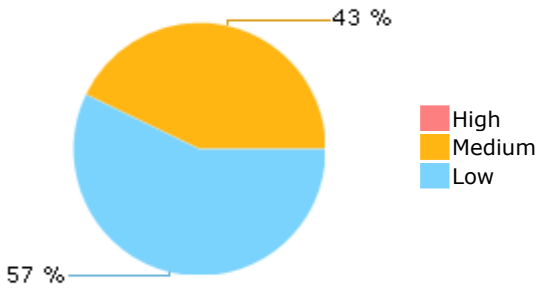
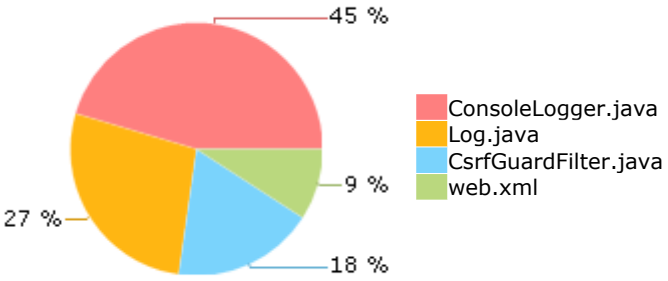
CSRFGuard-3.1 Scan Report

Project Name	CSRFGuard-3.1
Scan Start	16-Jul-2014 02:14
Preset	OWASP TOP 10 - 2013
Scan Time	00h:01m:30s
Lines Of Code Scanned	4,245
Files Scanned	53
Report Creation Time	16-Jul-2014 02:17
Scan Type	Full
Source Origin	LocalPath
Density	0/100 (Vulnerabilities/LOC)
Scan Comments	

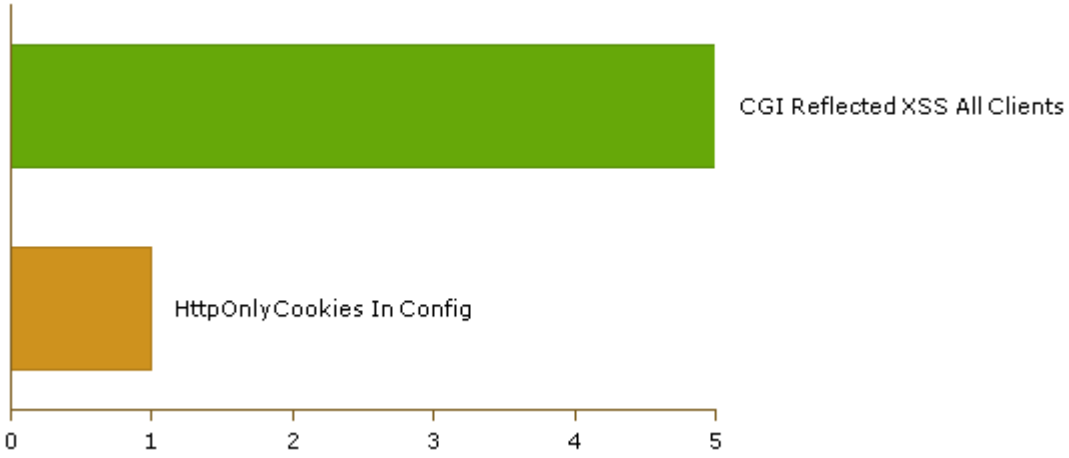
Result Summary



Most Vulnerable Files



Top 5 Vulnerabilities



Results Distribution By Status

First scan of the project

	High	Medium	Low	Information	Total
New Issues	0	6	8	7	21
Recurrent Issues	0	0	0	0	0
Total	0	6	8	7	21
Fixed Issues	0	0	0	0	0

Results Distribution By State

	High	Medium	Low	Information	Total
To Verify	0	6	8	7	21
Not Exploitable	0	0	0	0	0
Confirmed	0	0	0	0	0
Urgent	0	0	0	0	0
Total	0	6	8	7	21

Result Summary

Vulnerability Type	Occurrences	Severity
CGI Reflected XSS All Clients	5	Medium
HttpOnlyCookies In Config	1	Medium
Data Leak Between Sessions	8	Low
Pages Without Global Error Handler	4	Information
Exposure of Resource to Wrong Sphere	3	Information

10 Most Vulnerable Files

High and Medium Vulnerabilities

File Name	Issues Found
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java	5
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Log.java	3
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\CsrfGuardFilter.java	2
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\WEB-INF\web.xml	1

Scanned Files

File Name	File Size KB	Checksum
\OWASP-CSRFGuard-fix\csrfguard\pom.xml	5	9587df505c23df59d478ea6264e89eba
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\AbstractAction.java	3	00d36736e8a518cf75e90d88fd016601
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Empty.java	3	f2ad53d1aff1c44ca594c7edaf6ceabc
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Error.java	3	167b1831f2dfcbc2a6b95e7bd6b6f3b9
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Forward.java	3	0b2a8a3f32f0340279a4308e1d8ead6c
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\IAction.java	3	67f6e883c0f632ae88d9af5596d1f766
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Invalidate.java	3	8053e673b5a519cd65f1e3d59af8b75f
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Log.java	4	29557d86f9d84c90ed63b776a495b2ef
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Redirect.java	3	a7fd78381e5f6ff47af844c9189a25d6
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\RequestAttribute.java	3	07c23fac6310408fe5a87f2bab2b8dcc
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Rotate.java	4	1b67aa2855e02ada913c0e8b91e87141
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\SessionAttribute.java	3	aebb33df9db3fe9026b32595f16c1f86
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\CsrfGuard.java	23	2006ef5940bfabe9a98ae4833791c3c0
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\CsrfGuardException.java	2	9f77083a2a163793486cd6fb295c758c
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\CsrfGuardFilter.java	4	07d584142f187eb088a40b7352dbd630
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\CsrfGuardHttpSessionListener.java	1	cc08b62024b85b902aceb94f2be83bf8
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfgu	3	19e57d02d9a04064d6171d14438f0812

