WebJpa Scan Report

Project Name	WebJpa
Scan Start	Friday, June 19, 2020 2:02:43 PM
Preset	Checkmarx Default
Scan Time	00h:00m:00s
Lines Of Code Scanned	587
Files Scanned	17
Report Creation Time	Friday, June 19, 2020 2:03:01 PM
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7
Team	CxServer
Checkmarx Version	8.9.0.210 HF14
Scan Type	Full
Source Origin	LocalPath
Density	3/100 (Vulnerabilities/LOC)
Visibility	Public

Filter Settings

Severity

Included: High, Medium, Low, Information

Excluded: None

Result State

Included: Confirmed, Not Exploitable, To Verify, Urgent, Proposed Not Exploitable

Excluded: None

Assigned to

Included: All

Categories

Included:

Uncategorized	All
Custom	All
PCI DSS v3.2	All
OWASP Top 10 2013	All
FISMA 2014	All
NIST SP 800-53	All
OWASP Top 10 2017	All
OWASP Mobile Top 10 2016	All

Excluded:

Uncategorized	None
Custom	None
PCI DSS v3.2	None
OWASP Top 10 2013	None
FISMA 2014	None

NIST SP 800-53	None
OWASP Top 10 2017	None
OWASP Mobile Top 10 2016	None

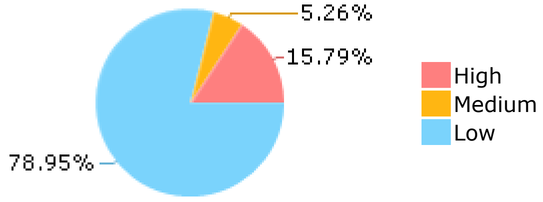
Results Limit

Results limit per query was set to 50

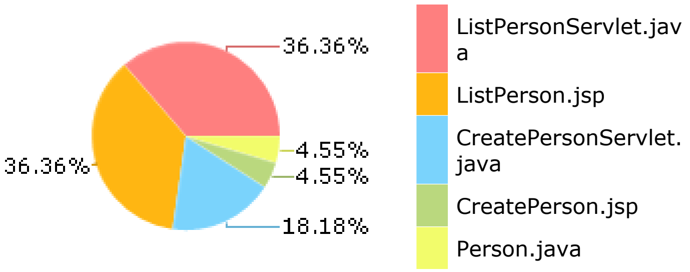
Selected Queries

Selected queries are listed in Result Summary

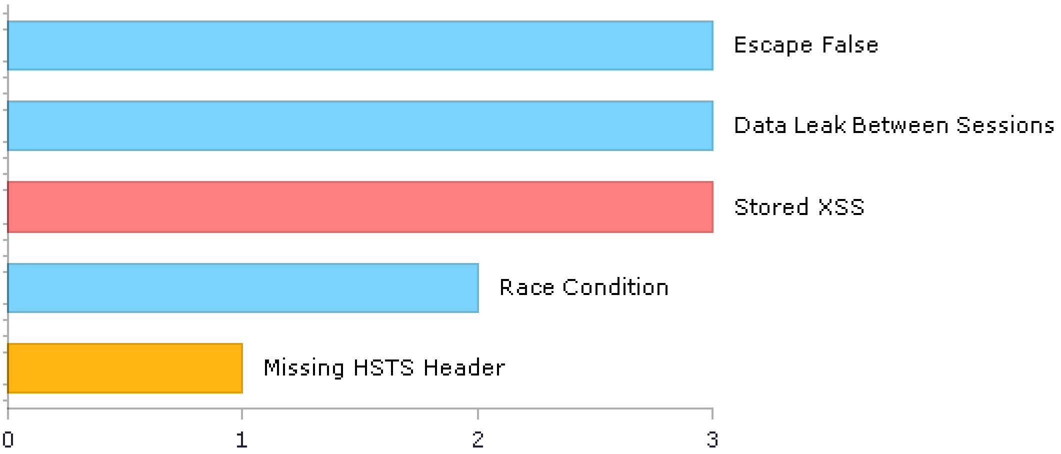
Result Summary



Most Vulnerable Files



Top 5 Vulnerabilities



Scan Summary - OWASP Top 10 2017

Further details and elaboration about vulnerabilities and risks can be found at: OWASP Top 10 2017

Category	Threat Agent	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact	Issues Found	Best Fix Locations
A1-Injection*	App. Specific	EASY	COMMON	EASY	SEVERE	App. Specific	0	0
A2-Broken Authentication*	App. Specific	EASY	COMMON	AVERAGE	SEVERE	App. Specific	3	0
A3-Sensitive Data Exposure*	App. Specific	AVERAGE	WIDESPREAD	AVERAGE	SEVERE	App. Specific	0	0
A4-XML External Entities (XXE)	App. Specific	AVERAGE	COMMON	EASY	SEVERE	App. Specific	0	0
A5-Broken Access Control*	App. Specific	AVERAGE	COMMON	AVERAGE	SEVERE	App. Specific	1	0
A6-Security Misconfiguration *	App. Specific	EASY	WIDESPREAD	EASY	MODERATE	App. Specific	2	0
A7-Cross-Site Scripting (XSS)*	App. Specific	EASY	WIDESPREAD	EASY	MODERATE	App. Specific	4	0
A8-Insecure Deserialization	App. Specific	DIFFICULT	COMMON	AVERAGE	SEVERE	App. Specific	0	0
A9-Using Components with Known Vulnerabilities*	App. Specific	AVERAGE	WIDESPREAD	AVERAGE	MODERATE	App. Specific	0	0
A10-Insufficient Logging & Monitoring	App. Specific	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	App. Specific	0	0

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - OWASP Top 10 2013

Further details and elaboration about vulnerabilities and risks can be found at: OWASP Top 10 2013

Category	Threat Agent	Attack Vectors	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact	Issues Found	Best Fix Locations
A1-Injection*	EXTERNAL, INTERNAL, ADMIN USERS	EASY	COMMON	AVERAGE	SEVERE	ALL DATA	0	0
A2-Broken Authentication and Session Management*	EXTERNAL, INTERNAL USERS	AVERAGE	WIDESPREAD	AVERAGE	SEVERE	AFFECTED DATA AND FUNCTIONS	3	0
A3-Cross-Site Scripting (XSS)*	EXTERNAL, INTERNAL, ADMIN USERS	AVERAGE	VERY WIDESPREAD	EASY	MODERATE	AFFECTED DATA AND SYSTEM	4	0
A4-Insecure Direct Object References*	SYSTEM USERS	EASY	COMMON	EASY	MODERATE	EXPOSED DATA	0	0
A5-Security Misconfiguration *	EXTERNAL, INTERNAL, ADMIN USERS	EASY	COMMON	EASY	MODERATE	ALL DATA AND SYSTEM	0	0
A6-Sensitive Data Exposure*	EXTERNAL, INTERNAL, ADMIN USERS, USERS BROWSERS	DIFFICULT	UNCOMMON	AVERAGE	SEVERE	EXPOSED DATA	0	0
A7-Missing Function Level Access Control*	EXTERNAL, INTERNAL USERS	EASY	COMMON	AVERAGE	MODERATE	EXPOSED DATA AND FUNCTIONS	0	0
A8-Cross-Site Request Forgery (CSRF)*	USERS BROWSERS	AVERAGE	COMMON	EASY	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0
A9-Using Components with Known Vulnerabilities*	EXTERNAL USERS, AUTOMATED TOOLS	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0
A10-Unvalidated Redirects and Forwards	USERS BROWSERS	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - PCI DSS v3.2

Category	Issues Found	Best Fix Locations
PCI DSS (3.2) - 6.5.1 - Injection flaws - particularly SQL injection	0	0
PCI DSS (3.2) - 6.5.2 - Buffer overflows	0	0
PCI DSS (3.2) - 6.5.3 - Insecure cryptographic storage*	0	0
PCI DSS (3.2) - 6.5.4 - Insecure communications*	0	0
PCI DSS (3.2) - 6.5.5 - Improper error handling*	0	0
PCI DSS (3.2) - 6.5.7 - Cross-site scripting (XSS)	4	0
PCI DSS (3.2) - 6.5.8 - Improper access control*	0	0
PCI DSS (3.2) - 6.5.9 - Cross-site request forgery*	0	0
PCI DSS (3.2) - 6.5.10 - Broken authentication and session management	4	0

