# 跨網站Scripting

## 應用程式

## WASC 威脅分類

用戶端攻擊：跨網站Scripting
http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml

## CVE 參照

N/A

## 安全風險

有可能竊取或操作客戶階段作業和Cookie，其可能用來假冒合法的使用者，讓駭客可以檢視或
變更使用者記錄，以及以該使用者的身分來執行交易

## 可能原因

未正確地消毒使用者所輸入的危險字元

## 技術說明

「跨網站Scripting」攻擊是一種隱私權違規，可讓攻擊者獲取合法使用者的認證，在與特定網站互動時假冒這位使用者。這個攻擊立足於下列事實：網站中有Script 在未事先消毒的情況下，傳回使用者在HTML 頁面中的輸入（通常是參數值）。當Script 在回應頁面中傳回這個輸入時，瀏覽器便得以執行JavaScript 程式碼所組成的輸入。結果便有可能形成通往網站的若干鏈結，且其中一個參數是由惡意的JavaScript 程式碼組成。這個程式碼將由使用者瀏覽器在網站環境定義中執行，透過使用者的瀏覽器，它便能存取使用者所擁有的網站Cookie以及網站的其他視窗。攻擊依照下列方式繼續進行：攻擊者誘使合法使用者按一下攻擊者所設下的鏈結。當使用者按下這個鏈結，便會產生對於網站的要求，其中的參數值含有惡意的JavaScript 程式碼。如果網站將這個參數值內嵌在回應的HTML 頁面中（這正是這個網站問題的本質），惡意程式碼便會在使用者瀏覽器中執行。
Script 可能執行的動作如下：
[1] 將使用者的Cookie（用於合法網站）傳給攻擊者。
[2] 將可透過DOM（URL、表單欄位等）存取的資訊傳給攻擊者。結果是在有漏洞的網站上，使用者成為受害者，安全和隱私權受到侵害。
部分附註如下：
[1] 雖然受攻擊的網站涉入其中，但它沒有直接受到侵害。它只是攻擊者傳送之惡意Script 的「跳板」，用來以合法身分返回受害者的瀏覽器。

不過，由於受害者的隱私權是在特定網站的環境中受到侵害，並且因為網站有直接責任，所以這視為網站的安全缺失。

[2] 如果受害者造訪由攻擊者所維護的網站，攻擊者便可以利用網站鏈結來提供惡意的鏈結。如果攻擊者知道使用者的電子郵件位址，且使用者的電子郵件用戶端利用瀏覽器來呈現HTML訊息，惡意的鏈結也可以由電子郵件來提供。

[3] 使用者輸入最常在表單欄位值中找到（也就是URL 參數），但也有已知的攻擊將惡意的程式碼內嵌在路徑、查詢或HTTP Referer 標頭中，甚至是內嵌在Cookie 中。

[4] AppScan 會傳送許多類型的「跨網站Scripting」攻擊，其中包括只作用於特定瀏覽器或特定瀏覽器版本的攻擊。AppScan 的「在瀏覽器中顯示」特性利用Internet Explorer 來顯示漏洞。在Internet Explorer 沒有漏洞而其他瀏覽器有漏洞的變式案例中，「在瀏覽器中顯示」機能無法運作且不會出現蹦現畫面。將輸入傳給很容易遭到跨網站Scripting 攻擊的Web 應用程式，有兩個可能的實務：

A. 在回應頁面中，傳回傳給CGI Script 的參數值，內嵌在HTML 中。

例如：

[要求]

GET /cgi-bin/script.pl?name=JSmith HTTP/1.0

[回應]

HTTP/1.1 200 OK

Server: SomeServer

Date: Sun, 01 Jan 2002 00:31:19 GMT

Content-Type: text/html

Accept-Ranges: bytes

Content-Length: 27

<HTML>

Hello JSmith

</HTML>

B. 在HTML 參數值上下文中，傳回傳給CGI Script 的參數值。

例如：

[要求]

GET /cgi-bin/script.pl?name=JSmith HTTP/1.0

[回應]

HTTP/1.1 200 OK

Server: SomeServer

Date: Sun, 01 Jan 2002 00:31:19 GMT

Content-Type: text/html

Accept-Ranges: bytes

Content-Length: 254

<HTML>

Please fill in your zip code:

<FORM METHOD=GET ACTION="/cgi-bin/script.pl">

<INPUT TYPE=text NAME="name" value="JSmith"> <br>

<INPUT TYPE=text NAME="zip" value="Enter zip code here"> <br>

<INPUT TYPE=submit value="Submit">

</FORM>

</HTML>

範例 1 - 實務A

使用者送出下列要求：

[攻擊要求]

