.CreateDB.main
org.owasp.webgoat.session.WebgoatProperties.main
org.owasp.webgoat.util.Exec.main

File System:
java.io.FileInputStream.FileInputStream

Private Information:
null.null.null
javax.crypto.SecretKeyFactory.generateSecret
org.owasp.webgoat.session.Employee.getCcn
org.owasp.webgoat.session.Employee.getSsn

Java Properties:
java.lang.System.getProperty
java.util.Properties.load
javax.servlet.GenericServlet.getInitParameter
javax.servlet.ServletContext.getInitParameter

Stream:
java.io.InputStream.read

System Information:
null.null.null
java.lang.Throwable.getLocalizedMessage
java.lang.Throwable.getMessage
java.net.URISyntaxException.getMessage
javax.servlet.ServletContext.getInitParameter
javax.servlet.ServletContext.getRealPath
javax.servlet.ServletContext.getResourcePaths

Web:
java.net.URLConnection.getInputStream
javax.servlet.ServletRequest.getRemoteAddr
javax.servlet.ServletRequest.getRemoteHost
javax.servlet.http.HttpServletRequest.getMethod

Filter Set Summary

Current Enabled Filter Set:
Security Auditor View

Filter Set Details:

Folder Filters:
If [fortify priority order] contains critical Then set folder to Critical
If [fortify priority order] contains high Then set folder to High
If [fortify priority order] contains medium Then set folder to Medium
If [fortify priority order] contains low Then set folder to Low

Audit Guide Summary

Audit guide not enabled

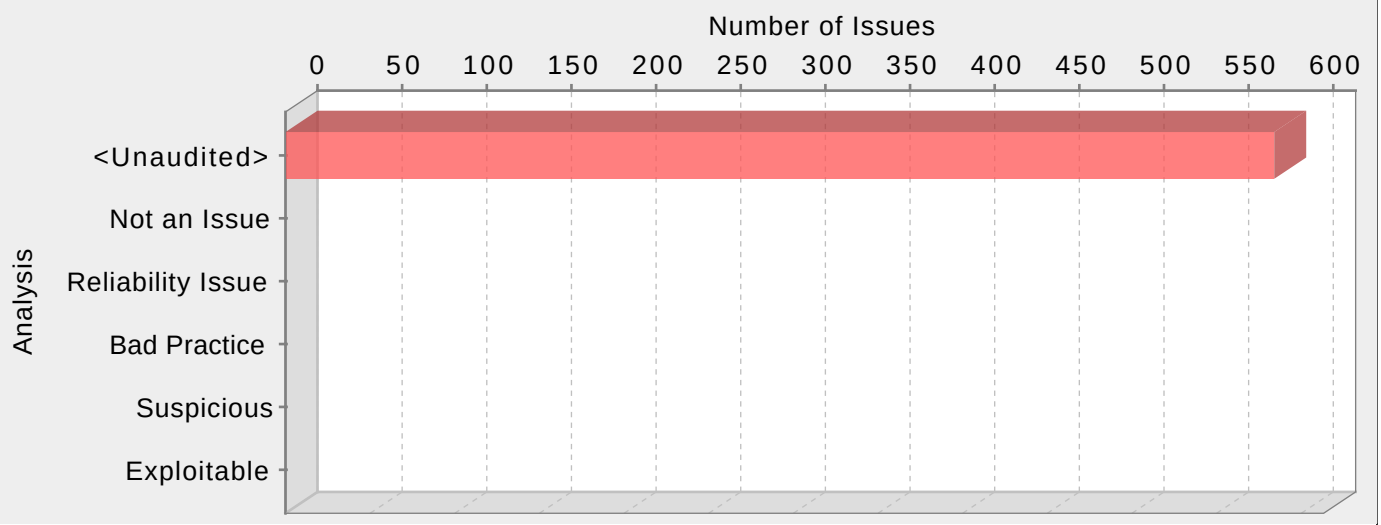
Results Outline

Overall number of results

The scan found 2320 issues.

Vulnerability Examples by Category

Category: Cross-Site Scripting: Persistent (584 Issues)



Abstract:

The method concept1() in BackDoors.java sends unvalidated data to a web browser on line 125, which can result in the browser executing malicious code.

Explanation:

Cross-site scripting (XSS) vulnerabilities occur when:

1. Data enters a web application through an untrusted source. In the case of Persistent (also known as Stored) XSS, the untrusted source is typically a database or other back-end datastore, while in the case of Reflected XSS it is typically a web request.

2. The data is included in dynamic content that is sent to a web user without being validated for malicious code.

The malicious content sent to the web browser often takes the form of a segment of JavaScript, but may also include HTML, Flash or any other type of code that the browser may execute. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data like cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

Example 1: The following JSP code segment queries a database for an employee with a given ID and prints the corresponding employee's name.

```
<%...
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from emp where id="+eid);
if (rs != null) {
rs.next();
String name = rs.getString("name");
}
%>

Employee Name: <%= name %>
```

This code functions correctly when the values of name are well-behaved, but it does nothing to prevent exploits if they are not. This code can appear less dangerous because the value of name is read from a database, whose contents are apparently managed by the application. However, if the value of name originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the database, an attacker can execute malicious commands in the user's web browser. This type of exploit, known as Persistent (or Stored) XSS, is particularly insidious because the indirection caused by the data store makes it more difficult to identify the threat and increases the possibility that the attack will affect multiple users. XSS got its start in this form with web sites that offered a "guestbook" to visitors. Attackers would include JavaScript in their guestbook entries, and all subsequent visitors to the guestbook page would execute the malicious code.

Example 2: The following JSP code segment reads an employee ID, eid, from an HTTP request and displays it to the user.

```
<% String eid = request.getParameter("eid"); %>
...
Employee ID: <%= eid %>
```

As in Example 1, this code operates correctly if eid contains only standard alphanumeric text. If eid has a value that includes meta-characters or source code, then the code will be executed by the web browser as it displays the HTTP response.

Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use e-mail or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers. This mechanism of exploiting vulnerable web applications is known as Reflected XSS.

As the examples demonstrate, XSS vulnerabilities are caused by code that includes unvalidated data in an HTTP response. There are three vectors by which an XSS attack can reach a victim:

- As in Example 1, the application stores dangerous data in a database or other trusted data store. The dangerous data is subsequently read back into the application and included in dynamic content. Persistent XSS exploits occur when an attacker injects dangerous content into a data store that is later read and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user.

- As in Example 2, data is read directly from the HTTP request and reflected back in the HTTP response. Reflected XSS exploits occur when an attacker causes a user to supply dangerous content to a vulnerable web application, which is then reflected back to the user and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to victims. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces victims to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the user, the content is executed and proceeds to transfer private information, such as cookies that may include session information, from the user's machine to the attacker or perform other nefarious activities.

- A source outside the application stores dangerous data in a database or other data store, and the dangerous data is subsequently read back into the application as trusted data and included in dynamic content.

A number of modern web frameworks provide mechanisms for performing validation of user input. Struts and Struts 2 are among them. To highlight the unvalidated sources of input, the rulepacks dynamically re-prioritize the issues reported by HP Fortify Static Code Analyzer by lowering their probability of exploit and providing pointers to the supporting evidence whenever the framework validation mechanism is in use. We refer to this feature as Context-Sensitive Ranking. To further assist the HP Fortify user with the auditing process, the Fortify Security Research Group makes available the Data Validation project template that groups the issues into folders based on the validation mechanism applied to their source of input.

Recommendations:

The solution to XSS is to ensure that validation occurs in the correct places and checks for the correct properties.

Since XSS vulnerabilities occur when an application includes malicious data in its output, one logical approach is to validate data immediately before it leaves the application. However, because web applications often have complex and intricate code for generating dynamic content, this method is prone to errors of omission (missing validation). An effective way to mitigate this risk is to also perform input validation for XSS.

Web applications must validate their input to prevent other vulnerabilities, such as SQL injection, so augmenting an application's existing input validation mechanism to include checks for XSS is generally relatively easy. Despite its value, input validation for XSS does not take the place of rigorous output validation. An application may accept input through a shared data store or other trusted source, and that data store may accept input from a source that does not perform adequate input validation. Therefore, the application cannot implicitly rely on the safety of this or any other data. This means the best way to prevent XSS vulnerabilities is to validate everything that enters the application or leaves the application destined for the user.

The most secure approach to validation for XSS is to create a whitelist of safe characters that are allowed to appear in HTTP content and accept input composed exclusively of characters in the approved set. For example, a valid username might only include alpha-numeric characters or a phone number might only include digits 0-9. However, this solution is often infeasible in web applications because many characters that have special meaning to the browser should still be considered valid input once they are encoded, such as a web design bulletin board that must accept HTML fragments from its users.

A more flexible, but less secure approach is known as blacklisting, which selectively rejects or escapes potentially dangerous characters before using the input. In order to form such a list, you first need to understand the set of characters that hold special meaning for web browsers. Although the HTML standard defines what characters have special meaning, many web browsers try to correct common mistakes in HTML and may treat other characters as special in certain contexts. The CERT(R) Coordination Center at the Software Engineering Institute at Carnegie Mellon University provides the following details about special characters in various contexts [1]:

In the content of a block-level element (in the middle of a paragraph of text):

- "<" is special because it introduces a tag.

- "&" is special because it introduces a character entity.

- ">" is special because some browsers treat it as special, on the assumption that the author of the page intended to include an opening "<", but omitted it in error.

The following principles apply to attribute values:

- In attribute values enclosed with double quotes, the double quotes are special because they mark the end of the attribute value.

- In attribute values enclosed with single quote, the single quotes are special because they mark the end of the attribute value.

- In attribute values without any quotes, white-space characters, such as space and tab, are special.

- "&" is special when used with certain attributes, because it introduces a character entity.

In URLs, for example, a search engine might provide a link within the results page that the user can click to re-run the search. This can be implemented by encoding the search query inside the URL, which introduces additional special characters:

- Space, tab, and new line are special because they mark the end of the URL.

- "&" is special because it either introduces a character entity or separates CGI parameters.

- Non-ASCII characters (that is, everything above 128 in the ISO-8859-1 encoding) are not allowed in URLs, so they are considered to be special in this context.

- The "%" symbol must be filtered from input anywhere parameters encoded with HTTP escape sequences are decoded by server -side code. For example, "%" must be filtered if input such as "%68%65%6C%6C%6F" becomes "hello" when it appears on the web page in question.

Within the body of a <SCRIPT> </SCRIPT>:

- The semicolon, parenthesis, curly braces, and new line should be filtered in situations where text could be inserted directly into a pre-existing script tag.

Server-side scripts:

- Server-side scripts that convert any exclamation characters (!) in input to double-quote characters (") on output might require additional filtering.

Other possibilities:

- If an attacker submits a request in UTF-7, the special character '<' appears as '+ADw-' and may bypass filtering. If the output is included in a page that does not explicitly specify an encoding format, then some browsers try to intelligently identify the encoding based on the content (in this case, UTF-7).

Once you identify the correct points in an application to perform validation for XSS attacks and what special characters the validation should consider, the next challenge is to identify how your validation handles special characters. If special characters are not considered valid input to the application, then you can reject any input that contains special characters as invalid. A second option in this situation is to remove special characters with filtering. However, filtering has the side effect of changing any visual representation of the filtered content and may be unacceptable in circumstances where the integrity of the input must be preserved for display.

If input containing special characters must be accepted and displayed accurately, validation must encode any special characters to remove their significance. A complete list of ISO 8859-1 encoded values for special characters is provided as part of the official HTML specification [2].

Many application servers attempt to limit an application's exposure to cross-site scripting vulnerabilities by providing implementations for the functions responsible for setting certain specific HTTP response content that perform validation for the characters essential to a cross-site scripting attack. Do not rely on the server running your application to make it secure. When an application is developed there are no guarantees about what application servers it will run on during its lifetime. As standards and known exploits evolve, there are no guarantees that application servers will also stay in sync.

Tips:

1. The HP Fortify Secure Coding Rulepacks treat the database as a source of untrusted data, which can lead to XSS vulnerabilities. If the database is a trusted resource in your environment, use custom filters to filter out dataflow issues that include the DATABASE taint flag or originate from database sources.

2. Even though URL encoding untrusted data protects against many XSS attacks, some browsers (specifically, Internet Explorer 6 and 7 and possibly others) automatically decode content at certain locations within the Document Object Model (DOM) prior to passing it to the JavaScript interpreter. To reflect this danger, the rulepacks no longer treat URL encoding routines as sufficient to protect against Cross-Site Scripting. Data values that are URL encoded and subsequently output will cause Fortify to report Cross-Site Scripting: Poor Validation vulnerabilities.

3. Fortify RTA adds protection against this category.

BackDoors.java, line 125 (Cross-Site Scripting: Persistent)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		

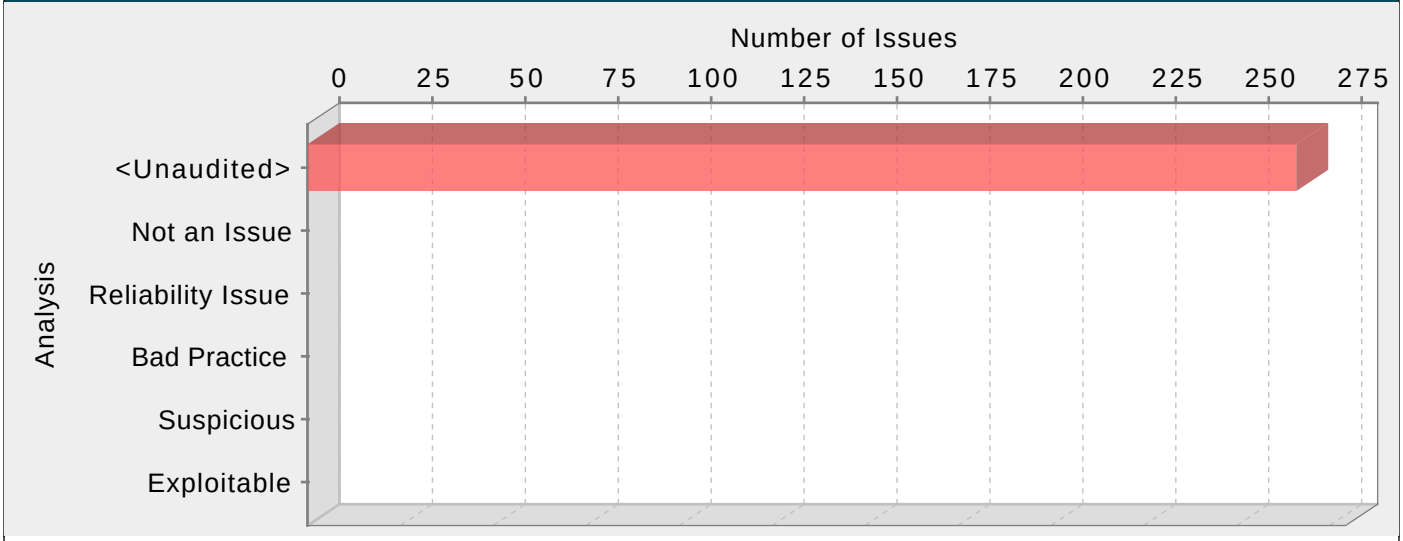
Abstract:	The method concept1() in BackDoors.java sends unvalidated data to a web browser on line 125, which can result in the browser executing malicious code.
Source:	BackDoors.java:113 java.sql.Statement.executeQuery()

```
111   }
112
113   ResultSet rs = statement.executeQuery(arrSQL[0]);
114   if (rs.next())
115   {
```

Sink: BackDoors.java:125 org.apache.ecs.html.TD.TD()

```
123       t.addElement(tr);
124       tr = new TR();
125       tr.addElement(new TD(rs.getString("userid")));
126       tr.addElement(new TD(rs.getString("password")));
127       tr.addElement(new TD(rs.getString("ssn")));
```

Category: Cross-Site Scripting: Reflected (266 Issues)



Abstract:

The method getSource() in AbstractLesson.java sends unvalidated data to a web browser on line 688, which can result in the browser executing malicious code.