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - FISMA 2014

Category	Description	Issues Found	Best Fix Locations
Access Control*	Organizations must limit information system access to authorized users, processes acting on behalf of authorized users, or devices (including other information systems) and to the types of transactions and functions that authorized users are permitted to exercise.	0	0
Audit And Accountability*	Organizations must: (i) create, protect, and retain information system audit records to the extent needed to enable the monitoring, analysis, investigation, and reporting of unlawful, unauthorized, or inappropriate information system activity; and (ii) ensure that the actions of individual information system users can be uniquely traced to those users so they can be held accountable for their actions.	0	0
Configuration Management*	Organizations must: (i) establish and maintain baseline configurations and inventories of organizational information systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles; and (ii) establish and enforce security configuration settings for information technology products employed in organizational information systems.	4	0
Identification And Authentication*	Organizations must identify information system users, processes acting on behalf of users, or devices and authenticate (or verify) the identities of those users, processes, or devices, as a prerequisite to allowing access to organizational information systems.	1	0
Media Protection	Organizations must: (i) protect information system media, both paper and digital; (ii) limit access to information on information system media to authorized users; and (iii) sanitize or destroy information system media before disposal or release for reuse.	0	0
System And Communications Protection	Organizations must: (i) monitor, control, and protect organizational communications (i.e., information transmitted or received by organizational information systems) at the external boundaries and key internal boundaries of the information systems; and (ii) employ architectural designs, software development techniques, and systems engineering principles that promote effective information security within organizational information systems.	0	0
System And Information Integrity*	Organizations must: (i) identify, report, and correct information and information system flaws in a timely manner; (ii) provide protection from malicious code at appropriate locations within organizational information systems; and (iii) monitor information system security alerts and advisories and take appropriate actions in response.	6	0

***** Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - NIST SP 800-53

Category	Issues Found	Best Fix Locations
AC-12 Session Termination (P2)	0	0
AC-3 Access Enforcement (P1)*	3	0
AC-4 Information Flow Enforcement (P1)	0	0
AC-6 Least Privilege (P1)	0	0
AU-9 Protection of Audit Information (P1)	0	0
CM-6 Configuration Settings (P2)	0	0
IA-5 Authenticator Management (P1)	0	0
IA-6 Authenticator Feedback (P2)	0	0
IA-8 Identification and Authentication (Non-Organizational Users) (P1)	0	0
SC-12 Cryptographic Key Establishment and Management (P1)	0	0
SC-13 Cryptographic Protection (P1)	0	0
SC-17 Public Key Infrastructure Certificates (P1)	0	0
SC-18 Mobile Code (P2)	1	0
SC-23 Session Authenticity (P1)*	0	0
SC-28 Protection of Information at Rest (P1)*	0	0
SC-4 Information in Shared Resources (P1)	3	0
SC-5 Denial of Service Protection (P1)*	0	0
SC-8 Transmission Confidentiality and Integrity (P1)	1	0
SI-10 Information Input Validation (P1)*	0	0
SI-11 Error Handling (P2)*	0	0
SI-15 Information Output Filtering (P0)*	7	0
SI-16 Memory Protection (P1)*	0	0

***** Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - OWASP Mobile Top 10 2016

Category	Description	Issues Found	Best Fix Locations
M1-Improper Platform Usage*	This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk.	0	0
M2-Insecure Data Storage*	This category covers insecure data storage and unintended data leakage.	0	0
M3-Insecure Communication*	This category covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.	0	0
M4-Insecure Authentication*	This category captures notions of authenticating the end user or bad session management. This can include: -Failing to identify the user at all when that should be required -Failure to maintain the user's identity when it is required -Weaknesses in session management	0	0
M5-Insufficient Cryptography*	The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that probably belongs in M2. This category is for issues where cryptography was attempted, but it wasnt done correctly.	0	0
M6-Insecure Authorization*	This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.). If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure.	0	0
M7-Client Code Quality*	This category is the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.	0	0
M8-Code Tampering*	This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or	0	0

	modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.		
M9-Reverse Engineering*****	This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.	0	0
M10-Extraneous Functionality*****	Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.	0	0

***** Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - Custom

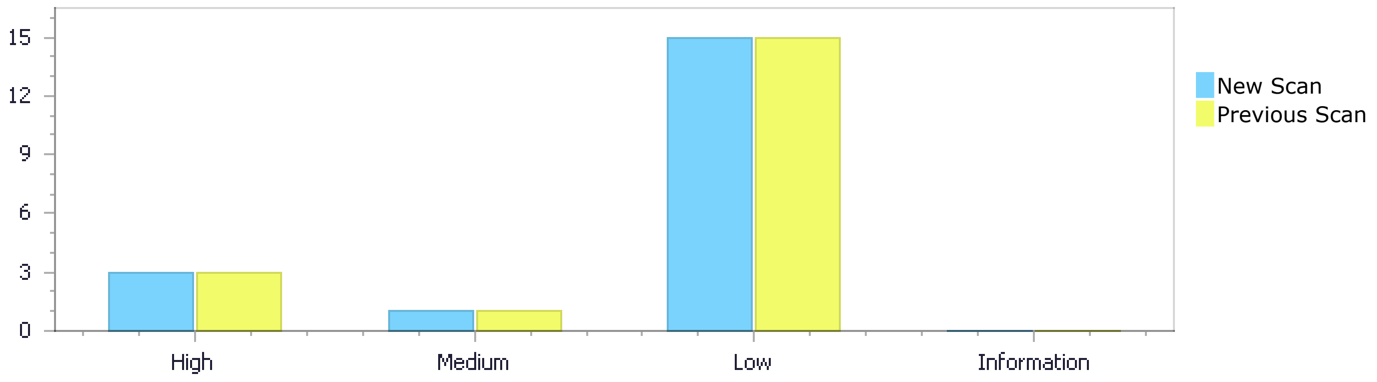
Category	Issues Found	Best Fix Locations
Must audit	0	0
Check	0	0
Optional	0	0

Results Distribution By Status

Compared to project scan from 6/19/2020 1:41 PM

	High	Medium	Low	Information	Total
New Issues	0	0	0	0	0
Recurrent Issues	3	1	15	0	19
Total	3	1	15	0	19

Fixed Issues	0	0	0	0	0



Results Distribution By State

	High	Medium	Low	Information	Total
Confirmed	0	0	0	0	0
Not Exploitable	0	0	0	0	0
To Verify	3	1	15	0	19
Urgent	0	0	0	0	0
Proposed Not Exploitable	0	0	0	0	0
Total	3	1	15	0	19

Result Summary

Vulnerability Type	Occurrences	Severity
Stored XSS	3	High
Missing HSTS Header	1	Medium
Data Leak Between Sessions	3	Low
Escape False	3	Low
Race Condition	2	Low

Client Insufficient ClickJacking Protection	1	Low
Improper Resource Access Authorization	1	Low
Improper Transaction Handling	1	Low
Missing Content Security Policy	1	Low
Missing X Frame Options	1	Low
Potential Stored XSS	1	Low
Stored Boundary Violation	1	Low

10 Most Vulnerable Files

High and Medium Vulnerabilities

File Name	Issues Found
web/ListPerson.jsp	4
src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java	3

Scan Results Details

Stored XSS

Query Path:
Java\Cx\Java High Risk\Stored XSS Version:1

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.7 - Cross-site scripting (XSS)
OWASP Top 10 2013: A3-Cross-Site Scripting (XSS)
FISMA 2014: System And Information Integrity
NIST SP 800-53: SI-15 Information Output Filtering (P0)
OWASP Top 10 2017: A7-Cross-Site Scripting (XSS)

Description

Stored XSS\Path 1:

Severity	High
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=6
Status	Recurrent

Method processRequest at line 56 of src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java gets data from the database, for the getResultList element. This element's value then flows through the code without being properly filtered or encoded and is eventually displayed to the user in method ${person.lastName} at line 57 of web/ListPerson.jsp. This may enable a Stored Cross-Site-Scripting attack.