GET

/cgi-bin/script.pl?name=>"'><script>alert('Watchfire%20XSS%20Test%20Successful')</script>

HTTP/1.0

[攻擊回應實務A]

HTTP/1.1 200 OK

Server: SomeServer

Date: Sun, 01 Jan 2002 00:31:19 GMT

Content-Type: text/html

Accept-Ranges: bytes

Content-Length: 83

<HTML>

Hello >"'><script>alert('Watchfire XSS Test Successful')</script>

</HTML>

在這個情況下，瀏覽器會執行JavaScript 程式碼（>"'> 部分在這裡並不相關）。

範例 2 - 實務B

利用範例 1 的相同Script 和輸入來呼叫攻擊：

[攻擊回應實務B]

HTTP/1.1 200 OK

Server: SomeServer

Date: Sun, 01 Jan 2002 00:31:19 GMT

Content-Type: text/html

Accept-Ranges: bytes

Content-Length: 310

<HTML>

Please fill in your zip code:

<FORM METHOD=GET ACTION="/cgi-bin/script.pl">

<INPUT TYPE=text NAME="name" value=">"'><script>alert('Watchfire XSS Test Successful')

</script>"> <br>

<INPUT TYPE=text NAME="zip" value="Enter zip code here"> <br>

<INPUT TYPE=submit value="Submit">

</FORM>

</HTML>

>"'> 字首用來跳脫出參數值的上下文。

先關閉參數值欄位 (>)，再關閉 <INPUT> 標籤 (>)，會使瀏覽器執行JavaScript，但不會視為已當作JavaScript 程式碼來剖析或執行的參數值。以下列出不同的測試變式：

[1] >'><script>alert('Watchfire XSS Test Successful')</script>

[2] >"><script>alert("Watchfire XSS Test Successful")</script>

[3] </TextArea><script>alert('Watchfire XSS Test Successful')</script>

[4] >'"><img src="javascript:alert('Watchfire XSS Test Successful')">

[5] >'"><img src=javascript:alert("Watchfire XSS Test Successful")>

[6] " style="background:url(javascript:alert('Watchfire XSS Test Successful'))" OA="

[7] --><script>alert('Watchfire XSS Test Successful')</script>

[8] '+alert('Watchfire XSS Test Successful')+'

[9] "+alert('Watchfire XSS Test Successful')+"

[10] >'><%00script>alert('Watchfire XSS Test Successful')</script> (.NET 1.1 specific variant)

[11] >"><%00script>alert("Watchfire XSS Test Successful")</script> (.NET 1.1 specific variant)

[12] >+ACI-+AD4-+ADw-SCRIPT+AD4-alert(1234)+ADw-/SCRIPT+AD4-

[13]

%A7%A2%BE%Bc%F3%E3%F2%E9%F0%F4%Be%E1%Ec%E5%F2%F4%A8%A7Watchfire%20XSS%20Test%20Successful%A7%A9%Bc%Af%F3%E3%F2%E9%F0%F4%Be

變式詳細資料：

測試 [1] 和 [2]：這些都是最基本的跨網站Scripting 變式。兩個變式之間的差異是在JavaScript 程式碼中使用引號或單引號。部分Web 應用程式設計師只消毒使用者輸入中的單引號或引號，未同時消毒兩者。執行這兩個變式都會偵測到這個漏洞。

測試變式 [3]：這項測試變式是專為了內嵌在 <TEXTAREA> 參數中傳回的使用者輸入而設計。在測試期間，試圖跳脫出參數值（文字區）來強制瀏覽器執行JavaScript。

測試變式 [4]：部分Web 應用程式設計師會消毒使用者輸入中的 <SCRIPT> 標籤，但忘了消毒HTML 鏈結中所能使用的 "javascript:" 指定元。在這項測試期間，試圖利用 <img> 標籤來內嵌惡意的JavaScript 程式碼，以JavaScript 鏈結為其來源。

測試變式 [5]：這個變式非常類似於 4 號。它利用HTML 實體來略過消毒 <, >、引號和 "javascript:" 指定元等使用者輸入的安全措施。

測試變式 [6]：這個變式使用最少非標準字元。它不像先前的變式，並不使用 &、>、<、# 或 ; 等字元。假設使用者輸入內嵌在HTML 表單參數值中（在 <INPUT> 標籤內），惡意字串先從參數值上下文中跳出，然後繼續將STYLE 屬性加到 <INPUT> 標籤，使惡意的JavaScript 程式碼內嵌在其中。附註：只有在實務B 中，或使用者輸入是內嵌在其他HTML 元素的屬性

