RIS2.0--7boss-style--master--7bossStyle Scan Report

Project Name	RIS2.0--7boss-style--master--7bossStyle
Scan Start	Wednesday, April 15, 2020 1:07:04 PM
Preset	High and Medium - 7-11
Scan Time	00h:02m:07s
Lines Of Code Scanned	18089
Files Scanned	21
Report Creation Time	Wednesday, April 15, 2020 1:09:25 PM
Online Results	http://EC2AMAZ-KD9UIP5/CxWebClient/ViewerMain.aspx?scanid=1071981&projectid=4040
Team	DevOps
Checkmarx Version	8.9.0.210 HF5
Scan Type	Full
Source Origin	LocalPath
Density	2/10000 (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 v3.2	All
OWASP Top 10 2013	All
FISMA 2014	All
NIST SP 800-53	All
OWASP Top 10 2017	All
OWASP Mobile Top 10 2016	All

Excluded:

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

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

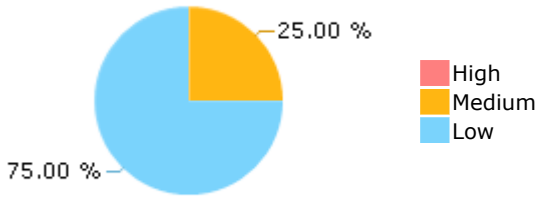
Results Limit

Results limit per query was set to 50

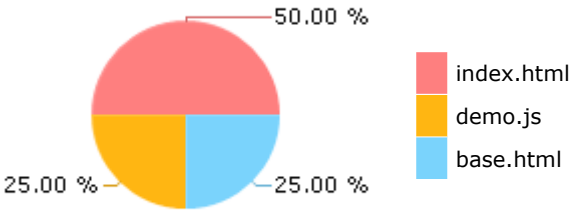
Selected Queries

Selected queries are listed in Result Summary

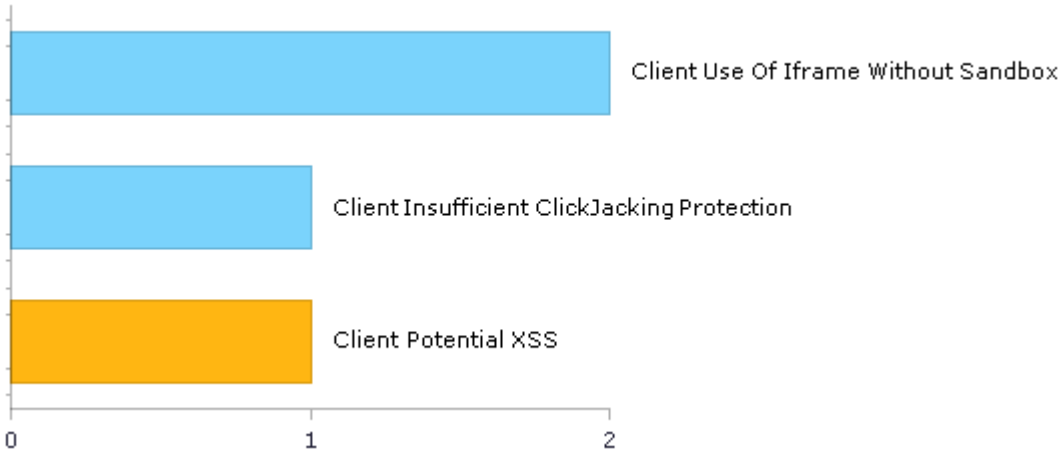
Result Summary



Most Vulnerable Files



Top 5 Vulnerabilities



Scan Summary - OWASP Top 10 2017

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

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

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

Scan Summary - OWASP Top 10 2013

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

Category	Threat Agent	Attack Vectors	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact	Issues Found	Best Fix Locations
A1-Injection	EXTERNAL, INTERNAL, ADMIN USERS	EASY	COMMON	AVERAGE	SEVERE	ALL DATA	0	0
A2-Broken Authentication and Session Management*	EXTERNAL, INTERNAL USERS	AVERAGE	WIDESPREAD	AVERAGE	SEVERE	AFFECTED DATA AND FUNCTIONS	0	0
A3-Cross-Site Scripting (XSS)*	EXTERNAL, INTERNAL, ADMIN USERS	AVERAGE	VERY WIDESPREAD	EASY	MODERATE	AFFECTED DATA AND SYSTEM	1	1
A4-Insecure Direct Object References	SYSTEM USERS	EASY	COMMON	EASY	MODERATE	EXPOSED DATA	0	0
A5-Security Misconfiguration	EXTERNAL, INTERNAL, ADMIN USERS	EASY	COMMON	EASY	MODERATE	ALL DATA AND SYSTEM	0	0
A6-Sensitive Data Exposure*	EXTERNAL, INTERNAL, ADMIN USERS, USERS BROWSERS	DIFFICULT	UNCOMMON	AVERAGE	SEVERE	EXPOSED DATA	0	0
A7-Missing Function Level Access Control	EXTERNAL, INTERNAL USERS	EASY	COMMON	AVERAGE	MODERATE	EXPOSED DATA AND FUNCTIONS	0	0
A8-Cross-Site Request Forgery (CSRF)*	USERS BROWSERS	AVERAGE	COMMON	EASY	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0
A9-Using Components with Known Vulnerabilities	EXTERNAL USERS, AUTOMATED TOOLS	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0
A10-Unvalidated Redirects and Forwards*	USERS BROWSERS	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0