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java	web/ListPerson.jsp
Line	64	57
Object	getResultList	BinaryExpr

Code Snippet
File Name src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java
Method protected void processRequest(HttpServletRequest request, HttpServletResponse response)

```
....
64.                 List persons = em.createQuery("select p from Person
p").getResultList();
```

File Name web/ListPerson.jsp
Method <td>${person.lastName}&nbsp;&nbsp;</td>

```
....
57.       <td>${person.lastName}&nbsp;&nbsp;</td>
```

Stored XSS\Path 2:

Severity	High
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=7
Status	Recurrent

Method processRequest at line 56 of src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java gets data from the database, for the getResultList element. This element's value then flows through the code without being properly filtered or encoded and is eventually displayed to the user in method ${person.firstName} at line 56 of web/ListPerson.jsp. This may enable a Stored Cross-Site-Scripting attack.

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java	web/ListPerson.jsp
Line	64	56
Object	getResultList	BinaryExpr

Code Snippet

File Name src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java

Method protected void processRequest(HttpServletRequest request, HttpServletResponse response)

```
....
64.                 List persons = em.createQuery("select p from Person
p").getResultList();
```

File Name web/ListPerson.jsp

Method <td>${person.firstName}&nbsp;&nbsp;</td>

```
....
56.       <td>${person.firstName}&nbsp;&nbsp;</td>
```

Stored XSS\Path 3:

Severity	High
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=8
Status	Recurrent

Method processRequest at line 56 of src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java gets data from the database, for the getResultList element. This element's value then flows through the code without being properly filtered or encoded and is eventually displayed to the user in method ${person.id} at line 55 of web/ListPerson.jsp. This may enable a Stored Cross-Site-Scripting attack.

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java	web/ListPerson.jsp
Line	64	55

Object	getResultList	BinaryExpr

Code Snippet

File Name	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java
Method	protected void processRequest(HttpServletRequest request, HttpServletResponse response)

```
....
64.                List persons = em.createQuery("select p from Person
p").getResultList();
```

File Name	web/ListPerson.jsp
Method	<td>${person.id}&nbsp;&nbsp;</td>

```
....
55.        <td>${person.id}&nbsp;&nbsp;</td>
```

Missing HSTS Header

Query Path:
Java\Cx\Java Medium Threat\Missing HSTS Header Version:1

Description

Missing HSTS Header\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=19
Status	Recurrent

The web-application does not define an HSTS header, leaving it vulnerable to attack.

	Source	Destination
File	web/ListPerson.jsp	web/ListPerson.jsp
Line	55	55
Object	write	write

Code Snippet

File Name	web/ListPerson.jsp
Method	<td>${person.id}&nbsp;&nbsp;</td>

```
....
55.        <td>${person.id}&nbsp;&nbsp;</td>
```

Escape False

Query Path:
Java\Cx\Java Low Visibility\Escape False Version:0

Categories

FISMA 2014: Configuration Management
NIST SP 800-53: SI-15 Information Output Filtering (P0)

Description

Escape False\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=1
Status	Recurrent

The web application uses the wrong kind of encoding to protect against Cross-Site Scripting (XSS), specifically using standard HTML encoding, escapeXml at web/ListPerson.jsp:55, to output JavaScript.

	Source	Destination
File	web/ListPerson.jsp	web/ListPerson.jsp
Line	55	55
Object	escapeXml	escapeXml

Code Snippet
File Name web/ListPerson.jsp
Method <td>${person.id}&nbsp;&nbsp;</td>

```
....
55.       <td>${person.id}&nbsp;&nbsp;</td>
```

Escape False\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=2
Status	Recurrent

The web application uses the wrong kind of encoding to protect against Cross-Site Scripting (XSS), specifically using standard HTML encoding, escapeXml at web/ListPerson.jsp:56, to output JavaScript.

	Source	Destination
File	web/ListPerson.jsp	web/ListPerson.jsp
Line	56	56
Object	escapeXml	escapeXml

Code Snippet
File Name web/ListPerson.jsp
Method <td>${person.firstName}&nbsp;&nbsp;</td>

```
....
56.       <td>${person.firstName}&nbsp;&nbsp;</td>
```

Escape False\Path 3:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=3
Status	Recurrent

The web application uses the wrong kind of encoding to protect against Cross-Site Scripting (XSS), specifically using standard HTML encoding, escapeXml at web/ListPerson.jsp:57, to output JavaScript.

	Source	Destination
File	web/ListPerson.jsp	web/ListPerson.jsp
Line	57	57
Object	escapeXml	escapeXml

Code Snippet
File Name web/ListPerson.jsp
Method <td>${person.lastName}&nbsp;&nbsp;</td>

```
....
57.         <td>${person.lastName}&nbsp;&nbsp;</td>
```

Data Leak Between Sessions

Query Path:
Java\Cx\Java Low Visibility\Data Leak Between Sessions Version:1

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.10 - Broken authentication and session management
OWASP Top 10 2013: A2-Broken Authentication and Session Management
NIST SP 800-53: SC-4 Information in Shared Resources (P1)
OWASP Top 10 2017: A2-Broken Authentication

Description

Data Leak Between Sessions\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=10
Status	Recurrent

The concurrent process @PersistenceUnit found in the file src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java at line 52 influences the shared resource CreatePersonServlet in the file src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java at line 49. When performed concurrently, an unexpected race condition may occur.

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet	src/java/enterprise/web_jpa_war/servlet

	/CreatePersonServlet.java	/CreatePersonServlet.java
Line	52	49
Object	emf	CreatePersonServlet

Code Snippet
File Name src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java
Method @PersistenceUnit

```
....
52.      @PersistenceUnit
```

▼

File Name src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java
Method @WebServlet(name="CreatePersonServlet", urlPatterns={"/CreatePerson"})

```
....
49.  @WebServlet(name="CreatePersonServlet",
urlPatterns={"/CreatePerson"})
```

Data Leak Between Sessions\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=11
Status	Recurrent

The concurrent process @Resource found in the file src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java at line 56 influences the shared resource CreatePersonServlet in the file src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java at line 49. When performed concurrently, an unexpected race condition may occur.

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet /CreatePersonServlet.java	src/java/enterprise/web_jpa_war/servlet /CreatePersonServlet.java
Line	56	49
Object	utx	CreatePersonServlet

Code Snippet
File Name src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java
Method @Resource

```
....
56.      @Resource
```

▼

File Name src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java

Method @WebServlet(name="CreatePersonServlet", urlPatterns={"/CreatePerson"})

```
....
49.  @WebServlet(name="CreatePersonServlet",
urlPatterns={"/CreatePerson"})
```

Data Leak Between Sessions\Path 3:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=12
Status	Recurrent

The concurrent process @PersistenceUnit found in the file src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java at line 49 influences the shared resource ListPersonServlet in the file src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java at line 46. When performed concurrently, an unexpected race condition may occur.

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java
Line	49	46
Object	emf	ListPersonServlet

Code Snippet

File Name src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java

Method @PersistenceUnit

```
....
49.      @PersistenceUnit
```

File Name src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java

Method @WebServlet(name="ListPersonServlet", urlPatterns={"/ListPerson"})

```
....
46.  @WebServlet(name="ListPersonServlet", urlPatterns={"/ListPerson"})
```

Race Condition

Query Path:
Java\Cx\Java Low Visibility\Race Condition Version:1

Categories

FISMA 2014: System And Information Integrity
NIST SP 800-53: AC-3 Access Enforcement (P1)

Description

Race Condition\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=14
Status	Recurrent

The concurrent process processRequest found in the file src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java at line 64 influences the shared resource emf in the file src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java at line 64. When performed concurrently, an unexpected race condition may occur.

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java	src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java
Line	83	83
Object	emf	emf

Code Snippet

File Name	src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java
Method	protected void processRequest(HttpServletRequest request, HttpServletResponse response)

```
....
83.                 em = emf.createEntityManager();
```

Race Condition\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=15
Status	Recurrent

The concurrent process processRequest found in the file src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java at line 56 influences the shared resource emf in the file src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java at line 56. When performed concurrently, an unexpected race condition may occur.

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java
Line	61	61
Object	emf	emf

Code Snippet

File Name	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java

Method protected void processRequest(HttpServletRequest request, HttpServletResponse response)

```
....
61.                 em = emf.createEntityManager();
```

Potential Stored XSS

Query Path:
Java\Cx\Java Potential\Potential Stored XSS Version:0

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.7 - Cross-site scripting (XSS)
OWASP Top 10 2013: A3-Cross-Site Scripting (XSS)
FISMA 2014: System And Information Integrity
NIST SP 800-53: SI-15 Information Output Filtering (P0)
OWASP Top 10 2017: A7-Cross-Site Scripting (XSS)

Description

Potential Stored XSS\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=4
Status	Recurrent

Method processRequest at line 56 of src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java gets data from the database, for the getResultList element. This element's value then flows through the code without being properly filtered or encoded and is eventually displayed to the user in method processRequest at line 56 of src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java. This may enable a Stored Cross-Site-Scripting attack.

