



Veracode Detailed Report
Application Security Report
As of 7 Oct 2016

Prepared for:	Veracode POV - Cyberseed
Prepared on:	October 10, 2016
Application:	Team_6
Industry:	Not Specified
Business Criticality:	BC3 (Medium)
Required Analysis:	Any
Type(s) of Analysis Conducted:	Static
Scope of Static Scan:	1 of 1 Modules Analyzed

Inside This Report

Executive Summary	1
Summary of Flaws by Severity	1
Action Items	1
Flaw Types by Category	2
Policy Summary	3
Findings & Recommendations	6
Methodology	

While every precaution has been taken in the preparation of this document, Veracode, Inc. assumes no responsibility for errors, omissions, or for damages resulting from the use of the information herein. The Veracode platform uses static and/or dynamic analysis techniques to discover potentially exploitable flaws. Due to the nature of software security testing, the lack of discoverable flaws does not mean the software is 100% secure.

Veracode Detailed Report Application Security Report As of 7 Oct 2016

Veracode Level: VL1

Rated: Oct 7, 2016

Application: Team_6
Target Level: None

Business Criticality: Medium
Published Rating: C

Scans Included in Report

Static Scan	Dynamic Scan	Manual Scan
Baseline Score: 52 Completed: 10/7/16	Not Included in Report	Not Included in Report

Executive Summary

This report contains a summary of the security flaws identified in the application using automated static, automated dynamic and/or manual security analysis techniques. This is useful for understanding the overall security quality of an individual application or for comparisons between applications.

Application Business Criticality: BC3 (Medium)

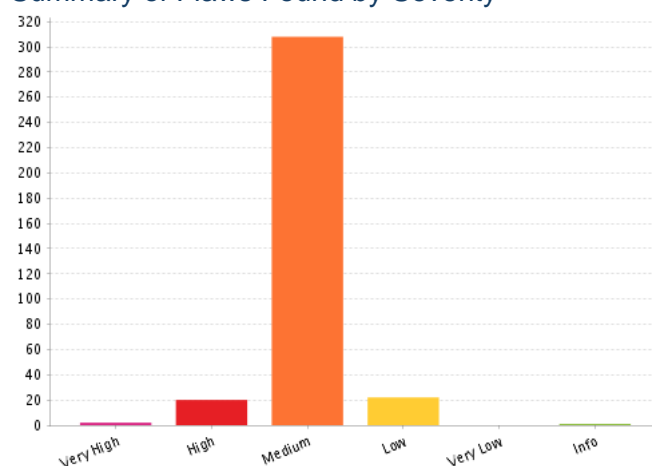
Impacts: Operational Risk (Low), Financial Loss (Medium)

An application's business criticality is determined by business risk factors such as: reputation damage, financial loss, operational risk, sensitive information disclosure, personal safety, and legal violations. The Veracode Level and required assessment techniques are selected based on the policy assigned to the application.

Analyses Performed vs. Required

	Any	Static	Dynamic	Manual
Performed:		●	○	○
Required:	●	○	○	○

Summary of Flaws Found by Severity



Action Items:

Veracode recommends the following approaches ranging from the most basic to the strong security measures that a vendor can undertake to increase the overall security level of the application.

Flaws To Fix By Expires Date

A grace period is specified for any flaw that violates the rules contained in your policy. These include CWE, Rollup Category, Issue Severity, Industry Standards as well as any flaws the prevent an application from achieving a minimum Veracode Level and/or score. To maintain policy compliance you must fix these flaws and resubmit your application for scanning before the grace period expires. The detailed flaw listing will badge the flaws that must be fixed and show the fix by date as well.

→ The grace period has expired [10/7/16] for 303 flaws that were found in your Static Scan.

Longer Timeframe (6 - 12 months)

→ Certify that software engineers have been trained on application security principles and practices.

Scope of Static Scan

It is important to note that this application may include additional modules which were not included in this analysis. We recommend that you contact the vendor to determine whether all modules have been included.

Engine Version: 102039

The following modules were included in the application scan:

Module Name	Compiler	Operating Environment	Engine Version
WebGoat-6.0.1.war	JAVAC_7	Java J2SE 7	102039

Flaw Types by Severity and Category

Static Scan Security Quality Score = 52			
Very High	2		
OS Command Injection	2		
High	20		
SQL Injection	20		
Medium	308		
CRLF Injection	10		
Code Quality	3		
Credentials Management	21		
Cross-Site Scripting	258		
Cryptographic Issues	7		
Directory Traversal	3		
Encapsulation	2		
Insufficient Input Validation	3		
Session Fixation	1		
Low	22		
API Abuse	3		
Code Quality	4		
Cryptographic Issues	5		
Information Leakage	10		
Very Low	0		
Informational	1		
Potential Backdoor	1		
Total	353		

Policy Evaluation

Policy Name: CyberSeed Compliance

Revision: 1

Policy Status: Did Not Pass

Description

Competition policy

Rules

Rule type	Requirement	Findings	Status
CWE	80 Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)	Flaws found: 258	Did not pass
CWE	89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	Flaws found: 20	Did not pass
CWE	93 Improper Neutralization of CRLF Sequences ('CRLF Injection')	Flaws found: 2	Did not pass
CWE	113 Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')	Flaws found: 4	Did not pass
CWE	78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	Flaws found: 2	Did not pass
CWE	117 Improper Output Neutralization for Logs	Flaws found: 4	Did not pass
CWE	256 Plaintext Storage of a Password	Flaws found: 1	Did not pass
CWE	326 Inadequate Encryption Strength	Flaws found: 2	Did not pass
CWE	327 Use of a Broken or Risky Cryptographic Algorithm	Flaws found: 3	Did not pass
CWE	331 Insufficient Entropy	Flaws found: 2	Did not pass
CWE	597 Use of Wrong Operator in String Comparison	Flaws found: 4	Did not pass
CWE	601 URL Redirection to Untrusted Site ('Open Redirect')	Flaws found: 1	Did not pass

Scan Requirements

Scan Type	Frequency	Last performed	Status
Any	Once	10/7/16	Passed

Remediation

Flaw Severity	Grace Period	Flaws Exceeding	Status
Very High	0 days	2	Did not pass
High	0 days	20	Did not pass

Flaw Severity	Grace Period	Flaws Exceeding	Status
Medium	0 days	277	Did not pass
Low	0 days	4	Did not pass
Very Low	0 days	0	Passed
Informational	0 days	0	Passed

Type	Grace Period	Exceeding	Status
Min Analysis Score	0 days	0	Passed

Unsupported Frameworks

This report may have incomplete results based on the following unsupported frameworks identified during the static scan:

- * Apache ECS

The lack of support for all frameworks in use by this application and/or its supporting libraries may prevent the static discovery of some flaws in the application, however, it does not invalidate the flaws that were found.

Findings & Recommendations

Detailed Flaws by Severity

Very High (2 flaws)

→ OS Command Injection(2 flaws)

Description

OS command injection vulnerabilities occur when data enters an application from an untrusted source and is used to dynamically construct and execute a system command. This allows an attacker to either alter the command executed by the application or append additional commands. The command is typically executed with the privileges of the executing process and gives an attacker a privilege or capability that he would not otherwise have.

Recommendations

Careful handling of all untrusted data is critical in preventing OS command injection attacks. Using one or more of the following techniques provides defense-in-depth and minimizes the likelihood of an vulnerability.

- * If possible, use library calls rather than external processes to recreate the desired functionality.
- * Validate user-supplied input using positive filters (white lists) to ensure that it conforms to the expected format, using centralized data validation routines when possible.
- * Select safe API routines. Some APIs that execute system commands take an array of strings as input rather than a single string, which protects against some forms of command injection by ensuring that a user-supplied argument cannot be interpreted as part of the command.

Associated Flaws by CWE ID:

→ Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (CWE ID 78)(2 flaws)

 **Fix Required by Policy**

Description

This call contains a command injection flaw. The argument to the function is constructed using user-supplied input. If an attacker is allowed to specify all or part of the command, it may be possible to execute commands on the server with the privileges of the executing process. The level of exposure depends on the effectiveness of input validation routines, if any.


Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

Recommendations

Validate all user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible. When using black lists, be sure that the sanitizing routine performs a sufficient number of iterations to remove all instances of disallowed characters. Most APIs that execute system commands also have a "safe" version of the method that takes an array of strings as input rather than a single string, which protects against some forms of command injection.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
 322	23	-	WebGoat-6.0.1.war	org/.../webgoat/util/Exec.java 107	10/7/16

Flaw Id	Module #	Class #	Module	Location	Fix By
 40	23	-	WebGoat-6.0.1.war	org/.../webgoat/util/Exec.java 289	10/7/16

High (20 flaws)

→ SQL Injection(20 flaws)

Description

SQL injection vulnerabilities occur when data enters an application from an untrusted source and is used to dynamically construct a SQL query. This allows an attacker to manipulate database queries in order to access, modify, or delete arbitrary data. Depending on the platform, database type, and configuration, it may also be possible to execute administrative operations on the database, access the filesystem, or execute arbitrary system commands. SQL injection attacks can also be used to subvert authentication and authorization schemes, which would enable an attacker to gain privileged access to restricted portions of the application.

Recommendations

Several techniques can be used to prevent SQL injection attacks. These techniques complement each other and address security at different points in the application. Using multiple techniques provides defense-in-depth and minimizes the likelihood of a SQL injection vulnerability.

- * Use parameterized prepared statements rather than dynamically constructing SQL queries. This will prevent the database from interpreting the contents of bind variables as part of the query and is the most effective defense against SQL injection.
- * Validate user-supplied input using positive filters (white lists) to ensure that it conforms to the expected format, using centralized data validation routines when possible.
- * Normalize all user-supplied data before applying filters or regular expressions, or submitting the data to a database. This means that all URL-encoded (%xx), HTML-encoded (&#xx;), or other encoding schemes should be reduced to the internal character representation expected by the application. This prevents attackers from using alternate encoding schemes to bypass filters.
- * When using database abstraction libraries such as Hibernate, do not assume that all methods exposed by the API will automatically prevent SQL injection attacks. Most libraries contain methods that pass arbitrary queries to the database in an unsafe manner.

