掃描結果詳細資料

Client DOM XSS

查詢路徑:

JavaScript\Cx\JavaScript High Risk\Client DOM XSS 版本:8

類別

PCI DSS v3.2.1: PCI DSS (3.2.1) - 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)
CWE top 25: CWE top 25
MOIS(KISA) Secure Coding 2021: MOIS(KISA) Verification and representation of input data
OWASP ASVS: V05 Validation, Sanitization and Encoding
OWASP Top 10 2021: A3-Injection
SANS top 25: SANS top 25
ASA Premium: ASA Premium
ASD STIG 5.2: APSC-DV-002490 - CAT I The application must protect from Cross-Site Scripting (XSS) vulnerabilities.
Top Tier: Top Tier

描述

Client DOM XSS\路徑 1:

嚴重程度：	高風險
結果狀態：	校驗
線上結果	http://10.10.2.164/CxWebClient/ViewerMain.aspx?scanid=85690&projectid=31418&pathid=1
狀態	新的
Detection Date	6/25/2024 10:55:04 AM

方法checkAndRedirect在cicd-form-frontend-checkmars/src/index.html的第33行使用location將不受信任的資料嵌入生成的輸出。這些不受信任的資料被嵌入輸出而沒有進行適當的消毒或編碼，使攻擊者能夠將惡意程式碼注入生成的網頁。

	來源	目的地
檔案	cicd-form-frontend-checkmars/src/index.html	cicd-form-frontend-checkmars/src/index.html
行	37	48
物件	toString	location

代碼片斷

檔案名稱 cicd-form-frontend-checkmars/src/index.html

方法 function checkAndRedirect() {

```
....
37.  var currentLocation = self.location.toString();
....
48.  top.location = sanitizedLocation;
```

Potential Clickjacking on Legacy Browsers

查詢路徑:

JavaScript\Cx\JavaScript Low Visibility\Potential Clickjacking on Legacy Browsers 版本:7

類別

CWE top 25: CWE top 25
MOIS(KISA) Secure Coding 2021: MOIS(KISA) Verification and representation of input data
OWASP ASVS: V05 Validation, Sanitization and Encoding
OWASP Top 10 2021: A8-Software and Data Integrity Failures
SANS top 25: SANS top 25
ASD STIG 5.2: APSC-DV-002330 - CAT II The application must protect the confidentiality and integrity of stored information when required by DoD policy or the information owner.

描述

Potential Clickjacking on Legacy Browsers\路徑 1:

嚴重程度：	低風險
結果狀態：	校驗
線上結果	http://10.10.2.164/CxWebClient/ViewerMain.aspx?scanid=85690&projectid=31418&pathid=2
狀態	反覆出現的問題
Detection Date	6/13/2024 10:45:28 AM

應用程式未使用 framebusting 腳本保護 cicd-form-frontend-checkmars/src/app/app.component.html網頁免受舊版瀏覽器中的 clickjacking 攻擊。

	來源	目的地
檔案	cicd-form-frontend-checkmars/src/app/app.component.html	cicd-form-frontend-checkmars/src/app/app.component.html
行	1	1
物件	<	<

代碼片斷

檔案名稱 cicd-form-frontend-checkmars/src/app/app.component.html

方法 <app-navbar [logoutTime]="logoutTime"></app-navbar>

```
....
1.  <app-navbar [logoutTime]="logoutTime"></app-navbar>
```

Client DOM XSS

風險

可能發生什麼問題

成功的跨站腳本 (XSS) 攻擊可讓攻擊者重寫網頁並插入惡意腳本，從而改變原本的輸出。這包括 HTML 片段、CSS 樣式規則、任意 JavaScript 或第三方程式碼的引用。攻擊者可以利用此漏洞來竊取使用者的密碼，收集個人資料，例如信用卡詳細訊息，提供虛假訊息或運行惡意軟件。從受害者的角度來看，這是由正確的網站執行，並且受害者會責怪網站造成的損害。

DOM XSS 還存在另一個風險，就是含有惡意的資料不需要通過伺服器。由於伺服器未參與這些輸入的消毒，因此伺服器端驗證不太可能知道 XSS 攻擊正在發生，任何伺服器端上的安全解決方案，例如 WAF，對 DOM XSS 的緩解可能會無效。

原因

如何發生

該應用程式建立的網頁包含不受信任的資料，無論是從使用者輸入、應用程式資料庫或其他外部來源。這些不受信任的資料直接嵌入頁面的 HTML 中，導致瀏覽器將其顯示為網頁的一部分。如果輸入包含 HTML 片段或 JavaScript，這些也會被顯示，而使用者無法判斷這不是預期的網頁。此漏洞是由於直接嵌入任意資料，而未先將其編碼為防止瀏覽器將其視為 HTML 或程式碼而非純文本的格式所導致。

當DOM XSS發生時，用戶端的程式碼可以操控本地端網頁上的DOM，並引用惡意的內容。

一般建議

如何避免

- 在輸出前，必須將所有動態資料完全編碼，不論來源為何。
- 編碼應該是對上下文敏感的。例如：
  - 對於HTML內容使用HTML編碼
  - 對於輸出到屬性值的資料使用HTML屬性編碼
  - 對於伺服器生成的JavaScript使用JavaScript編碼
- 建議使用平台提供的編碼功能，或已知的安全庫來編碼輸出。
- 實施Content Security Policy（CSP），僅為應用程序的資源明確白名單。
- 作為額外的保護層，無論來源如何，驗證所有不受信任的資料（注意，這不是編碼的替代）。驗證應基於白名單：僅接受符合指定結構的資料，而不是拒絕不良模式。檢查：
  - 資料類型
  - 大小
  - 範圍
  - 格式
  - 預期值
- 在Content-Type HTTP回應標頭中，明確定義整個頁面的字符編碼（字符集）。
- 為"深度防禦"，在會話cookie上設置HTTPOnly標誌，以防止任何成功的XSS攻擊竊取cookie。