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java
Line	64	65
Object	getResultList	persons

Code Snippet
File Name src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java
Method protected void processRequest(HttpServletRequest request, HttpServletResponse response)

```
....
64.                 List persons = em.createQuery("select p from Person
p").getResultList();
65.                 request.setAttribute("personList",persons);
```

Stored Boundary Violation

Query Path:
Java\Cx\Java Stored\Stored Boundary Violation Version:0

Categories

OWASP Top 10 2017: A5-Broken Access Control

Description

Stored Boundary Violation\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=5
Status	Recurrent

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java
Line	64	65
Object	getResultList	persons

Code Snippet

File Name	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java
Method	protected void processRequest(HttpServletRequest request, HttpServletResponse response)

```
....
64.                 List persons = em.createQuery("select p from Person
p").getResultList();
65.                 request.setAttribute("personList",persons);
```

Improper Transaction Handling

Query Path:
Java\Cx\Java Low Visibility\Improper Transaction Handling Version:1

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.10 - Broken authentication and session management

Description

Improper Transaction Handling\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=9
Status	Recurrent

The application's processRequest method in src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java creates and opens a connection to the database, and enlists it in a transaction. Though the application wraps the connection in a `try { }` block to handle exceptions, the database transaction is not always rolled back on errors.

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java	src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java

Line	68	68
Object	try	try

Code Snippet
File Name src/java/enterprise/web_jpa_war/servlet/CreatePersonServlet.java
Method protected void processRequest(HttpServletRequest request, HttpServletResponse response)

```
....
68.            try {
```

Client Insufficient ClickJacking Protection

Query Path:
JavaScript\Cx\JavaScript Low Visibility\Client Insufficient ClickJacking Protection Version:1

Categories

FISMA 2014: Configuration Management
NIST SP 800-53: SC-8 Transmission Confidentiality and Integrity (P1)

Description
Client Insufficient ClickJacking Protection\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=13
Status	Recurrent

The application does not protect the web page web/CreatePerson.jsp from clickjacking attacks in legacy browsers, by using framebusting scripts.

	Source	Destination
File	web/CreatePerson.jsp	web/CreatePerson.jsp
Line	1	1
Object	CxJSNS_1627838578	CxJSNS_1627838578

Code Snippet
File Name web/CreatePerson.jsp
Method <!--

```
....
1.  <!--
```

Missing X Frame Options

Query Path:
Java\Cx\Java Low Visibility\Missing X Frame Options Version:1

Categories

NIST SP 800-53: SC-18 Mobile Code (P2)
OWASP Top 10 2017: A6-Security Misconfiguration

Description

Missing X Frame Options\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=16
Status	Recurrent

The web-application does not properly utilize the "X-FRAME-OPTIONS" header to restrict embedding web-pages inside of a frame.

	Source	Destination
File	src/java/enterprise/web_jpa_war/entity/Person.java	src/java/enterprise/web_jpa_war/entity/Person.java
Line	32	32
Object	entity	entity

Code Snippet

File Name	src/java/enterprise/web_jpa_war/entity/Person.java
Method	package enterprise.web_jpa_war.entity;

```
....
32.  package enterprise.web_jpa_war.entity;
```

Improper Resource Access Authorization

Query Path:
Java\Cx\Java Low Visibility\Improper Resource Access Authorization Version:1

Categories

FISMA 2014: Identification And Authentication
NIST SP 800-53: AC-3 Access Enforcement (P1)

Description

Improper Resource Access Authorization\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=17
Status	Recurrent

	Source	Destination
File	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java
Line	64	64
Object	getResultList	getResultList

Code Snippet

File Name	src/java/enterprise/web_jpa_war/servlet/ListPersonServlet.java

Method	protected void processRequest(HttpServletRequest request, HttpServletResponse response)

```
....
64.                 List persons = em.createQuery("select p from Person
p").getResultList();
```

Missing Content Security Policy

Query Path:
Java\Cx\Java Low Visibility\Missing Content Security Policy Version:1

Categories

OWASP Top 10 2017: A6-Security Misconfiguration

Description

Missing Content Security Policy\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EDUARDOM-LAPTOP/CxWebClient/ViewerMain.aspx?scanid=1000032&projectid=7&pathid=18
Status	Recurrent

A Content Security Policy is not explicitly defined within the web-application.

	Source	Destination
File	web/ListPerson.jsp	web/ListPerson.jsp
Line	55	55
Object	Param	Param

Code Snippet

File Name	web/ListPerson.jsp
Method	<td>${person.id}&nbsp;&nbsp;</td>

```
....
55.       <td>${person.id}&nbsp;&nbsp;</td>
```

Stored XSS

Risk

What might happen

An attacker could use legitimate access to the application to submit engineered data to the application’s database. When another user subsequently accesses this data, web pages may be rewritten and malicious scripts may be activated.

Cause

How does it happen

The application creates web pages that include data from the application’s database. The data is embedded directly in the page's HTML, causing the browser to display it as part of the web page. This data may have originated in input from another user. If the data includes HTML fragments or Javascript, these are displayed too, and the user cannot tell that this is not the intended page. The vulnerability is the result of embedding arbitrary database data without first encoding it in a format that would prevent the browser from treating it like HTML instead of plain text.

General Recommendations

How to avoid it

1. Validate all dynamic data, regardless of source. Validation should be based on a whitelist: accept only data fitting a specified structure, rather than reject bad patterns. Check for:
   - Data type
   - Size
   - Range
   - Format
   - Expected values
2. Validation is not a replacement for encoding. Fully encode all dynamic data, regardless of source, before embedding it in output. Encoding should be context-sensitive. For example:
   - HTML encoding for HTML content
   - HTML attribute encoding for data output to attribute values
   - Javascript encoding for server-generated Javascript.
3. Consider using either the ESAPI encoding library, or its built-in functions. For earlier versions of ASP.NET, consider using the AntiXSS library.
4. In the Content-Type HTTP response header, explicitly define character encoding (charset) for the entire page.
5. Set the httpOnly flag on the session cookie, to prevent XSS exploits from stealing the cookie.

Source Code Examples

CSharp

Data obtained from the execution of an SQL command is output to a label

```
public class StoredXss
{
        public string foo(Label lblOutput, SqlConnection connection, int id)
    {
                string sql = "select email from CustomerLogin where customerNumber = @id";
                SqlCommand cmd = new SqlCommand(sql, connection);
                cmd.Prepare();
                cmd.Parameters.AddWithValue("@id", id);
                string output = (string)cmd.ExecuteScalar();
                lblOutput.Text = String.IsNullOrEmpty(output) ? "Customer Number does not
exist" : output;
        }
}
```

The outputed string is Html encoded before it is displayed in the label

```
public class StoredXssFixed
{
        public string foo(Label lblOutput, SqlConnection connection, HttpServerUtility
Server, int id)
    {
                string sql = "select email from CustomerLogin where customerNumber = @id";
                SqlCommand cmd = new SqlCommand(sql, connection);
                cmd.Prepare();
                cmd.Parameters.AddWithValue("@id", id);
                string output = (string)cmd.ExecuteScalar()
                lblOutput.Text = String.IsNullOrEmpty(output) ? "Customer Number does not
exist" : Server.HtmlEncode(output);
        }
}
```

Java

Data obtained from the excecution of an SQL command is output to a label

```
public class Stored_XSS {
     public static void XSSExample(Statement stmt) throws SQLException {
            Label label = new Label();
            ResultSet rs;
            rs = stmt.executeQuery("SELECT * FROM Customers WHERE UserName = Mickey");
            String lastNames = "";
            while (rs.next()) {
                   lastNames += rs.getString("Lname") + ", ";
            }
            label.setText("Mickey last names are: " + lastNames + " ");
     }
}
```