Associated Flaws by CWE ID:

→ Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (CWE ID 89)(20 flaws)

 **Fix Required by Policy**

Description

This database query contains a SQL injection flaw. The function call constructs a dynamic SQL query using a variable derived from user-supplied input. An attacker could exploit this flaw to execute arbitrary SQL queries against the database.

Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

Recommendations

Avoid dynamically constructing SQL queries. Instead, use parameterized prepared statements to prevent the database from interpreting the contents of bind variables as part of the query. Always validate user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
	326	1	-	WebGoat-6.0.1.war	org/.../lessons/BackDoors.java 141	10/7/16
	193	1	-	WebGoat-6.0.1.war	org/.../lessons/BackDoors.java 148	10/7/16
	105	1	-	WebGoat-6.0.1.war	org/.../lessons/BackDoors.java 180	10/7/16
	84	3	-	WebGoat-6.0.1.war	.../BlindNumericSqlInjection.java 114	10/7/16
	164	5	-	WebGoat-6.0.1.war	.../BlindStringSqlInjection.java 114	10/7/16
	67	6	-	WebGoat-6.0.1.war	org/.../Challenge2Screen.java 226	10/7/16
	279	11	-	WebGoat-6.0.1.war	org/.../lessons/DOS_Login.java 106	10/7/16
	336	11	-	WebGoat-6.0.1.war	org/.../lessons/DOS_Login.java 122	10/7/16
	244	40	-	WebGoat-6.0.1.war	org/.../Login.java 127	10/7/16
	262	39	-	WebGoat-6.0.1.war	org/.../SQLInjection/Login.java 131	10/7/16
	38	39	-	WebGoat-6.0.1.war	org/.../SQLInjection/Login.java 166	10/7/16
	107	66	-	WebGoat-6.0.1.war	org/.../lessons/SqlAddData.java 99	10/7/16
	251	67	-	WebGoat-6.0.1.war	org/.../lessons/SqlModifyData.java 107	10/7/16
	31	68	-	WebGoat-6.0.1.war	.../SqlNumericInjection.java 118	10/7/16
	331	69	-	WebGoat-6.0.1.war	.../SqlStringInjection.java 100	10/7/16
	268	70	-	WebGoat-6.0.1.war	.../ThreadSafetyProblem.java 98	10/7/16
	333	73	-	WebGoat-6.0.1.war	org/.../admin/ViewDatabase.java 82	10/7/16
	91	74	-	WebGoat-6.0.1.war	org/.../ViewProfile.java 108	10/7/16
	69	74	-	WebGoat-6.0.1.war	org/.../ViewProfile.java 157	10/7/16
	297	87	-	WebGoat-6.0.1.war	org/.../WsSqlInjection.java 233	10/7/16

Medium (308 flaws)

→ CRLF Injection(10 flaws)

Description

The acronym CRLF stands for "Carriage Return, Line Feed" and refers to the sequence of characters used to denote the end of a line of text. CRLF injection vulnerabilities occur when data enters an application from an untrusted source and is not properly validated before being used. For example, if an attacker is able to inject a CRLF into a log file, he could append falsified log entries, thereby misleading administrators or cover traces of the attack. If an attacker is able to inject CRLFs into an HTTP response header, he can use this ability to carry out other attacks such as cache poisoning. CRLF vulnerabilities primarily affect data integrity.

Recommendations

Apply robust input filtering for all user-supplied data, using centralized data validation routines when possible. Use output filters to sanitize all output derived from user-supplied input, replacing non-alphanumeric characters with their HTML entity equivalents.

Associated Flaws by CWE ID:

→ Improper Neutralization of CRLF Sequences ('CRLF Injection') (CWE ID 93)(2 flaws)

 **Fix Required by Policy**

Description



A function call contains a CRLF Injection flaw. Writing unsanitized user-supplied input to an interface or external application that treats the CRLF (carriage return line feed) sequence as a delimiter to separate lines or records can result in that data being misinterpreted. FTP and SMTP are examples of protocols that treat CRLF as a delimiter when parsing commands.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Sanitize CRLF sequences from user-supplied input when the data is being passed to an entity that may incorrectly interpret it.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
	55	71	-	WebGoat-6.0.1.war	org/.../UncheckedEmail.java 364	10/7/16
	25	71	-	WebGoat-6.0.1.war	org/.../UncheckedEmail.java 367	10/7/16

→ Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') (CWE ID 113)(4 flaws)

 **Fix Required by Policy**

Description

A function call contains an HTTP response splitting flaw. Writing unsanitized user-supplied input into an HTTP header allows an attacker to manipulate the HTTP response rendered by the browser, leading to cache poisoning and cross-site scripting attacks.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Remove unexpected carriage returns and line feeds from user-supplied data used to construct an HTTP response. Always validate user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
 305	27	-	WebGoat-6.0.1.war	org/.../lessons/HttpOnly.java 217	10/7/16
 200	27	-	WebGoat-6.0.1.war	org/.../lessons/HttpOnly.java 235	10/7/16
 303	53	-	WebGoat-6.0.1.war	/lessons/.../General/redirect.jsp 10	10/7/16
 30	84	-	WebGoat-6.0.1.war	org/.../session/WebSession.java 276	10/7/16

→ Improper Output Neutralization for Logs (CWE ID 117)(4 flaws)

 **Fix Required by Policy**

Description

A function call could result in a log forging attack. Writing unsanitized user-supplied data into a log file allows an attacker to forge log entries or inject malicious content into log files. Corrupted log files can be used to cover an attacker's tracks or as a delivery mechanism for an attack on a log viewing or processing utility. For example, if a web administrator uses a browser-based utility to review logs, a cross-site scripting attack might be possible.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Avoid directly embedding user input in log files when possible. Sanitize user-supplied data used to construct log entries by using a safe logging mechanism such as the OWASP ESAPI Logger, which will automatically remove unexpected carriage returns and line feeds and can be configured to use HTML entity encoding for non-alphanumeric data. Only write custom blacklisting code when absolutely necessary. Always validate user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
 194	2	-	WebGoat-6.0.1.war	org/.../service/BaseService.java 60	10/7/16
 309	25	-	WebGoat-6.0.1.war	org/.../webgoat/HammerHead.java 261	10/7/16
 44	25	-	WebGoat-6.0.1.war	org/.../webgoat/HammerHead.java 262	10/7/16
 214	84	-	WebGoat-6.0.1.war	org/.../session/WebSession.java 924	10/7/16

→ Code Quality(3 flaws)

Description

Code quality issues stem from failure to follow good coding practices and can lead to unpredictable behavior. These may include but are not limited to:

- * Neglecting to remove debug code or dead code
- * Improper resource management, such as using a pointer after it has been freed
- * Using the incorrect operator to compare objects
- * Failing to follow an API or framework specification
- * Using a language feature or API in an unintended manner

While code quality flaws are generally less severe than other categories and usually are not directly exploitable, they may serve as indicators that developers are not following practices that increase the reliability and security of an application. For an attacker, code quality issues may provide an opportunity to stress the application in unexpected ways.

Recommendations

The wide variance of code quality issues makes it impractical to generalize how these issues should be addressed. Refer to individual categories for specific recommendations.

Associated Flaws by CWE ID:

→ Leftover Debug Code (CWE ID 489)(3 flaws)

Description

A method may be leftover debug code that creates an unintended entry point in a web application. Although this is an acceptable practice during product development, classes that are part of a production J2EE application should not define a main() method. Whether this method can be remotely invoked depends on the configuration of the J2EE container and the application itself.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Remove debug code prior to deploying the application. Eliminate unnecessary entry points in deployed web applications to reduce the attack surface.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
202	19	-	WebGoat-6.0.1.war	org/.../lessons/Encoding.java 559	
100	23	-	WebGoat-6.0.1.war	org/.../webgoat/util/Exec.java 496	
285	83	-	WebGoat-6.0.1.war	org/.../WebgoatProperties.java 109	

→ Credentials Management(21 flaws)

Description

Improper management of credentials, such as usernames and passwords, may compromise system security. In particular, storing passwords in plaintext or hard-coding passwords directly into application code are design issues that cannot be easily remedied. Not only does embedding 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 a hard-coded password is compromised in a commercial product, all deployed instances may be vulnerable to attack, putting customers at risk.

One variation on hard-coding plaintext passwords is to hard-code a constant string which is the result of a cryptographic one-way hash. For example, instead of storing the word "secret," the application stores an MD5 hash of the word. This is a common mechanism for obscuring hard-coded passwords from casual viewing but does not significantly reduce risk. However, using cryptographic hashes for data stored outside the application code can be an effective practice.

Recommendations

Avoid storing passwords in easily accessible locations, and never store any type of sensitive data in plaintext. Avoid using hard-coded usernames, passwords, or hash constants whenever possible, particularly in relation to security-critical components. Store passwords out-of-band from the application code. Follow best practices for protecting credentials stored in alternate locations such as configuration or properties files.

Associated Flaws by CWE ID:

- **Plaintext Storage of a Password (CWE ID 256)(1 flaw)**  **Fix Required by Policy**

Description

A method reads and/or stores sensitive information in plaintext, making the data more susceptible to compromise.

Effort to Fix: 4 - Simple design error. Requires redesign and up to 5 days to fix.

Recommendations

Never store sensitive data in plaintext. Consider using cryptographic hashes as an alternative to plaintext.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
	27	9	-	WebGoat-6.0.1.war	org/.../DatabaseUtilities.java 100	10/7/16

- **Use of Hard-coded Password (CWE ID 259)(20 flaws)**

Description

A method uses a hard-coded password that may compromise system security in a way that cannot be easily remedied. The use of a hard-coded password significantly increases the possibility that the account being protected will be compromised. Moreover, the password cannot be changed without patching the software. If a hard-coded password is compromised in a commercial product, all deployed instances may be vulnerable to attack.

Effort to Fix: 4 - Simple design error. Requires redesign and up to 5 days to fix.