程式碼範例

JavaScript

DOM XSS in img Attribute

```
var url = new URL(window.location.href);
var imgsrc = url.searchParams.get("imageLocation");
document.write('<img id="myImage" src=' + imgsrc +' ></img>'); // The payload
"imageLocation=1 onerror=alert(1)" will result in an alert prompt, demonstrating XSS
```

Use Javascript to Construct DOM Elements, Rather Than Manually Concatenating Values

```
var url = new URL(window.location.href);
var imgsrc = url.searchParams.get("imageLocation");
var myImg = document.createElement("IMG");
myImg.src = imgsrc;
someDiv.append(myImg);
```

DOM XSS When Using "eval()" to Parse JSON in Javascript

```
var url = new URL(window.location.href);
var val = url.searchParams.get("val");
var json = `[{"val": "${val}"}]`;
var obj = eval(json); // The payload json=","a":alert(1),"b":" will result in an alert
prompt, demonstrating XSS
```

Replacing "eval()" with "JSON.parse()" to Avoid XSS

```
var url = new URL(window.location.href);
var val = url.searchParams.get("val");
var json = `[{"val": "${val}"}]`;
var obj =  JSON.parse(json); // JSON.parse() does not eval JS code
```

DOM XSS in iFrame "src" Attribute

```
var url = new URL(window.location.href);
var iframeLocation = url.searchParams.get("iframeLocation");
document.getElementById("myFrame").src = iframeLocation; // The payload
"iframeLocation=javascript:alert(1)" will result in an alert prompt, demonstrating XSS. This
is also vulnerable to open redirection.
```

Prepending iFrame "src" Attribute to Prevent Malicious URI Schemes

```
var url = new URL(window.location.href);
var iframeLocation = url.searchParams.get("iframeLocation");
document.getElementById("myFrame").src = "/example/"+iframeLocation; // Prepending
iframeLocation prevents changing the URI scheme to "javascript:", mitigating XSS
```

Potential Clickjacking on Legacy Browsers

風險

可能發生什麼問題

Clickjacking 攻擊允許攻擊者構築無形的應用程式並將其疊加在偽冒網站上，藉此「劫持」使用者在網頁上的滑鼠點擊。使用者以為自己在點擊網站的連結或按鈕，但實際上點擊的是一個看不見的惡意網頁。
這使攻擊者能夠設計覆蓋層，點擊後會導致在脆弱的應用程式中執行非預期操作，例如啟用使用者的網路攝影機、刪除所有使用者紀錄、更改使用者設置或導致 clickfraud。

原因

如何發生

Clickjacking 漏洞的根本原因是應用程式網頁可以被載入到另一個網站的框架中。應用程式未實作可以防止頁面被載入到其他框架的 framebusting 腳本。注意還有很多類型的簡化重新導向腳本會使應用程式容易受到 Clickjacking 技術的影響，因此建議不要使用。
大部分現代瀏覽器可以支援應用程式所發佈的 Content-Security-Policy 或 X-Frame-Options header，從而禁止框架，避免此漏洞。但舊版瀏覽器不支援此功能，因此需要在 Javascript 中手動實作一個 framebusting 腳本，已保證在舊版本瀏覽器中也不會遭遇攻擊。

一般建議

如何避免

通用指南：

- 在伺服器端定義並實作內容安全策略 (CSP)，包括 frame-ancestors 指令。在所有相關網頁上實作 CSP。
- 如果需要將某些網頁載入到框架中，請定義具體的白名單目標 URL。
- 也可在所有 HTTP responses上返回一個 "X-Frame-Options" header。如果需要允許將特定網頁載入到框架中，可定義具體的白名單目標 URL。
- 對於舊版本瀏覽器的支援，可使用 Javascript 和 CSS 實作 framebusting 程式，確保頁面被框架化後不會顯示，並嘗試導航到框架以防止攻擊。即使無法導航，頁面也不會顯示，因此沒有交互性，也可以減少受到 Clickjacking 攻擊的機會。

具體建議：

- 在客戶端上實作不容易受到 frame-buster-busting 攻擊的 framebuster 腳本。
    - 代碼應該先禁用 UI，這樣即使成功繞過防框架代碼，也無法點擊 UI。這可以通過在 "body" 或 "html" 標籤中，將 "display" 屬性的 CSS 值設為 "none" 來禁用UI。這樣做是因為，如果框架嘗試重新導向並成為主視窗，仍然可以通過各種技術阻止惡意主視窗重新導向。
    - 然後程式應通過比較 self === top 來確定是否沒有框架發生；如果結果為 true，則可以啟用 UI。如果為 false，可將 top.location 屬性設置為 self.location 來嘗試離開框架頁面。

程式碼範例

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>
```

檢測的語言

語言	HASH值	變更的日期
JavaScript	5693733879119650	2024/6/11
VbScript	0386000544005133	2023/4/6
Common	1330881790325397	2024/6/11