The outputed string is encoded to hard-coded string before it is displayed in the label

```
public class Stored_XSS_Fix {
     public static void XSSExample(Statement stmt) throws SQLException {
            Label label = new Label();
            ResultSet rs;
            HashMap<String, String> sanitize = new HashMap<String, String>();
            sanitize.put("A", "Cohen");
            sanitize.put("B", "Smith");
            sanitize.put("C", "Bond");
            rs = stmt.executeQuery("SELECT * FROM Customers WHERE UserName = Mickey");
            String lastNames = "";
            while (rs.next()) {
                   lastNames += sanitize.get(rs.getString("Lname")) + ", ";
            }
            label.setText("Mickey last names are: " + lastNames + " ");
     }
}
```

JavaScript

Data obtained from the execution of an SQL command is rendered to a web-page template

```
function renderUserProfileTable(res, connection, user_id) {
      connection.query('SELECT id,name,description from user WHERE id= ?',
[user_id],function(err, results) {
            var table = "<table>"
            table += "<table class='profile-html-table'>"
            table += "<tr><td>" + results[0].name + "</td></tr>"
            table += "<tr><td>" + results[0].description + "</td></tr>"
            table += "</table>"
            res.render("profile", table)
      });
}
```

Data obtained from the execution of an SQL command is encoded and then rendered to a web-page template

```
var htmlencoder = require('htmlencode');

function renderUserProfileTable(res, connection, user_id) {
      connection.query('SELECT id,name,description from user WHERE id= ?',
[user_id],function(err, results) {
            var table = "<table>"
            table += "<table class='profile-html-table'>"
            table += "<tr><td>" + htmlencoder.htmlEncode(results[0].name) + "</td></tr>"
            table += "<tr><td>" + htmlencoder.htmlEncode(results[0].description) +
"</td></tr>"
            table += "</table>"
            res.render("profile", table)
      });
}
```

Missing HSTS Header

Risk

What might happen

Failure to set an HSTS header and provide it with a reasonable "max-age" value of at least one year may leave users vulnerable to Man-in-the-Middle attacks.

Cause

How does it happen

Many users browse to websites by simply typing the domain name into the address bar, without the protocol prefix. The browser will automatically assume that the user's intended protocol is HTTP, instead of the encrypted HTTPS protocol.

When this initial request is made, an attacker can perform a Man-in-the-Middle attack and manipulate it to redirect users to a malicious web-site of the attacker's choosing. To protect the user from such an occurence, the HTTP Strict Transport Security (HSTS) header instructs the user's browser to disallow use of an unsecure HTTP connection to the the domain associated with the HSTS header.

Once a browser that supports the HSTS feature has visited a web-site and the header was set, it will no longer allow communicating with the domain over an HTTP connection.

Once an HSTS header was issued for a specific website, the browser is also instructed to prevent users from manually overriding and accepting an untrusted SSL certificate for as long as the "max-age" value still applies. The recommended "max-age" value is for at least one year in seconds, or 31536000.

General Recommendations

How to avoid it

- Before setting the HSTS header - consider the implications it may have:
  - Forcing HTTPS will prevent any future use of HTTP, which could hinder some testing
  - Disabling HSTS is not trivial, as once it is disabled on the site, it must also be disabled on the browser
- Set the HSTS header either explicitly within application code, or using web-server configurations.
- Ensure the "max-age" value for HSTS headers is set to 31536000 to ensure HSTS is strictly enforced for at least one year.
- Once HSTS has been enforced, submit the web-application's address to an HSTS preload list - this will ensure that, even if a client is accessing the web-application for the first time (implying HSTS has not yet been set by the web-application), a browser that respects the HSTS preload list would still treat the web-application as if it had already issued an HSTS header. Note that this requires the server to have a trusted SSL certificate, and issue an HSTS header with a maxAge of 1 year (31536000)
- Note that this query is designed to return one result per application. This means that if more than one vulnerable response without an HSTS header is identified, only the first identified instance of this issue will be highlighted as a result. Since HSTS is required to be enforced across the entire application to be considered a secure deployment of HSTS functionality, fixing this issue only where the query highlights this result is likely to produce subsequent results in other sections of the application; therefore, when adding this header via code, ensure it is uniformly deployed across the entire application. If this header is added via configuration, ensure that this configuration applies to the entire application.

Source Code Examples

Java

Setting an HSTS Header in an HTTP Response

```
response.setHeader("Strict-Transport-Security", "max-age=31536000");
```

Spring - Setting HSTS via Configuration Web.Xml

```
<http>
      <!-- This is a default value -->
      <headers>
             <hsts
                    include-subdomains="true"
                    max-age-seconds="31536000" />
      </headers>
</http>
```

JBoss - Setting HSTS via Configuration Web.Xml

```
<system.webServer>
    <httpProtocol>
        <customHeaders>
            <add name="Strict-Transport-Security" value="max-age=31536000"/>
        </customHeaders>
    </httpProtocol>
</system.webServer>
```

Tomcat - Enable Header Security in Web.Xml

```
<filter>
      <filter-name>httpHeaderSecurity</filter-name>
    <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>

      <init-param>
             <param-name>hstsMaxAgeSeconds</param-name>
            <param-value>31536000</param-value>
      </init-param>
</filter>
```

Escape False

Risk

What might happen

Since the application is not using context-specific encoding, it may be possible for an attacker to find a Cross-Site Scripting (XSS) vulnerability, enabling them to steal the user's password, request the user’s credit card information, provide false information, or run malicious code. All this would still be in context of the original web application, from the victim's perspective, allowing the attacker to effectively control the user's experience on the target website.

Cause

How does it happen

The application dynamically creates JavaScript scripts in the website's pages, embedding data generated on the server directly into the client JavaScript. This data might be controllable from user input. In order to protect this page against Cross-Site Scripting (XSS) attacks, the application does ensure the data is encoded before being output into the response.

However, the application does not perform proper context-specific encoding, in particular encoding the input for HTML instead of JavaScript-specific encoding. This could allow an attacker to inject a JavaScript payload into the response, despite the encoding. This JavaScript would be executed by the victim's browser.

General Recommendations

How to avoid it

Generic Guidance

- Avoid generating JavaScript code dynamically, based on user input or other data.
- Fully encode all dynamic data before embedding it in output.
- Encoding should be context-sensitive. For example:
  - HTML encoding for HTML content
  - HTML Attribute encoding for data output to attribute values
  - JavaScript encoding for server-generated JavaScript
- Consider using the `System.Web.Security.AntiXss.AntiXssEncoder` class (or the AntiXss library in older versions), to provide context-specific encoding methods.
- In the Content-Type HTTP response header, explicitly define character encoding (charset) for the entire page.
- Set the httpOnly flag on the session cookie, to prevent XSS exploits from stealing the cookie.
- Implement Content Security Policy (CSP) headers, to constrain and restrict the source of JavaScript code.

Specific Recommendations

- Remove the dynamically generated JavaScript code, and seperate any source of data from the source code.
- If it is necessary to dynamically generate JavaScript from data, use context-specific encoding on all data to properly mitigate XSS and simliar attacks.
- Consider using Razor's built-in context encoding nuggets.

Source Code Examples

CSharp

Generating JavaScript from Server Code

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="javascriptGenerator.aspx.cs"
                   Inherits="javascriptGenerator" %>
<script>
      var dataFromServer = "<%= Server.HtmlEncode(this.variable); %>";