Recommendations

Store passwords out-of-band from the application code. Follow best practices for protecting credentials stored in locations such as configuration or properties files.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
57	6	-	WebGoat-6.0.1.war	org/.../Challenge2Screen.java 1	
204	6	-	WebGoat-6.0.1.war	org/.../Challenge2Screen.java 121	
131	11	-	WebGoat-6.0.1.war	org/.../lessons/DOS_Login.java 1	
154	12	-	WebGoat-6.0.1.war	org/.../session/ECSFactory.java 1	
152	22	-	WebGoat-6.0.1.war	/lessons/Ajax/eval.jsp 28	
231	24	-	WebGoat-6.0.1.war	.../GoatHillsFinancial.java 1	
161	26	-	WebGoat-6.0.1.war	org/.../lessons/HtmlClues.java 1	
5	28	-	WebGoat-6.0.1.war	org/.../lessons/InsecureLogin.java 1	
18	47	-	WebGoat-6.0.1.war	org/.../lessons/LogSpoofing.java 1	
4	49	-	WebGoat-6.0.1.war	org/.../MultiLevelLogin1.java 1	
128	50	-	WebGoat-6.0.1.war	org/.../MultiLevelLogin2.java 1	
245	62	-	WebGoat-6.0.1.war	org/.../SessionFixation.java 1	
175	71	-	WebGoat-6.0.1.war	org/.../UncheckedEmail.java 1	
209	71	-	WebGoat-6.0.1.war	org/.../UncheckedEmail.java 1	
299	71	-	WebGoat-6.0.1.war	org/.../UncheckedEmail.java 70	
123	81	-	WebGoat-6.0.1.war	.../WeakAuthenticationCookie.java 1	
282	82	-	WebGoat-6.0.1.war	org/.../lessons/WeakSessionID.java 1	
8	86	-	WebGoat-6.0.1.war	org/.../WsSAXInjection.java 1	
167	86	-	WebGoat-6.0.1.war	org/.../WsSAXInjection.java 184	
21	89	-	WebGoat-6.0.1.war	org/.../XPathInjection.java 1	

→ Cross-Site Scripting(258 flaws)

Description

Cross-site scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed occur whenever a web application uses untrusted data in the output it generates without validating or encoding it. XSS vulnerabilities are commonly exploited to steal or manipulate cookies, modify presentation of content, and compromise sensitive information, with new attack vectors being discovered on a regular basis. XSS is also commonly referred to as HTML injection.

XSS vulnerabilities can be either persistent or transient (often referred to as stored and reflected, respectively). In a persistent XSS vulnerability, the injected code is stored by the application, for example within a blog comment or message board. The attack occurs whenever a victim views the page containing the malicious script. In a transient XSS vulnerability, the injected code is included directly in the HTTP request. These attacks are often carried out via malicious URLs sent via email or another website and requires the victim to browse to that link. The consequence of an XSS attack to a victim is the same regardless of whether it is persistent or transient; however, persistent XSS vulnerabilities are likely to affect a greater number of victims due to its delivery mechanism.

Recommendations

Several techniques can be used to prevent XSS attacks. These techniques complement each other and address security at different points in the application. Using multiple techniques provides defense-in-depth and minimizes the likelihood of a XSS vulnerability.

- * Use output filtering to sanitize all output generated from user-supplied input, selecting the appropriate method of encoding based on the use case of the untrusted data. For example, if the data is being written to the body of an HTML page, use HTML entity encoding. However, if the data is being used to construct generated Javascript or if it is consumed by client-side methods that may interpret it as code (a common technique in Web 2.0 applications), additional restrictions may be necessary beyond simple HTML encoding.
- * Validate user-supplied input using positive filters (white lists) to ensure that it conforms to the expected format, using centralized data validation routines when possible.
- * Do not permit users to include HTML content in posts, notes, or other data that will be displayed by the application. If users are permitted to include HTML tags, then carefully limit access to specific elements or attributes, and use strict validation filters to prevent abuse.

Associated Flaws by CWE ID:

→ Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (CWE ID 80)(258 flaws)

 **Fix Required by Policy**

Description

This call contains a cross-site scripting (XSS) flaw. The application populates the HTTP response with user-supplied input, allowing an attacker to embed malicious content, such as Javascript code, which will be executed in the context of the victim's browser. XSS vulnerabilities are commonly exploited to steal or manipulate cookies, modify presentation of content, and compromise confidential information, with new attack vectors being discovered on a regular basis.































Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.































Recommendations































Use contextual escaping on all untrusted data before using it to construct any portion of an HTTP response. The escaping method should be chosen based on the specific use case of the untrusted data, otherwise it may not protect fully against the attack. For example, if the data is being written to the body of an HTML page, use HTML entity escaping; if the data is being written to an attribute, use attribute escaping; etc. When a web framework provides built-in support for automatic XSS escaping, do not disable it. Both the OWASP Java Encoder library for Java and the Microsoft AntiXSS library provide contextual escaping methods. For more details on contextual escaping, see https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet. In addition, as a best practice, always validate user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible.































Instances found via Static Scan































Flaw Id	Module #	Class #	Module	Location	Fix By
 56	8	-	WebGoat-6.0.1.war	org/.../service/CookieService.java 62	10/7/16
 87	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 8	10/7/16
 141	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 8	10/7/16
 171	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 8	10/7/16
 59	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 8	10/7/16































Flaw Id	Module #	Class #	Module	Location	Fix By
 196	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 8	10/7/16
 232	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 8	10/7/16
 153	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 10	10/7/16
 156	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 10	10/7/16
 98	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 10	10/7/16
 103	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 10	10/7/16
 173	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 10	10/7/16
 345	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 10	10/7/16
 22	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 16	10/7/16
 181	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 16	10/7/16
 281	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 16	10/7/16
 230	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 16	10/7/16
 308	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 16	10/7/16
 343	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 16	10/7/16
 199	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 21	10/7/16
 213	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 21	10/7/16
 119	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 22	10/7/16
 76	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 22	10/7/16
 228	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 22	10/7/16
 304	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 22	10/7/16
 316	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 28	10/7/16
 234	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 28	10/7/16
 78	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 29	10/7/16
 184	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 29	10/7/16
 337	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 29	10/7/16
 270	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 29	10/7/16
 348	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 33	10/7/16
 342	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 33	10/7/16
 150	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 34	10/7/16
 47	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 34	10/7/16































Flaw Id	Module #	Class #	Module	Location	Fix By
 48	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 34	10/7/16
 346	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 34	10/7/16
 117	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 40	10/7/16
 293	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 40	10/7/16
 92	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 41	10/7/16
 104	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 41	10/7/16
 318	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 41	10/7/16
 208	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 41	10/7/16
 134	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 46	10/7/16
 106	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 46	10/7/16
 146	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 47	10/7/16
 88	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 47	10/7/16
 24	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 47	10/7/16
 306	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 47	10/7/16
 176	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 53	10/7/16
 260	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 53	10/7/16
 93	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 54	10/7/16
 16	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 54	10/7/16
 277	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 54	10/7/16
 212	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 54	10/7/16
 224	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 66	10/7/16
 237	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 66	10/7/16
 144	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 67	10/7/16
 323	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 67	10/7/16
 233	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 67	10/7/16
 210	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 67	10/7/16
 311	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 79	10/7/16
 324	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 79	10/7/16
 115	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 80	10/7/16
 162	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 80	10/7/16































Flaw Id	Module #	Class #	Module	Location	Fix By
 320	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 80	10/7/16
 301	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 80	10/7/16
 14	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 89	10/7/16
 32	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 89	10/7/16
 83	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 94	10/7/16
 124	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 94	10/7/16
 158	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 94	10/7/16
 28	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 94	10/7/16
 222	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 94	10/7/16
 198	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 94	10/7/16
 120	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 103	10/7/16
 182	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 103	10/7/16
 108	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 103	10/7/16
 273	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 103	10/7/16
 179	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 109	10/7/16
 10	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 109	10/7/16
 314	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 109	10/7/16
 325	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 109	10/7/16
 58	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 110	10/7/16
 61	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 110	10/7/16
 79	14	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 123	10/7/16
 114	16	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 123	10/7/16
 11	18	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 123	10/7/16
 339	13	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 123	10/7/16
 140	15	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 126	10/7/16
 280	17	-	WebGoat-6.0.1.war	/lessons/.../EditProfile.jsp 126	10/7/16
 289	20	-	WebGoat-6.0.1.war	/lessons/.../error.jsp 10	10/7/16
 341	21	-	WebGoat-6.0.1.war	/lessons/.../error.jsp 10	10/7/16
 221	22	-	WebGoat-6.0.1.war	/lessons/Ajax/eval.jsp 31	10/7/16
 34	22	-	WebGoat-6.0.1.war	/lessons/Ajax/eval.jsp 35	10/7/16














Flaw Id	Module #	Class #	Module	Location	Fix By
 287	30	-	WebGoat-6.0.1.war	/lesson_content.jsp 26	10/7/16
 332	31	-	WebGoat-6.0.1.war	org/.../LessonPlanService.java 62	10/7/16
 276	32	-	WebGoat-6.0.1.war	org/.../webgoat/LessonSource.java 182	10/7/16
 122	36	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 8	10/7/16
 90	34	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 8	10/7/16
 19	37	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 8	10/7/16
 136	33	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 8	10/7/16
 143	35	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 8	10/7/16
 225	38	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 8	10/7/16
 112	34	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 13	10/7/16
 206	36	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 13	10/7/16
 189	33	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 14	10/7/16
 292	37	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 14	10/7/16
 246	35	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 14	10/7/16
 340	38	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 14	10/7/16
 36	34	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 22	10/7/16
 43	36	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 22	10/7/16
 65	35	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 23	10/7/16
 137	33	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 23	10/7/16
 271	37	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 23	10/7/16
 272	38	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 23	10/7/16
 81	34	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 26	10/7/16
 102	36	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 26	10/7/16
 151	35	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 27	10/7/16
 291	33	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 27	10/7/16
 242	37	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 27	10/7/16
 219	38	-	WebGoat-6.0.1.war	/lessons/.../ListStaff.jsp 27	10/7/16
 149	41	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 9	10/7/16
 129	43	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 9	10/7/16
 163	44	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 9	10/7/16