ard\CsrfGuardServletContextListener.java		
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\http\InterceptRedirectResponse.java	2	93707f1e9d0c4384aedd14175816f0fd
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java	3	1378f40d6e275910986cfec9eee16fbd
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ILogger.java	2	21048a7d2dfdbd01fd05bc110f3a3604
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\JavaLogger.java	2	cf98e238dc3b45947ca6d984c4a5897f
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\LogLevel.java	2	26c2553454eb8fb0c8ab6c31fc7eff23
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java	10	e19103b133aa848a903e4f83f180ad59
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\tag\AbstractTag.java	2	280dd1dd9b137c802062a6ce1c313f40
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\tag\AbstractUriTag.java	2	95fa82195c1c58f26eb8f0c4b5f0f95c
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\tag\ATag.java	4	e02236db58c0c6f74f514a3fcf237e5a
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\tag\FormTag.java	4	1438c5db5a3d9bbec2508d04b00f159c
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\tag\TokenNameTag.java	3	abf6bca38409af95adceef10b6b265af
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\tag\TokenTag.java	3	5413c9f7a614f08749bb9a538b5c9cef
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\tag\TokenValueTag.java	3	2aba60121f6b94eeda805ad705d695c5
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\util\BrowserEncoder.java	3	cb7ee1ef8022b4e4114009af37fd0d50
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\util\RandomGenerator.java	3	1ca8fdfd4a64316322d5279e49c284ad
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\util\Streams.java	3	909efbe04c6514ef94726ee3415b697c
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\util\Strings.java	1	8ff234ffc6135ab9a556a594785ff9c1
\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\util\Writers.java	3	a0a37ce2941e82bf0afd3d1d20262c2f
\OWASP-CSRFGuard-	14	331dc9208fdb6a1cd7b64a6607c53a7

fix\csrfguard\src\main\resources\csrfguard.js		d
\OWASP-CSRFGuard-fix\csrfguard\src\main\resources\csrfguard.properties	19	74b733cc89ebd6ba2aeeeda932df2a96
\OWASP-CSRFGuard-fix\csrfguard\src\main\resources\csrfguard.tld	3	5017ad375901fd7560ebe12229391e89
\OWASP-CSRFGuard-fix\csrfguard-test\pom.xml	5	6a0c68aaa4bc91daadebbb7ab8f01345
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\java\org\owasp\csrfguard\test\HelloServlet.java	2	97af48cafb75651cf71a007b8029cd94
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\ajax.html	3	863548e11f1fe395098d378db1f5186c
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\error.html	1	8e183b316793293ad0dea08b98b12efb
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\forward.jsp	1	b8e9de5a8a02b137ff3a0c0821b5943b
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\index.html	1	8af914b3014b193ee44406fae4d3c5c8
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\javascript.html	2	b37ab631fd5585ef27dd55c6d89f49c4
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\protect.html	1	9bd57e92fe3a64470b0828c9e78c23c4
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\redirect.jsp	1	f9a1c112a4b091702795c147d5c6d2b4
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\script\csrfguard.js	13	9886518cfa7971757b132b22cb8622d9
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\session.jsp	1	21588de3248a5fd3870371f8d48ed6a6
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\tag.jsp	2	7357f150c075ae4f0c08129e757c3024
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\upload.html	1	50a47bed167b9355121c577a49150967
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\WEB-INF\csrfguard.properties	18	6636413f9a2c5cc33e22ef386c5c87fe
\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\WEB-INF\web.xml	3	3249442296a865042838867ae77df697

Scanned Queries

Query Name	
Data Leak Between Sessions	8
CGI Reflected XSS All Clients	5
Pages Without Global Error Handler	4
Exposure of Resource to Wrong Sphere	3
HttpOnlyCookies In Config	1
Dynamic SQL Queries	0
Use of Obsolete Functions	0
Potentially Serializable Class With Sensitive Data	0
Use of Inner Class Containing Sensitive Data	0
GWT DOM XSS	0
GWT Reflected XSS	0

Heuristic 2nd Order SQL Injection	0
Heuristic CGI Stored XSS	0
Heuristic DB Parameter Tampering	0
Heuristic Parameter Tampering	0
Heuristic SQL Injection	0
Heuristic Stored XSS	0
Heuristic XSRF	0
Code Injection	0
Command Injection	0
Connection String Injection	0
LDAP Injection	0
Reflected XSS All Clients	0
Resource Injection	0
Second Order SQL Injection	0
SQL Injection	0
Stored XSS	0
XPath Injection	0
Blind SQL Injections	0
Sensitive Cookie in HTTPS Session Without Secure Attribute	0
Insufficiently Protected Credentials	0
Information Leak Through Persistent Cookies	0
Potential ReDoS By Injection	0
Use of RSA Algorithm without OAEP	0
Creation of Temp File With Insecure Permissions	0
Use of Client Side Authentication	0
Using Referer Field for Authentication	0
Stored Absolute Path Traversal	0
Storing Passwords in a Recoverable Format	0
Missing Password Field Masking	0
Plaintext Storage in a Cookie	0
Relative Path Traversal	0
Reversible One Way Hash	0
Serializable Class Containing Sensitive Data	0
Creation of Temp File in Dir with Incorrect Permissions	0
Channel Accessible by NonEndpoint	0
Exposure of System Data	0
Information Leak Through Comments	0
Information Exposure Through Debug Log	0
Information Exposure Through Server Log	0
Information Leak Through Shell Error Message	0
Open Redirect	0
UTF7 XSS	0
Use of Broken or Risky Cryptographic Algorithm	0
Stored Command Injection	0
Stored Relative Path Traversal	0
Authorization Bypass Through User Controlled SQL PrimaryKey	0
CGI Stored XSS	0
DB Parameter Tampering	0
External Control of System or Config Setting	0
Parameter Tampering	0
Privacy Violation	0
Unvalidated Forwards	0
Use of a One Way Hash with a Predictable Salt	0