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

Scan Summary - PCI DSS v3.2

Category	Issues Found	Best Fix Locations
PCI DSS (3.2) - 6.5.1 - Injection flaws - particularly SQL injection	0	0
PCI DSS (3.2) - 6.5.2 - Buffer overflows	0	0
PCI DSS (3.2) - 6.5.3 - Insecure cryptographic storage	0	0
PCI DSS (3.2) - 6.5.4 - Insecure communications	0	0
PCI DSS (3.2) - 6.5.5 - Improper error handling	0	0
PCI DSS (3.2) - 6.5.7 - Cross-site scripting (XSS)	1	1
PCI DSS (3.2) - 6.5.8 - Improper access control*	0	0
PCI DSS (3.2) - 6.5.9 - Cross-site request forgery*	0	0
PCI DSS (3.2) - 6.5.10 - Broken authentication and session management	0	0

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

Scan Summary - FISMA 2014

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

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

Scan Summary - NIST SP 800-53

Category	Issues Found	Best Fix Locations
AC-12 Session Termination (P2)	0	0
AC-3 Access Enforcement (P1)	0	0
AC-4 Information Flow Enforcement (P1)	0	0
AC-6 Least Privilege (P1)	0	0
AU-9 Protection of Audit Information (P1)	0	0
CM-6 Configuration Settings (P2)	0	0
IA-5 Authenticator Management (P1)	0	0
IA-6 Authenticator Feedback (P2)	0	0
IA-8 Identification and Authentication (Non-Organizational Users) (P1)	0	0
SC-12 Cryptographic Key Establishment and Management (P1)	0	0
SC-13 Cryptographic Protection (P1)	0	0
SC-17 Public Key Infrastructure Certificates (P1)	0	0
SC-18 Mobile Code (P2)	2	2
SC-23 Session Authenticity (P1)*	0	0
SC-28 Protection of Information at Rest (P1)	0	0
SC-4 Information in Shared Resources (P1)	0	0
SC-5 Denial of Service Protection (P1)	0	0
SC-8 Transmission Confidentiality and Integrity (P1)	1	1
SI-10 Information Input Validation (P1)*	0	0
SI-11 Error Handling (P2)	0	0
SI-15 Information Output Filtering (P0)*	1	1
SI-16 Memory Protection (P1)	0	0

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

Scan Summary - OWASP Mobile Top 10 2016

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

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

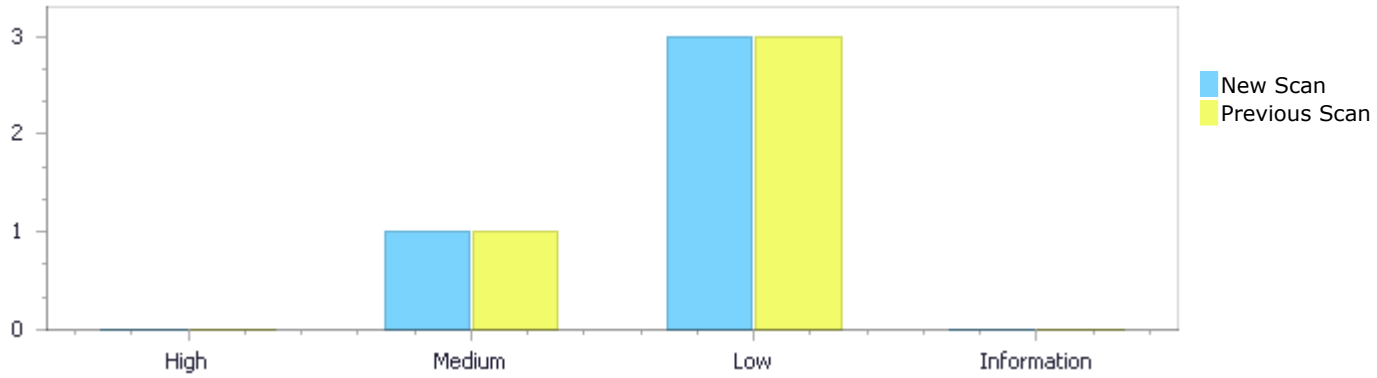
Scan Summary - Custom

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

Results Distribution By Status

Compared to project scan from 4/10/2020 2:49 PM

	High	Medium	Low	Information	Total
New Issues	0	0	0	0	0
Recurrent Issues	0	1	3	0	4
Total	0	1	3	0	4
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	0	1	3	0	4
Urgent	0	0	0	0	0
Proposed Not Exploitable	0	0	0	0	0
Total	0	1	3	0	4