Explanation:

Cross-site scripting (XSS) vulnerabilities occur when:

1. Data enters a web application through an untrusted source. In the case of Reflected XSS, the untrusted source is typically a web request, while in the case of Persisted (also known as Stored) XSS it is typically a database or other back-end datastore.

2. The data is included in dynamic content that is sent to a web user without being validated for malicious code.

The malicious content sent to the web browser often takes the form of a segment of JavaScript, but may also include HTML, Flash or any other type of code that the browser may execute. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data like cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

Example 1: The following JSP code segment reads an employee ID, eid, from an HTTP request and displays it to the user.

```
<% String eid = request.getParameter("eid"); %>
...
Employee ID: <%= eid %>
```

The code in this example operates correctly if eid contains only standard alphanumeric text. If eid has a value that includes meta-characters or source code, then the code will be executed by the web browser as it displays the HTTP response.

Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use e-mail or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers. This mechanism of exploiting vulnerable web applications is known as Reflected XSS.

Example 2: The following JSP code segment queries a database for an employee with a given ID and prints the corresponding employee's name.

```
<%...
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from emp where id="+eid);
if (rs != null) {
rs.next();
String name = rs.getString("name");
}
%>
Employee Name: <%= name %>
```

As in Example 1, this code functions correctly when the values of name are well-behaved, but it does nothing to prevent exploits if they are not. Again, this code can appear less dangerous because the value of name is read from a database, whose contents are apparently managed by the application. However, if the value of name originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the database, an attacker can execute malicious commands in the user's web browser. This type of exploit, known as Persistent (or Stored) XSS, is particularly insidious because the indirection caused by the data store makes it more difficult to identify the threat and increases the possibility that the attack will affect multiple users. XSS got its start in this form with web sites that offered a "guestbook" to visitors. Attackers would include JavaScript in their guestbook entries, and all subsequent visitors to the guestbook page would execute the malicious code.

As the examples demonstrate, XSS vulnerabilities are caused by code that includes unvalidated data in an HTTP response. There are three vectors by which an XSS attack can reach a victim:

- As in Example 1, data is read directly from the HTTP request and reflected back in the HTTP response. Reflected XSS exploits occur when an attacker causes a user to supply dangerous content to a vulnerable web application, which is then reflected back to the user and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to victims. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces victims to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the user, the content is executed and proceeds to transfer private information, such as cookies that may include session information, from the user's machine to the attacker or perform other nefarious activities.

- As in Example 2, the application stores dangerous data in a database or other trusted data store. The dangerous data is subsequently read back into the application and included in dynamic content. Persistent XSS exploits occur when an attacker injects dangerous content into a data store that is later read and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user.

- A source outside the application stores dangerous data in a database or other data store, and the dangerous data is subsequently read back into the application as trusted data and included in dynamic content.

A number of modern web frameworks provide mechanisms for performing validation of user input. Struts and Struts 2 are among them. To highlight the unvalidated sources of input, the rulepacks dynamically re-prioritize the issues reported by HP Fortify Static Code Analyzer by lowering their probability of exploit and providing pointers to the supporting evidence whenever the framework validation mechanism is in use. We refer to this feature as Context-Sensitive Ranking. To further assist the HP Fortify user with the auditing process, the Fortify Security Research Group makes available the Data Validation project template that groups the issues into folders based on the validation mechanism applied to their source of input.

Recommendations:

The solution to XSS is to ensure that validation occurs in the correct places and checks for the correct properties.

Since XSS vulnerabilities occur when an application includes malicious data in its output, one logical approach is to validate data immediately before it leaves the application. However, because web applications often have complex and intricate code for generating dynamic content, this method is prone to errors of omission (missing validation). An effective way to mitigate this risk is to also perform input validation for XSS.

Web applications must validate their input to prevent other vulnerabilities, such as SQL injection, so augmenting an application's existing input validation mechanism to include checks for XSS is generally relatively easy. Despite its value, input validation for XSS does not take the place of rigorous output validation. An application may accept input through a shared data store or other trusted source, and that data store may accept input from a source that does not perform adequate input validation. Therefore, the application cannot implicitly rely on the safety of this or any other data. This means the best way to prevent XSS vulnerabilities is to validate everything that enters the application or leaves the application destined for the user.

The most secure approach to validation for XSS is to create a whitelist of safe characters that are allowed to appear in HTTP content and accept input composed exclusively of characters in the approved set. For example, a valid username might only include alpha-numeric characters or a phone number might only include digits 0-9. However, this solution is often infeasible in web applications because many characters that have special meaning to the browser should still be considered valid input once they are encoded, such as a web design bulletin board that must accept HTML fragments from its users.

A more flexible, but less secure approach is known as blacklisting, which selectively rejects or escapes potentially dangerous characters before using the input. In order to form such a list, you first need to understand the set of characters that hold special meaning for web browsers. Although the HTML standard defines what characters have special meaning, many web browsers try to correct common mistakes in HTML and may treat other characters as special in certain contexts. The CERT(R) Coordination Center at the Software Engineering Institute at Carnegie Mellon University provides the following details about special characters in various contexts [1]:

In the content of a block-level element (in the middle of a paragraph of text):

- "<" is special because it introduces a tag.

- "&" is special because it introduces a character entity.

- ">" is special because some browsers treat it as special, on the assumption that the author of the page intended to include an opening "<", but omitted it in error.

The following principles apply to attribute values:

- In attribute values enclosed with double quotes, the double quotes are special because they mark the end of the attribute value.

- In attribute values enclosed with single quote, the single quotes are special because they mark the end of the attribute value.

- In attribute values without any quotes, white-space characters, such as space and tab, are special.

- "&" is special when used with certain attributes, because it introduces a character entity.

In URLs, for example, a search engine might provide a link within the results page that the user can click to re-run the search. This can be implemented by encoding the search query inside the URL, which introduces additional special characters:

- Space, tab, and new line are special because they mark the end of the URL.

- "&" is special because it either introduces a character entity or separates CGI parameters.

- Non-ASCII characters (that is, everything above 128 in the ISO-8859-1 encoding) are not allowed in URLs, so they are considered to be special in this context.

- The "%" symbol must be filtered from input anywhere parameters encoded with HTTP escape sequences are decoded by server -side code. For example, "%" must be filtered if input such as "%68%65%6C%6C%6F" becomes "hello" when it appears on the web page in question.

Within the body of a <SCRIPT> </SCRIPT>:

- The semicolon, parenthesis, curly braces, and new line should be filtered in situations where text could be inserted directly into a pre-existing script tag.

Server-side scripts:

- Server-side scripts that convert any exclamation characters (!) in input to double-quote characters (") on output might require additional filtering.

Other possibilities:

- If an attacker submits a request in UTF-7, the special character '<' appears as '+ADw-' and may bypass filtering. If the output is included in a page that does not explicitly specify an encoding format, then some browsers try to intelligently identify the encoding based on the content (in this case, UTF-7).

Once you identify the correct points in an application to perform validation for XSS attacks and what special characters the validation should consider, the next challenge is to identify how your validation handles special characters. If special characters are not considered valid input to the application, then you can reject any input that contains special characters as invalid. A second option in this situation is to remove special characters with filtering. However, filtering has the side effect of changing any visual representation of the filtered content and may be unacceptable in circumstances where the integrity of the input must be preserved for display.

If input containing special characters must be accepted and displayed accurately, validation must encode any special characters to remove their significance. A complete list of ISO 8859-1 encoded values for special characters is provided as part of the official HTML specification [2].

Many application servers attempt to limit an application's exposure to cross-site scripting vulnerabilities by providing implementations for the functions responsible for setting certain specific HTTP response content that perform validation for the characters essential to a cross-site scripting attack. Do not rely on the server running your application to make it secure. When an application is developed there are no guarantees about what application servers it will run on during its lifetime. As standards and known exploits evolve, there are no guarantees that application servers will also stay in sync.

Tips:

1. The HP Fortify Secure Coding Rulepacks treat the database as a source of untrusted data, which can lead to XSS vulnerabilities. If the database is a trusted resource in your environment, use custom filters to filter out dataflow issues that include the DATABASE taint flag or originate from database sources.

2. Even though URL encoding untrusted data protects against many XSS attacks, some browsers (specifically, Internet Explorer 6 and 7 and possibly others) automatically decode content at certain locations within the Document Object Model (DOM) prior to passing it to the JavaScript interpreter. To reflect this danger, the rulepacks no longer treat URL encoding routines as sufficient to protect against Cross-Site Scripting. Data values that are URL encoded and subsequently output will cause Fortify to report Cross-Site Scripting: Poor Validation vulnerabilities.

3. Fortify RTA adds protection against this category.

AbstractLesson.java, line 688 (Cross-Site Scripting: Reflected)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	The method getSource() in AbstractLesson.java sends unvalidated data to a web browser on line 688, which can result in the browser executing malicious code.		
Source:	ParameterParser.java:627 javax.servlet.ServletRequest.getParameterValues()		

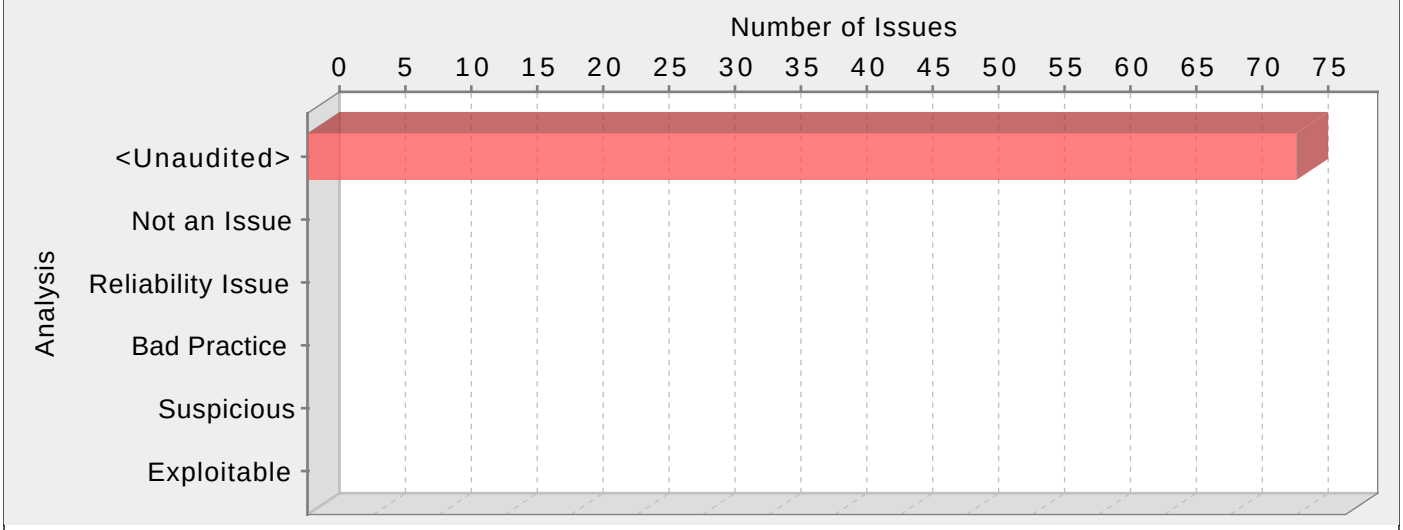
```
625                 throws ParameterNotFoundException
626             {
```

```
627                     String[] values = request.getParameterValues(name);
628
629                     if (values == null)
```

Sink: AbstractLesson.java:688 org.apache.ecs.html.Title.Title()

```
686
687                     Head head = new Head();
688                     head.addElement(new Title(getSourceFileName()));
689                     head.addElement(new StringElement(
690                      "<meta name=\"Author\" content=\"Bruce Mayhew\">"));
```

Category: Privacy Violation (75 Issues)



Abstract:

The method ParameterNotFoundException() in ParameterNotFoundException.java mishandles confidential information, which can compromise user privacy and is often illegal.

Explanation:

Privacy violations occur when:

1. Private user information enters the program.

2. The data is written to an external location, such as the console, file system, or network.

Example: The following code contains a logging statement that tracks the contents of records added to a database by storing them in a log file. Among other values that are stored, the getPassword() function returns the user-supplied plaintext password associated with the account.

```
pass = getPassword();
...
dbmsLog.println(id+":"+pass+":"+type+":"+tstamp);
```

The code in the example above logs a plaintext password to the filesystem. Although many developers trust the filesystem as a safe storage location for data, it should not be trusted implicitly, particularly when privacy is a concern.

Private data can enter a program in a variety of ways:

- Directly from the user in the form of a password or personal information

- Accessed from a database or other data store by the application

- Indirectly from a partner or other third party

Sometimes data that is not labeled as private can have a privacy implication in a different context. For example, student identification numbers are usually not considered private because there is no explicit and publicly-available mapping to an individual student's personal information. However, if a school generates identification numbers based on student social security numbers, then the identification numbers should be considered private.