Flaw Id	Module #	Class #	Module	Location	Fix By
 310	45	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 9	10/7/16
 223	42	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 9	10/7/16
 264	46	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 9	10/7/16
 82	42	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 20	10/7/16
 73	43	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 20	10/7/16
 62	44	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 20	10/7/16
 321	45	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 20	10/7/16
 294	46	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 20	10/7/16
 298	41	-	WebGoat-6.0.1.war	/lessons/.../Login.jsp 20	10/7/16
 12	48	-	WebGoat-6.0.1.war	/main.jsp 99	10/7/16
 238	48	-	WebGoat-6.0.1.war	/main.jsp 163	10/7/16
 256	48	-	WebGoat-6.0.1.war	/main.jsp 168	10/7/16
 135	48	-	WebGoat-6.0.1.war	/main.jsp 173	10/7/16
 72	48	-	WebGoat-6.0.1.war	/main.jsp 179	10/7/16
 188	48	-	WebGoat-6.0.1.war	/main.jsp 182	10/7/16
 216	48	-	WebGoat-6.0.1.war	/main.jsp 184	10/7/16
 307	48	-	WebGoat-6.0.1.war	/main.jsp 220	10/7/16
 236	48	-	WebGoat-6.0.1.war	/main.jsp 230	10/7/16
 263	48	-	WebGoat-6.0.1.war	/main.jsp 238	10/7/16
 187	48	-	WebGoat-6.0.1.war	/main.jsp 247	10/7/16
 29	48	-	WebGoat-6.0.1.war	/main.jsp 250	10/7/16
 190	48	-	WebGoat-6.0.1.war	/main.jsp 266	10/7/16
 275	51	-	WebGoat-6.0.1.war	org/.../ParameterService.java 65	10/7/16
 64	54	-	WebGoat-6.0.1.war	/reportBug.jsp 48	10/7/16
 313	54	-	WebGoat-6.0.1.war	/reportBug.jsp 55	10/7/16
 334	55	-	WebGoat-6.0.1.war	.../RestartLessonService.java 57	10/7/16
 147	56	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 11	10/7/16
 15	57	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 11	10/7/16
 165	60	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 11	10/7/16
 183	59	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 11	10/7/16

Flaw Id	Module #	Class #	Module	Location	Fix By
 226	58	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 11	10/7/16
 266	61	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 11	10/7/16
 6	56	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 15	10/7/16
 45	58	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 15	10/7/16
 97	59	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 15	10/7/16
 317	61	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 15	10/7/16
 211	60	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 15	10/7/16
 252	57	-	WebGoat-6.0.1.war	/lessons/.../SearchStaff.jsp 15	10/7/16
 192	63	-	WebGoat-6.0.1.war	.../SilentTransactions.java 89	10/7/16
 259	64	-	WebGoat-6.0.1.war	org/.../SolutionService.java 59	10/7/16
 46	65	-	WebGoat-6.0.1.war	org/.../service/SourceService.java 64	10/7/16
 166	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 7	10/7/16
 155	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 9	10/7/16
 70	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 9	10/7/16
 349	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 9	10/7/16
 296	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 9	10/7/16
 132	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 12	10/7/16
 157	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 14	10/7/16
 85	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 16	10/7/16
 42	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 16	10/7/16
 185	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 16	10/7/16
 207	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 16	10/7/16
 352	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 19	10/7/16
 269	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 20	10/7/16
 177	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 22	10/7/16
 174	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 22	10/7/16
 288	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 22	10/7/16
 253	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 22	10/7/16
 23	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 25	10/7/16
 99	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 27	10/7/16

Flaw Id	Module #	Class #	Module	Location	Fix By
 17	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 29	10/7/16
 138	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 29	10/7/16
 283	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 29	10/7/16
 261	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 29	10/7/16
 284	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 32	10/7/16
 2	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 34	10/7/16
 20	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 34	10/7/16
 344	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 34	10/7/16
 329	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 34	10/7/16
 239	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 34	10/7/16
 110	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 39	10/7/16
 127	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 39	10/7/16
 170	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 41	10/7/16
 278	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 41	10/7/16
 227	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 41	10/7/16
 201	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 41	10/7/16
 148	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 45	10/7/16
 218	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 46	10/7/16
 13	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 47	10/7/16
 160	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 47	10/7/16
 77	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 47	10/7/16
 243	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 47	10/7/16
 126	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 52	10/7/16
 186	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 52	10/7/16
 80	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 54	10/7/16
 37	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 54	10/7/16
 257	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 54	10/7/16
 338	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 54	10/7/16
 101	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 59	10/7/16
 302	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 65	10/7/16

Flaw Id	Module #	Class #	Module	Location	Fix By
 50	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 67	10/7/16
 350	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 67	10/7/16
 330	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 67	10/7/16
 250	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 67	10/7/16
 52	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 72	10/7/16
 60	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 78	10/7/16
 7	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 80	10/7/16
 68	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 80	10/7/16
 71	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 80	10/7/16
 248	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 80	10/7/16
 197	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 87	10/7/16
 94	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 91	10/7/16
 53	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 91	10/7/16
 315	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 91	10/7/16
 109	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 93	10/7/16
 265	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 93	10/7/16
 33	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 95	10/7/16
 203	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 95	10/7/16
 241	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 97	10/7/16
 142	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 99	10/7/16
 347	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 99	10/7/16
 247	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 100	10/7/16
 255	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 106	10/7/16
 195	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 110	10/7/16
 86	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 112	10/7/16
 335	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 112	10/7/16
 39	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 116	10/7/16
 258	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 116	10/7/16
 41	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 119	10/7/16
 111	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 122	10/7/16

Flaw Id	Module #	Class #	Module	Location	Fix By
 327	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 125	10/7/16
 217	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 125	10/7/16
 51	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 128	10/7/16
 254	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 128	10/7/16
 121	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 131	10/7/16
 300	75	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 135	10/7/16
 9	76	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 138	10/7/16
 168	80	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 138	10/7/16
 95	77	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 141	10/7/16
 290	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 141	10/7/16
 274	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 144	10/7/16
 145	78	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 151	10/7/16
 1	79	-	WebGoat-6.0.1.war	/lessons/.../ViewProfile.jsp 154	10/7/16

→ Cryptographic Issues(7 flaws)

Description

Applications commonly use cryptography to implement authentication mechanisms and to ensure the confidentiality and integrity of sensitive data, both in transit and at rest. The proper and accurate implementation of cryptography is extremely critical to its efficacy. Configuration or coding mistakes as well as incorrect assumptions may negate a large degree of the protection it affords, leaving the crypto implementation vulnerable to attack.

Common cryptographic mistakes include, but are not limited to, selecting weak keys or weak cipher modes, unintentionally exposing sensitive cryptographic data, using predictable entropy sources, and mismanaging or hard-coding keys.

Developers often make the dangerous assumption that they can improve security by designing their own cryptographic algorithm; however, one of the basic tenets of cryptography is that any cipher whose effectiveness is reliant on the secrecy of the algorithm is fundamentally flawed.

Recommendations

Select the appropriate type of cryptography for the intended purpose. Avoid proprietary encryption algorithms as they typically rely on "security through obscurity" rather than sound mathematics. Select key sizes appropriate for the data being protected; for high assurance applications, 256-bit symmetric keys and 2048-bit asymmetric keys are sufficient. Follow best practices for key storage, and ensure that plaintext data and key material are not inadvertently exposed.

Associated Flaws by CWE ID:

→ Inadequate Encryption Strength (CWE ID 326)(2 flaws)

 **Fix Required by Policy**

Description

Insufficiently strong encryption schemes may not adequately secure secret data from attackers. This can result from poor cipher selection, insufficient key size, or weak key selection.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Use a cryptographic algorithm that has been subject to public scrutiny. Follow security best practices when selecting key sizes and when generating key material.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
 312	19	-	WebGoat-6.0.1.war	org/.../lessons/Encoding.java 321	10/7/16
 89	19	-	WebGoat-6.0.1.war	org/.../lessons/Encoding.java 366	10/7/16

→ Insufficient Entropy (CWE ID 331)(2 flaws)

 **Fix Required by Policy**

Description

Standard random number generators do not provide a sufficient amount of entropy when used for security purposes. Attackers can brute force the output of pseudorandom number generators such as rand().

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

If this random number is used where security is a concern, such as generating a session identifier or cryptographic key, use a trusted cryptographic random number generator instead.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
 267	62	-	WebGoat-6.0.1.war	org/.../SessionFixation.java 797	10/7/16
 178	82	-	WebGoat-6.0.1.war	org/.../lessons/WeakSessionID.java 77	10/7/16

→ Use of a Broken or Risky Cryptographic Algorithm (CWE ID 327)(3 flaws)

 **Fix Required by Policy**

Description

The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the disclosure of sensitive information.

Effort to Fix: 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
	172	19	-	WebGoat-6.0.1.war	org/.../lessons/Encoding.java 325	10/7/16
	191	19	-	WebGoat-6.0.1.war	org/.../lessons/Encoding.java 370	10/7/16
	286	19	-	WebGoat-6.0.1.war	org/.../lessons/Encoding.java 465	10/7/16

→ Directory Traversal(3 flaws)

Description

Allowing user input to control paths used in filesystem operations may enable an attacker to access or modify otherwise protected system resources that would normally be inaccessible to end users. In some cases, the user-provided input may be passed directly to the filesystem operation, or it may be concatenated to one or more fixed strings to construct a fully-qualified path.

When an application improperly cleanses special character sequences in user-supplied filenames, a path traversal (or directory traversal) vulnerability may occur. For example, an attacker could specify a filename such as "../../../etc/passwd", which resolves to a file outside of the intended directory that the attacker would not normally be authorized to view.

Recommendations

Assume all user-supplied input is malicious. Validate all user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible. When using black lists, be sure that the sanitizing routine performs a sufficient number of iterations to remove all instances of disallowed characters and ensure that the end result is not dangerous.