    doSomething(dataFromServer);
</script>
```

Using Modern Razor Auto Encoding

```
@{
      LayoutPage = "SiteLayout.cshtml";
}
<script>
      var dataFromServer = "@this.variable";
    doSomething(dataFromServer);
</script>
```

Potential Stored XSS

Risk

What might happen

An attacker could use legitimate access to the application to submit engineered data to the application’s database. When another user subsequently accesses this data, web pages may be rewritten and malicious scripts may be activated.

Cause

How does it happen

The application creates web pages that include data from the application’s database. The data is embedded directly in the page's HTML, causing the browser to display it as part of the web page. This data may have originated in input from another user. If the data includes HTML fragments or Javascript, these are displayed too, and the user cannot tell that this is not the intended page. The vulnerability is the result of embedding arbitrary database data without first encoding it in a format that would prevent the browser from treating it like HTML instead of plain text.

General Recommendations

How to avoid it

1. Validate all dynamic data, regardless of source. Validation should be based on a whitelist: accept only data fitting a specified structure, rather than reject bad patterns. Check for:
    - Data type
    - Size
    - Range
    - Format
    - Expected values
2. Validation is not a replacement for encoding. Fully encode all dynamic data, regardless of source, before embedding it in output. Encoding should be context-sensitive. For example:
    - HTML encoding for HTML content
    - HTML attribute encoding for data output to attribute values
    - Javascript encoding for server-generated Javascript.
3. Consider using either the ESAPI encoding library, or its built-in functions. For earlier versions of ASP.NET, consider using the AntiXSS library.
4. In the Content-Type HTTP response header, explicitly define character encoding (charset) for the entire page.
5. Set the httpOnly flag on the session cookie, to prevent XSS exploits from stealing the cookie.

Source Code Examples

Reliance on File Name or Extension of Externally-Supplied File

Weakness ID: 646 *(Weakness Variant)* **Status:** Incomplete

Description

Description Summary

The software allows a file to be uploaded, but it relies on the file name or extension of the file to determine the appropriate behaviors. This could be used by attackers to cause the file to be misclassified and processed in a dangerous fashion.

Extended Description

An application might use the file name or extension of of a user-supplied file to determine the proper course of action, such as selecting the correct process to which control should be passed, deciding what data should be made available, or what resources should be allocated. If the attacker can cause the code to misclassify the supplied file, then the wrong action could occur. For example, an attacker could supply a file that ends in a ".php.gif" extension that appears to be a GIF image, but would be processed as PHP code. In extreme cases, code execution is possible, but the attacker could also cause exhaustion of resources, denial of service, information disclosure of debug or system data (including application source code), or being bound to a particular server side process. This weakness may be due to a vulnerability in any of the technologies used by the web and application servers, due to misconfiguration, or resultant from another flaw in the application itself.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

All

Common Consequences

Scope	Effect
Confidentiality	Information Leakage
Availability	Denial of Service
Access Control	Privilege Escalation

Likelihood of Exploit

High

Enabling Factors for Exploitation

There is reliance on file name and/or file extension on the server side for processing.

Potential Mitigations

Make decisions on the server side based on file content and not on file name or extension.

Properly configure web and applications servers.

Install the latest security patches for all of the technologies being used on the server side.

Relationships

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

ChildOf	Weakness Class	345	Insufficient Verification of Data Authenticity	**Development Concepts (primary)699** **Research Concepts (primary)1000**
ChildOf	Category	442	Web Problems	Development Concepts699

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version: 1.5)*
209	Cross-Site Scripting Using MIME Type Mismatch	

Content History

Submissions

Submission Date	Submitter	Organization	Source
2008-01-30	Evgeny Lebanidze	Cigital	External Submission

Modifications

Modification Date	Modifier	Organization	Source
2008-09-08	CWE Content Team	MITRE	Internal
	updated Common Consequences, Relationships, Observed Example		
2008-10-13	CWE Content Team	MITRE	Internal
	Significant clarification of the weakness description.		
2008-10-14	CWE Content Team	MITRE	Internal
	updated Description, Name, Observed Examples, Relationships		
2009-07-27	CWE Content Team	MITRE	Internal
	updated Related Attack Patterns		
2009-10-29	CWE Content Team	MITRE	Internal
	updated Common Consequences		

Previous Entry Names

Change Date	Previous Entry Name
2008-10-14	Taking Actions based on File Name or Extension of a User Supplied File

BACK TO TOP

Improper Transaction Handling

Risk

What might happen

Database transactions that are abandoned (if their associated connection is closed, before the transaction is committed or rolled back) can have several different results, depending on implementation and specific technologies in use. While in some scenarios the database will automatically roll back the transaction if the connection is closed, more often it will either automatically commit the transaction in its abortive state, or leave the transaction open indefinitely (depending on its configured timeout value).

In the first case, a transaction that is committed after a runtime exception is likely to be in an inconsistent state, incompatible with the current runtime conditions. This would result in situation detrimental to the system's integrity and possibly even stability.

In the second case, a transaction that is kept active indefinitely would cause the database server to retain its locks on all records and tables affected by the transaction. This could cause general reliability and availability problems, leading to delays, degraded performance, or even deadlocks as one thread waits for the locks to be released.

In either case this results in unexpected state, and is dependent on external factors such that the application is not controlling the result.

Cause

How does it happen

The application creates a connection to the database, and explicitly manages the database transaction by committing it when appropriate. However, the code does not explicitly roll back failed transactions, for example in the case of exceptions. This causes the application to rely on implementation-specific behavior, depending on the specific combination of technologies (such as the database server) and resultant configuration.

General Recommendations

How to avoid it

- Always open database connections and begin transactions within a `try { }` block.
- Ensure there are no active uncommitted transactions before closing a database connection.
- Always rollback active transactions in the case of exceptions.
- After handling the exception, ensure the transaction is rolled back in the `catch { }` block, or possibly in the `finally { }` block.

Source Code Examples

Java
Implicit Failed Transaction Handling

```
public void updateValues(int id, String value) {
      Connection con;
      PreparedStatement orderStmt = null;
      PreparedStatement custStmt = null;

    try {
            con = DriverManager.getConnection(CONN_STRING);
        con.setAutoCommit(false);

            String orderSql = "update ORDERS set DESCRIPTION = ? where CUSTID = ?";
        orderStmt = con.prepareStatement(orderSql);
            orderStmt.setString(1, value);
            orderStmt.setInt(2, id);
            orderStmt.executeUpdate();

            String custSql = "update CUSTOMERS set TOTAL = TOTAL + 1 where ID = ?"
        custStmt = con.prepareStatement(custSql);
            custStmt.setInt(1, id);
            custStmt.executeUpdate();

            con.commit();
    } catch (SQLException e ) {
        handleException(e);
    } finally {
        if (orderStmt != null)
            orderStmt.close();
        if (custStmt != null)
            custStmt.close();
        if (con != null)
                   con.close();  // In case of exception, transaction might be rolled back,
might not...
    }
}
```

Explicit Rollback of Failed Transaction

```
public void updateValues(int id, String value) {
      Connection con;
      PreparedStatement orderStmt = null;
      PreparedStatement custStmt = null;

    try {
            con = DriverManager.getConnection(CONN_STRING);
        con.setAutoCommit(false);

            String orderSql = "update ORDERS set DESCRIPTION = ? where CUSTID = ?";
        orderStmt = con.prepareStatement(orderSql);
            orderStmt.setString(1, value);
            orderStmt.setInt(2, id);
            orderStmt.executeUpdate();

            String custSql = "update CUSTOMERS set TOTAL = TOTAL + 1 where ID = ?"
        custStmt = con.prepareStatement(custSql);
            custStmt.setInt(1, id);
            custStmt.executeUpdate();

            con.commit();
    } catch (SQLException e ) {
        handleException(e);
            if (con != null) {
                   try {
```

```
                  // Explicitly rollback active transaction in case of exception
                         con.rollback();
            } catch (SQLException innerEx) {
                         handleException(innerEx);
            }
            }
    } finally {
        if (orderStmt != null)
            orderStmt.close();
        if (custStmt != null)
            custStmt.close();
        if (con != null)
                  con.close();
    }
}
```

Data Leak Between Sessions

Risk

What might happen

A race condition can cause erratic or unexpected application behavior. Additionally, if a race condition can be influenced directly by users, an attacker may choose to replay a race condition until they obtain a desired result, allowing them to induce certain application behavior that is not part of its intentional design.

Cause

How does it happen

Concurrent instances, such as threads or servlets, can run concurrently. Failure to ensure all of their operations on a shared resource are done atomically and in a synchronized manner may result in threads overriding each other's actions or outputs. For example, a logical state may have been validated prior to an update operation, but the update operation itself was only performed after the previously validated state has changed due to code concurrency.

General Recommendations

How to avoid it

Consider allocating independent resources for concurrent logical processes, unless required.

Where resource sharing is required, utilize a locking mechanism when handling shared resources to ensure that actions are performed atomically by the current process, so that these logical processes are not overridden or disrupted by any other concurrent process.

Source Code Examples

Java

Threads Incrementing The Same Variable Without Locking

```
        public static class FooIncrement {
                static int foo = 0;
                static Object lock = new Object();
                public static int generateFoo() throws InterruptedException {
                        IncrementFooThread[] threadArray = new IncrementFooThread[10000];
                        for (int i = 0; i < threadArray.length; i++) {
                                threadArray[i] = new IncrementFooThread();
                                threadArray[i].start();
                        }
                        for (IncrementFooThread threaderThread : threadArray) {
                                threaderThread.join();
                        }
                        return foo;
                }
                public static class IncrementFooThread extends Thread {
                        public void run()