Security and privacy concerns often seem to compete with each other. From a security perspective, you should record all important operations so that any anomalous activity can later be identified. However, when private data is involved, this practice can in fact create risk.

Although there are many ways in which private data can be handled unsafely, a common risk stems from misplaced trust. Programmers often trust the operating environment in which a program runs, and therefore believe that it is acceptable to store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted. For example, in 2004, an unscrupulous employee at AOL sold approximately 92 million private customer e-mail addresses to a spammer marketing an offshore gambling web site [1].

In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated. Depending on its location, the type of business it conducts, and the nature of any private data it handles, an organization may be required to comply with one or more of the following federal and state regulations:

- Safe Harbor Privacy Framework [3]

- Gramm-Leach Bliley Act (GLBA) [4]

- Health Insurance Portability and Accountability Act (HIPAA) [5]

- California SB-1386 [6]

Despite these regulations, privacy violations continue to occur with alarming frequency.

Recommendations:

When security and privacy demands clash, privacy should usually be given the higher priority. To accomplish this and still maintain required security information, cleanse any private information before it exits the program.

To enforce good privacy management, develop and strictly adhere to internal privacy guidelines. The guidelines should specifically describe how an application should handle private data. If your organization is regulated by federal or state law, ensure that your privacy guidelines are sufficiently strenuous to meet the legal requirements. Even if your organization is not regulated, you must protect private information or risk losing customer confidence.

The best policy with respect to private data is to minimize its exposure. Applications, processes, and employees should not be granted access to any private data unless the access is required for the tasks that they are to perform. Just as the principle of least privilege dictates that no operation should be performed with more than the necessary privileges, access to private data should be restricted to the smallest possible group.

Tips:

1. As part of any thorough audit for privacy violations, ensure that custom rules have been written to identify all sources of private or otherwise sensitive information entering the program. Most sources of private data cannot be identified automatically. Without custom rules, your check for privacy violations is likely to be substantially incomplete.

2. The Fortify Java Annotations FortifyPassword, FortifyNotPassword, FortifyPrivate and FortifyNotPrivate can be used to indicate which fields and variables represent passwords and private data.

3. A number of modern web frameworks provide mechanisms for performing validation of user input. Struts and Struts 2 are among them. To highlight the unvalidated sources of input, the HP Fortify Secure Coding Rulepacks dynamically re-prioritize the issues reported by HP Fortify Static Code Analyzer by lowering their probability of exploit and providing pointers to the supporting evidence whenever the framework validation mechanism is in use. We refer to this feature as Context-Sensitive Ranking. To further assist the HP Fortify user with the auditing process, the Fortify Security Research Group makes available the Data Validation project template that groups the issues into folders based on the validation mechanism applied to their source of input.

4. Fortify RTA adds protection against this category.

ParameterNotFoundException.java, line 54 (Privacy Violation)

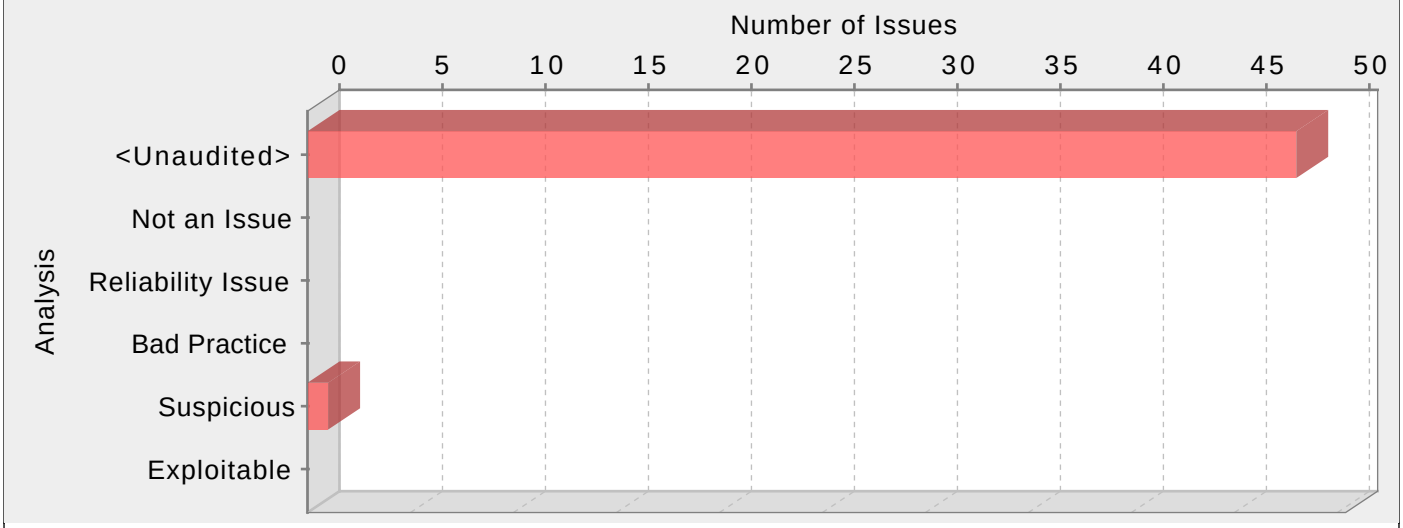
Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	The method ParameterNotFoundException() in ParameterNotFoundException.java mishandles confidential information, which can compromise user privacy and is often illegal.		
Source:	DOS_Login.java:89 Read org.owasp.webgoat.lessons.DOS_Login.PASSWORD()		

```
87                  String password =******
88                  username = s.getParser().getRawParameter(USERNAME);
89                  password =******
90
91                  // don;t allow user name from other lessons.  it would be too simple.
```

Sink: ParameterNotFoundException.java:54 java.lang.Exception.Exception()

```
52              public ParameterNotFoundException(String s)
53              {
54               super(s);
55              }
56          }
```

Category: SQL Injection (49 Issues)



Abstract:

On line 90 of ViewDatabase.java, the method createContent() invokes a SQL query built using input coming from an untrusted source. This call could allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.

Explanation:

SQL injection errors occur when:

1. Data enters a program from an untrusted source.

2. The data is used to dynamically construct a SQL query.

Example 1: The following code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where the owner matches the user name of the currently-authenticated user.

```
...
String userName = ctx.getAuthenticatedUserName();
String itemName = request.getParameter("itemName");
String query = "SELECT * FROM items WHERE owner = '"
+ userName + "' AND itemname = '"
+ itemName + "'";
ResultSet rs = stmt.execute(query);
...
```

The query that this code intends to execute follows:

```
SELECT * FROM items
WHERE owner = <userName>
AND itemname = <itemName>;
```

However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string "name' OR 'a'='a" for itemName, then the query becomes the following:

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name' OR 'a'='a';
```

The addition of the OR 'a'='a' condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

```
SELECT * FROM items;
```

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

Example 2: This example examines the effects of a different malicious value passed to the query constructed and executed in Example 1. If an attacker with the user name wiley enters the string "name'; DELETE FROM items; --" for itemName, then the query becomes the following two queries:

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name';

DELETE FROM items;

--'
```

Many database servers, including Microsoft(R) SQL Server 2000, allow multiple SQL statements separated by semicolons to be executed at once. While this attack string results in an error on Oracle and other database servers that do not allow the batch-execution of statements separated by semicolons, on databases that do allow batch execution, this type of attack allows the attacker to execute arbitrary commands against the database.

Notice the trailing pair of hyphens (--), which specifies to most database servers that the remainder of the statement is to be treated as a comment and not executed [4]. In this case the comment character serves to remove the trailing single-quote left over from the modified query. On a database where comments are not allowed to be used in this way, the general attack could still be made effective using a trick similar to the one shown in Example 1. If an attacker enters the string "name'); DELETE FROM items; SELECT * FROM items WHERE 'a'='a", the following three valid statements will be created:

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name';

DELETE FROM items;

SELECT * FROM items WHERE 'a'='a';
```

One traditional approach to preventing SQL injection attacks is to handle them as an input validation problem and either accept only characters from a whitelist of safe values or identify and escape a blacklist of potentially malicious values. Whitelisting can be a very effective means of enforcing strict input validation rules, but parameterized SQL statements require less maintenance and can offer more guarantees with respect to security. As is almost always the case, blacklisting is riddled with loopholes that make it ineffective at preventing SQL injection attacks. For example, attackers can:

- Target fields that are not quoted

- Find ways to bypass the need for certain escaped meta-characters

- Use stored procedures to hide the injected meta-characters

Manually escaping characters in input to SQL queries can help, but it will not make your application secure from SQL injection attacks.

Another solution commonly proposed for dealing with SQL injection attacks is to use stored procedures. Although stored procedures prevent some types of SQL injection attacks, they fail to protect against many others. Stored procedures typically help prevent SQL injection attacks by limiting the types of statements that can be passed to their parameters. However, there are many ways around the limitations and many interesting statements that can still be passed to stored procedures. Again, stored procedures can prevent some exploits, but they will not make your application secure against SQL injection attacks.

Recommendations:

The root cause of a SQL injection vulnerability is the ability of an attacker to change context in the SQL query, causing a value that the programmer intended to be interpreted as data to be interpreted as a command instead. When a SQL query is constructed, the programmer knows what should be interpreted as part of the command and what should be interpreted as data. Parameterized SQL statements can enforce this behavior by disallowing data-directed context changes and preventing nearly all SQL injection attacks. Parameterized SQL statements are constructed using strings of regular SQL, but where user-supplied data needs to be included, they include bind parameters, which are placeholders for data that is subsequently inserted. In other words, bind parameters allow the programmer to explicitly specify to the database what should be treated as a command and what should be treated as data. When the program is ready to execute a statement, it specifies to the database the runtime values to use for each of the bind parameters without the risk that the data will be interpreted as a modification to the command.

The previous example can be rewritten to use parameterized SQL statements (instead of concatenating user supplied strings) as follows:

```
...
String userName = ctx.getAuthenticatedUserName();
String itemName = request.getParameter("itemName");
String query =
"SELECT * FROM items WHERE itemname=? AND owner=?";
PreparedStatement stmt = conn.prepareStatement(query);
stmt.setString(1, itemName);
```

```
stmt.setString(2, userName);
ResultSet results = stmt.execute();
...
```

More complicated scenarios, often found in report generation code, require that user input affect the structure of the SQL statement, for instance by adding a dynamic constraint in the WHERE clause. Do not use this requirement to justify concatenating user input to create a query string. Prevent SQL injection attacks where user input must affect command structure with a level of indirection: create a set of legitimate strings that correspond to different elements you might include in a SQL statement. When constructing a statement, use input from the user to select from this set of application-controlled values.

Tips:

1. A common mistake is to use parameterized SQL statements that are constructed by concatenating user-controlled strings. Of course, this defeats the purpose of using parameterized SQL statements. If you are not certain that the strings used to form parameterized statements are constants controlled by the application, do not assume that they are safe because they are not being executed directly as SQL strings. Thoroughly investigate all uses of user-controlled strings in SQL statements and verify that none can be used to modify the meaning of the query.

2. A number of modern web frameworks provide mechanisms for performing validation of user input. Struts and Struts 2 are among them. To highlight the unvalidated sources of input, the HP Fortify Secure Coding Rulepacks dynamically re-prioritize the issues reported by HP Fortify Static Code Analyzer by lowering their probability of exploit and providing pointers to the supporting evidence whenever the framework validation mechanism is in use. We refer to this feature as Context-Sensitive Ranking. To further assist the HP Fortify user with the auditing process, the Fortify Security Research Group makes available the Data Validation project template that groups the issues into folders based on the validation mechanism applied to their source of input.

3. Fortify RTA adds protection against this category.

ViewDatabase.java, line 90 (SQL Injection)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	On line 90 of ViewDatabase.java, the method createContent() invokes a SQL query built using input coming from an untrusted source. This call could allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.		

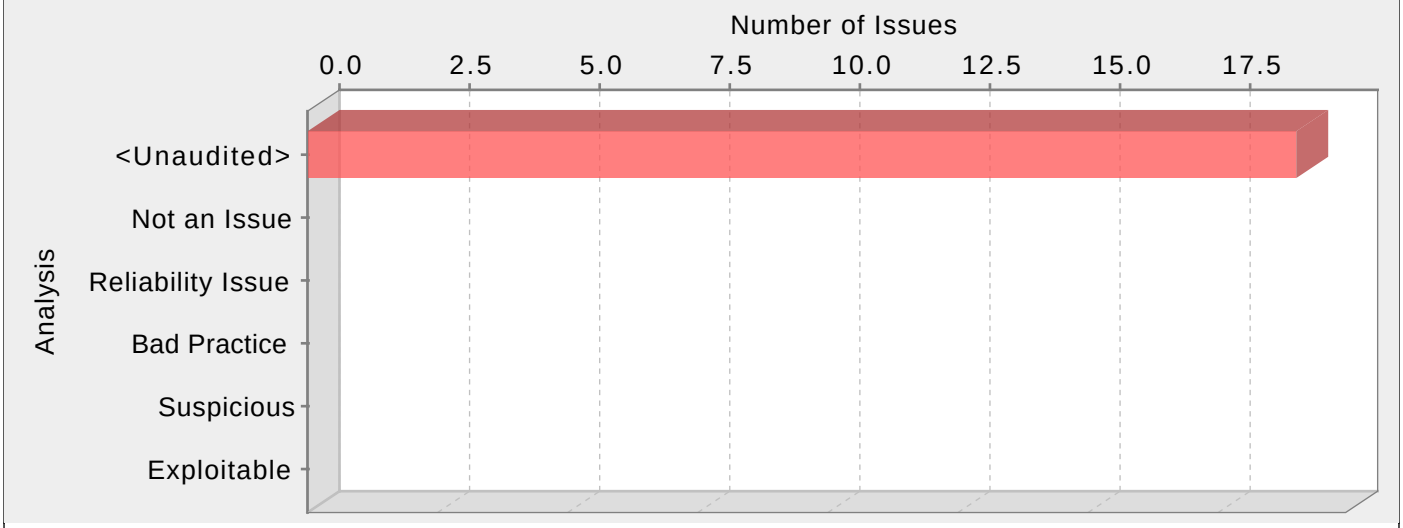
Source: ParameterParser.java:627 javax.servlet.ServletRequest.getParameterValues()

```
625                 throws ParameterNotFoundException
626                {
627              String[] values = request.getParameterValues(name);
628
629              if (values == null)
```

Sink: ViewDatabase.java:90 java.sql.Statement.executeQuery()