Result Summary

Vulnerability Type	Occurrences	Severity
Client Potential XSS	1	Medium
Client Use Of Iframe Without Sandbox	2	Low
Client Insufficient ClickJacking Protection	1	Low

10 Most Vulnerable Files

High and Medium Vulnerabilities

File Name	Issues Found
src/icomoon/demo-files/demo.js	1

Scan Results Details

Client Potential XSS

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

Categories

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

Description
Client Potential XSS\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://EC2AMAZ-KD9UIP5/CxWebClient/ViewerMain.aspx?scanid=1071981&projectid=4040&pathid=4
Status	Recurrent

Method updateTest at line 17 of src/icomoon/demo-files/demo.js gets user input for the value 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 updateTest at line 17 of src/icomoon/demo-files/demo.js. This may enable a Cross-Site-Scripting attack.

	Source	Destination
File	src/icomoon/demo-files/demo.js	src/icomoon/demo-files/demo.js
Line	18	18
Object	value	innerHTML

Code Snippet
File Name src/icomoon/demo-files/demo.js
Method function updateTest() {

```
....
18.          testDrive.innerHTML = testText.value ||
String.fromCharCode(160);
```

Client Use Of Iframe Without Sandbox

Query Path:
JavaScript\Cx\JavaScript Low Visibility\Client Use Of Iframe Without Sandbox Version:1

Categories

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

Description
Client Use Of Iframe Without Sandbox\Path 1:

Severity	Low

Result State	To Verify
Online Results	http://EC2AMAZ-KD9UIP5/CxWebClient/ViewerMain.aspx?scanid=1071981&projectid=4040&pathid=2
Status	Recurrent

The application loads an external library or source code file using iframe___158069358, at line 459 of index.html. An attacker might be able to exploit this and cause the application to load arbitrary code.

	Source	Destination
File	index.html	index.html
Line	459	459
Object	iframe___158069358	iframe___158069358

Code Snippet
File Name index.html
Method <iframe src="file:///Users/gher3001/source/repos/7boss-style/7bossStyle/base.html">

```
....
459.              <iframe
src="file:///Users/gher3001/source/repos/7boss-
style/7bossStyle/base.html">
```

Client Use Of Iframe Without Sandbox\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://EC2AMAZ-KD9UIP5/CxWebClient/ViewerMain.aspx?scanid=1071981&projectid=4040&pathid=3
Status	Recurrent

The application loads an external library or source code file using iframe____533647770, at line 2900 of index.html. An attacker might be able to exploit this and cause the application to load arbitrary code.

	Source	Destination
File	index.html	index.html
Line	2900	2900
Object	iframe____533647770	iframe____533647770

Code Snippet
File Name index.html
Method <iframe src="file:///Users/gher3001/source/repos/7boss-style/7bossStyle/spinner.html">

```
....
2900.            <iframe src="file:///Users/gher3001/source/repos/7boss-
style/7bossStyle/spinner.html">
```

Client Insufficient ClickJacking Protection

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

Categories

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

Description

Client Insufficient ClickJacking Protection\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://EC2AMAZ-KD9UIP5/CxWebClient/ViewerMain.aspx?scanid=1071981&projectid=4040&pathid=1
Status	Recurrent

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

	Source	Destination
File	base.html	base.html
Line	1	1
Object	CxJSNS_250039106	CxJSNS_250039106

Code Snippet
File Name base.html
Method <!doctype html>

```
....
1.  <!doctype html>
```

Client Potential XSS

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

Bad - 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);
        }
}
```

Good - 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));
        }
}
```

Bad - 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;
        }
}
```

Good - 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);
        }
}
```

Bad - 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));
        }
}
```

Good - 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

```
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

```
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");
            }
      }
}
```

Client Insufficient ClickJacking Protection

Risk

What might happen

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

Cause

How does it happen

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

General Recommendations

How to avoid it

Generic Guidance:

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

Specific Recommendations:

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

Source Code Examples

JavaScript
Clickjackable Webpage

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

Bustable Framebuster

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

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

Proper Framebusterbusterbusting

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

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

Client Use Of Iframe Without Sandbox

Risk

What might happen

If an attacker can select the name of the library, or the location of the code file that is loaded by the application, they would be able to cause the application to execute arbitrary code. This effectively allows the attacker to control the code run by the application.

Cause

How does it happen

The application uses untrusted data to specify the library or code file, without proper sanitization. This causes the application to load any arbitrary code, as specified. The loaded code will then be executed.

General Recommendations

How to avoid it

- Do not dynamically load code libraries, especially not based on user input.
- If it is necessary to use untrusted data to select the library to be loaded, verify the selected library name matches a predefined set of whitelisted library names. Alternatively, use the value as an identifier to select from the whitelisted libraries.
- Validate any untrusted data used to load or process libraries or code files.

Source Code Examples

Scanned Languages

Language	Hash Number	Change Date
JavaScript	1003522720031683	10/10/2019
VbScript	9340222351170833	10/10/2019
Typescript	1488217042171263	10/10/2019
Common	0114668597102001	10/10/2019