                        {

                                foo++; //two threads may separately read the same value of foo before
   any of them update it,
                            //resulting in two threads setting the same value of foo
                   }

            }
     }

```

Adding A Lock Object to Synchronize a Shared Resource

```
     public static class FooIncrement {
            static int foo = 0;
            static Object lock = new Object();
            public static int generateFoo() throws InterruptedException {
                   IncrementFooThread[] threadArray = new IncrementFooThread[10000];
                   for (int i = 0; i < threadArray.length; i++) {
                          threadArray[i] = new IncrementFooThread();
                          threadArray[i].start();
                   }
                   for (IncrementFooThread threaderThread : threadArray) {
                          threaderThread.join();
                   }
                   return foo;
            }
            public static class IncrementFooThread extends Thread {
                   public void run()
                   {

                          synchronized (lock) {
                                 foo++; //foo is atomically incremented
                          }
                   }

            }

```

CSharp

ASPX Setting and Then Reading A Singleton Variable That May Have Been Set By Another Request

```
// Default.aspx.cs
using System;
using System.ComponentModel.DataAnnotations;
using System.Web.DynamicData;

public partial class _Default : System.Web.UI.Page {
    public static string secret = "None";
    protected void Page_Load(object sender, EventArgs e) {
        secret = Request.Params["secret"];
    }
}

// Default.aspx - the "secret" printed here may be different to the secret obtained and set
from Request,
// if another user makes a request with a different "secret", writing a different "secret"
altogether
<%@ Page Language="C#" MasterPageFile="~/Site.master" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <%= secret %>
</asp:Content>

```

Client Insufficient ClickJacking Protection

Risk

What might happen

Clickjacking attacks allow an attacker to "hijack" a user's mouse clicks on a webpage, by invisibly framing the application, and superimposing it in front of a bogus site. When the user is convinced to click on the bogus website, e.g. on a link or a button, the user's mouse is actually clicking on the target webpage, despite being invisible.
This could allow the attacker to craft an overlay that, when clicked, would lead the user to perform undesirable actions in the vulnerable application, e.g. enabling the user's webcam, deleting all the user's records, changing the user's settings, or causing clickfraud.

Cause

How does it happen

The root cause of vulnerability to a clickjacking attack, is that the application's web pages can be loaded into a frame of another website. The application does not implement a proper frame-busting script, that would prevent the page from being loaded into another frame. Note that there are many types of simplistic redirection scripts that still leave the application vulnerable to clickjacking techniques, and should not be used.
When dealing with modern browsers, applications mitigate this vulnerability by issuing appropriate Content-Security-Policy or X-Frame-Options headers to indicate to the browser to disallow framing. However, many legacy browsers do not support this feature, and require a more manual approach by implementing a mitigation in Javascript. To ensure legacy support, a framebusting script is required.

General Recommendations

How to avoid it

Generic Guidance:

- Define and implement a a Content Security Policy (CSP) on the server side, including a frame-ancestors directive. Enforce the CSP on all relevant webpages.
- If certain webpages are required to be loaded into a frame, define a specific, whitelisted target URL.
- Alternatively, return a "X-Frame-Options" header on all HTTP responses. If it is necessary to allow a particular webpage to be loaded into a frame, define a specific, whitelisted target URL.
- For legacy support, implement framebusting code using Javascript and CSS to ensure that, if a page is framed, it is never displayed, and attempt to navigate into the frame to prevent attack. Even if navigation fails, the page is not displayed and is therefore not interactive, mitigating potential clickjacking attacks.

Specific Recommendations:

- Implement a proper framebuster script on the client, that is not vulnerable to frame-buster-busting attacks.
    - Code should first disable the UI, such that even if frame-busting is successfully evaded, the UI cannot be clicked. This can be done by setting the CSS value of the "display" attribute to "none" on either the "body" or "html" tags. This is done because, if a frame attempts to redirect and become the parent, the malicious parent can still prevent redirection via various techniques.
    - Code should then determine whether no framing occurs by comparing self === top; if the result is true, can the UI be enabled. If it is false, attempt to navigate away from the framing page by setting the top.location attribute to self.location.

Source Code Examples

JavaScript
Clickjackable Webpage

```
<html>
    <body>

    <button onclick="clicked();">
            Click here if you love ducks
        </button>
    </body>
</html>
```

Bustable Framebuster

```
<html>
    <head>

    <script>
            if ( window.self.location != window.top.location ) {
                   window.top.location = window.self.location;
            }
        </script>
    </head>

    <body>
        <button onclick="clicked();">
            Click here if you love ducks
        </button>
    </body>
</html>
```

Proper Framebusterbusterbusting

```
<html>
    <head>

    <style> html {display : none; } </style>
        <script>
            if ( self === top ) {
                   document.documentElement.style.display = 'block';
            }
            else {
                   top.location = self.location;
            }
        </script>
    </head>

    <body>
        <button onclick="clicked();">
            Click here if you love ducks
        </button>
    </body>
</html>
```

Race Condition

Risk

What might happen

A race condition can cause erratic or unexpected application behavior. Additionally, if a race condition can be influenced directly by users, an attacker may choose to replay a race condition until they obtain a desired result, allowing them to induce certain application behavior that is not part of its intentional design.

Cause

How does it happen

Concurrent instances, such as threads or servlets, can run concurrently. Failure to ensure all of their operations on a shared resource are done atomically and in a synchronized manner may result in threads overriding each other's actions or outputs. For example, a logical state may have been validated prior to an update operation, but the update operation itself was only performed after the previously validated state has changed due to code concurrency.

General Recommendations

How to avoid it

Consider allocating independent resources for concurrent logical processes, unless required.

Where resource sharing is required, utilize a locking mechanism when handling shared resources to ensure that actions are performed atomically by the current process, so that these logical processes are not overridden or disrupted by any other concurrent process.

Source Code Examples

Missing X Frame Options

Risk

What might happen

Allowing setting of web-pages inside of a frame in an untrusted web-page will leave these web-pages vulnerable to Clickjacking, otherwise known as a redress attack. This may allow an attacker to redress a vulnerable web-page by setting it inside a frame within a malicious web-page. By crafting a convincing malicious web-page, the attacker can then use the overlayed redress to convince the user to click a certain area of the screen, unknowingly clicking inside the frame containing the vulnerable web-page, and thus performing actions within the user's context on the attacker's behalf.

Cause

How does it happen

Failure to utilize the "X-FRAME-OPTIONS" header will likely allow attackers to perform Clickjacking attacks. Properly utilizing the "X-FRAME-OPTIONS" header would indicate to the browser to disallow embedding the web-page within a frame, mitigating this risk, if the browser supports this header. All modern browsers support this header by default.

General Recommendations

How to avoid it

Utilize the "X-FRAME-OPTIONS" header flags according to business requirements to restrict browsers that support this header from allowing embedding web-pages in a frame:

- "X-Frame-Options: DENY" will indicate to the browser to disallow embedding any web-page inside a frame, including the current web-site.
- "X-Frame-Options: SAMEORIGIN" will indicate to the browser to disallow embedding any web-page inside a frame, excluding the current web-site.
- "X-Frame-Options: ALLOW-FROM https://example.com/" will indicate to the browser to disallow embedding any web-page inside a frame, excluding the web-site listed after the ALLOW-FROM parameter.

Source Code Examples

Java

Setting the "DENY" Flag on a Response