```
88                 ResultSet.CONCUR_READ_ONLY);
89               ResultSet results = statement.executeQuery(sqlStatement
90                .toString());
91
92               if ((results != null) && (results.first() == true))
```

Category: Password Management: Hardcoded Password (19 Issues)



Abstract:

Hardcoded passwords can compromise system security in a way that cannot be easily remedied.

Explanation:

It is never a good idea to hardcode a password. Not only does hardcoding a password allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. Once the code is in production, the password cannot be changed without patching the software. If the account protected by the password is compromised, the owners of the system will be forced to choose between security and availability.

Example: The following code uses a hardcoded password to connect to a database:

```
...
DriverManager.getConnection(url, "scott", "tiger");
...
```

This code will run successfully, but anyone who has access to it will have access to the password. Once the program has shipped, there is no going back from the database user "scott" with a password of "tiger" unless the program is patched. A devious employee with access to this information can use it to break into the system. Even worse, if attackers have access to the bytecode for the application they can use the javap -c command to access the disassembled code, which will contain the values of the passwords used. The result of this operation might look something like the following for the example above:

```
javap -c ConnMngr.class
22: ldc   #36; //String jdbc:mysql://ixne.com/rxsql
24: ldc   #38; //String scott
26: ldc   #17; //String tiger
```

Recommendations:

Passwords should never be hardcoded and should generally be obfuscated and managed in an external source. Storing passwords in plaintext anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the password.

Some third party products claim the ability to manage passwords in a more secure way. For example, WebSphere Application Server 4.x uses a simple XOR encryption algorithm for obfuscating values, but be skeptical about such facilities. WebSphere and other application servers offer outdated and relatively weak encryption mechanisms that are insufficient for security-sensitive environments. For a secure solution, the only viable option available today appears to be a proprietary one that you create.

Tips:

1. The Fortify Java Annotations FortifyPassword and FortifyNotPassword can be used to indicate which fields and variables represent passwords.

2. When identifying null, empty, or hardcoded passwords, default rules only consider fields and variables that contain the word password. However, the Custom Rules Editor provides the Password Management wizard that makes it easy to create rules for detecting password management issues on custom-named fields and variables.

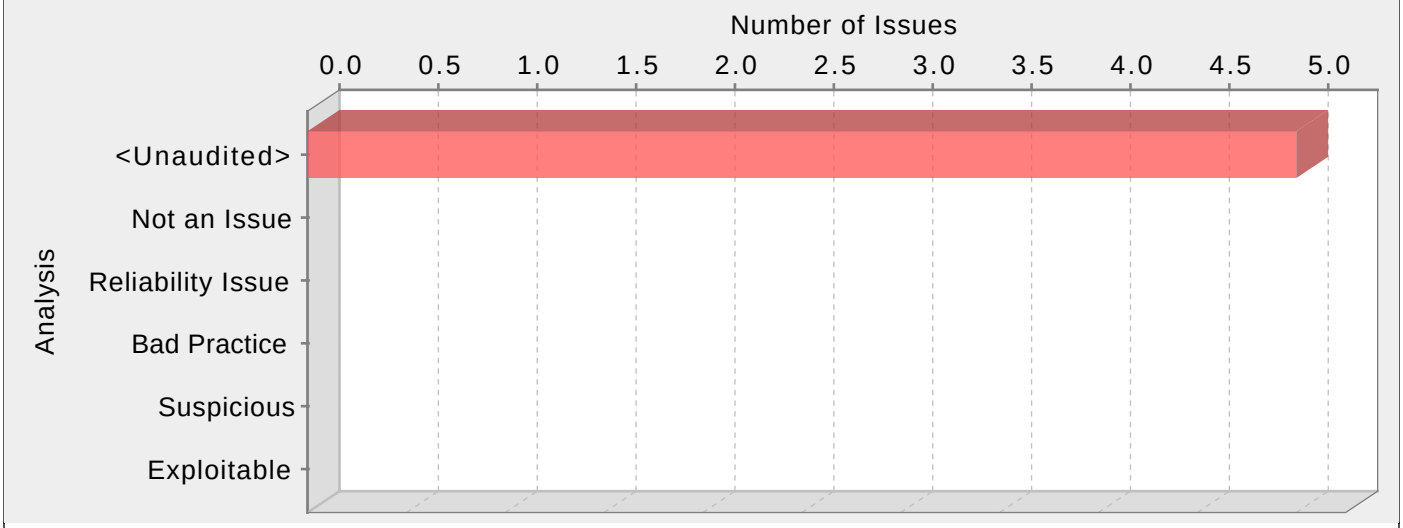
Challenge2Screen.java, line 108 (Password Management: Hardcoded Password)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	Hardcoded passwords can compromise system security in a way that cannot be easily remedied.		

Sink: Challenge2Screen.java:108 FieldAccess: PASSWORD()

```
106                  *  Description of the Field
107                  */
108                 protected final static String PASSWORD =******
109
110                 /**
```

Category: Path Manipulation (5 Issues)



Abstract:

Attackers can control the filesystem path argument to File() at CommandInjection.java line 172, which allows them to access or modify otherwise protected files.

Explanation:

Path manipulation errors occur when the following two conditions are met:

1. An attacker can specify a path used in an operation on the filesystem.

2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted.

For example, the program may give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker.

Example 1: The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "../../tomcat/conf/server.xml", which causes the application to delete one of its own configuration files.

```
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
...
rFile.delete();
```

Example 2: The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension .txt.

```
fis = new FileInputStream(cfg.getProperty("sub")+".txt");
amt = fis.read(arr);
out.println(arr);
```

Recommendations:

The best way to prevent path manipulation is with a level of indirection: create a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list. With this approach the input provided by the user is never used directly to specify the resource name.

In some situations this approach is impractical because the set of legitimate resource names is too large or too hard to keep track of. Programmers often resort to blacklisting in these situations. Blacklisting selectively rejects or escapes potentially dangerous characters before using the input. However, any such list of unsafe characters is likely to be incomplete and will almost certainly become out of date. A better approach is to create a white list of characters that are allowed to appear in the resource name and accept input composed exclusively of characters in the approved set.

Tips:

1. If the program is performing input validation, satisfy yourself that the validation is correct, and use the Custom Rules Editor to create a cleanse rule for the validation routine.

2. It is notoriously difficult to correctly implement a blacklist. If the validation logic relies on blacklisting, be skeptical. Consider different types of input encoding and different sets of meta-characters that might have special meaning when interpreted by different operating systems, databases, or other resources. Determine whether or not the blacklist can be updated easily, correctly, and completely if these requirements ever change.

3. A number of modern web frameworks provide mechanisms for performing validation of user input. Struts and Struts 2 are among them. To highlight the unvalidated sources of input, the HP Fortify Secure Coding Rulepacks dynamically re-prioritize the issues reported by HP Fortify Static Code Analyzer by lowering their probability of exploit and providing pointers to the supporting evidence whenever the framework validation mechanism is in use. We refer to this feature as Context-Sensitive Ranking. To further assist the HP Fortify user with the auditing process, the Fortify Security Research Group makes available the Data Validation project template that groups the issues into folders based on the validation mechanism applied to their source of input.

CommandInjection.java, line 172 (Path Manipulation)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	Attackers can control the filesystem path argument to File() at CommandInjection.java line 172, which allows them to access or modify otherwise protected files.		

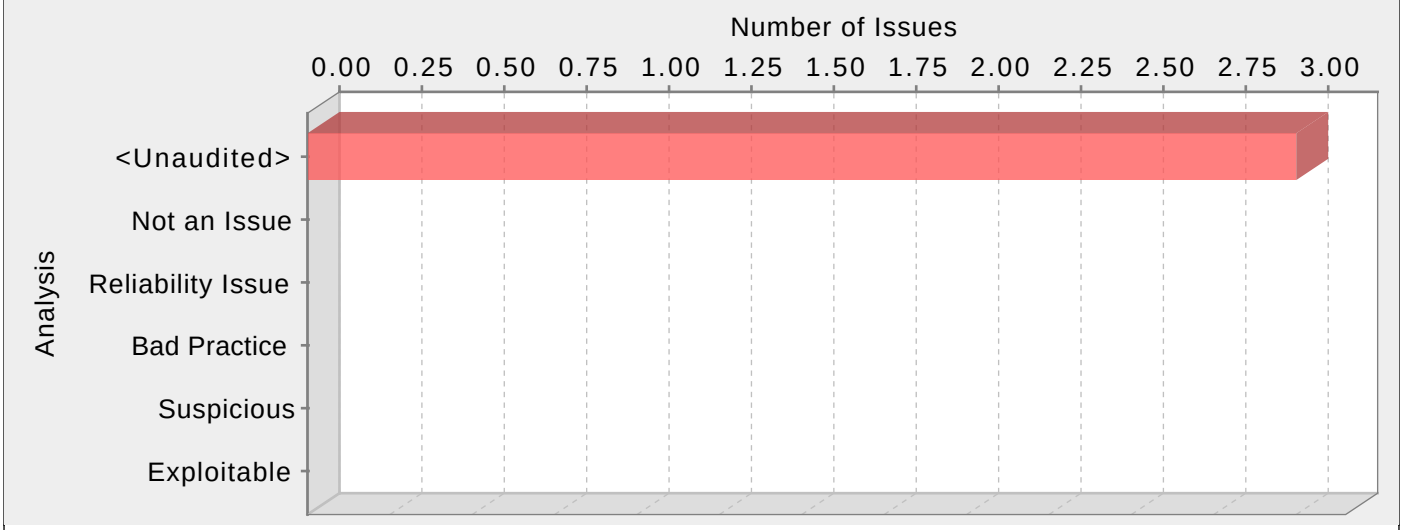
Source: ParameterParser.java:627 javax.servlet.ServletRequest.getParameterValues()

```
625                 throws ParameterNotFoundException
626                {
627                 String[] values = request.getParameterValues(name);
628
629                 if (values == null)
```

Sink: CommandInjection.java:172 java.io.File.File()

```
170                   + safeDir.getPath() + "\"");
171                  fileData = exec(s, "cmd.exe /c type \""
172                   + new File(safeDir, helpFile).getPath() + "\"");
173
174                 }
```

Category: Command Injection (3 Issues)



Abstract:

The method accessWGService() in WSDLScanning.java calls setOperationName() with a command built from untrusted data. This call can cause the program to execute malicious commands on behalf of an attacker.

Explanation:

Command injection vulnerabilities take two forms:

- An attacker can change the command that the program executes: the attacker explicitly controls what the command is.

- An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means.

In this case we are primarily concerned with the first scenario, the possibility that an attacker may be able to control the command that is executed. Command injection vulnerabilities of this type occur when:

1. Data enters the application from an untrusted source.

2. The data is used as or as part of a string representing a command that is executed by the application.

3. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

Example 1: The following code from a system utility uses the system property APPHOME to determine the directory in which it is installed and then executes an initialization script based on a relative path from the specified directory.

```
...
String home = System.getProperty("APPHOME");
String cmd = home + INITCMD;
java.lang.Runtime.getRuntime().exec(cmd);
...
```

The code in Example 1 allows an attacker to execute arbitrary commands with the elevated privilege of the application by modifying the system property APPHOME to point to a different path containing a malicious version of INITCMD. Because the program does not validate the value read from the environment, if an attacker can control the value of the system property APPHOME, then they can fool the application into running malicious code and take control of the system.

Example 2: The following code is from an administrative web application designed allow users to kick off a backup of an Oracle database using a batch-file wrapper around the rman utility and then run a cleanup.bat script to delete some temporary files. The script rmanDB.bat accepts a single command line parameter, which specifies what type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

```
...
String btype = request.getParameter("backuptype");
String cmd = new String("cmd.exe /K
\"c:\\util\\rmanDB.bat "+btype+"&&c:\\utl\\cleanup.bat\"")
System.Runtime.getRuntime().exec(cmd);
...
```

The problem here is that the program does not do any validation on the backuptype parameter read from the user. Typically the Runtime.exec() function will not execute multiple commands, but in this case the program first runs the cmd.exe shell in order to run multiple commands with a single call to Runtime.exec(). Once the shell is invoked, it will happily execute multiple commands separated by two ampersands. If an attacker passes a string of the form "&& del c:\\dbms*.*", then the application will execute this command along with the others specified by the program. Because of the nature of the application, it runs with the privileges necessary to interact with the database, which means whatever command the attacker injects will run with those privileges as well.

Example 3: The following code is from a web application that allows users access to an interface through which they can update their password on the system. Part of the process for updating passwords in certain network environments is to run a make command in the /var/yp directory, the code for which is shown below.