Use of a One Way Hash without a Salt	0
Direct Use of Unsafe JNI	0
Session Fixation	0
HttpOnlyCookies	0
Multiple Binds to the Same Port	0
Plaintext Storage of a Password	0
Process Control	0
Reliance on Cookies without Validation	0
Stored LDAP Injection	0
Use of Cryptographically Weak PRNG	0
Spring ModelView Injection	0
SQL Injection Evasion Attack	0
XSRF	0
Absolute Path Traversal	0
Cleartext Submission of Sensitive Information	0
External Control of Critical State Data	0
Stored Code Injection	0
Stored Open Redirect	0
Stored XPath Injection	0
Client DOM Code Injection	0
Client DOM XSS	0
Client DOM Stored Code Injection	0
Client DOM Stored XSS	0
Client Resource Injection	0
Client Second Order Sql Injection	0
Client SQL Injection	0
Client JQuery Deprecated Symbols	0
Client DOM Open Redirect	0
Client Weak Password Authentication	0
Client Weak Cryptographic Hash	0
Client Weak Encryption	0
Client Password In Comment	0
Client Located JQuery Outdated Lib File	0
Client Use Of Deprecated SQL Database	0
Client Untrusted Activex	0
Client ReDoS From Regex Injection	0
Client DOM XSRF	0
Client HTML5 Information Exposure	0
Client HTML5 Store Sensitive data In Web Storage	0
Client Use Of JQuery Outdated Version	0
Client DB Parameter Tampering	0
DOM XSRF	0
DOM Open Redirect	0
Weak Password Authentication	0
DOM Code Injection	0
DOM XSS	0
Hardcoded password in Connection String	0
HTTP Response Splitting	0
Use of Insufficiently Random Values	0
Path Traversal	0
Reflected XSS	0

Scan Results Details

CGI Reflected XSS All Clients

Detailed Vulnerability Description

CGI Reflected XSS All Clients\Path 1:

Severity	Medium
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Log.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java
Line	61	47
Object	getRequestURI	println

Code Snippet

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Log.java

Method public void execute(HttpServletRequest request, HttpServletResponse response, CsrfGuardException csrfe, CsrfGuard csrfGuard) throws CsrfGuardException {

```
....
61.  logMessage = logMessage.replaceAll("%request_uri%",
request.getRequestURI());
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java

Method public void log(LogLevel level, String msg) {

```
....
47.  System.out.println(String.format("[%s] [%s] %s", new Date(),
String.valueOf(level), msg));
```

CGI Reflected XSS All Clients\Path 2:

Severity	Medium
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Log.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java

Line	62	47
Object	getRequestURL	println

Code Snippet

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Log.java

Method public void execute(HttpServletRequest request, HttpServletResponse response, CsrfGuardException csrfe, CsrfGuard csrfGuard) throws CsrfGuardException {

```
....
62.  logMessage = logMessage.replaceAll("%request_url%",
request.getRequestURL().toString());
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java

Method public void log(LogLevel level, String msg) {

```
....
47.  System.out.println(String.format("[%s] [%s] %s", new Date(),
String.valueOf(level), msg));
```

CGI Reflected XSS All Clients\Path 3:

Severity	Medium
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Log.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java
Line	66	47
Object	getRemoteUser	println

Code Snippet

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\action\Log.java

Method public void execute(HttpServletRequest request, HttpServletResponse response, CsrfGuardException csrfe, CsrfGuard csrfGuard) throws CsrfGuardException {

```
....
66.  logMessage = logMessage.replaceAll("%user%",
request.getRemoteUser());
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java

Method public void log(LogLevel level, String msg) {

```
....
47.  System.out.println(String.format("[%s] [%s] %s", new Date(),
String.valueOf(level), msg));
```

CGI Reflected XSS All Clients\Path 4:

Severity	Medium
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\CsrfGuardFilter.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java
Line	68	47
Object	getRequestURI	println

Code Snippet

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\CsrfGuardFilter.java

Method public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) throws IOException, ServletException {

```
....
68.  csrfGuard.getLogger().log(String.format("CsrfGuard analyzing
request %s", httpRequest.getRequestURI()));
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java

Method public void log(LogLevel level, String msg) {

```
....
47.  System.out.println(String.format("[%s] [%s] %s", new Date(),
String.valueOf(level), msg));
```

CGI Reflected XSS All Clients\Path 5:

Severity	Medium
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\CsrfGuardFilter.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java
Line	68	61
Object	getRequestURI	println

Code Snippet	
File Name	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\CsrfGuardFilter.java
Method	public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) throws IOException, ServletException {

```
....
68.  csrfGuard.getLogger().log(String.format("CsrfGuard analyzing
request %s", httpRequest.getRequestURI()));
```

File Name	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\log\ConsoleLogger.java
Method	public void log(LogLevel level, Exception exception) {

```
....
61.  System.out.println(String.format("[%s] [%s] %s", new Date(),
String.valueOf(level), String.valueOf(exception)));
```

HttpOnlyCookies In Config

Detailed Vulnerability Description

HttpOnlyCookies In Config\Path 1:

Severity	Medium
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\WEB-INF\web.xml	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\WEB-INF\web.xml
Line	1	1
Object	CxXmlConfigClass299943797	CxXmlConfigClass299943797

Code Snippet	
File Name	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\WEB-INF\web.xml
Method	<?xml version="1.0" encoding="UTF-8"?>

```
....
1.  <?xml version="1.0" encoding="UTF-8"?>
```

Data Leak Between Sessions

Detailed Vulnerability Description

Data Leak Between Sessions\Path 1:

Severity	Low
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Line	66	40
Object	templateCode	JavaScriptServlet

Code Snippet
File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method private String templateCode = null;

```
....
66.  private String templateCode = null;
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method public final class JavaScriptServlet extends HttpServlet {

```
....
40.  public final class JavaScriptServlet extends HttpServlet {
```

Data Leak Between Sessions\Path 2:

Severity Low
Result State To Verify
Result Comment
Status New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Line	68	40
Object	sourceFile	JavaScriptServlet

Code Snippet
File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method private String sourceFile = null;

```
....
68.  private String sourceFile = null;
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method public final class JavaScriptServlet extends HttpServlet {

```
....
40.  public final class JavaScriptServlet extends HttpServlet {
```

Data Leak Between Sessions\Path 3:

Severity	Low
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Line	70	40
Object	injectIntoForms	JavaScriptServlet

Code Snippet

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java

Method private String injectIntoForms = null;

```
....
70.  private String injectIntoForms = null;
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java

Method public final class JavaScriptServlet extends HttpServlet {

```
....
40.  public final class JavaScriptServlet extends HttpServlet {
```

Data Leak Between Sessions\Path 4:

Severity	Low
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Line	72	40
Object	injectIntoAttributes	JavaScriptServlet

Code Snippet

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method private String injectIntoAttributes = null;

```
....
72.  private String injectIntoAttributes = null;
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method public final class JavaScriptServlet extends HttpServlet {

```
....
40.  public final class JavaScriptServlet extends HttpServlet {
```

Data Leak Between Sessions\Path 5:

Severity Low
Result State To Verify
Result Comment
Status New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Line	74	40
Object	domainStrict	JavaScriptServlet

Code Snippet
File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method private String domainStrict = null;

```
....
74.  private String domainStrict = null;
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method public final class JavaScriptServlet extends HttpServlet {

```
....
40.  public final class JavaScriptServlet extends HttpServlet {
```

Data Leak Between Sessions\Path 6:

Severity Low
Result State To Verify
Result Comment
Status New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Line	76	40
Object	cacheControl	JavaScriptServlet

Code Snippet
File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method private String cacheControl = null;

```
....
76.  private String cacheControl = null;
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method public final class JavaScriptServlet extends HttpServlet {

```
....
40.  public final class JavaScriptServlet extends HttpServlet {
```

Data Leak Between Sessions\Path 7:

Severity Low
Result State To Verify
Result Comment
Status New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Line	78	40
Object	refererPattern	JavaScriptServlet

Code Snippet
File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method private Pattern refererPattern = null;

```
....
78.  private Pattern refererPattern = null;
```

File Name \OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java

Method	public final class JavaScriptServlet extends HttpServlet {

```
....
40.  public final class JavaScriptServlet extends HttpServlet {
```

Data Leak Between Sessions\Path 8:

Severity	Low
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Line	80	40
Object	xRequestedWith	JavaScriptServlet

Code Snippet

File Name	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method	private String xRequestedWith = null;

```
....
80.  private String xRequestedWith = null;
```

File Name	\OWASP-CSRFGuard-fix\csrfguard\src\main\java\org\owasp\csrfguard\servlet\JavaScriptServlet.java
Method	public final class JavaScriptServlet extends HttpServlet {

```
....
40.  public final class JavaScriptServlet extends HttpServlet {
```

Pages Without Global Error Handler

Detailed Vulnerability Description

Pages Without Global Error Handler\Path 1:

Severity	Information
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\forward.jsp	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\forward.jsp
Line	1	1
Object	forward_jsp	forward_jsp

Code Snippet
File Name \OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\forward.jsp
Method <%@ page language="java" contentType="text/html; charset=ISO-8859-1"

```
....
1.  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

Pages Without Global Error Handler\Path 2:

Severity Information
Result State To Verify
Result Comment
Status New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\redirect.jsp	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\redirect.jsp
Line	1	1
Object	redirect_jsp	redirect_jsp

Code Snippet
File Name \OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\redirect.jsp
Method <%@ page language="java" contentType="text/html; charset=ISO-8859-1"

```
....
1.  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

Pages Without Global Error Handler\Path 3:

Severity Information
Result State To Verify
Result Comment
Status New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\session.jsp	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\session.jsp
Line	1	1
Object	session_jsp	session_jsp

Code Snippet
File Name \OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\session.jsp
Method <%@ page language="java" contentType="text/html; charset=ISO-8859-1"

```
....
1.  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

Pages Without Global Error Handler\Path 4:

Severity Information

Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\tag.jsp	\OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\tag.jsp
Line	1	1
Object	tag_jsp	tag_jsp

Code Snippet
File Name \OWASP-CSRFGuard-fix\csrfguard-test\src\main\webapp\tag.jsp
Method <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

```
....
1.  <%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
```

Exposure of Resource to Wrong Sphere

Detailed Vulnerability Description

Exposure of Resource to Wrong Sphere\Path 1:

Severity	Information
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File		
Line	4	4
Object	actionErrors	actionErrors

Code Snippet
File Name
Method

```
....
4.
```

Exposure of Resource to Wrong Sphere\Path 2:

Severity	Information
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File		

Line	5	5
Object	actionMessages	actionMessages

Code Snippet
File Name
Method

```
....
5.
```

Exposure of Resource to Wrong Sphere\Path 3:

Severity	Information
Result State	To Verify
Result Comment	
Status	New

	Source	Destination
File		
Line	6	6
Object	fieldErrors	fieldErrors

Code Snippet
File Name
Method