Associated Flaws by CWE ID:

→ External Control of File Name or Path (CWE ID 73)(3 flaws)

Description

This call contains a path manipulation flaw. The argument to the function is a filename constructed using user-supplied input. If an attacker is allowed to specify all or part of the filename, it may be possible to gain unauthorized access to files on the server, including those outside the webroot, that would be normally be inaccessible to end users. The level of exposure depends on the effectiveness of input validation routines, if any.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Validate all user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible. When using black lists, be sure that the sanitizing routine performs a sufficient number of iterations to remove all instances of disallowed characters.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
220	7	-	WebGoat-6.0.1.war	org/.../CommandInjection.java 142	
139	7	-	WebGoat-6.0.1.war	org/.../CommandInjection.java 149	
180	52	-	WebGoat-6.0.1.war	.../PathBasedAccessControl.java 128	

→ Encapsulation(2 flaws)

Description

Encapsulation is about defining strong security boundaries governing data and processes. Within an application, it might mean differentiation between validated and unvalidated data, between public and private members, or between one user's data and another's.

In object-oriented programming, the term encapsulation is used to describe the grouping together of data and functionality within an object and the ability to provide users with a well-defined interface in a way which hides their internal workings. Though there is some overlap with the above definition, the two definitions should not be confused as being interchangeable.

Recommendations

The wide variance of encapsulation issues makes it impractical to generalize how these issues should be addressed, beyond stating that encapsulation boundaries should be well-defined and adhered to. Refer to individual categories for specific recommendations.

Associated Flaws by CWE ID:

→ Trust Boundary Violation (CWE ID 501)(2 flaws)

Description

A trust boundary violation occurs when a program blurs the line between what is trusted and what is untrusted. This application mixes trusted and untrusted data in the same data structure. By doing so, it becomes easier for programmers to mistakenly trust unvalidated data. Without well-established and maintained trust boundaries, programmers will inevitably lose track of which pieces of data have been validated and which have not. This confusion will eventually allow some data to be used without first being validated. A common manifestation of this flaw is in J2EE application, when a Session object is used to store untrusted data from the HTTP request.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Avoid storing untrusted data alongside trusted data in the same data structure. Establish and maintain trust boundaries to avoid losing track of which pieces of data have been validated and which have not.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
319	10	-	WebGoat-6.0.1.war	.../DefaultLessonAction.java 89	
235	25	-	WebGoat-6.0.1.war	org/.../webgoat/HammerHead.java 390	

→ Insufficient Input Validation(3 flaws)

Description

Weaknesses in this category are related to an absent or incorrect protection mechanism that fails to properly validate input that can affect the control flow or data flow of a program.

Recommendations

Validate input from untrusted sources before it is used. The untrusted data sources may include HTTP requests, file systems, databases, and any external systems that provide data to the application. In the case of HTTP requests, validate all parts of the request, including headers, form fields, cookies, and URL components that are used to transfer information from the browser to the server side application.

Duplicate any client-side checks on the server side. This should be simple to implement in terms of time and difficulty, and will greatly reduce the likelihood of insecure parameter values being used in the application.

Associated Flaws by CWE ID:

→ URL Redirection to Untrusted Site ('Open Redirect') (CWE ID 601)(1 flaw)

 **Fix Required by Policy**

Description

A web application accepts a user-controlled input that specifies a link to an external site, and uses that link to generate a redirect. This enables phishing attacks.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Always validate user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible. Check the supplied URL against a whitelist of approved URLs or domains before redirecting.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
	328	53	-	WebGoat-6.0.1.war	/lessons/.../General/redirect.jsp 10	10/7/16

→ Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') (CWE ID 470)(2 flaws)

Description

A call uses reflection in an unsafe manner. An attacker can specify the class name to be instantiated, which may create unexpected control flow paths through the application. Depending on how reflection is being used, the attack vector may allow the attacker to bypass security checks or otherwise cause the application to behave in an unexpected manner. Even if the object does not implement the specified interface and a `ClassCastException` is thrown, the constructor of the user-supplied class name will have already executed.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Validate the class name against a combination of white and black lists to ensure that only expected behavior is produced.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
75	4	-	WebGoat-6.0.1.war	org/.../lessons/BlindScript.java 243	
96	9	-	WebGoat-6.0.1.war	org/.../DatabaseUtilities.java 93	

→ Session Fixation(1 flaw)

Description

Authenticating a user without invalidating any existing session identifier gives an attacker the opportunity to steal authenticated sessions. Session fixation vulnerabilities occur when:

- * A web application authenticates a user without first invalidating the existing session ID, thereby continuing to use the session ID already associated with the user.
- * An attacker is able to force a known session ID on a user so that, once the user authenticates, the attacker has access to the authenticated session.

In the generic exploit of session fixation vulnerabilities, an attacker creates a new session on a web application and records the associated session identifier. The attacker then causes the victim to authenticate against the server using the same session identifier, giving the attacker access to the user's account through the active session. A similar, passive version of this attack could be carried out by sniffing the session identifier at some point between the victim and the server prior to authentication.

Failing to destroy a session once a user has logged out, or failing to provide a mechanism for logging out of the application, is another form of session fixation.

Recommendations

Invalidate any existing session after the user has authenticated and issue a new session identifier. Also, invalidate the session object when a user logs out, otherwise the session will remain valid on the server.

Associated Flaws by CWE ID:

→ Session Fixation (CWE ID 384)(1 flaw)

Description

The application never invalidates user sessions, which can lead to session fixation attacks. As a result, the session identifier stays the same before, during, and after a user has logged in or out. An attacker may attempt to force a user into using a specific session identifier, then hijack the session once the user has logged in.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Invalidate any existing session after the user has authenticated but before calling methods that establish the UserPrincipal. Also, invalidate the session object when a user logs out, otherwise the session will remain valid on the server.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
240	85	-	WebGoat-6.0.1.war	org/.../controller/Welcome.java 36	

Low (22 flaws)

→ API Abuse(3 flaws)

Description

An API is a contract between a caller and a callee. Incorrect usage of certain API functions can result in exploitable security vulnerabilities.

The most common forms of API abuse are caused by the caller failing to honor its end of this contract. For example, if a program fails to call `chdir()` after calling `chroot()`, it violates the contract that specifies how to change the active root directory in a secure fashion. Providing too few arguments to a varargs function such as `printf()` also violates the API contract and will cause the missing parameters to be populated with unexpected data from the stack.

Another common mishap is when the caller makes false assumptions about the callee's behavior. One example of this is when a caller expects a DNS-related function to return trustworthy information that can be used for authentication purposes. This is a bad assumption because DNS responses can be easily spoofed.

Recommendations

When calling API functions, be sure to fully understand and adhere to the specifications to avoid introducing security vulnerabilities. Do not make assumptions about trustworthiness of the data returned from API calls or use the data in a context that was unintended by that API.

Associated Flaws by CWE ID:

→ J2EE Bad Practices: Direct Management of Connections (CWE ID 245)(3 flaws)

Description

The J2EE application directly manages connections rather than using the container's resource management facilities to obtain connections as specified in the J2EE standard. Every major web application container provides pooled database connection management as part of its resource management framework. Duplicating this functionality in an application is difficult and error prone, which is part of the reason it is forbidden under the J2EE standard.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Request the connection from the container rather than attempting to access it directly.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
49	9	-	WebGoat-6.0.1.war	org/.../DatabaseUtilities.java 100	

Flaw Id	Module #	Class #	Module	Location	Fix By
66	9	-	WebGoat-6.0.1.war	org/.../DatabaseUtilities.java 112	
118	72	-	WebGoat-6.0.1.war	org/.../session/UserDatabase.java 43	

→ Code Quality(4 flaws)

Description

Code quality issues stem from failure to follow good coding practices and can lead to unpredictable behavior. These may include but are not limited to:

- * Neglecting to remove debug code or dead code
- * Improper resource management, such as using a pointer after it has been freed
- * Using the incorrect operator to compare objects
- * Failing to follow an API or framework specification
- * Using a language feature or API in an unintended manner

While code quality flaws are generally less severe than other categories and usually are not directly exploitable, they may serve as indicators that developers are not following practices that increase the reliability and security of an application. For an attacker, code quality issues may provide an opportunity to stress the application in unexpected ways.

Recommendations

The wide variance of code quality issues makes it impractical to generalize how these issues should be addressed. Refer to individual categories for specific recommendations.

Associated Flaws by CWE ID:

→ Use of Wrong Operator in String Comparison (CWE ID 597)(4 flaws)

 **Fix Required by Policy**

Description


Using '=' to compare two strings for equality or '!=' for inequality actually compares the object references rather than their values. It is unlikely that this reflects the intended application logic.

Effort to Fix: 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

Recommendations

Use the equals() method to compare strings, not the '==' or '!=' operator

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
 351	81	-	WebGoat-6.0.1.war	.../WeakAuthenticationCookie.java 141	10/7/16
 133	88	-	WebGoat-6.0.1.war	org/.../lessons/XMLInjection.java 234	10/7/16
 113	88	-	WebGoat-6.0.1.war	org/.../lessons/XMLInjection.java 236	10/7/16
 125	88	-	WebGoat-6.0.1.war	org/.../lessons/XMLInjection.java 246	10/7/16

→ Cryptographic Issues(5 flaws)

Description

Applications commonly use cryptography to implement authentication mechanisms and to ensure the confidentiality and integrity of sensitive data, both in transit and at rest. The proper and accurate implementation of cryptography is extremely critical to its efficacy. Configuration or coding mistakes as well as incorrect assumptions may negate a large degree of the protection it affords, leaving the crypto implementation vulnerable to attack.

Common cryptographic mistakes include, but are not limited to, selecting weak keys or weak cipher modes, unintentionally exposing sensitive cryptographic data, using predictable entropy sources, and mismanaging or hard-coding keys.

Developers often make the dangerous assumption that they can improve security by designing their own cryptographic algorithm; however, one of the basic tenets of cryptography is that any cipher whose effectiveness is reliant on the secrecy of the algorithm is fundamentally flawed.

Recommendations

Select the appropriate type of cryptography for the intended purpose. Avoid proprietary encryption algorithms as they typically rely on "security through obscurity" rather than sound mathematics. Select key sizes appropriate for the data being protected; for high assurance applications, 256-bit symmetric keys and 2048-bit asymmetric keys are sufficient. Follow best practices for key storage, and ensure that plaintext data and key material are not inadvertently exposed.