```
...
System.Runtime.getRuntime().exec("make");
...
```

The problem here is that the program does not specify an absolute path for make and fails to clean its environment prior to executing the call to Runtime.exec(). If an attacker can modify the $PATH variable to point to a malicious binary called make and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's make will now be run with these privileges, possibly giving the attacker complete control of the system.

Recommendations:

Do not allow users to have direct control over the commands executed by the program. In cases where user input must affect the command to be run, use the input only to make a selection from a predetermined set of safe commands. If the input appears to be malicious, the value passed to the command execution function should either default to some safe selection from this set or the program should decline to execute any command at all.

In cases where user input must be used as an argument to a command executed by the program, this approach often becomes impractical because the set of legitimate argument values is too large or too hard to keep track of. Developers often fall back on blacklisting in these situations. Blacklisting selectively rejects or escapes potentially dangerous characters before using the input. Any list of unsafe characters is likely to be incomplete and will be heavily dependent on the system where the commands are executed. A better approach is to create a white list of characters that are allowed to appear in the input and accept input composed exclusively of characters in the approved set.

An attacker can indirectly control commands executed by a program by modifying the environment in which they are executed. The environment should not be trusted and precautions should be taken to prevent an attacker from using some manipulation of the environment to perform an attack. Whenever possible, commands should be controlled by the application and executed using an absolute path. In cases where the path is not known at compile time, such as for cross-platform applications, an absolute path should be constructed from trusted values during execution. Command values and paths read from configuration files or the environment should be sanity-checked against a set of invariants that define valid values.

Other checks can sometimes be performed to detect if these sources may have been tampered with. For example, if a configuration file is world-writable, the program might refuse to run. In cases where information about the binary to be executed is known in advance, the program may perform checks to verify the identity of the binary. If a binary should always be owned by a particular user or have a particular set of access permissions assigned to it, these properties can be verified programmatically before the binary is executed.

Although it may be impossible to completely protect a program from an imaginative attacker bent on controlling the commands the program executes, be sure to apply the principle of least privilege wherever the program executes an external command: do not hold privileges that are not essential to the execution of the command.

Tips:

1. A number of modern web frameworks provide mechanisms for performing validation of user input. Struts and Struts 2 are among them. To highlight the unvalidated sources of input, the HP Fortify Secure Coding Rulepacks dynamically re-prioritize the issues reported by HP Fortify Static Code Analyzer by lowering their probability of exploit and providing pointers to the supporting evidence whenever the framework validation mechanism is in use. We refer to this feature as Context-Sensitive Ranking. To further assist the HP Fortify user with the auditing process, the Fortify Security Research Group makes available the Data Validation project template that groups the issues into folders based on the validation mechanism applied to their source of input.

WSDLScanning.java, line 143 (Command Injection)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	The method accessWGService() in WSDLScanning.java calls setOperationName() with a command built from untrusted data. This call can cause the program to execute malicious commands on behalf of an attacker.		
Source:	ParameterParser.java:593 javax.servlet.ServletRequest.getParameterValues()		

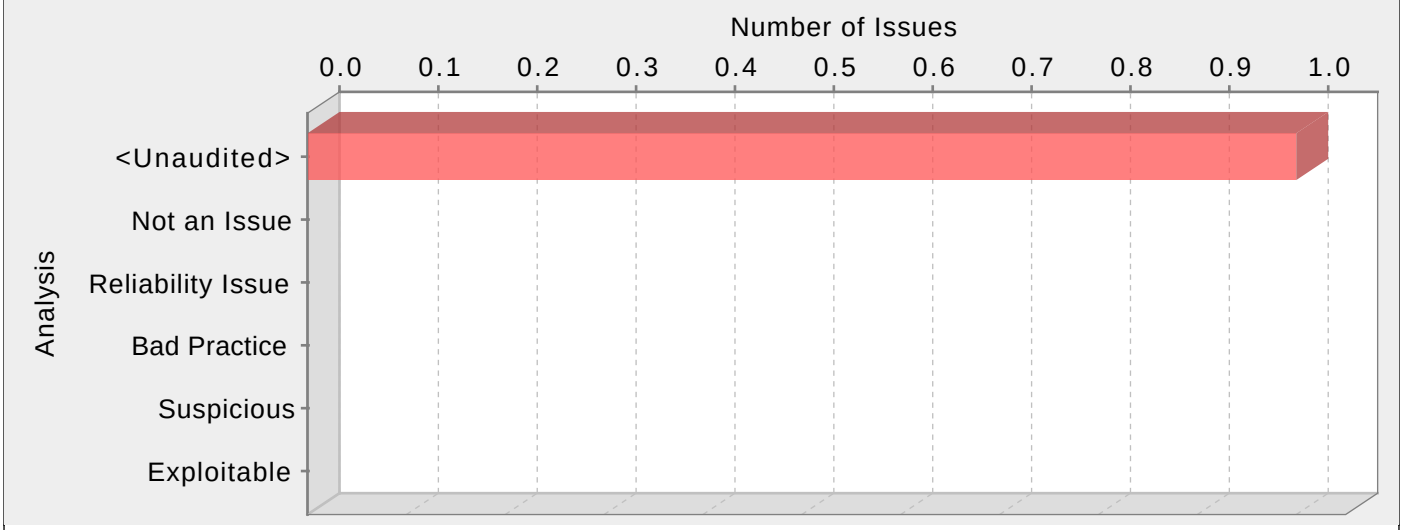
```
591  }
592
593  return request.getParameterValues(name);
```

```
594                 }
```

Sink: WSDLScanning.java:143 org.apache.axis.client.Call.setOperationName()

```
141                 Service service = new Service();
142                 Call call = (Call) service.createCall();
143                 call.setOperationName(operationName);
144                 call.addParameter(parameterName, serviceName, ParameterMode.INOUT);
145                 call.setReturnType(XMLType.XSD_STRING);
```

Category: Race Condition: Singleton Member Field (1 Issues)



Abstract:

The class HammerHead is a singleton, so the member field mySession is shared between users. The result is that one user could see another user's data.

Explanation:

Many Servlet developers do not understand that a Servlet is a singleton. There is only one instance of the Servlet, and that single instance is used and re-used to handle multiple requests that are processed simultaneously by different threads.

A common result of this misunderstanding is that developers use Servlet member fields in such a way that one user may inadvertently see another user's data. In other words, storing user data in Servlet member fields introduces a data access race condition.

Example 1: The following Servlet stores the value of a request parameter in a member field and then later echoes the parameter value to the response output stream.

```
public class GuestBook extends HttpServlet {

String name;

protected void doPost (HttpServletRequest req,
HttpServletResponse res) {
name = req.getParameter("name");
...
out.println(name + ", thanks for visiting!");
}
}
```

While this code will work perfectly in a single-user environment, if two users access the Servlet at approximately the same time, it is possible for the two request handler threads to interleave in the following way:

Thread 1: assign "Dick" to name
Thread 2: assign "Jane" to name
Thread 1: print "Jane, thanks for visiting!"
Thread 2: print "Jane, thanks for visiting!"

Thereby showing the first user the second user's name.

Recommendations:

Do not use Servlet member fields for anything but constants. (i.e. make all member fields static final).

Developers are often tempted to use Servlet member fields for user data when they need to transport data from one region of code to another. If this is your aim, consider declaring a separate class and using the Servlet only to "wrap" this new class.

Example 2: The bug in the example above can be corrected in the following way:

```
public class GuestBook extends HttpServlet {

protected void doPost (HttpServletRequest req,
HttpServletResponse res) {
GBRequestHandler handler = new GBRequestHandler();
```

handler.handle(req, res);
}
}

public class GBRequestHandler {

String name;

public void handle(HttpServletRequest req,
HttpServletResponse res) {
name = req.getParameter("name");
...
out.println(name + ", thanks for visiting!");
}

}

Alternatively, a Servlet can utilize synchronized blocks to access servlet instance variables but using synchronized blocks may cause significant performance problems.

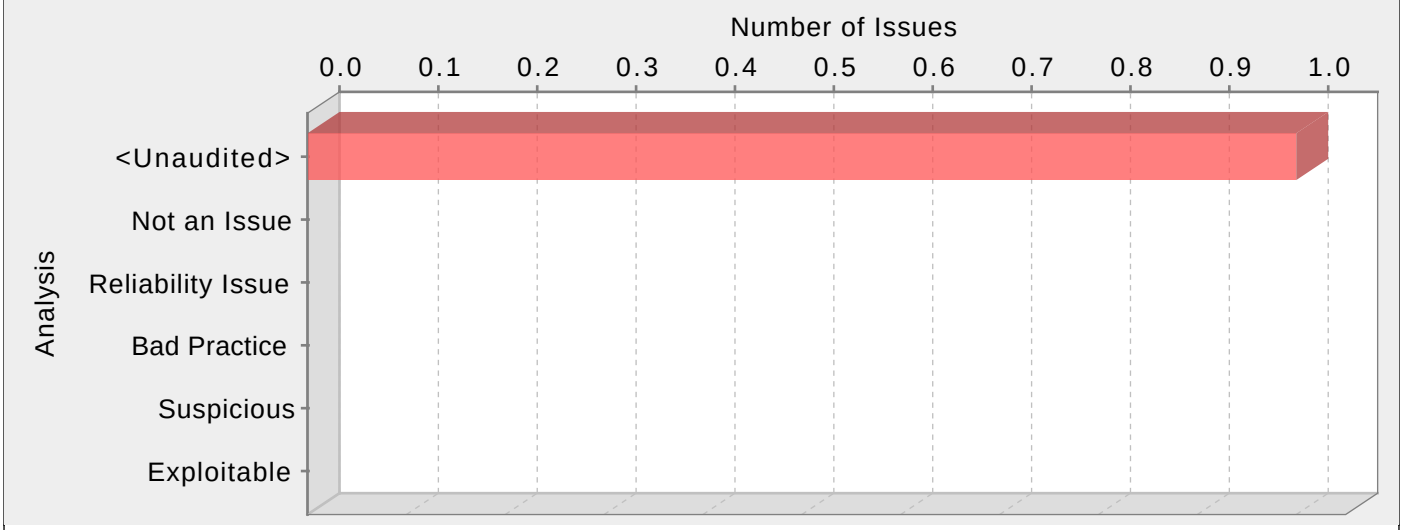
HammerHead.java, line 135 (Race Condition: Singleton Member Field)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Time and State		
Abstract:	The class HammerHead is a singleton, so the member field mySession is shared between users. The result is that one user could see another user's data.		

Sink: HammerHead.java:135 AssignmentStatement()

```
133                 // FIXME: If a response is written by updateSession(), do not
134                 // call makeScreen() and writeScreen()
135                 mySession = updateSession(request, response, context);
136                 if (response.isCommitted())
137               return;
```

Category: XPath Injection (1 Issues)



Abstract:

On line 158 of XPATHInjection.java, the method createContent() invokes an XPath query built using unvalidated input. This call could allow an attacker to modify the statement's meaning or to execute arbitrary XPath queries.

Explanation:

XPath injection occurs when:

1. Data enters a program from an untrusted source.

2. The data used to dynamically construct an XPath query.

Example 1: The following code dynamically constructs and executes an XPath query that retrieves an e-mail address for a given account ID. The account ID is read from an HTTP request, and is therefore untrusted.

```
...
String acctID = request.getParameter("acctID");
String query = null;
if(acctID != null) {
StringBuffer sb = new StringBuffer("/accounts/account[acctID='");
sb.append(acctID);
sb.append("']/email/text()");
query = sb.toString();
}

DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
domFactory.setNamespaceAware(true);
DocumentBuilder builder = domFactory.newDocumentBuilder();
Document doc = builder.parse("accounts.xml");
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
XPathExpression expr = xpath.compile(query);
Object result = expr.evaluate(doc, XPathConstants.NODESET);
...
```

Under normal conditions, such as searching for an e-mail address that belongs to the account number 1, the query that this code executes will look like the following:

```
/accounts/account[acctID='1']/email/text()
```

However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if acctID does not contain a single-quote character. If an attacker enters the string 1' or '1' = '1 for acctID, then the query becomes the following:

```
/accounts/account[acctID='1' or '1' = '1']/email/text()
```

The addition of the 1' or '1' = '1 condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

//email/text()

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all e-mail addresses stored in the document, regardless of their specified owner.

Recommendations:

The root cause of XPath injection vulnerability is the ability of an attacker to change context in the XPath query, causing a value that the programmer intended to be interpreted as data to be interpreted as a command instead. When an XPath query is constructed, the programmer knows what should be interpreted as part of the command and what should be interpreted as data.

To prevent an attacker from violating the programmer's expectations, use a whitelist to ensure that user-controlled values used in an XPath query are composed from only the expected set of characters and do not contain any XPath metacharacters given the context in which they are used. If a user-controlled value requires that it contain XPath metacharacters, use an appropriate encoding mechanism to remove their significance within the XPath query.

Example 2

```
...
String acctID = request.getParameter("acctID");
String query = null;
if(acctID != null) {
Integer iAcctID = -1;
try {
iAcctID = Integer.parseInt(acctID);
}
catch (NumberFormatException e) {
throw new InvalidParameterException();
}
StringBuffer sb = new StringBuffer("/accounts/account[acctID='");
sb.append(iAcctID.toString());
sb.append("']/email/text()");
query = sb.toString();
}

DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
domFactory.setNamespaceAware(true);
DocumentBuilder builder = domFactory.newDocumentBuilder();
Document doc = builder.parse("accounts.xml");
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
XPathExpression expr = xpath.compile(query);
Object result = expr.evaluate(doc, XPathConstants.NODESET);
...
```

Tips:

1. A number of modern web frameworks provide mechanisms for performing validation of user input. Struts and Struts 2 are among them. To highlight the unvalidated sources of input, the HP Fortify Secure Coding Rulepacks dynamically re-prioritize the issues reported by HP Fortify Static Code Analyzer by lowering their probability of exploit and providing pointers to the supporting evidence whenever the framework validation mechanism is in use. We refer to this feature as Context-Sensitive Ranking. To further assist the HP Fortify user with the auditing process, the Fortify Security Research Group makes available the Data Validation project template that groups the issues into folders based on the validation mechanism applied to their source of input.

XPATHInjection.java, line 158 (XPath Injection)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	On line 158 of XPATHInjection.java, the method createContent() invokes an XPath query built using unvalidated input. This call could allow an attacker to modify the statement's meaning or to execute arbitrary XPath queries.		
Source:	ParameterParser.java:627 javax.servlet.ServletRequest.getParameterValues()		

```
625         throws ParameterNotFoundException
```

```
626                     {
627                 String[] values = request.getParameterValues(name);
628
629                 if (values == null)
```

Sink: XPATHInjection.java:158 javax.xml.xpath.XPath.evaluate()