```
....
6.
```

Failure to Preserve Web Page Structure ('Cross-site Scripting')

Weakness ID: 79 *(Weakness Base)* **Status:** Usable

Description

Description Summary

The software does not sufficiently validate, filter, escape, and/or encode user-controllable input before it is placed in output that is used as a web page that is served to other users.

Extended Description

Cross-site scripting (XSS) vulnerabilities occur when:

1. Untrusted data enters a web application, typically from a web request.

2. The web application dynamically generates a web page that contains this untrusted data.

3. During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTML tags, HTML attributes, mouse events, Flash, ActiveX, etc.

4. A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data.

5. Since the script comes from a web page that was sent by the web server, the victim's web browser executes the malicious script in the context of the web server's domain.

6. This effectively violates the intention of the web browser's same-origin policy, which

states that scripts in one domain should not be able to access resources or run code in a different domain.

There are three main kinds of XSS:

Type 1: Reflected XSS (or Non-Persistent)

The server reads data directly from the HTTP request and reflects it back in the HTTP response. Reflected XSS exploits occur when an attacker causes a victim to supply dangerous content to a vulnerable web application, which is then reflected back to the victim and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces a victim to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the victim, the content is executed by the victim's browser.

Type 2: Stored XSS (or Persistent)

The application stores dangerous data in a database, message forum, visitor log, or other trusted data store. At a later time, the dangerous data is subsequently read back into the application and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user. For example, the attacker might inject XSS into a log message, which might not be handled properly when an administrator views the logs.

Type 0: DOM-Based XSS

In DOM-based XSS, the client performs the injection of XSS into the page; in the other types, the server performs the injection. DOM-based XSS generally involves server-controlled, trusted script that is sent to the client, such as Javascript that performs sanity checks on a form before the user submits it. If the server-supplied script processes user-supplied data and then injects it back into the web page (such as with dynamic HTML), then DOM-based XSS is possible.

Once the malicious script is injected, the attacker can perform a variety of malicious activities. The attacker could transfer private information, such as cookies that may include session information, from the victim's machine to the attacker. The attacker could send malicious requests to a web site on behalf of the victim, which could be especially dangerous to the site if the victim has administrator privileges to manage that site. Phishing attacks could be used to emulate trusted web sites and trick the victim into entering a password, allowing the attacker to compromise the victim's account on that web site. Finally, the script could exploit a vulnerability in the web browser itself possibly taking over the victim's machine, sometimes referred to as "drive-by hacking."

In many cases, the attack can be launched without the victim even being aware of it. Even with careful users, attackers frequently use a variety of methods to encode the malicious portion of the attack, such as URL encoding or Unicode, so the request looks less suspicious.

Alternate Terms

XSS

CSS: "CSS" was once used as the acronym for this problem, but this could cause confusion with "Cascading Style Sheets," so usage of this acronym has declined significantly.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

Language-independent

Architectural Paradigms

Web-based: *(Often)*

Technology Classes

Web-Server: *(Often)*

Platform Notes

XSS flaws are very common in web applications since they require a great deal of developer discipline to avoid them.

Common Consequences

Scope	Effect
Confidentiality	The most common attack performed with cross-site scripting involves the disclosure of information stored in user cookies. Typically, a malicious user will craft a client-side script, which -- when parsed by a web browser -- performs some activity (such as sending all site cookies to a given E-mail address). This script will be loaded and run by each user visiting the web site. Since the site requesting to run the script has access to the cookies in question, the malicious script does also.
Access Control	In some circumstances it may be possible to run arbitrary code on a victim's computer when cross-site scripting is combined with other flaws.
Confidentiality Integrity Availability	The consequence of an XSS attack is the same regardless of whether it is stored or reflected. The difference is in how the payload arrives at the server. XSS can cause a variety of problems for the end user that range in severity from an annoyance to complete account compromise. Some cross-site scripting vulnerabilities can be exploited to manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on the end user systems for a variety of nefarious purposes. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirecting the user to some other page or site, running "Active X" controls (under Microsoft Internet Explorer) from sites that a user perceives as trustworthy, and modifying presentation of content.

Likelihood of Exploit

High to Very High

Enabling Factors for Exploitation

Cross-site scripting attacks may occur anywhere that possibly malicious users are allowed to post unregulated material to a trusted web site for the consumption of other valid users, commonly on places such as bulletin-board web sites which provide web based mailing list-style functionality.

Stored XSS got its start with web sites that offered a "guestbook" to visitors. Attackers would include JavaScript in their guestbook entries, and all subsequent visitors to the guestbook page would execute the malicious code. As the examples demonstrate, XSS vulnerabilities are caused by code that includes unvalidated data in an HTTP response.

Detection Methods

Automated Static Analysis

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible, especially when multiple components are involved.

Effectiveness: Moderate

Black Box

Use the XSS Cheat Sheet [REF-14] or automated test-generation tools to help launch a wide variety of attacks against your web application. The Cheat Sheet contains many subtle XSS variations that are specifically targeted against weak XSS defenses.

Effectiveness: Moderate

With Stored XSS, the indirection caused by the data store can make it more difficult to find the problem. The tester must first inject the XSS string into the data store, then find the appropriate application functionality in which the XSS string is sent to other users of the application. These are two distinct steps in which the activation of the XSS can take place minutes, hours, or days after the XSS was originally injected into the data store.

Demonstrative Examples

Example 1

This example covers a Reflected XSS (Type 1) scenario.

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

(Bad Code)

Example Language: **JSP**

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

The following ASP.NET code segment reads an employee ID number from an HTTP request and displays it to the user.

(Bad Code)

Example Language: **ASP.NET**

```
...
protected System.Web.UI.WebControls.TextBox Login;
protected System.Web.UI.WebControls.Label EmployeeID;
...
EmployeeID.Text = Login.Text;
... (HTML follows) ...
<p><asp:label id="EmployeeID" runat="server" /></p>
...
```

The code in this example operates correctly if the Employee ID variable contains only standard alphanumeric text. If it has a value that includes meta-characters or source code, then the code will be executed by the web browser as it displays the HTTP response. Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use e-mail or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers.

Example 2

This example covers a Stored XSS (Type 2) scenario.

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

(Bad Code)

Example Language: **JSP**

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

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

The following ASP.NET code segment queries a database for an employee with a given employee ID and prints the name corresponding with the ID.

(Bad Code)

Example Language: **ASP.NET**

```
protected System.Web.UI.WebControls.Label EmployeeName;
...
string query = "select * from emp where id=" + eid;
sda = new SqlDataAdapter(query, conn);
sda.Fill(dt);
string name = dt.Rows[0]["Name"];
```