Associated Flaws by CWE ID:

→ Sensitive Cookie in HTTPS Session Without 'Secure' Attribute (CWE ID 614)(5 flaws)

Description

Setting the Secure attribute on an HTTP cookie instructs the web browser to send it only over a secure channel, such as an SSL connection. Issuing a cookie without the Secure attribute allows the browser to transmit it over unencrypted connections, which are susceptible to eavesdropping. It is particularly important to set the Secure attribute on any cookies containing sensitive data, such as authentication information (e.g. "remember me" style functionality).

Effort to Fix: 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

Recommendations

Set the Secure attribute for all cookies used by HTTPS sessions.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
3	6	-	WebGoat-6.0.1.war	org/.../Challenge2Screen.java 172	
130	6	-	WebGoat-6.0.1.war	org/.../Challenge2Screen.java 195	
26	81	-	WebGoat-6.0.1.war	.../WeakAuthenticationCookie.java 145	
116	82	-	WebGoat-6.0.1.war	org/.../lessons/WeakSessionID.java 200	
169	84	-	WebGoat-6.0.1.war	org/.../session/WebSession.java 276	

→ Information Leakage(10 flaws)

Description

An information leak is the intentional or unintentional disclosure of information that is either regarded as sensitive within the product's own functionality or provides information about the product or its environment that could be useful in an attack. Information leakage issues are commonly overlooked because they cannot be used to directly exploit the application. However, information leaks should be viewed as building blocks that an attacker uses to carry out other, more complicated attacks.

There are many different types of problems that involve information leaks, with severities that can range widely depending on the type of information leaked and the context of the information with respect to the application. Common sources of information leakage include, but are not limited to:

- * Source code disclosure
- * Browsable directories
- * Log files or backup files in web-accessible directories
- * Unfiltered backend error messages
- * Exception stack traces
- * Server version information
- * Transmission of uninitialized memory containing sensitive data

Recommendations

Configure applications and servers to return generic error messages and to suppress stack traces from being displayed to end users. Ensure that errors generated by the application do not provide insight into specific backend issues.

Remove all backup files, binary archives, alternate versions of files, and test files from web-accessible directories of production servers. The only files that should be present in the application's web document root are files required by the application. Ensure that deployment procedures include the removal of these file types by an administrator. Keep web and application servers fully patched to minimize exposure to publicly-disclosed information leakage vulnerabilities.

Associated Flaws by CWE ID:

→ Information Exposure Through Sent Data (CWE ID 201)(10 flaws)

Description

Sensitive information may be exposed as a result of outbound network connections made by the application. This can manifest in a couple of different ways.

In C/C++ applications, sometimes the developer fails to zero out a buffer before populating it with data. This can cause information leakage if, for example, the buffer contains a data structure for which only certain fields were populated. The uninitialized fields would contain whatever data is present at that memory location. Sensitive information from previously allocated variables could then be leaked when the buffer is sent over the network.

Mobile applications may also transmit sensitive information such as email or SMS messages, address book entries, GPS location data, and anything else that can be accessed by the mobile API. This behavior is common in mobile spyware applications designed to exfiltrate data to a listening post or other data collection point. This flaw is categorized as low severity because it only impacts confidentiality, not integrity or availability. However, in the context of a mobile application, the significance of an information leak may be much greater, especially if misaligned with user expectations or data privacy policies.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

In C/C++ applications, ensure that all struct elements are initialized or zeroed before being sent. In mobile applications, ensure that the transfer of sensitive data is intended and that it does not violate application security policy or user expectations.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
35	4	-	WebGoat-6.0.1.war	org/.../lessons/BlindScript.java 179	
63	29	-	WebGoat-6.0.1.war	org/.../lessons/JSONInjection.java 80	
159	30	-	WebGoat-6.0.1.war	/lesson_content.jsp 25	
229	32	-	WebGoat-6.0.1.war	org/.../webgoat/LessonSource.java 182	
74	48	-	WebGoat-6.0.1.war	/main.jsp 230	
353	48	-	WebGoat-6.0.1.war	/main.jsp 250	
295	48	-	WebGoat-6.0.1.war	/main.jsp 265	
215	54	-	WebGoat-6.0.1.war	/reportBug.jsp 48	
54	54	-	WebGoat-6.0.1.war	/reportBug.jsp 55	
205	88	-	WebGoat-6.0.1.war	org/.../lessons/XMLInjection.java 112	

Very Low (0 flaws)

No flaws of this type were found

Info (1 flaw)

→ Potential Backdoor(1 flaw)

Description

Application backdoors are modifications to programs that are designed to bypass security mechanisms or inject malicious functionality. Backdoors are often inserted by rogue developers with legitimate access to the source code or distribution binaries. Backdoors can take many forms, such as hard-coded credentials, "easter egg" style functionality, rootkits, or time bombs, among others.

Recommendations

Investigate all potential backdoors thoroughly to ensure there is no undesirable functionality. If the backdoors are real, eliminate them, and initiate a broader effort to inspect the entire codebase for malicious code. This may require a detailed review of all code, as it is possible to hide a serious attack in only one or two lines of code. These lines may be located almost anywhere in an application and may have been intentionally obfuscated by the attacker.

Associated Flaws by CWE ID:

→ Reliance on Security Through Obscurity (CWE ID 656)(1 flaw)

Description

The strength of a security mechanism depends heavily on its obscurity, such that knowledge of its algorithms or key data is sufficient to allow the mechanism to be compromised.

Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

Instances found via Static Scan

Flaw Id	Module #	Class #	Module	Location	Fix By
249	19	-	WebGoat-6.0.1.war	org/.../lessons/Encoding.java 335	

About Veracode's Methodology

The Veracode platform uses static and dynamic analysis (for web applications) to inspect executables and identify security flaws in your applications. Using both static and dynamic analysis helps reduce false negatives and detect a broader range of security flaws. The static binary analysis engine models the binary executable into an intermediate representation, which is then verified for security flaws using a set of automated security scans. Dynamic analysis uses an automated penetration testing technique to detect security flaws at runtime. Once the automated process is complete, a security technician verifies the output to ensure the lowest false positive rates in the industry. The end result is an accurate list of security flaws for the classes of automated scans applied to the application.

Veracode Rating System Using Multiple Analysis Techniques

Higher assurance applications require more comprehensive analysis to accurately score their security quality. Because each analysis technique (automated static, automated dynamic, manual penetration testing or manual review) has differing false negative (FN) rates for different types of security flaws, any single analysis technique or even combination of techniques is bound to produce a certain level of false negatives. Some false negatives are acceptable for lower business critical applications, so a less expensive analysis using only one or two analysis techniques is acceptable. At higher business criticality the FN rate should be close to zero, so multiple analysis techniques are recommended.

Application Security Policies

The Veracode platform allows an organization to define and enforce a uniform application security policy across all applications in its portfolio. The elements of an application security policy include the target Veracode Level for the application; types of flaws that should not be in the application (which may be defined by flaw severity, flaw category, CWE, or a common standard including OWASP, CWE/SANS Top 25, or PCI); minimum Veracode security score; required scan types and frequencies; and grace period within which any policy-relevant flaws should be fixed.

Policy constraints

Policies have three main constraints that can be applied: rules, required scans, and remediation grace periods.

Evaluating applications against a policy

When an application is evaluated against a policy, it can receive one of four assessments:

Not assessed The application has not yet had a scan published

Passed The application has passed all the aspects of the policy, including rules, required scans, and grace period.

Did not pass The application has not completed all required scans; has not achieved the target Veracode Level; or has one or more policy relevant flaws that have exceeded the grace period to fix.

Conditional pass The application has one or more policy relevant flaws that have not yet exceeded the grace period to fix.

Understand Veracode Levels

The Veracode Level (VL) achieved by an application is determined by type of testing performed on the application, and the severity and types of flaws detected. A minimum security score (defined below) is also required for each level.

There are five Veracode Levels denoted as VL1, VL2, VL3, VL4, and VL5. VL1 is the lowest level and is achieved by demonstrating that security testing, automated static or dynamic, is utilized during the SDLC. VL5 is the highest level and is achieved by performing automated and manual testing and removing all significant flaws. The Veracode Levels VL2, VL3, and VL4 form a continuum of increasing software assurance between VL1 and VL5.

For IT staff operating applications, Veracode Levels can be used to set application security policies. For deployment scenarios of different business criticality, differing VLs should be made requirements. For example, the policy for applications that handle credit card transactions, and therefore have PCI compliance requirements, should be VL5. A medium business criticality internal application could have a policy requiring VL3.

Software developers can decide which VL they want to achieve based on the requirements of their customers. Developers of software that is mission critical to most of their customers will want to achieve VL5. Developers of general purpose business software may want

to achieve VL3 or VL4. Once the software has achieved a Veracode Level it can be communicated to customers through a Veracode Report or through the Veracode Directory on the Veracode web site.

Criteria for achieving Veracode Levels

The following table defines the details to achieve each Veracode Level. The criteria for all columns: Flaw Severities Not Allowed, Flaw Categories not Allowed, Testing Required, and Minimum Score.

*Dynamic is only an option for web applications.

Veracode Level	Flaw Severities Not Allowed	Testing Required*	Minimum Score
VL5	V.High, High, Medium	Static AND Manual	90
VL4	V.High, High, Medium	Static	80
VL3	V.High, High	Static	70
VL2	V.High	Static OR Dynamic OR Manual	60
VL1		Static OR Dynamic OR Manual	

When multiple testing techniques are used it is likely that not all testing will be performed on the exact same build. If that is the case the latest test results from a particular technique will be used to calculate the current Veracode Level. After 6 months test results will be deemed out of date and will no longer be used to calculate the current Veracode Level.

Business Criticality

The foundation of the Veracode rating system is the concept that more critical applications require higher security quality scores to be acceptable risks. Less business critical applications can tolerate lower security quality. The business criticality is dictated by the typical deployed environment and the value of data used by the application. Factors that determine business criticality are: reputation damage, financial loss, operational risk, sensitive information disclosure, personal safety, and legal violations.