```
response.addHeader("X-Frame-Options", "DENY");
```

Improper Access Control (Authorization)

Weakness ID: 285 *(Weakness Class)* **Status:** Draft

Description

Description Summary

The software does not perform or incorrectly performs access control checks across all potential execution paths.

Extended Description

When access control checks are not applied consistently - or not at all - users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information leaks, denial of service, and arbitrary code execution.

Alternate Terms

AuthZ:	"AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is also distinct from "AuthC," which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

Language-independent

Technology Classes

Web-Server: *(Often)*

Database-Server: *(Often)*

Modes of Introduction

A developer may introduce authorization weaknesses because of a lack of understanding about the underlying technologies. For example, a developer may assume that attackers cannot modify certain inputs such as headers or cookies.

Authorization weaknesses may arise when a single-user application is ported to a multi-user environment.

Common Consequences

Scope	Effect
Confidentiality	An attacker could read sensitive data, either by reading the data directly from a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to read the data.
Integrity	An attacker could modify sensitive data, either by writing the data directly to a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to write the data.
Integrity	An attacker could gain privileges by modifying or reading critical data directly, or by accessing insufficiently-protected, privileged functionality.

Likelihood of Exploit

High

Detection Methods

Automated Static Analysis

Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries.

Generally, automated static analysis tools have difficulty detecting custom authorization schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an authorization check; an automated technique that detects the absence of authorization may report false positives.

Effectiveness: Limited

Automated Dynamic Analysis

Automated dynamic analysis may find many or all possible interfaces that do not require authorization, but manual analysis is required to determine if the lack of authorization violates business logic

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Specifically, manual static analysis is useful for evaluating the correctness of custom authorization mechanisms.

Effectiveness: Moderate

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

Demonstrative Examples

Example 1

The following program could be part of a bulletin board system that allows users to send private messages to each other. This program intends to authenticate the user before deciding whether a private message should be displayed. Assume that LookupMessageObject() ensures that the $id argument is numeric, constructs a filename based on that id, and reads the message details from that file. Also assume that the program stores all private messages for all users in the same directory.

(Bad Code)

Example Language: **Perl**

```
sub DisplayPrivateMessage {
my($id) = @_;
my $Message = LookupMessageObject($id);
print "From: " . encodeHTML($Message->{from}) . "<br>\n";
print "Subject: " . encodeHTML($Message->{subject}) . "\n";
print "<hr>\n";
print "Body: " . encodeHTML($Message->{body}) . "\n";
}

my $q = new CGI;
# For purposes of this example, assume that CWE-309 and
# CWE-523 do not apply.
if (! AuthenticateUser($q->param('username'), $q->param('password'))) {
ExitError("invalid username or password");
}

my $id = $q->param('id');
DisplayPrivateMessage($id);
```

While the program properly exits if authentication fails, it does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other users.

One way to avoid this problem would be to ensure that the "to" field in the message object matches the username of the authenticated user.

Observed Examples

Reference	Description
CVE-2009-3168	Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords.

CVE-2009-2960	Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users.
CVE-2009-3597	Web application stores database file under the web root with insufficient access control (CWE-219), allowing direct request.
CVE-2009-2282	Terminal server does not check authorization for guest access.
CVE-2009-3230	Database server does not use appropriate privileges for certain sensitive operations.
CVE-2009-2213	Gateway uses default "Allow" configuration for its authorization settings.
CVE-2009-0034	Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges.
CVE-2008-6123	Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect.
CVE-2008-5027	System monitoring software allows users to bypass authorization by creating custom forms.
CVE-2008-7109	Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client.
CVE-2008-3424	Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access.
CVE-2009-3781	Content management system does not check access permissions for private files, allowing others to view those files.
CVE-2008-4577	ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions.
CVE-2008-6548	Product does not check the ACL of a page accessed using an "include" directive, allowing attackers to read unauthorized files.
CVE-2007-2925	Default ACL list for a DNS server does not set certain ACLs, allowing unauthorized DNS queries.
CVE-2006-6679	Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header.
CVE-2005-3623	OS kernel does not check for a certain privilege before setting ACLs for files.
CVE-2005-2801	Chain: file-system code performs an incorrect comparison (CWE-697), preventing defauls ACLs from being properly applied.
CVE-2001-1155	Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions.

Potential Mitigations

Phase: Architecture and Design

Divide your application into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) to enforce the roles at the appropriate boundaries.

Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

Phase: Architecture and Design

Ensure that you perform access control checks related to your business logic. These checks may be different than the access control checks that you apply to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor.

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.

For example, consider using authorization frameworks such as the JAAS Authorization Framework and the OWASP ESAPI Access Control feature.

Phase: Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page.

One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

Phases: System Configuration; Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Category	254	Security Features	**Seven Pernicious Kingdoms (primary)700**
ChildOf	Weakness Class	284	Access Control (Authorization) Issues	**Development Concepts (primary)699** **Research Concepts (primary)1000**
ChildOf	Category	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	**Weaknesses in OWASP Top Ten (2007) (primary)629**
ChildOf	Category	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ChildOf	Category	753	2009 Top 25 - Porous Defenses	**Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)750**
ChildOf	Category	803	2010 Top 25 - Porous Defenses	**Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)800**
ParentOf	Weakness Variant	219	Sensitive Data Under Web Root	**Research Concepts (primary)1000**
ParentOf	Weakness Base	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	**Development Concepts (primary)699** Research Concepts1000
ParentOf	Weakness Class	638	Failure to Use Complete Mediation	Research Concepts1000
ParentOf	Weakness Base	804	Guessable CAPTCHA	**Development Concepts (primary)699** **Research Concepts (primary)1000**

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Missing Access Control
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version: 1.5)*
1	Accessing Functionality Not Properly Constrained by ACLs	
13	Subverting Environment Variable Values	

17	Accessing, Modifying or Executing Executable Files
87	Forceful Browsing
39	Manipulating Opaque Client-based Data Tokens
45	Buffer Overflow via Symbolic Links
51	Poison Web Service Registry
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
77	Manipulating User-Controlled Variables
76	Manipulating Input to File System Calls
104	Cross Zone Scripting

References

NIST. "Role Based Access Control and Role Based Security". <http://csrc.nist.gov/groups/SNS/rbac/>.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 4, "Authorization" Page 114; Chapter 6, "Determining Appropriate Access Control" Page 171. 2nd Edition. Microsoft. 2002.

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 Time of Introduction		
2008-08-15		Veracode	External
	Suggested OWASP Top Ten 2004 mapping		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Relationships, Other Notes, Taxonomy Mappings		
2009-01-12	CWE Content Team	MITRE	Internal
	updated Common Consequences, Description, Likelihood of Exploit, Name, 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 Description, Related Attack Patterns		
2009-07-27	CWE Content Team	MITRE	Internal
	updated Relationships		
2009-10-29	CWE Content Team	MITRE	Internal
	updated Type		
2009-12-28	CWE Content Team	MITRE	Internal
	updated Applicable Platforms, Common Consequences, Demonstrative Examples, Detection Factors, Modes of Introduction, Observed Examples, Relationships		
2010-02-16	CWE Content Team	MITRE	Internal
	updated Alternate Terms, Detection Factors, Potential Mitigations, References, Relationships		
2010-04-05	CWE Content Team	MITRE	Internal
	updated Potential Mitigations		

Previous Entry Names

Change Date	Previous Entry Name
2009-01-12	Missing or Inconsistent Access Control

BACK TO TOP

Missing Content Security Policy

Risk

What might happen

The Content-Security-Policy header enforces that the source of content, such as the origin of a script, embedded (child) frame, embedding (parent) frame or image, are trusted and allowed by the current web-page; if, within the web-page, a content's source does not adhere to a strict Content Security Policy, it is promptly rejected by the browser. Failure to define a policy may leave the application's users exposed to Cross-Site Scripting (XSS) attacks, Clickjacking attacks, content forgery and more.

Cause

How does it happen

The Content-Security-Policy header is used by modern browsers as an indicator for trusted sources of content, including media, images, scripts, frames and more. If these policies are not explicitly defined, default browser behavior would allow untrusted content.

The application creates web responses, but does not properly set a Content-Security-Policy header.

General Recommendations

How to avoid it

Explicitly set the Content-Security-Policy headers for all applicable policy types (frame, script, form, script, media, img etc.) according to business requirements and deployment layout of external file hosting services. Specifically, do not use a wildcard, '*', to specify these policies, as this would allow content from any external resource.

The Content-Security-Policy can be explicitly defined within web-application code, as a header managed by web-server configurations, or within <meta> tags in the HTML `<head>` section.

Source Code Examples

Java

HTTP Response With CSP Header Set

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
  // handle request
  response.setHeader("Content-Security-Policy", "default-src 'self'"); // default-src is the
most restric mode of CSP and covers all applicable policy types
}
```

HTTP Response with CSP Header in Spring

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Override
protected void configure(HttpSecurity http) throws Exception {
 http
   .headers()
   .contentSecurityPolicy("default-src 'self'"); // default-src is the most restric mode of CSP and covers all applicable policy types
      }
}
```

Scanned Languages

Language	Hash Number	Change Date
Java	0125956323748501	3/25/2020
JavaScript	1003522720031683	3/25/2020
Common	0114668597102001	3/25/2020