```
...
EmployeeName.Text = name;
```

This code can appear less dangerous because the value of name is read from a database, whose contents are apparently managed by the application. However, if the value of name originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the database, an attacker can execute malicious commands in the user's web browser.

Observed Examples

Reference	Description
CVE-2008-5080	Chain: protection mechanism failure allows XSS
CVE-2006-4308	Chain: only checks "javascript:" tag
CVE-2007-5727	Chain: only removes SCRIPT tags, enabling XSS
CVE-2008-5770	Reflected XSS using the PATH INFO in a URL
CVE-2008-4730	Reflected XSS not properly handled when generating an error message
CVE-2008-5734	Reflected XSS sent through email message.
CVE-2008-0971	Stored XSS in a security product.
CVE-2008-5249	Stored XSS using a wiki page.
CVE-2006-3568	Stored XSS in a guestbook application.
CVE-2006-3211	Stored XSS in a guestbook application using a javascript: URI in a bbcode img tag.
CVE-2006-3295	Chain: library file is not protected against a direct request (CWE-425), leading to reflected XSS.

Potential Mitigations

Phase: Architecture and Design

Strategy: Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

Phases: Implementation; Architecture and Design

Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.

For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.

Parts of the same output document may require different encodings, which will vary depending on whether the output is in the:

- HTML body
- Element attributes (such as src="XYZ")
- URIs
- JavaScript sections
- Cascading Style Sheets and style property

etc. Note that HTML Entity Encoding is only appropriate for the HTML body.

Consult the XSS Prevention Cheat Sheet [REF-16] for more details on the types of encoding and escaping that are needed.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Implementation

Use and specify a strong character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can open you up to subtle XSS attacks related to that encoding. See CWE-116 for more mitigations related to encoding/escaping.

Phase: Implementation

With Struts, you should write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Phase: Implementation

Strategy: Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."

When dynamically constructing web pages, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. All input should be validated and cleansed, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. It is common to see data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Note that proper output encoding, escaping, and quoting is the most effective solution for preventing XSS, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent XSS, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, in a chat application, the heart emoticon ("<3") would likely pass the validation step, since it is commonly used. However, it cannot be directly inserted into the web page because it contains the "<" character, which would need to be escaped or otherwise handled. In this case, stripping the "<" might reduce the risk of XSS, but it would produce incorrect behavior because the emoticon would not be recorded. This might seem to be a minor inconvenience, but it would be more important in a mathematical forum that wants to represent inequalities.

Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.

Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.

Phase: Operation

Use an application firewall that can detect attacks against this weakness. This might not catch all attacks, and it might require some effort for customization. However, it can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Background Details

Same Origin Policy

The same origin policy states that browsers should limit the resources accessible to scripts running on a given web site , or "origin", to the resources associated with that web site on the client-side, and not the client-side resources of any other sites or "origins". The goal is to prevent one site from being able to modify or read the contents of an unrelated site. Since the World Wide Web involves interactions between many sites, this policy is important for browsers to enforce.

Domain

The Domain of a website when referring to XSS is roughly equivalent to the resources associated with that website on the client-side of the connection. That is, the domain can be thought of as all resources the browser is storing for the user's interactions with this particular site.

Weakness Ordinalities

Ordinality	Description
Resultant	*(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to	Named Chain(s) this relationship pertains to

ChildOf	Weakness Class	20	Improper Input Validation	**Seven Pernicious Kingdoms (primary)700**	
ChildOf	Weakness Class	74	Failure to Sanitize Data into a Different Plane ('Injection')	**Development Concepts (primary)699 Research Concepts (primary)1000**	
ChildOf	Category	442	Web Problems	Development Concepts699	
ChildOf	Category	712	OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)	**Weaknesses in OWASP Top Ten (2007) (primary)629**	
ChildOf	Category	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	Weaknesses in OWASP Top Ten (2004)711	
ChildOf	Category	725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	**Weaknesses in OWASP Top Ten (2004) (primary)711**	
ChildOf	Category	751	2009 Top 25 - Insecure Interaction Between Components	**Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)750**	
ChildOf	Category	801	2010 Top 25 - Insecure Interaction Between Components	**Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)800**	
CanPrecede	Weakness Base	494	Download of Code Without Integrity Check	Research Concepts1000	
PeerOf	Compound Element: Composite	352	Cross-Site Request Forgery (CSRF)	Research Concepts1000	
ParentOf	Weakness Variant	80	Improper Sanitization of Script-Related HTML Tags in a Web Page (Basic XSS)	**Development Concepts (primary)699 Research Concepts (primary)1000**	
ParentOf	Weakness Variant	81	Improper Sanitization of Script in an Error Message Web Page	**Development Concepts (primary)699 Research Concepts (primary)1000**	
ParentOf	Weakness Variant	83	Improper Neutralization of Script in Attributes in a Web Page	**Development Concepts (primary)699 Research Concepts (primary)1000**	
ParentOf	Weakness Variant	84	Failure to Resolve Encoded URI Schemes in a Web Page	**Development Concepts (primary)699 Research Concepts (primary)1000**	
ParentOf	Weakness Variant	85	Doubled Character XSS Manipulations	**Development Concepts (primary)699 Research Concepts (primary)1000**	
ParentOf	Weakness Variant	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages	**Development Concepts (primary)699 Research Concepts (primary)1000**	
ParentOf	Weakness Variant	87	Failure to Sanitize Alternate XSS Syntax	**Development Concepts (primary)699 Research Concepts (primary)1000**	
MemberOf	View	635	Weaknesses Used by NVD	**Weaknesses Used by NVD (primary)635**	
CanFollow	Weakness Base	113	Failure to Sanitize CRLF Sequences in HTTP Headers ('HTTP Response Splitting')	Research Concepts1000	
CanFollow	Weakness Base	184	Incomplete Blacklist	Research Concepts1000	Incomplete Blacklist to Cross-Site Scripting692

f Causal Nature

Explicit

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Cross-site scripting (XSS)
7 Pernicious Kingdoms			Cross-site Scripting
CLASP			Cross-site scripting
OWASP Top Ten 2007	A1	Exact	Cross Site Scripting (XSS)
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A4	Exact	Cross-Site Scripting (XSS) Flaws
WASC	8		Cross-site Scripting

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version: 1.5)*
232	Exploitation of Privilege/Trust	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	

32	Embedding Scripts in HTTP Query Strings
18	Embedding Scripts in Nonscript Elements
19	Embedding Scripts within Scripts
63	Simple Script Injection
91	XSS in IMG Tags
106	Cross Site Scripting through Log Files
198	Cross-Site Scripting in Error Pages
199	Cross-Site Scripting Using Alternate Syntax
209	Cross-Site Scripting Using MIME Type Mismatch
243	Cross-Site Scripting in Attributes
244	Cross-Site Scripting via Encoded URI Schemes
245	Cross-Site Scripting Using Doubled Characters, e.g. %3C%3Cscript
246	Cross-Site Scripting Using Flash
247	Cross-Site Scripting with Masking through Invalid Characters in Identifiers

References

[REF-15] Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager and Seth Fogie. "XSS Attacks". Syngress. 2007.

[REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 2: Web-Server Related Vulnerabilities (XSS, XSRF, and Response Splitting)." Page 31. McGraw-Hill. 2010.

[REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 3: Web-Client Related Vulnerabilities (XSS)." Page 63. McGraw-Hill. 2010.

"Cross-site scripting". Wikipedia. 2008-08-26. <http://en.wikipedia.org/wiki/Cross-site_scripting>.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 13, "Web-Specific Input Issues" Page 413. 2nd Edition. Microsoft. 2002.

[REF-14] RSnake. "XSS (Cross Site Scripting) Cheat Sheet". <http://ha.ckers.org/xss.html>.

Microsoft. "Mitigating Cross-site Scripting With HTTP-only Cookies". <http://msdn.microsoft.com/en-us/library/ms533046.aspx>.

Mark Curphey, Microsoft. "Anti-XSS 3.0 Beta and CAT.NET Community Technology Preview now Live!". <http://blogs.msdn.com/cisg/archive/2008/12/15/anti-xss-3-0-beta-and-cat-net-community-technology-preview-now-live.aspx>.

"OWASP Enterprise Security API (ESAPI) Project". <http://www.owasp.org/index.php/ESAPI>.

Ivan Ristic. "XSS Defense HOWTO". <http://blog.modsecurity.org/2008/07/do-you-know-how.html>.

OWASP. "Web Application Firewall". <http://www.owasp.org/index.php/Web_Application_Firewall>.

Web Application Security Consortium. "Web Application Firewall Evaluation Criteria". <http://www.webappsec.org/projects/wafec/v1/wasc-wafec-v1.0.html>.

RSnake. "Firefox Implements httpOnly And is Vulnerable to XMLHTTPRequest". 2007-07-19.

"XMLHttpRequest allows reading HTTPOnly cookies". Mozilla. <https://bugzilla.mozilla.org/show_bug.cgi?id=380418>.

"Apache Wicket". <http://wicket.apache.org/>.

[REF-16] OWASP. "XSS (Cross Site Scripting) Prevention Cheat Sheet". <http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet>.

Content History

Submissions

Submission	Submitter	Organization	Source

Date			
	PLOVER		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci	Cigital	External
	updated Time of Introduction		
2008-08-15		Veracode	External
	Suggested OWASP Top Ten 2004 mapping		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Alternate Terms, Applicable Platforms, Background Details, Common Consequences, Description, Relationships, Other Notes, References, Taxonomy Mappings, Weakness Ordinalities		
2009-01-12	CWE Content Team	MITRE	Internal
	updated Alternate Terms, Applicable Platforms, Background Details, Common Consequences, Demonstrative Examples, Description, Detection Factors, Enabling Factors for Exploitation, Name, Observed Examples, Other Notes, Potential Mitigations, References, Relationships		
2009-03-10	CWE Content Team	MITRE	Internal
	updated Potential Mitigations		
2009-05-27	CWE Content Team	MITRE	Internal
	updated Name		
2009-07-27	CWE Content Team	MITRE	Internal
	updated Description		
2009-10-29	CWE Content Team	MITRE	Internal
	updated Observed Examples, Relationships		
2009-12-28	CWE Content Team	MITRE	Internal
	updated Demonstrative Examples, Description, Detection Factors, Enabling Factors for Exploitation, Observed Examples		
2010-02-16	CWE Content Team	MITRE	Internal
	updated Applicable Platforms, Detection Factors, Potential Mitigations, References, Relationships, Taxonomy Mappings		
2010-04-05	CWE Content Team	MITRE	Internal
	updated Description, Potential Mitigations, Related Attack Patterns		

Previous Entry Names

Change Date	Previous Entry Name
2008-04-11	Cross-site Scripting (XSS)
2009-01-12	Failure to Sanitize Directives in a Web Page (aka 'Cross-site scripting' (XSS))
2009-05-27	Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')

BACK TO TOP

HTTPOnlyCookies XSS

Compound Element ID: 10706 Status: Draft

Description

Description Summary

When a cookie is not marked with "httpOnly" can be exposed by any client-side scripting code,and thus making the application vulnerable to XSS attacks.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

ASP.NET

Technology Classes

Web-Server

Demonstrative Examples

Example:

This example in ASP.NET shows us a vulnerable configuration of httpOnlyCookies in a web.config file:

(Bad Code)

Example Language: ASP.NET