US. Govt. OMB Memorandum M-04-04; NIST FIPS Pub. 199

Business Criticality Description

Very High	Mission critical for business/safety of life and limb on the line
High	Exploitation causes serious brand damage and financial loss with long term business impact
Medium	Applications connected to the internet that process financial or private customer information
Low	Typically internal applications with non-critical business impact
Very Low	Applications with no material business impact

Business Criticality Definitions

Very High (BC5) This is typically an application where the safety of life or limb is dependent on the system; it is mission critical the application maintain 100% availability for the long term viability of the project or business. Examples are control software for industrial, transportation or medical equipment or critical business systems such as financial trading systems.

High (BC4) This is typically an important multi-user business application reachable from the internet and is critical that the application maintain high availability to accomplish its mission. Exploitation of high criticality applications cause serious brand damage and business/financial loss and could lead to long term business impact.

Medium (BC3) This is typically a multi-user application connected to the internet or any system that processes financial or private customer information. Exploitation of medium criticality applications typically result in material business impact resulting

in some financial loss, brand damage or business liability. An example is a financial services company's internal 401K management system.

Low (BC2) This is typically an internal only application that requires low levels of application security such as authentication to protect access to non-critical business information and prevent IT disruptions. Exploitation of low criticality applications may lead to minor levels of inconvenience, distress or IT disruption. An example internal system is a conference room reservation or business card order system.

Very Low (BC1) Applications that have no material business impact should its confidentiality, data integrity and availability be affected. Code security analysis is not required for applications at this business criticality, and security spending should be directed to other higher criticality applications.

Scoring Methodology

The Veracode scoring system, Security Quality Score, is built on the foundation of two industry standards, the Common Weakness Enumeration (CWE) and Common Vulnerability Scoring System (CVSS). CWE provides the dictionary of security flaws and CVSS provides the foundation for computing severity, based on the potential Confidentiality, Integrity and Availability impact of a flaw if exploited.

The Security Quality Score is a single score from 0 to 100, where 0 is the most insecure application and 100 is an application with no detectable security flaws. The score calculation includes non-linear factors so that, for instance, a single Severity 5 flaw is weighted more heavily than five Severity 1 flaws, and so that each additional flaw at a given severity contributes progressively less to the score.

Veracode assigns a severity level to each flaw type based on three foundational application security requirements — Confidentiality, Integrity and Availability. Each of the severity levels reflects the potential business impact if a security breach occurs across one or more of these security dimensions.

Confidentiality Impact

According to CVSS, this metric measures the impact on confidentiality if a exploit should occur using the vulnerability on the target system. At the weakness level, the scope of the Confidentiality in this model is within an application and is measured at three levels of impact -None, Partial and Complete.

Integrity Impact

This metric measures the potential impact on integrity of the application being analyzed. Integrity refers to the trustworthiness and guaranteed veracity of information within the application. Integrity measures are meant to protect data from unauthorized modification. When the integrity of a system is sound, it is fully proof from unauthorized modification of its contents.

Availability Impact

This metric measures the potential impact on availability if a successful exploit of the vulnerability is carried out on a target application. Availability refers to the accessibility of information resources. Almost exclusive to this domain are denial-of-service vulnerabilities. Attacks that compromise authentication and authorization for application access, application memory, and administrative privileges are examples of impact on the availability of an application.

Security Quality Score Calculation

The overall Security Quality Score is computed by aggregating impact levels of all weaknesses within an application and representing the score on a 100 point scale. This score does not predict vulnerability potential as much as it enumerates the security weaknesses and their impact levels within the application code.

The Raw Score formula puts weights on each flaw based on its impact level. These weights are exponential and determined by empirical analysis by Veracode's application security experts with validation from industry experts. The score is normalized to a scale of 0 to 100, where a score of 100 is an application with 0 detected flaws using the analysis technique for the application's business criticality.

Understand Severity, Exploitability, and Remediation Effort

Severity and exploitability are two different measures of the seriousness of a flaw. Severity is defined in terms of the potential impact to confidentiality, integrity, and availability of the application as defined in the CVSS, and exploitability is defined in terms of the likelihood

or ease with which a flaw can be exploited. A high severity flaw with a high likelihood of being exploited by an attacker is potentially more dangerous than a high severity flaw with a low likelihood of being exploited.

Remediation effort, also called Complexity of Fix, is a measure of the likely effort required to fix a flaw. Together with severity, the remediation effort is used to give Fix First guidance to the developer.

Veracode Flaw Severities

Veracode flaw severities are defined as follows:

Severity	Description
Very High	The offending line or lines of code is a very serious weakness and is an easy target for an attacker. The code should be modified immediately to avoid potential attacks.
High	The offending line or lines of code have significant weakness, and the code should be modified immediately to avoid potential attacks.
Medium	A weakness of average severity. These should be fixed in high assurance software. A fix for this weakness should be considered after fixing the very high and high for medium assurance software.
Low	This is a low priority weakness that will have a small impact on the security of the software. Fixing should be consideration for high assurance software. Medium and low assurance software can ignore these flaws.
Very Low	Minor problems that some high assurance software may want to be aware of. These flaws can be safely ignored in medium and low assurance software.
Informational	Issues that have no impact on the security quality of the application but which may be of interest to the reviewer.

Informational findings

Informational severity findings are items observed in the analysis of the application that have no impact on the security quality of the application but may be interesting to the reviewer for other reasons. These findings may include code quality issues, API usage, and other factors.

Informational severity findings have no impact on the security quality score of the application and are not included in the summary tables of flaws for the application.

Exploitability

Each flaw instance in a static scan may receive an exploitability rating. The rating is an indication of the intrinsic likelihood that the flaw may be exploited by an attacker. Veracode recommends that the exploitability rating be used to prioritize flaw remediation within a particular group of flaws with the same severity and difficulty of fix classification.

The possible exploitability ratings include:

Exploitability	Description
V. Unlikely	Very unlikely to be exploited
Unlikely	Unlikely to be exploited

Exploitability	Description
Neutral	Neither likely nor unlikely to be exploited.
Likely	Likely to be exploited
V. Likely	Very likely to be exploited

Note: All reported flaws found via dynamic scans are assumed to be exploitable, because the dynamic scan actually executes the attack in question and verifies that it is valid.

Effort/Complexity of Fix

Each flaw instance receives an effort/complexity of fix rating based on the classification of the flaw. The effort/complexity of fix rating is given on a scale of 1 to 5, as follows:

Effort/Complexity of Fix	Description
5	Complex design error. Requires significant redesign.
4	Simple design error. Requires redesign and up to 5 days to fix.
3	Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.
2	Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.
1	Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

Flaw Types by Severity Level

The flaw types by severity level table provides a summary of flaws found in the application by Severity and Category. The table puts the Security Quality Score into context by showing the specific breakout of flaws by severity, used to compute the score as described above. If multiple analysis techniques are used, the table includes a breakout of all flaws by category and severity for each analysis type performed.

Flaws by Severity

The flaws by severity chart shows the distribution of flaws by severity. An application can get a mediocre security rating by having a few high risk flaws or many medium risk flaws.

Flaws in Common Modules

The flaws in common modules listing shows a summary of flaws in shared dependency modules in this application. A shared dependency is a dependency that is used by more than one analyzed module. Each module is listed with the number of executables that consume it as a dependency and a summary of the impact on the application's security score of the flaws found in the dependency.

The score impact represents the amount that the application score would increase if all the flaws in the shared dependency module were fixed. This information can be used to focus remediation efforts on common modules with a higher impact on the application security score.

Only common modules that were uploaded with debug information are included in the Flaws in Common Modules listing.

Action Items

The Action Items section of the report provides guidance on the steps required to bring the application to a state where it passes its assigned policy. These steps may include fixing or mitigating flaws or performing additional scans. The section also includes best practice recommendations to improve the security quality of the application.

Common Weakness Enumeration (CWE)

The Common Weakness Enumeration (CWE) is an industry standard classification of types of software weaknesses, or flaws, that can lead to security problems. CWE is widely used to provide a standard taxonomy of software errors. Every flaw in a Veracode report is classified according to a standard CWE identifier.

More guidance and background about the CWE is available at <http://cwe.mitre.org/data/index.html>.

About Manual Assessments

The Veracode platform can include the results from a manual assessment (usually a penetration test or code review) as part of a report. These results differ from the results of automated scans in several important ways, including objectives, attack vectors, and common attack patterns.

A manual penetration assessment is conducted to observe the application code in a run-time environment and to simulate real-world attack scenarios. Manual testing is able to identify design flaws, evaluate environmental conditions, compound multiple lower risk flaws into higher risk vulnerabilities, and determine if identified flaws affect the confidentiality, integrity, or availability of the application.

Objectives

The stated objectives of a manual penetration assessment are:

- Perform testing, using proprietary and/or public tools, to determine whether it is possible for an attacker to:
- Circumvent authentication and authorization mechanisms
- Escalate application user privileges
- Hijack accounts belonging to other users
- Violate access controls placed by the site administrator
- Alter data or data presentation
- Corrupt application and data integrity, functionality and performance
- Circumvent application business logic
- Circumvent application session management
- Break or analyze use of cryptography within user accessible components
- Determine possible extent access or impact to the system by attempting to exploit vulnerabilities
- Score vulnerabilities using the Common Vulnerability Scoring System (CVSS)
- Provide tactical recommendations to address security issues of immediate consequence

Provide strategic recommendations to enhance security by leveraging industry best practices

Attack vectors

In order to achieve the stated objectives, the following tests are performed as part of the manual penetration assessment, when applicable to the platforms and technologies in use:

- Cross Site Scripting (XSS)
- SQL Injection
- Command Injection
- Cross Site Request Forgery (CSRF)
- Authentication/Authorization Bypass
- Session Management testing, e.g. token analysis, session expiration, and logout effectiveness
- Account Management testing, e.g. password strength, password reset, account lockout, etc.
- Directory Traversal
- Response Splitting
- Stack/Heap Overflows
- Format String Attacks

- Cookie Analysis
- Server Side Includes Injection
- Remote File Inclusion
- LDAP Injection
- XPATH Injection
- Internationalization attacks
- Denial of Service testing at the application layer only
- AJAX Endpoint Analysis
- Web Services Endpoint Analysis
- HTTP Method Analysis
- SSL Certificate and Cipher Strength Analysis
- Forced Browsing

CAPEC Attack Pattern Classification

The following attack pattern classifications are used to group similar application flaws discovered during manual penetration testing. Attack patterns describe the general methods employed to access and exploit the specific weaknesses that exist within an application. CAPEC (Common Attack Pattern Enumeration and Classification) is an effort led by Cigital, Inc. and is sponsored by the United States Department of Homeland Security's National Cyber Security Division.