內，才能順利完成這個變式。

測試變式 [7]：部分Web 應用程式將使用者輸入內嵌在HTML 註解中。為了測試應用程式的這個漏洞，首先是關閉HTML 註解 ( --> )，然後內嵌惡意的JavaScript 程式碼。

測試變式 [8] 和 [9]：部分Web 應用程式將使用者輸入內嵌在JavaScript 字串文字中，例如：

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
var str = 'Hello $user_input';
...
</SCRIPT>
</HTML>
```

如果我們傳送下列參數值：'+alert('Watchfire XSS Test Successful')+'，結果回應頁面會看起來如

下：

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
var str = 'Hello ' + alert('Watchfire XSS Test Successful') + '';
...
</SCRIPT>
</HTML>
```

應用程式遭受矇騙，將惡意的JavaScript 程式碼連結在原始字串文字的中間，導致瀏覽器執行我們的JavaScript 程式碼。8 號和 9 號變式之間的差異是使用引號或單引號，它們用來自訂這兩種字串終止字元的攻擊。

測試變式 [10] 和 [11]：在Microsoft .NET 1.1 中，HttpRequest.ValidateInput 方法會驗證用戶端瀏覽器提交的資料，如果有潛在危險資料，便發出異常狀況。MSDN 指出：「如果頁面指引或配置已啟用驗證特性，在頁面的ProcessRequest 處理階段中，會呼叫這個方法。如果未啟用驗證特性，您的程式碼可以呼叫ValidateInput。驗證要求的方式是對照寫在程式中的潛在危險資料清單來檢查所有輸入資料。」要求驗證期間，會檢查下列成員中的輸入資料：

- HttpRequest.Form,
- HttpRequest.QueryString,
- HttpRequest.Cookies

** 附註：依預設，在ASP.NET 1.1 中，會啟用HttpRequest.ValidateInput

ASP.NET 1.1 會封鎖含有 '<'，且後面接著英數字元或驚嘆號的輸入（例如：<script>、<img、<!--，等等）。如果 '<' 字元後面先接著NULL 位元組，然後才是英數字元，型樣便不符，且輸入可以抵達Web 應用程式。

例如：

[*] ASP.NET 1.1 會封鎖 '<script>' 字串

[*] ASP.NET 1.1 會接受 '<%00script>' 字串

此外，大部分Web 瀏覽器（包括Microsoft Internet Explorer 的所有版本）的HTML 剖析器都會忽略NULL 位元組，將 <%00script> 當作 <script> 來剖析。當這一點與上述安全問題結合

起來，便可以透過ASP.NET 1.1 HttpRequest.ValidateInput 安全機制來注入任何HTML 標籤，結果會非常容易遭到跨網站 Scripting 及注入其他惡意 HTML 標籤的攻擊。

測試變式 [12]：許多輸入驗證功能都能適當濾除或跳出XSS 的常用字元（如角括弧 <>），只有少數會設法處理危險的UTF-7 編碼字串。因此，在許多情況下，當傳送以UTF-7 來編碼的XSS 攻擊內容時，內容會在回應中原封不動傳回。攻擊若要成功，受害者的瀏覽器應該將XSS 內容當作UTF-7 來處理，否則，不會執行Script。如果「編碼」設為「自動偵測」，且Internet Explorer 在回應內文的前 4096 個字元中找到UTF-7 字串，除非已施行另一個字元編碼，否則，它會將字集編碼自動設為UTF-7。這項自動編碼特性可協助惡意的使用者發動UTF-7 XSS 攻擊。這個變式攻擊成功的需求如下：

[*] 受害者使用Internet Explorer 用戶端，且「編碼」設為「自動偵測」。

[*] 未在下列位置施行字集編碼（除非施行UTF-7）：

[*] 回應標頭 ("Content-Type: text/html; charset=[encoding]")。

[*] 在回應HTML 中有 <meta http-equiv="Content-Type" (...) charset=[encoding]/> 標籤。

[*] 注入的文字出現在HTML 文字的前 4096 個字元中。

測試變式 [13]：這個變式的用途是不當運用Internet-Explorer 對於含有 'us-ascii' Content-Type 之回應的處理方式（它會捨棄每個字元的「最高有效位元」）。AppScan 可以變更XSS 內容的各字元最高有效位元，以避開標準輸入消毒函數。例如：%3C 是 "<" 的URL 編碼表示法，在這個攻擊中，會轉換成 %BC。伺服器端消毒函數不會將它視為危險字元，因此，完全不會變更，但Internet Explorer 會將它讀成 "<"，可能就會引發「跨網站Scripting」攻擊。