```
<configuration>
<system.web>
<httpCookies httpOnlyCookies="false">
```

Any form or a login page that requests an input and then echoes some of it back, may be susceptible to an XSS attack.

The following code is an example of an input that may expose senstive data:

(Attack)

Example Language: **HTML**

```
"<script>alert(document.cookie);</script>".
```

The following input is received. If no proper escaping is done ,the browser interprets the script and executes it, and by this revealing the user's cookie.

In case that the input received in a message borad or a forum, it might reveal sensitive information and make it public.

Attackers usually use such script code to retrieve the userâ€™s authentication token.

Potential Mitigations

Enable HttpOnlyCookies by setting the HttpOnlyCookies property of the HttpCookies object to true.
This way the cookies will be accessible only from server-side code, and not to any client-side scripting code.

Data Leak Between Sessions

Weakness ID: 488 *(Weakness Variant)* **Status:** Draft

Description

Description Summary

The product does not sufficiently enforce boundaries between the states of different sessions, causing data to be provided to, or used by, the wrong session.

Extended Description

Data can "bleed" from one session to another through member variables of singleton objects, such as Servlets, and objects from a shared pool.

In the case of Servlets, developers sometimes do not understand that, unless a Servlet implements the SingleThreadModel interface, the Servlet is a singleton; there is only one instance of the Servlet, and that single instance is used and re-used to handle multiple requests that are processed simultaneously by different threads. A common result is that developers use Servlet member fields in such a way that one user may inadvertently see another user's data. In other words, storing user data in Servlet member fields introduces a data access race condition.

Time of Introduction

- Implementation

Applicable Platforms

Languages

All

Demonstrative Examples

Example 1

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

(Bad Code)

Example Language: **Java**

```
public class GuestBook extends HttpServlet {
String name;

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

While this code will work perfectly in a single-user environment, if two users access the Servlet at approximately the same time, it is possible for the two request handler threads to interleave in the following way: Thread 1: assign "Dick" to name Thread 2: assign "Jane" to name Thread 1: print "Jane, thanks for visiting!" Thread 2: print "Jane, thanks for visiting!" Thereby showing the first user the second user's name.

Potential Mitigations

Protect the application's sessions from information leakage. Make sure that a session's data is not used or visible by other sessions.

Use a static analysis tool to scan the code for information leakage vulnerabilities (e.g. Singleton Member Field).

In multithreading environment, storing user data in Servlet member fields introduces a data access race condition. Do not use member fields to store information in the Servlet.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Weakness Class	485	Insufficient Encapsulation	**Development Concepts (primary)699** **Seven Pernicious Kingdoms (primary)700** **Research Concepts (primary)1000**
PeerOf	Weakness Base	567	Unsynchronized Access to Shared Data	Research Concepts1000

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Data Leaking Between Users

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version: 1.5)*
59	Session Credential Falsification through Prediction	
60	Reusing Session IDs (aka Session Replay)	

Content History

Submissions

Submission Date	Submitter	Organization	Source
	7 Pernicious Kingdoms		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci	Cigital	External
	updated Potential Mitigations, Time of Introduction		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Description, Relationships, Other Notes, Taxonomy Mappings		
2009-05-27	CWE Content Team	MITRE	Internal
	updated Demonstrative Examples		
2009-10-29	CWE Content Team	MITRE	Internal
	updated Description, Other Notes		

Previous Entry Names

Change Date	Previous Entry Name
2008-04-11	Data Leaking Between Users

BACK TO TOP

Failure to Use a Standardized Error Handling Mechanism

Weakness ID: 544 *(Weakness Base)* **Status:** Draft

Description

Description Summary

The software does not use a standardized method for handling errors throughout the code, which might introduce inconsistent error handling and resultant weaknesses.