```
156                     String expression = "/employees/employee[loginID/text()='"
157                      + username + "' and passwd/text()='" + password + "']";
158                     nodes = (NodeList) xPath.evaluate(expression, inputSource,
159                      XPathConstants.NODESET);
160                     int nodesLength = nodes.getLength();
```

Detailed Project Summary

Files Scanned

Code base location: C:/Program Files/HP_Fortify/HP_Fortify_SCA_and_Apps_3.90/Samples/advanced/webgoat/WebGoat5.0
Files Scanned:
JavaSource/org/owasp/webgoat/HammerHead.java java 124 Lines 15.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/LessonSource.java java 50 Lines 5.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/AbstractLesson.java java 199 Lines 27.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/AccessControlMatrix.java java 66 Lines 7.4 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/BackDoors.java java 111 Lines 8.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/BasicAuthentication.java java 104 Lines 10.3 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/BlindSqlInjection.java java 78 Lines 11.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/BufferOverflow.java java 11 Lines 2.9 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/CSRF.java java 129 Lines 10.4 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/Category.java java 12 Lines 2.2 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/Challenge2Screen.java java 304 Lines 21.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/CommandInjection.java java 128 Lines 11.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/CrossSiteScripting/CrossSiteScripting.java java 158 Lines 14.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/CrossSiteScripting/EditProfile.java java 76 Lines 6.4 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/CrossSiteScripting/FindProfile.java java 85 Lines 7.4 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/CrossSiteScripting/UpdateProfile.java java 180 Lines 13.9 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/CrossSiteScripting/ViewProfile.java java 102 Lines 7.9 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/DOMInjection.java java 65 Lines 5.9 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/DOS_Login.java java 87 Lines 7.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/DefaultLessonAction.java java 127 Lines 11.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/Encoding.java java 329 Lines 27.3 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/FailOpenAuthentication.java java 41 Lines 5.5 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/ForcedBrowsing.java java 36 Lines 4.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/ForgotPassword.java java 130 Lines 9.2 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/HiddenFieldTampering.java java 75 Lines 6.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/HtmlClues.java java 66 Lines 7.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/HttpBasics.java java 27 Lines 3.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/HttpOnly.java java 157 Lines 12.5 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/HttpSplitting.java java 97 Lines 8.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/JSONInjection.java java 79 Lines 8.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/JavaScriptValidation.java java 102 Lines 10.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/LessonAction.java java 18 Lines 924 bytes May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/LessonAdapter.java java 106 Lines 9.9 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/LogSpoofing.java java 58 Lines 5 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/NewLesson.java java 7 Lines 2.5 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/PathBasedAccessControl.java java 98 Lines 9 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/ReflectedXSS.java java 129 Lines 9.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/RemoteAdminFlaw.java java 16 Lines 3.3 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/RoleBasedAccessControl/DeleteProfile.java java 53 Lines 5.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/RoleBasedAccessControl/EditProfile.java java 76 Lines 6.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/RoleBasedAccessControl/FindProfile.java java 65 Lines 5.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/RoleBasedAccessControl/ListStaff.java java 57 Lines 5.5 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/RoleBasedAccessControl/Login.java java 72 Lines 6.3 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/RoleBasedAccessControl/Logout.java java 12 Lines 2.7 KB May 24, 2013 5:03:06 PM

JavaSource/org/owasp/webgoat/lessons/RoleBasedAccessControl/RoleBasedAccessControl.java java 190 Lines 15.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/RoleBasedAccessControl/SearchStaff.java java 2 Lines 1.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/RoleBasedAccessControl/UpdateProfile.java java 149 Lines 10.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/RoleBasedAccessControl/ViewProfile.java java 92 Lines 7.5 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/SQLInjection/ListStaff.java java 57 Lines 5.5 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/SQLInjection/Login.java java 112 Lines 8.2 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/SQLInjection/SQLInjection.java java 148 Lines 13.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/SQLInjection/ViewProfile.java java 111 Lines 8.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/SilentTransactions.java java 105 Lines 9.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/SoapRequest.java java 148 Lines 14.9 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/SqlNumericInjection.java java 125 Lines 9.9 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/SqlStringInjection.java java 97 Lines 8.3 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/StoredXss.java java 132 Lines 11.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/ThreadSafetyProblem.java java 55 Lines 6.2 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/TraceXSS.java java 126 Lines 9.4 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/UncheckedEmail.java java 92 Lines 8.2 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/WSDLScanning.java java 131 Lines 9.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/WeakAuthenticationCookie.java java 108 Lines 10.2 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/WeakSessionID.java java 83 Lines 7.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/WelcomeScreen.java java 33 Lines 4.3 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/WsSAXInjection.java java 68 Lines 6.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/WsSqlInjection.java java 87 Lines 8.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/XMLInjection.java java 129 Lines 9.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/XPATHInjection.java java 103 Lines 7.5 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/admin/AdminScreen.java java 8 Lines 2.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/admin/ProductsAdminScreen.java java 25 Lines 3.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/admin/RefreshDBScreen.java java 40 Lines 4.4 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/admin/ReportCardScreen.java java 99 Lines 8.9 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/admin/SummaryReportCardScreen.java java 117 Lines 8.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/admin/UserAdminScreen.java java 25 Lines 3.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/admin/ViewDatabase.java java 39 Lines 4.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/lessons/admin/WelcomeAdminScreen.java java 8 Lines 2.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/servlets/Controller.java java 9 Lines 2.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/Authorization.java java 4 Lines 1.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/Course.java java 115 Lines 11.2 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/CreateDB.java java 433 Lines 32.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/DatabaseUtilities.java java 39 Lines 4.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/ECSFactory.java java 134 Lines 15.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/Employee.java java 48 Lines 4.9 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/EmployeeStub.java java 10 Lines 2 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/ErrorScreen.java java 61 Lines 7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/LessonSession.java java 6 Lines 2 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/LessonTracker.java java 114 Lines 11.5 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/Parameter.java java 13 Lines 2.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/ParameterNotFoundException.java java 2 Lines 1.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/ParameterParser.java java 212 Lines 29.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/Screen.java java 46 Lines 8.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/UnauthenticatedException.java java 1 Lines 1.3 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/UnauthorizedException.java java 1 Lines 1.3 KB May 24, 2013 5:03:06 PM

JavaSource/org/owasp/webgoat/session/UserTracker.java java 61 Lines 6.4 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/ValidationException.java java 2 Lines 1.4 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/WebSession.java java 301 Lines 28 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/session/WebgoatProperties.java java 35 Lines 3.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/util/Exec.java java 163 Lines 13.8 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/util/ExecResults.java java 57 Lines 8.2 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/util/ExecutionException.java java 2 Lines 1.7 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/util/HtmlEncoder.java java 109 Lines 8.6 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/util/Interceptor.java java 34 Lines 4.1 KB May 24, 2013 5:03:06 PM
JavaSource/org/owasp/webgoat/util/ThreadWatcher.java java 10 Lines 2.9 KB May 24, 2013 5:03:06 PM
WebContent/WEB-INF/server-config.wsdd xml 3.4 KB May 24, 2013 5:03:07 PM
WebContent/WEB-INF/web-unix.xml xml 11 KB May 24, 2013 5:03:07 PM
WebContent/WEB-INF/web-windows.xml xml 11 KB May 24, 2013 5:03:07 PM
WebContent/WEB-INF/web.xml xml 11 KB May 24, 2013 5:03:07 PM
WebContent/WEB-INF/webgoat.properties java_properties 37 bytes May 24, 2013 5:03:07 PM
WebContent/javascript/javascript.js javascript 5 Lines 369 bytes May 24, 2013 5:03:06 PM
WebContent/javascript/lessonNav.js javascript 32 Lines 2.6 KB May 24, 2013 5:03:06 PM
WebContent/javascript/makeWindow.js javascript 4 Lines 239 bytes May 24, 2013 5:03:06 PM
WebContent/javascript/menu_system.js javascript 115 Lines 7.4 KB May 24, 2013 5:03:06 PM
WebContent/javascript/toggle.js javascript 25 Lines 1.2 KB May 24, 2013 5:03:06 PM
WebContent/lesson_plans/AccessControlMatrix.html html 912 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/BackDoors.html html 1.1 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/BasicAuthentication.html html 974 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/BlindSqlInjection.html html 1.3 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/BufferOverflow.html html 290 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/CSRF.html html 2.1 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/ChallengeScreen.html html 291 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/CommandInjection.html html 992 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/CrossSiteScripting.html html 1 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/DOMInjection.html html 898 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/DOS_Login.html html 618 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/Encoding.html html 380 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/FailOpenAuthentication.html html 681 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/ForcedBrowsing.html html 833 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/ForgotPassword.html html 758 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/HiddenFieldTampering.html html 707 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/HtmlClues.html html 540 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/HttpBasics.html html 1.5 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/HttpOnly.html html 936 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/HttpSplitting.html html 2.1 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/JSONInjection.html html 1.2 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/JavaScriptValidation.html html 908 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/Lesson_Plan_Template.html html 643 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/LogSpoofing.html html 733 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/NewLesson.html html 1.5 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/PathBasedAccessControl.html html 602 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/ReflectedXSS.html html 765 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/RemoteAdminFlaw.html html 731 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/RoleBasedAccessControl.html html 1.5 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/SilentTransactions.html html 1.2 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/SoapRequest.html html 757 bytes May 24, 2013 5:03:07 PM