Abuse of Functionality

Exploitation of business logic errors or misappropriation of programmatic resources. Application functions are developed to specifications with particular intentions, and these types of attacks serve to undermine those intentions.

Examples:

- Exploiting password recovery mechanisms
- Accessing unpublished or test APIs
- Cache poisoning

Spoofing

Impersonation of entities or trusted resources. A successful attack will present itself to a verifying entity with an acceptable level of authenticity.

Examples:

- Man in the middle attacks
- Checksum spoofing
- Phishing attacks

Probabilistic Techniques

Using predictive capabilities or exhaustive search techniques in order to derive or manipulate sensitive information. Attacks capitalize on the availability of computing resources or the lack of entropy within targeted components.

Examples:

- Password brute forcing
- Cryptanalysis
- Manipulation of authentication tokens

Exploitation of Authentication

Circumventing authentication requirements to access protected resources. Design or implementation flaws may allow authentication checks to be ignored, delegated, or bypassed.

Examples:

- Cross-site request forgery
- Reuse of session identifiers
- Flawed authentication protocol

Resource Depletion

Affecting the availability of application components or resources through symmetric or asymmetric consumption. Unrestricted access to computationally expensive functions or implementation flaws that affect the stability of the application can be targeted by an attacker in order to cause denial of service conditions.

Examples:

- Flooding attacks
- Unlimited file upload size
- Memory leaks

Exploitation of Privilege/Trust

Undermining the application's trust model in order to gain access to protected resources or gain additional levels of access as defined by the application. Applications that implicitly extend trust to resources or entities outside of their direct control are susceptible to attack.

Examples:

- Insufficient access control lists
- Circumvention of client side protections
- Manipulation of role identification information

Injection

Inserting unexpected inputs to manipulate control flow or alter normal business processing. Applications must contain sufficient data validation checks in order to sanitize tainted data and prevent malicious, external control over internal processing.

Examples:

- SQL Injection
- Cross-site scripting
- XML Injection

Data Structure Attacks

Supplying unexpected or excessive data that results in more data being written to a buffer than it is capable of holding. Successful attacks of this class can result in arbitrary command execution or denial of service conditions.

Examples:

- Buffer overflow
- Integer overflow
- Format string overflow

Data Leakage Attacks

Recovering information exposed by the application that may itself be confidential or may be useful to an attacker in discovering or exploiting other weaknesses. A successful attack may be conducted passive observation or active interception methods. This attack pattern often manifests itself in the form of applications that expose sensitive information within error messages.

Examples:

- Sniffing clear-text communication protocols
- Stack traces returned to end users
- Sensitive information in HTML comments

Resource Manipulation

Manipulating application dependencies or accessed resources in order to undermine security controls and gain unauthorized access to protected resources. Applications may use tainted data when constructing paths to local resources or when constructing processing environments.

Examples:

- Carriage Return Line Feed log file injection
- File retrieval via path manipulation
- User specification of configuration files

Time and State Attacks

Undermining state condition assumptions made by the application or capitalizing on time delays between security checks and performed operations. An application that does not enforce a required processing sequence or does not handle concurrency adequately will be susceptible to these attack patterns.

Examples:

- Bypassing intermediate form processing steps
- Time-of-check and time-of-use race conditions
- Deadlock triggering to cause a denial of service

Terms of Use

Use and distribution of this report are governed by the agreement between Veracode and its customer. In particular, this report and the results in the report cannot be used publicly in connection with Veracode's name without written permission.

Appendix A: Referenced Source Files

Id	Filename	Path
1	BackDoors.java	org/owasp/webgoat/lessons/
2	BaseService.java	org/owasp/webgoat/service/
3	BlindNumericSqlInjection.java	org/owasp/webgoat/lessons/
4	BlindScript.java	org/owasp/webgoat/lessons/
5	BlindStringSqlInjection.java	org/owasp/webgoat/lessons/
6	Challenge2Screen.java	org/owasp/webgoat/lessons/
7	CommandInjection.java	org/owasp/webgoat/lessons/
8	CookieService.java	org/owasp/webgoat/service/
9	DatabaseUtilities.java	org/owasp/webgoat/session/
10	DefaultLessonAction.java	org/owasp/webgoat/lessons/GoatHillsFinancial/
11	DOS_Login.java	org/owasp/webgoat/lessons/
12	ECSFactory.java	org/owasp/webgoat/session/
13	EditProfile.jsp	/lessons/CrossSiteScripting/
14	EditProfile.jsp	/lessons/DBSQLInjection/
15	EditProfile.jsp	/lessons/GoatHillsFinancial/
16	EditProfile.jsp	/lessons/DBCrossSiteScripting/
17	EditProfile.jsp	/lessons/RoleBasedAccessControl/
18	EditProfile.jsp	/lessons/SQLInjection/
19	Encoding.java	org/owasp/webgoat/lessons/
20	error.jsp	/lessons/GoatHillsFinancial/
21	error.jsp	/lessons/RoleBasedAccessControl/
22	eval.jsp	/lessons/Ajax/
23	Exec.java	org/owasp/webgoat/util/
24	GoatHillsFinancial.java	org/owasp/webgoat/lessons/GoatHillsFinancial/
25	HammerHead.java	org/owasp/webgoat/
26	HtmlClues.java	org/owasp/webgoat/lessons/
27	HttpOnly.java	org/owasp/webgoat/lessons/
28	InsecureLogin.java	org/owasp/webgoat/lessons/
29	JSONInjection.java	org/owasp/webgoat/lessons/
30	lesson_content.jsp	/
31	LessonPlanService.java	org/owasp/webgoat/service/
32	LessonSource.java	org/owasp/webgoat/
33	ListStaff.jsp	/lessons/RoleBasedAccessControl/
34	ListStaff.jsp	/lessons/DBCrossSiteScripting/
35	ListStaff.jsp	/lessons/GoatHillsFinancial/
36	ListStaff.jsp	/lessons/CrossSiteScripting/
37	ListStaff.jsp	/lessons/SQLInjection/
38	ListStaff.jsp	/lessons/DBSQLInjection/

Id	Filename	Path
39	Login.java	org/owasp/webgoat/lessons/SQLInjection/
40	Login.java	org/owasp/webgoat/lessons/GoatHillsFinancial/
41	Login.jsp	/lessons/DBCrossSiteScripting/
42	Login.jsp	/lessons/RoleBasedAccessControl/
43	Login.jsp	/lessons/SQLInjection/
44	Login.jsp	/lessons/DBSQLInjection/
45	Login.jsp	/lessons/CrossSiteScripting/
46	Login.jsp	/lessons/GoatHillsFinancial/
47	LogSpoofing.java	org/owasp/webgoat/lessons/
48	main.jsp	/
49	MultiLevelLogin1.java	org/owasp/webgoat/lessons/
50	MultiLevelLogin2.java	org/owasp/webgoat/lessons/
51	ParameterService.java	org/owasp/webgoat/service/
52	PathBasedAccessControl.java	org/owasp/webgoat/lessons/
53	redirect.jsp	/lessons/General/
54	reportBug.jsp	/
55	RestartLessonService.java	org/owasp/webgoat/service/
56	SearchStaff.jsp	/lessons/DBSQLInjection/
57	SearchStaff.jsp	/lessons/GoatHillsFinancial/
58	SearchStaff.jsp	/lessons/SQLInjection/
59	SearchStaff.jsp	/lessons/DBCrossSiteScripting/
60	SearchStaff.jsp	/lessons/CrossSiteScripting/
61	SearchStaff.jsp	/lessons/RoleBasedAccessControl/
62	SessionFixation.java	org/owasp/webgoat/lessons/
63	SilentTransactions.java	org/owasp/webgoat/lessons/
64	SolutionService.java	org/owasp/webgoat/service/
65	SourceService.java	org/owasp/webgoat/service/
66	SqlAddData.java	org/owasp/webgoat/lessons/
67	SqlModifyData.java	org/owasp/webgoat/lessons/
68	SqlNumericInjection.java	org/owasp/webgoat/lessons/
69	SqlStringInjection.java	org/owasp/webgoat/lessons/
70	ThreadSafetyProblem.java	org/owasp/webgoat/lessons/
71	UncheckedEmail.java	org/owasp/webgoat/lessons/
72	UserDatabase.java	org/owasp/webgoat/session/
73	ViewDatabase.java	org/owasp/webgoat/lessons/admin/
74	ViewProfile.java	org/owasp/webgoat/lessons/SQLInjection/
75	ViewProfile.jsp	/lessons/DBCrossSiteScripting/
76	ViewProfile.jsp	/lessons/DBSQLInjection/
77	ViewProfile.jsp	/lessons/GoatHillsFinancial/
78	ViewProfile.jsp	/lessons/RoleBasedAccessControl/

Id	Filename	Path
79	ViewProfile.jsp	/lessons/CrossSiteScripting/
80	ViewProfile.jsp	/lessons/SQLInjection/
81	WeakAuthenticationCookie.java	org/owasp/webgoat/lessons/
82	WeakSessionID.java	org/owasp/webgoat/lessons/
83	WebgoatProperties.java	org/owasp/webgoat/session/
84	WebSession.java	org/owasp/webgoat/session/
85	Welcome.java	org/owasp/webgoat/controller/
86	WsSAXInjection.java	org/owasp/webgoat/lessons/
87	WsSqlInjection.java	org/owasp/webgoat/lessons/
88	XMLInjection.java	org/owasp/webgoat/lessons/
89	XPATHInjection.java	org/owasp/webgoat/lessons/