### 一般修正建議

若干問題的補救有賴於對使用者輸入進行消毒。經由確認使用者輸入未包含危險的字元，便可能防止惡意的使用者讓您的應用程式執行非預期的作業，例如：啟動任意SQL 查詢、內嵌執行於用戶端的JavaScript 程式碼、執行各種作業系統指令等等。

建議濾除下列所有字元：

[1] |（垂直線符號）

[2] &（'&' 符號）

[3] ;（分號）

[4] $（錢幣符號）

[5] %（百分比符號）

[6] @（at 符號）

[7] '（單引號）

[8] "（引號）

[9] \'（反斜線跳出單引號）

[10] \"（反斜線跳出引號）

[11] <>（角括弧）

[12] ()（括弧）

[13] +（加號）

[14] CR（回車，ASCII 0x0d）

[15] LF（換行，ASCII 0x0a）

[16] ,（逗號）

[17] \（反斜線）

下列各節說明各種問題、問題之修正建議，以及可能觸發這些問題的危險字元：

SQL 注入和盲目的SQL 注入：

A. 確定使用者輸入的值和類型（如Integer、Date 等）有效，且符合應用程式預期。

B. 利用儲存程序，將資料存取抽象化，讓使用者無法直接存取表格或視圖。當使用儲存程序時，請利用ADO 指令物件來實作它們，以強化變數類型。

C. 消毒輸入以排除環境定義變更的符號，例如：

[1] '（單引號）

[2] "（引號）

[3] \'（反斜線跳出單引號）

[4] \"（反斜線跳出引號）

[5] )（右括弧）

[6] ;（分號）

跨網站Scripting：

A. 消毒使用者輸入，並濾除JavaScript 程式碼。我們建議您濾除下列字元：

[1] <>（角括弧）

[2] "（引號）

[3] '（單引號）

[4] %（百分比符號）

[5] ;（分號）

[6] ()（括弧）

[7] &（'&' 符號）

[8] +（加號）

B. 如果要修正 <%00script> 變式，請參閱MS 文章 821349

C. 對於UTF-7 攻擊：

[-] 可能的話，建議您施行特定字集編碼（使用 'Content-Type' 標頭或 <meta> 標籤）。

HTTP 回應分割：

消毒使用者輸入（至少是稍後內嵌在HTTP 回應中的輸入）。請確定輸入未包含惡意的字元，例如：

[1] CR（回車，ASCII 0x0d）

[2] LF（換行，ASCII 0x0a）

遠端指令執行：

消毒輸入來排除對執行作業系統指令有意義的符號，例如：

[1] |（垂直線符號）

[2] &（'&' 符號）

[3] ;（分號）

Shell 指令執行：

A. 絕不將未檢查的使用者輸入傳遞給eval()、open()、sysopen()、system() 之類的Perl 指令。

B. 確定輸入未包含惡意的字元，例如：

[1] $（錢幣符號）

[2] %（百分比符號）

[3] @（at 符號）

XPath 注入：

消毒輸入以排除環境定義變更的符號，例如：

[1] '（單引號）

[2] "（引號）

等

LDAP 注入：

A. 使用正面驗證。英數過濾 (A..Z,a..z,0..9) 適合大部分LDAP 查詢。

B. 應該濾除或跳出的特殊LDAP 字元：

[1] 在字串開頭的空格或 "#" 字元

[2] 在字串結尾的空格字元

[3] ,（逗號）

[4] +（加號）

[5] "（引號）

[6] \（反斜線）

[7] <>（角括弧）

[8];（分號）

[9] ()（括弧）

MX 注入：

應該濾除特殊MX 字元：

[1] CR（回車，ASCII 0x0d）

[2] LF（換行，ASCII 0x0a）

偽造日誌：

應該濾除特殊記載字元：

[1] CR（回車，ASCII 0x0d）

[2] LF（換行，ASCII 0x0a）

[3] BS（倒退鍵，ASCII 0x08）

ORM 注入：

A. 確定使用者輸入的值和類型（如Integer、Date 等）有效，且符合應用程式預期。

B. 利用儲存程序，將資料存取抽象化，讓使用者無法直接存取表格或視圖。

C. 使用參數化查詢API

D. 消毒輸入以排除環境定義變更的符號，例如 (*)：

[1] '（單引號）

[2] "（引號）

[3] \'（反斜線跳出單引號）

[4] \"（反斜線跳出引號）

[5] )（右括弧）

[6] ;（分號）

(*) 這適用於SQL。高階查詢語言可能需要不同的消毒機制。