Extended Description

If the application handles error messages individually, on a one-by-one basis, this is likely to result in inconsistent error handling. The causes of errors may be lost. Also, detailed information about the causes of an error may be unintentionally returned to the user.

Time of Introduction

- Architecture and Design

Potential Mitigations

Phase: Architecture and Design

define a strategy for handling errors of different severities, such as fatal errors versus basic log events. Use or create built-in language features, or an external package, that provides an easy-to-use API and define coding standards for the detection and handling of errors.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Category	388	Error Handling	**Development Concepts (primary)699**
ChildOf	Category	746	CERT C Secure Coding Section 12 - Error Handling (ERR)	**Weaknesses Addressed by the CERT C Secure Coding Standard (primary)734**
ChildOf	Weakness Class	755	Improper Handling of Exceptional Conditions	**Research Concepts (primary)1000**

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Anonymous Tool Vendor (under NDA)			
CERT C Secure Coding	ERR00-C		Adopt and implement a consistent and comprehensive error-handling policy

Content History

Submissions			
Submission Date	**Submitter**	**Organization**	**Source**
	Anonymous Tool Vendor (under NDA)		Externally Mined
Modifications			
Modification Date	**Modifier**	**Organization**	**Source**
2008-07-01	Eric Dalci	Cigital	External
	updated Potential Mitigations, Time of Introduction		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Description, Relationships, Taxonomy Mappings		
2008-10-14	CWE Content Team	MITRE	Internal
	updated Relationships		
2008-11-24	CWE Content Team	MITRE	Internal
	updated Relationships, Taxonomy Mappings		
2009-03-10	CWE Content Team	MITRE	Internal
	updated Description, Name, Relationships		
2009-10-29	CWE Content Team	MITRE	Internal
	updated Potential Mitigations, Time of Introduction		
Previous Entry Names			
Change Date	**Previous Entry Name**		
2009-03-10	Missing Error Handling Mechanism		

BACK TO TOP

Exposure of Resource to Wrong Sphere

Weakness ID: 668 *(Weakness Class)* **Status:** Draft

Description

Description Summary

The product exposes a resource to the wrong control sphere, providing unintended actors with inappropriate access to the resource.

Extended Description

Resources such as files and directories may be inadvertently exposed through mechanisms such as insecure permissions, or when a program accidentally operates on the wrong object. For example, a program may intend that private files can only be provided to a specific user. This effectively defines a control sphere that is intended to prevent attackers from accessing these private files. If the file permissions are insecure, then parties other than the user will be able to access those files.

A separate control sphere might effectively require that the user can only access the private files, but not any other files on the system. If the program does not ensure that the user is only requesting private files, then the user might be able to access other files on the system.

In either case, the end result is that a resource has been exposed to the wrong party.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Category	361	Time and State	**Development Concepts (primary)699**
ChildOf	Weakness Class	664	Improper Control of a Resource Through its Lifetime	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	8	J2EE Misconfiguration: Entity Bean Declared Remote	**Research Concepts (primary)1000**
ParentOf	Weakness Class	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	Research Concepts1000
ParentOf	Weakness Class	200	Information Exposure	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	220	Sensitive Data Under FTP Root	**Research Concepts (primary)1000**
ParentOf	Weakness Base	269	Improper Privilege Management	Research Concepts1000
ParentOf	Weakness Base	374	Mutable Objects Passed by Reference	**Research Concepts (primary)1000**
ParentOf	Weakness Base	375	Passing Mutable Objects to an Untrusted Method	**Research Concepts (primary)1000**
ParentOf	Weakness Base	377	Insecure Temporary File	**Research Concepts (primary)1000**
ParentOf	Weakness Class	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	**Research Concepts (primary)1000**
ParentOf	Weakness Base	419	Unprotected Primary Channel	**Research Concepts (primary)1000**
ParentOf	Weakness Base	420	Unprotected Alternate Channel	**Research Concepts (primary)1000**
ParentOf	Weakness Base	427	Uncontrolled Search Path Element	**Research Concepts (primary)1000**
ParentOf	Weakness Base	428	Unquoted Search Path or Element	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	491	Public cloneable() Method Without Final ('Object Hijack')	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	492	Use of Inner Class Containing Sensitive Data	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	493	Critical Public Variable Without Final Modifier	**Research Concepts**

				(primary)1000
ParentOf	Weakness Base	522	Insufficiently Protected Credentials	**Research Concepts (primary)1000**
ParentOf	Weakness Base	552	Files or Directories Accessible to External Parties	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	582	Array Declared Public, Final, and Static	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	583	finalize() Method Declared Public	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	608	Struts: Non-private Field in ActionForm Class	**Research Concepts (primary)1000**
ParentOf	Weakness Class	642	External Control of Critical State Data	**Research Concepts (primary)1000**
ParentOf	Compound Element: Composite	689	Permission Race Condition During Resource Copy	**Research Concepts (primary)1000**
ParentOf	Weakness Class	732	Incorrect Permission Assignment for Critical Resource	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	766	Critical Variable Declared Public	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	767	Access to Critical Private Variable via Public Method	**Research Concepts (primary)1000**
CanFollow	Weakness Variant	219	Sensitive Data Under Web Root	Research Concepts1000

Theoretical Notes

A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

Relevant Properties

- Accessibility

Content History

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci	Cigital	External
	updated Time of Introduction		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Relationships, Other Notes		
2008-11-24	CWE Content Team	MITRE	Internal
	updated Relationships		
2009-05-27	CWE Content Team	MITRE	Internal
	updated Relationships		
2009-07-22 ***(Critical)***	CWE Content Team	MITRE	Internal
	Clarified description to include permissions.		
2009-07-27	CWE Content Team	MITRE	Internal
	updated Description, Relationships		
2009-10-29	CWE Content Team	MITRE	Internal
	updated Other Notes, Theoretical Notes		
2009-12-28	CWE Content Team	MITRE	Internal
	updated Relationships		

BACK TO TOP