WebContent/lesson_plans/SqlNumericInjection.html html 1 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/SqlStringInjection.html html 1.1 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/StoredXss.html html 791 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/ThreadSafetyProblem.html html 1.3 KB May 24, 2013 5:03:07 PM
WebContent/lesson_plans/TraceXSS.html html 793 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/UncheckedEmail.html html 506 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/WSDLScanning.html html 555 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/WeakAuthenticationCookie.html html 899 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/WeakSessionID.html html 570 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/WelcomeScreeen.html html 710 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/WsSAXInjection.html html 725 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/WsSqlInjection.html html 689 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/XMLInjection.html html 791 bytes May 24, 2013 5:03:07 PM
WebContent/lesson_plans/XPATHInjection.html html 1.5 KB May 24, 2013 5:03:07 PM
WebContent/lesson_template/lessons.html html 2.7 KB May 24, 2013 5:03:07 PM
WebContent/lessons/ConfManagement/config.jsp jsp 4 Lines 558 bytes May 24, 2013 5:03:06 PM
WebContent/lessons/CrossSiteScripting/CrossSiteScripting.jsp jsp 7 Lines 764 bytes May 24, 2013 5:03:07 PM
WebContent/lessons/CrossSiteScripting/EditProfile.jsp jsp 29 Lines 4.8 KB May 24, 2013 5:03:07 PM
WebContent/lessons/CrossSiteScripting/ListStaff.jsp jsp 18 Lines 2.1 KB May 24, 2013 5:03:07 PM
WebContent/lessons/CrossSiteScripting/Login.jsp jsp 14 Lines 1.6 KB May 24, 2013 5:03:07 PM
WebContent/lessons/CrossSiteScripting/SearchStaff.jsp jsp 8 Lines 850 bytes May 24, 2013 5:03:07 PM
WebContent/lessons/CrossSiteScripting/ViewProfile.jsp jsp 35 Lines 4.8 KB May 24, 2013 5:03:07 PM
WebContent/lessons/CrossSiteScripting/error.jsp jsp 1 Lines 125 bytes May 24, 2013 5:03:07 PM
WebContent/lessons/General/redirect.jsp jsp 5 Lines 600 bytes May 24, 2013 5:03:07 PM
WebContent/lessons/RoleBasedAccessControl/EditProfile.jsp jsp 29 Lines 5 KB May 24, 2013 5:03:07 PM
WebContent/lessons/RoleBasedAccessControl/ListStaff.jsp jsp 18 Lines 2.1 KB May 24, 2013 5:03:07 PM
WebContent/lessons/RoleBasedAccessControl/Login.jsp jsp 14 Lines 1.6 KB May 24, 2013 5:03:07 PM
WebContent/lessons/RoleBasedAccessControl/RoleBasedAccessControl.jsp jsp 7 Lines 784 bytes May 24, 2013 5:03:07 PM
WebContent/lessons/RoleBasedAccessControl/SearchStaff.jsp jsp 8 Lines 871 bytes May 24, 2013 5:03:07 PM
WebContent/lessons/RoleBasedAccessControl/ViewProfile.jsp jsp 31 Lines 4.8 KB May 24, 2013 5:03:07 PM
WebContent/lessons/RoleBasedAccessControl/error.jsp jsp 4 Lines 623 bytes May 24, 2013 5:03:07 PM
WebContent/lessons/SQLInjection/EditProfile.jsp jsp 29 Lines 4.6 KB May 24, 2013 5:03:07 PM
WebContent/lessons/SQLInjection/ListStaff.jsp jsp 18 Lines 2 KB May 24, 2013 5:03:07 PM
WebContent/lessons/SQLInjection/Login.jsp jsp 14 Lines 1.5 KB May 24, 2013 5:03:07 PM
WebContent/lessons/SQLInjection/SQLInjection.jsp jsp 7 Lines 734 bytes May 24, 2013 5:03:07 PM
WebContent/lessons/SQLInjection/SearchStaff.jsp jsp 8 Lines 820 bytes May 24, 2013 5:03:07 PM
WebContent/lessons/SQLInjection/ViewProfile.jsp jsp 31 Lines 4.1 KB May 24, 2013 5:03:07 PM
WebContent/lessons/SQLInjection/error.jsp jsp 1 Lines 125 bytes May 24, 2013 5:03:07 PM
WebContent/lessons/XPATHInjection/EmployeesData.xml xml 517 bytes May 24, 2013 5:03:07 PM
WebContent/main.jsp jsp 85 Lines 10.3 KB May 24, 2013 5:03:07 PM
WebContent/sideWindow.jsp jsp 6 Lines 832 bytes May 24, 2013 5:03:07 PM
WebContent/webgoat.jsp jsp 1 Lines 4.6 KB May 24, 2013 5:03:07 PM
WebContent/webgoat_challenge.jsp jsp 1 Lines 3.6 KB May 24, 2013 5:03:07 PM

Reference Elements

Classpath:

C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\axis-ant.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-

INF\lib\axis.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\catalina.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\commons-collections-3.1.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\commons-digester.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\commons-discovery-0.2.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\commons-logging-1.0.4.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\ecs-1.4.2.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\idb.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\j2h.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\jaxrpc.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\jta-spec1_0_1.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\log4j-1.2.8.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\saaj.jar
C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat\.\WebGoat5.0\WebContent\WEB-INF\lib\wsdl4j-1.5.1.jar

Libdirs:

No libdirs specified during translation

Rulepacks

Valid Rulepacks:

Name: Fortify Secure Coding Rules, Core, ABAP
Version: 2013.1.0.0008
ID: 5A663288-B1F7-416A-AEED-E5F00DC3596F
SKU: RUL13095

Name: Fortify Secure Coding Rules, Core, ActionScript 3.0
Version: 2013.1.0.0008
ID: 92127AA2-E666-4F28-B1C1-C0F6A939A089
SKU: RUL13094

Name: Fortify Secure Coding Rules, Core, Android
Version: 2013.1.0.0008
ID: FF9890E6-D119-4EE8-A591-83DCF4CA6952
SKU: RUL13093

Name: Fortify Secure Coding Rules, Core, Annotations
Version: 2013.1.0.0008
ID: 14EE50EB-FA1C-4AE8-8B59-39F952E21E3B
SKU: RUL13078

Name: Fortify Secure Coding Rules, Core, ColdFusion
Version: 2013.1.0.0008
ID: EEA7C678-058E-462A-8A59-AF925F7B7164
SKU: RUL13024

Name: Fortify Secure Coding Rules, Core, COBOL
Version: 2013.1.0.0008
ID: 9EB21F2C-ED22-440B-82A3-E38DC2533F03
SKU: RUL13088

Name: Fortify Secure Coding Rules, Core, C/C++
Version: 2013.1.0.0008
ID: 711E0652-7494-42BE-94B1-DB3799418C7E
SKU: RUL13001

Name: Fortify Secure Coding Rules, Core, .NET
Version: 2013.1.0.0008
ID: D57210E5-E762-4112-97DD-019E61D32D0E
SKU: RUL13002

Name: Fortify Secure Coding Rules, Core, Java
Version: 2013.1.0.0008
ID: 06A6CC97-8C3F-4E73-9093-3E74C64A2AAF
SKU: RUL13003

Name: Fortify Secure Coding Rules, Core, JavaScript
Version: 2013.1.0.0008
ID: BD292C4E-4216-4DB8-96C7-9B607BFD9584
SKU: RUL13059

Name: Fortify Secure Coding Rules, Core, Objective-C
Version: 2013.1.0.0008
ID: B18E58BA-8B2D-4FC5-83BF-378594CAD260
SKU: RUL13099

Name: Fortify Secure Coding Rules, Core, PHP
Version: 2013.1.0.0008
ID: 343CBB32-087C-4A4E-8BD8-273B5F876069
SKU: RUL13058

Name: Fortify Secure Coding Rules, Core, Python
Version: 2013.1.0.0008
ID: FD15CBE4-E059-4CBB-914E-546BDCEB422B
SKU: RUL13083

Name: Fortify Secure Coding Rules, Core, SQL
Version: 2013.1.0.0008
ID: 6494160B-E1DB-41F5-9840-2B1609EE7649
SKU: RUL13004

Name: Fortify Secure Coding Rules, Core, Classic ASP, VBScript, and VB6
Version: 2013.1.0.0008
ID: 1D426B6F-8D33-4AD6-BBCE-237ABAFAB924
SKU: RUL13060

Name: Fortify Secure Coding Rules, Extended, Configuration
Version: 2013.1.0.0008
ID: CD6959FC-0C37-45BE-9637-BAA43C3A4D56
SKU: RUL13005

Name: Fortify Secure Coding Rules, Extended, Content
Version: 2013.1.0.0008
ID: 9C48678C-09B6-474D-B86D-97EE94D38F17
SKU: RUL13067

Name: Fortify Secure Coding Rules, Extended, C/C++
Version: 2013.1.0.0008
ID: BD4641AD-A6FF-4401-A8F4-6873272F2748
SKU: RUL13006

Name: Fortify Secure Coding Rules, Extended, .NET
Version: 2013.1.0.0008
ID: 557BCC56-CD42-43A7-B4FE-CDD00D58577E
SKU: RUL13027

Name: Fortify Secure Coding Rules, Extended, Java
Version: 2013.1.0.0008
ID: AAAC0B10-79E7-4FE5-9921-F4903A79D317
SKU: RUL13007

Name: Fortify Secure Coding Rules, Extended, JSP
Version: 2013.1.0.0008
ID: 00403342-15D0-48C9-8E67-4B1CFBDEFCD2
SKU: RUL13026

Name: Fortify Secure Coding Rules, Extended, SQL
Version: 2013.1.0.0008
ID: 4BC5B2FA-C209-4DBC-9C3E-1D3EEFAF135A
SKU: RUL13025

External Metadata:

Name: CWE
ID: 3ADB9EE4-5761-4289-8BD3-CBFCC593EBBC
Common Weakness Enumeration description

Name: FISMA
ID: B40F9EE0-3824-4879-B9FE-7A789C89307C
FIPS200 description

Name: OWASP Top 10 2004
ID: 771C470C-9274-4580-8556-C023E4D3ADB4
The OWASP Top Ten 2004 provides a powerful awareness document for web application security. The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are. Project members include a variety of security experts from around the world who have shared their expertise to produce this list.

Name: OWASP Top 10 2007
ID: 1EB1EC0E-74E6-49A0-BCE5-E6603802987A
The OWASP Top Ten 2007 provides a powerful awareness document for web application security. The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are. Project members include a variety of security experts from around the world who have shared their expertise to produce this list.

Name: OWASP Top 10 2010
ID: FDCECA5E-C2A8-4BE8-BB26-76A8ECD0ED59
The OWASP Top Ten 2010 provides a powerful awareness document for web application security. The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are. Project members include a variety of security experts from around the world who have shared their expertise to produce this list.

Name: PCI 1.1
ID: CBDB9D4D-FC20-4C04-AD58-575901CAB531
Payment Card Industry Data Security Standard Version 1.1 description

Name: PCI 1.2
ID: 57940BDB-99F0-48BF-BF2E-CFC42BA035E5
Payment Card Industry Data Security Standard Version 1.2 description

Name: PCI 2.0
ID: 8970556D-7F9F-4EA7-8033-9DF39D68FF3E
The PCI DSS 2.0 compliance standard, particularly sections 6.3, 6.5, and 6.6, references the OWASP Top 10 vulnerability categories as the core categories that must be tested for and remediated. The following table summarizes the number of issues identified accross the different PCI DSS requirements and broken down by Fortify Priority Order.

Name: SANS Top 25 2009
ID: 939EF193-507A-44E2-ABB7-C00B2168B6D8
The 2009 CWE/SANS Top 25 Programming Errors list the most significant programming errors that can lead to serious software vulnerabilities. They occur frequently, are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all. The list is the result of collaboration between the SANS Institute, MITRE, and many top software security experts.

Name: SANS Top 25 2010
ID: 72688795-4F7B-484C-88A6-D4757A6121CA
SANS Top 25 2010 description

Name: SANS Top 25 2011
ID: 92EB4481-1FD9-4165-8E16-F2DE6CB0BD63
SANS Top 25 2011 Most Dangerous Software Errors provides an enumeration of the most widespread and critical errors, categorized by Common Weakness Enumeration (CWE) identifiers, that lead to serious vulnerabilities in software

(http://cwe.mitre.org/). These software errors are often easy to find and exploit. The inherent danger in these errors is that they can allow an attacker to completely take over the software, steal data, or prevent the software from working at all.

Name: STIG 3
ID: F2FA57EA-5AAA-4DDE-90A5-480BE65CE7E7
Security Technical Implementation Guide Version 3.1 description

Name: STIG 3.4
ID: 58E2C21D-C70F-4314-8994-B859E24CF855
<P>Each requirement or recommendation identified by the DISA STIG is represented by a STIG identifier (STIGID) which corresponds to a checklist item and a severity code [APP<I>ID</I>: CAT <I>SEV</I>]. DISA STIG identifies several severities with respect to vulnerabilities:</P> <UL>
<LI>CAT I: allow an attacker immediate access into a machine, allow super user access, or bypass a firewall.</LI>
<LI>CAT II: provide information that have a high potential of giving access to an intruder.</LI>
<LI>CAT III: provide information that potentially could lead to compromise.</LI> </UL> <P>The following table summarizes the number of issues identified accross the different STIGIDs and broken down by Fortify Priority Order. The status of a STIGID is considered "In Place" when there are no issues reported for a given STIGID.</P>

Name: WASC 24 + 2
ID: 9DC61E7F-1A48-4711-BBFD-E9DFF537871F
Web Application Security Consortium 24 + 2 description

Properties

WinForms.CollectionMutationMonitor.Label=WinFormsDataSource
WinForms.ExtractEventHandlers=true
WinForms.TransformChangeNotificationPattern=true
WinForms.TransformDataBindings=true
WinForms.TransformMessageLoops=true
awt.toolkit=sun.awt.windows.WToolkit
com.fortify.AuthenticationKey=C:\Users\fortify\AppData\Local/Fortify/config/tools
com.fortify.Core=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Core
com.fortify.InstallRoot=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90
com.fortify.InstallationUserName=fortify
com.fortify.SCAExecutablePath=C:Program FilesHP_FortifyHP_Fortify_SCA_and_Apps_3.90binsourceanalyzer.exe
com.fortify.TotalPhysicalMemory=51522015232
com.fortify.VS.RequireASPPrecompilation=true
com.fortify.WorkingDirectory=C:\Users\fortify\AppData\Local/Fortify
com.fortify.locale=en
com.fortify.sca.AddImpliedMethods=true
com.fortify.sca.AntCompilerClass=com.fortify.dev.ant.SCACompiler
com.fortify.sca.BuildID=WebGoat5.0
com.fortify.sca.BundleControlflowIssues=true
com.fortify.sca.CollectPerformanceData=true
com.fortify.sca.CustomRulesDir=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Core\config\customrules
com.fortify.sca.DaemonCompilers=com.fortify.sca.util.compilers.GppCompiler,com.fortify.sca.util.compilers.GccCompiler,com.fortify.sca.util.compilers.AppleGppCompiler,com.fortify.sca.util.compilers.AppleGccCompiler,com.fortify.sca.util.compilers.MicrosoftCompiler,com.fortify.sca.util.compilers.MicrosoftLinker,com.fortify.sca.util.compilers.LdCompiler,com.fortify.sca.util.compilers.ArUtil,com.fortify.sca.util.compilers.SunCCompiler,com.fortify.sca.util.compilers.SunCppCompiler,com.fortify.sca.util.compilers.IntelCompiler,com.fortify.sca.util.compilers.ExternalCppAdapter,com.fortify.sca.util.compilers.ClangCompiler

com.fortify.sca.DeadCodeFilter=true
com.fortify.sca.DeadCodeIgnoreTrivialPredicates=true
com.fortify.sca.DefaultAnalyzers=semantic:dataflow:controlflow:nullptr:configuration:content:structural:buffer
com.fortify.sca.DefaultFileTypes=java,jsp,jspx,tag,tagx,tld,sql,cfm,php,phtml,ctp,pks,pkh,pkb,xml,config,settings,properties,dll,exe,inc,asp,vbscript,js,ini,bas,cls,vbs,frm,ctl,html,htm,xsd,wsdd,xmi,py,cfml,cfc,abap,xhtml,cpx,xcfg,jsff,as,mxml
com.fortify.sca.DefaultJarsDirs=default_jars
com.fortify.sca.DefaultRulesDir=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Core\config\rules
com.fortify.sca.DisableDeadCodeElimination=false
com.fortify.sca.DisableFunctionPointers=false
com.fortify.sca.DisableGlobals=false
com.fortify.sca.DisplayProgress=true
com.fortify.sca.EnableInterproceduralConstantResolution=false
com.fortify.sca.EnableNestedWrappers=false
com.fortify.sca.EnableStructuralMatchCache=true
com.fortify.sca.EnableWrapperDetection=false
com.fortify.sca.FVDLAllowUnifiedVulnerability=true
com.fortify.sca.FVDLDisableDescriptions=false
com.fortify.sca.FVDLDisableProgramData=false
com.fortify.sca.FVDLDisableSnippets=false
com.fortify.sca.FVDLStylesheet=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Core/resources/sca/fvdl2html.xsl
com.fortify.sca.IndirectCallGraphBuilders=com.fortify.sca.analyzer.callgraph.WinFormsAdHocFunctionBuilder,com.fortify.sca.analyzer.callgraph.VirtualCGBuilder,com.fortify.sca.analyzer.callgraph.J2EEIndirectCGBuilder,com.fortify.sca.analyzer.callgraph.JNICGBuilder,com.fortify.sca.analyzer.callgraph.StoredProcedureResolver,com.fortify.sca.analyzer.callgraph.JavaWSCGBuilder,com.fortify.sca.analyzer.callgraph.StrutsCGBuilder,com.fortify.sca.analyzer.callgraph.DotNetWSCGBuilder,com.fortify.sca.analyzer.callgraph.SqlServerSPResolver,com.fortify.sca.analyzer.callgraph.ASPCGBuilder,com.fortify.sca.analyzer.callgraph.ScriptedCGBuilder,com.fortify.sca.analyzer.callgraph.NewJspCustomTagCGBuilder,com.fortify.sca.analyzer.callgraph.DotNetCABCGBuilder,com.fortify.sca.analyzer.callgraph.StateInjectionCGBuilder,com.fortify.sca.analyzer.callgraph.SqlServerSPResolver2,com.fortify.sca.analyzer.callgraph.PHPLambdaResolver
com.fortify.sca.JVMArgs=-Dcom.sun.management.jmxremote=true -XX:SoftRefLRUPolicyMSPerMB=100 -Xss1M -Xmx600M -Xms300M -server
com.fortify.sca.JdkVersion=1.4
com.fortify.sca.LowSeverityCutoff=1.0
com.fortify.sca.NoNestedOutTagOutput=org.apache.taglibs.standard.tag.rt.core.RemoveTag,org.apache.taglibs.standard.tag.rt.core.SetTag
com.fortify.sca.PID=5060
com.fortify.sca.ParentProcessIsCygwinShell=true
com.fortify.sca.PrintPerformanceDataAfterScan=false
com.fortify.sca.ProjectRoot=C:\Users\fortify\AppData\Local/Fortify
com.fortify.sca.ProjectRoot=C:\Users\fortify\AppData\Local/Fortify
com.fortify.sca.RequireMapKeys=never
com.fortify.sca.ResultsFile=WebGoat5.0.fpr
com.fortify.sca.SolverTimeout=15
com.fortify.sca.SqlLanguage=TSQL
com.fortify.sca.SuppressLowSeverity=true
com.fortify.sca.UnicodeInputFile=true
com.fortify.sca.analyzer.controlflow.EnableLivenessOptimization=false
com.fortify.sca.analyzer.controlflow.EnableMachineFiltering=false
com.fortify.sca.analyzer.controlflow.EnableRefRuleOptimization=false
com.fortify.sca.analyzer.controlflow.EnableTimeOut=true
com.fortify.sca.compilers.ant=com.fortify.sca.util.compilers.AntAdapter

com.fortify.sca.compilers.ar=com.fortify.sca.util.compilers.ArUtil
com.fortify.sca.compilers.armcc=com.fortify.sca.util.compilers.ArmCcCompiler
com.fortify.sca.compilers.armcpp=com.fortify.sca.util.compilers.ArmCppCompiler
com.fortify.sca.compilers.c++=com.fortify.sca.util.compilers.GppCompiler
com.fortify.sca.compilers.c89=com.fortify.sca.util.compilers.C89Compiler
com.fortify.sca.compilers.cc=com.fortify.sca.util.compilers.GccCompiler
com.fortify.sca.compilers.cl=com.fortify.sca.util.compilers.MicrosoftCompiler
com.fortify.sca.compilers.clearmake=com.fortify.sca.util.compilers.TouchlessCompiler
com.fortify.sca.compilers.devenv=com.fortify.sca.util.compilers.DevenvNetAdapter
com.fortify.sca.compilers.fortify=com.fortify.sca.util.compilers.FortifyCompiler
com.fortify.sca.compilers.g++=com.fortify.sca.util.compilers.GppCompiler
com.fortify.sca.compilers.g++-*=com.fortify.sca.util.compilers.GppCompiler
com.fortify.sca.compilers.g++2*=com.fortify.sca.util.compilers.GppCompiler
com.fortify.sca.compilers.g++3*=com.fortify.sca.util.compilers.GppCompiler
com.fortify.sca.compilers.g++4*=com.fortify.sca.util.compilers.GppCompiler
com.fortify.sca.compilers.gcc=com.fortify.sca.util.compilers.GccCompiler
com.fortify.sca.compilers.gcc-*=com.fortify.sca.util.compilers.GccCompiler
com.fortify.sca.compilers.gcc2*=com.fortify.sca.util.compilers.GccCompiler
com.fortify.sca.compilers.gcc3*=com.fortify.sca.util.compilers.GccCompiler
com.fortify.sca.compilers.gcc4*=com.fortify.sca.util.compilers.GccCompiler
com.fortify.sca.compilers.gmake=com.fortify.sca.util.compilers.TouchlessCompiler
com.fortify.sca.compilers.icc=com.fortify.sca.util.compilers.IntelCompiler
com.fortify.sca.compilers.icl=com.fortify.sca.util.compilers.MicrosoftCompiler
com.fortify.sca.compilers.icpc=com.fortify.sca.util.compilers.IntelCompiler
com.fortify.sca.compilers.jam=com.fortify.sca.util.compilers.TouchlessCompiler
com.fortify.sca.compilers.javac=com.fortify.sca.util.compilers.JavacCompiler
com.fortify.sca.compilers.ld=com.fortify.sca.util.compilers.LdCompiler
com.fortify.sca.compilers.link=com.fortify.sca.util.compilers.MicrosoftLinker
com.fortify.sca.compilers.make=com.fortify.sca.util.compilers.TouchlessCompiler
com.fortify.sca.compilers.msbuild=com.fortify.sca.util.compilers.MSBuildAdapter
com.fortify.sca.compilers.msdev=com.fortify.sca.util.compilers.DevenvAdapter
com.fortify.sca.compilers.mvn=com.fortify.sca.util.compilers.MavenAdapter
com.fortify.sca.compilers.nmake=com.fortify.sca.util.compilers.TouchlessCompiler
com.fortify.sca.compilers.tcc=com.fortify.sca.util.compilers.ArmCcCompiler
com.fortify.sca.compilers.tcpp=com.fortify.sca.util.compilers.ArmCppCompiler
com.fortify.sca.compilers.touchless=com.fortify.sca.util.compilers.FortifyCompiler
com.fortify.sca.compilers.xilink=com.fortify.sca.util.compilers.MicrosoftLinker
com.fortify.sca.cpfe.command=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Core/private-bin/sca/cpfe.exe
com.fortify.sca.cpfe.file.option=--gen_c_file_name
com.fortify.sca.cpfe.new.command=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Core/private-bin/sca/cpfe441
com.fortify.sca.cpfe.options=--remove_unneeded_entities --suppress_vtbl -tused
com.fortify.sca.cpfe.options=--remove_unneeded_entities --suppress_vtbl -tused
com.fortify.sca.env.exesearchpath=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat;C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\bin;C:\Program Files\Common Files\Microsoft Shared\Windows Live;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0;C:\Program Files (x86)\Microsoft SQL Server\100\Tools\Binn;C:\Program Files\Microsoft SQL Server\100\Tools\Binn;C:\Program Files\Microsoft SQL Server\100\DTS\Binn;C:\Program Files\Perforce;C:\Program Files (x86)\Borland\SilkTest;C:\Program Files (x86)\Silk\SilkTest
com.fortify.sca.fileextensions.ABAP=ABAP

com.fortify.sca.fileextensions.abap=ABAP
com.fortify.sca.fileextensions.as=ACTIONSCRIPT
com.fortify.sca.fileextensions.asp=ASP
com.fortify.sca.fileextensions.bas=VB6
com.fortify.sca.fileextensions.cfc=CFML
com.fortify.sca.fileextensions.cfm=CFML
com.fortify.sca.fileextensions.cfml=CFML
com.fortify.sca.fileextensions.cls=VB6
com.fortify.sca.fileextensions.config=XML
com.fortify.sca.fileextensions.cpx=XML
com.fortify.sca.fileextensions.cs=CSHARP
com.fortify.sca.fileextensions.ctl=VB6
com.fortify.sca.fileextensions.ctp=PHP
com.fortify.sca.fileextensions.dll=MSIL
com.fortify.sca.fileextensions.exe=MSIL
com.fortify.sca.fileextensions.faces=JSPX
com.fortify.sca.fileextensions.frm=VB6
com.fortify.sca.fileextensions.htm=HTML
com.fortify.sca.fileextensions.html=HTML
com.fortify.sca.fileextensions.ini=JAVA_PROPERTIES
com.fortify.sca.fileextensions.java=JAVA
com.fortify.sca.fileextensions.js=JAVASCRIPT
com.fortify.sca.fileextensions.jsff=JSPX
com.fortify.sca.fileextensions.jsp=JSP
com.fortify.sca.fileextensions.jspx=JSPX
com.fortify.sca.fileextensions.mdl=MSIL
com.fortify.sca.fileextensions.mod=MSIL
com.fortify.sca.fileextensions.mxml=MXML
com.fortify.sca.fileextensions.php=PHP
com.fortify.sca.fileextensions.phtml=PHP
com.fortify.sca.fileextensions.pkb=PLSQL
com.fortify.sca.fileextensions.pkh=PLSQL
com.fortify.sca.fileextensions.pks=PLSQL
com.fortify.sca.fileextensions.properties=JAVA_PROPERTIES
com.fortify.sca.fileextensions.py=PYTHON
com.fortify.sca.fileextensions.settings=XML
com.fortify.sca.fileextensions.sql=SQL
com.fortify.sca.fileextensions.tag=JSP
com.fortify.sca.fileextensions.tagx=JSP
com.fortify.sca.fileextensions.tld=TLD
com.fortify.sca.fileextensions.vb=VB
com.fortify.sca.fileextensions.vbs=VB6
com.fortify.sca.fileextensions.vbscript=VBSCRIPT
com.fortify.sca.fileextensions.wsdd=XML
com.fortify.sca.fileextensions.xcfg=XML
com.fortify.sca.fileextensions.xhtml=JSPX
com.fortify.sca.fileextensions.xmi=XML
com.fortify.sca.fileextensions.xml=XML
com.fortify.sca.fileextensions.xsd=XML
com.fortify.sca.jsp.UseNativeParser=true
com.fortify.search.defaultSyntaxVer=2

com.sun.management.jmxremote=true
dotnet.install.dir=C:\Windows\Microsoft.NET\Framework
dotnet.v30.referenceAssemblies=C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\v3.0
dotnet.v35.referenceAssemblies=C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\v3.5
file.encoding=Cp1252
file.encoding.pkg=sun.io
file.separator=\
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
java.awt.printerjob=sun.awt.windows.WPrinterJob
java.class.path=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Core\lib\exe\sca-exe.jar
java.class.version=50.0
java.endorsed.dirs=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\lib\endorsed
java.ext.dirs=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\lib\ext;C:\Windows\Sun\Java\lib\ext
java.home=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre
java.io.tmpdir=C:\Users\fortify\AppData\Local\Temp\
java.library.path=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\bin;C:\Windows\Sun\Java\bin;C:\Windows\system32;C:\Windows;C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\bin;C:\Program Files\Common Files\Microsoft Shared\Windows Live;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0;C:\Program Files (x86)\Microsoft SQL Server\100\Tools\Binn;C:\Program Files\Microsoft SQL Server\100\Tools\Binn;C:\Program Files\Microsoft SQL Server\100\DTS\Binn;C:\Program Files\Perforce;C:\Program Files (x86)\Borland\SilkTest;C:\Program Files (x86)\Silk\SilkTest;.
java.rmi.server.randomIDs=true
java.runtime.name=Java(TM) SE Runtime Environment
java.runtime.version=1.6.0_43-b01
java.specification.name=Java Platform API Specification
java.specification.vendor=Sun Microsystems Inc.
java.specification.version=1.6
java.vendor=Sun Microsystems Inc.
java.vendor.url=http://java.sun.com/
java.vendor.url.bug=http://java.sun.com/cgi-bin/bugreport.cgi
java.version=1.6.0_43
java.vm.info=mixed mode
java.vm.name=Java HotSpot(TM) Server VM
java.vm.specification.name=Java Virtual Machine Specification
java.vm.specification.vendor=Sun Microsystems Inc.
java.vm.specification.version=1.0
java.vm.vendor=Sun Microsystems Inc.
java.vm.version=20.14-b01
line.separator=

max.file.path.length=255
os.arch=x86
os.name=Windows 7
os.version=6.1
path.separator=;
stderr.isatty=true
stdout.isatty=true
sun.arch.data.model=32
sun.boot.class.path=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\lib\resources.jar;C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\lib\rt.jar;C:\Program

Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\lib\sunrsasign.jar;C:\Program
Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\lib\jsse.jar;C:\Program
Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\lib\jce.jar;C:\Program
Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\lib\charsets.jar;C:\Program
Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\lib\modules\jdk.boot.jar;C:\Program
Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\classes
sun.boot.library.path=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\jre\bin
sun.cpu.endian=little
sun.cpu.isalist=pentium_pro+mmx pentium_pro pentium+mmx pentium i486 i386 i86
sun.desktop=windows
sun.io.unicode.encoding=UnicodeLittle
sun.java.command=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Core\lib\exe\sca-exe.jar -b WebGoat5.0 -scan -f WebGoat5.0.fpr
sun.java.launcher=SUN_STANDARD
sun.jnu.encoding=Cp1252
sun.management.compiler=HotSpot Tiered Compilers
sun.os.patch.level=Service Pack 1
user.country=US
user.dir=C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_3.90\Samples\advanced\webgoat
user.home=C:\Users\fortify
user.language=en
user.name=fortify
user.timezone=America/Los_Angeles
user.variant=
vs.100.dotnet.clr.version=v4.0.30319
vs.100.dotnet.install.dir=c:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE
win32.LocalAppdata=C:\Users\fortify\AppData\Local

Commandline Arguments

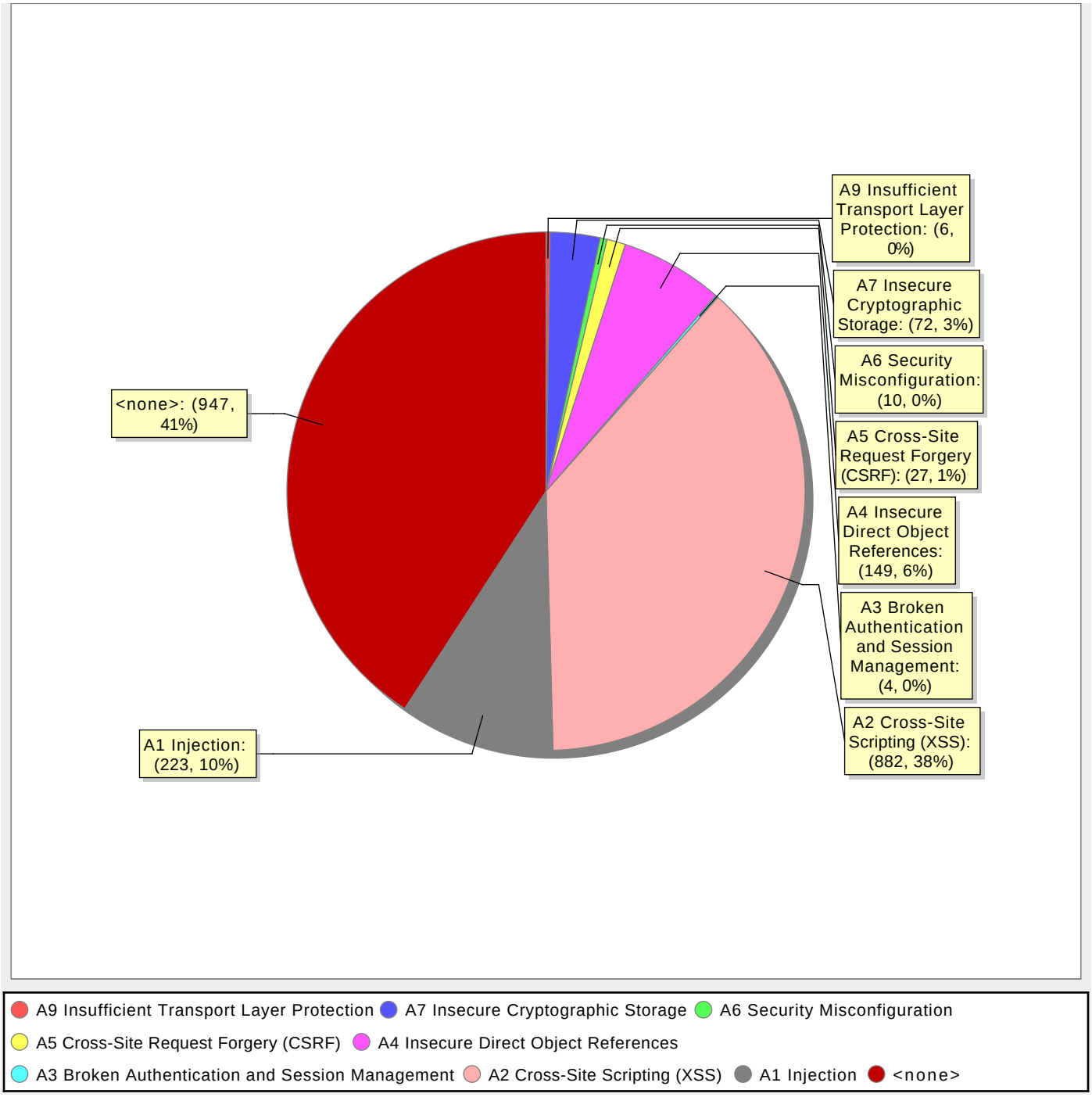
-b
WebGoat5.0
-scan
-f
WebGoat5.0.fpr

Warnings

No warnings occurred during analysis

Issue Count by Category

Issues by OWASP Top 10 2010



Issue Breakdown by Analysis	
Issues by PCI 2.0	
Requirement 6.5.9	27
Requirement 6.5.8, Requirement 8.5.15	1
Requirement 6.5.8	245
Requirement 6.5.7	882
Requirement 6.5.5	640
Requirement 6.5.3	7
Requirement 6.5.1, Requirement 10.5.2	25
Requirement 6.5.1	105
Requirement 4.1, Requirement 6.5.4	6
Requirement 3.4, Requirement 6.5.3, Requirement 8.4	65
Requirement 3.2, Requirement 3.4, Requirement 4.2, Requirement 6.5.5, Requirement 8.4	101
<none>	216