

ASP.NET 防駭指南

作者: 李明儒

ASP.NET 的淺顯易學，讓許多初學者靠著翻書自修就打造起自己的網站王國。可是，有些書上沒教的事，卻可能讓你的網站變成駭客的后院，在本文中，我們就來看看這些常被忽視的網頁安全漏洞

對抗駭客是件堅苦而漫長的工作，就算網站的設計再嚴謹，考慮再周全，只要一個漏洞就可以讓你前功盡棄。因此網站設計人員必須從設計之初就熟知如何避免各種安全上的漏洞，落實在每一行程式中，才能達到“滴水不漏”的境界。

以下我整理了 ASP.NET 程式開發時可能會犯的一些資安錯誤，並說明如何防範。

SQL Injection

所有的網頁漏洞中，SQL Injection(有人翻譯成資料庫隱碼攻擊)算是最廣為人知的，大部分的開發者應該都略有所聞。它不限平台、不限資料庫、也不限程式語言，是一種設計邏輯上的疏失造成的，後果卻相當嚴重。可是很不幸地，它似乎還是駭客事件裡的一哥，用 Google 簡單搜尋一下，就可以找到相當多與 SQL Injection 有關的網站入侵新聞。[見參考資料 1]

舉個例子來說，程式 1 中的程式片段應該會讓很多人大為緊張。(如果你不知道為什麼要緊張，就該認真看完這一節)

程式 1

```
protected void Button1_Click(object sender, EventArgs e)
{
    using (SqlConnection cn = new SqlConnection("Data Source=(local);User Id=sa;
Password=mypassword; Initial Catalog=Lab;"))
    {
        cn.Open();
        string sqlText = "SELECT UserName FROM tblAccount WHERE Account='" + txtUID.Text + "' AND
Password='" + txtPWD.Text + "'";
        SqlCommand cmd = new SqlCommand(sqlText, cn);
        SqlDataReader dr = cmd.ExecuteReader();
        if (dr.Read())
            Response.Write("歡迎登入! " + dr["UserName"].ToString());
        else
            Response.Write("帳號或密碼錯誤!");
        dr.Close();
        cn.Close();
    }
}
```

這段程式碼由 Request 物件中取出使用者輸入的帳號與密碼，並與 tblAccount 中的資料比對，若有資料，就傳回使用者的姓名，並認定為合法使用者。否則就顯示帳號密碼有誤，拒絕其登入。但這段寫法的最大的問題在於“直接將使用者的輸入資料變成 SQL 指令的一部分”，這提供了絕佳的 SQL Injection 管道。

一般而言，駭客先生來到一個陌生的網站，看到輸入帳號密碼的地方，應該就會順手敲入一個單引號，並登入看看。如果看到了如圖 1 的錯誤，便會有發現新大陸的感覺。以圖 1 錯誤訊息中，明確地指出 SQL 指令中有不對稱的單引號，明確地通知程式人員直接將使用者輸入內容組合成 SQL 指令的一部分。而這位苦主犯的第二個大錯是未設定 web.config 中的 customErrors，同時還使用編譯出來的還是 Debug 版本的 DLL，因此 IIS 很“貼心地”送上詳細的錯誤訊息並顯露了一小段錯誤相關程式碼。結果原本為了偵錯方便的貼心設計，當場變成了指引駭客如何入侵的幫凶。

由於看到了程式會檢查 SqlDataReader 有無資料來決定是否為合法使用者，共由資料表中取出 UserName 欄位。因此，只需在 txtPWD Textbox 中輸入 ' OR 1=1 --，程式就會讀取到 tblAccount 中的第一筆使用者，進而放行。

如果駭客只是想搗蛋的話，可以下個 UNION SELECT name as UserName FROM sys.tables --，找出資料表的名稱，藉著 Response.Write 顯示出來，進行資料庫訊息的蒐集，接著隨興玩個 DROP TABLE 之類的把戲，資料庫就糟殃了。

前述程式裡還犯的第三個大錯，是沒為此應用程式另外建立專屬帳號，而直接使用 sa 帳號。sa 帳號具有許多特殊的權限，以 SQL Server 為例，有些很方便(也很危險的)的特殊指令，例如 xp_cmdshell 可以從 SQL 中執行外部的 Windows 程式，而它需要 sa 權限才可以呼叫，這下又為駭客開了方便之門，要偷 Windows 帳號、殖入木馬程式，或是格式化硬碟，悉聽尊便。

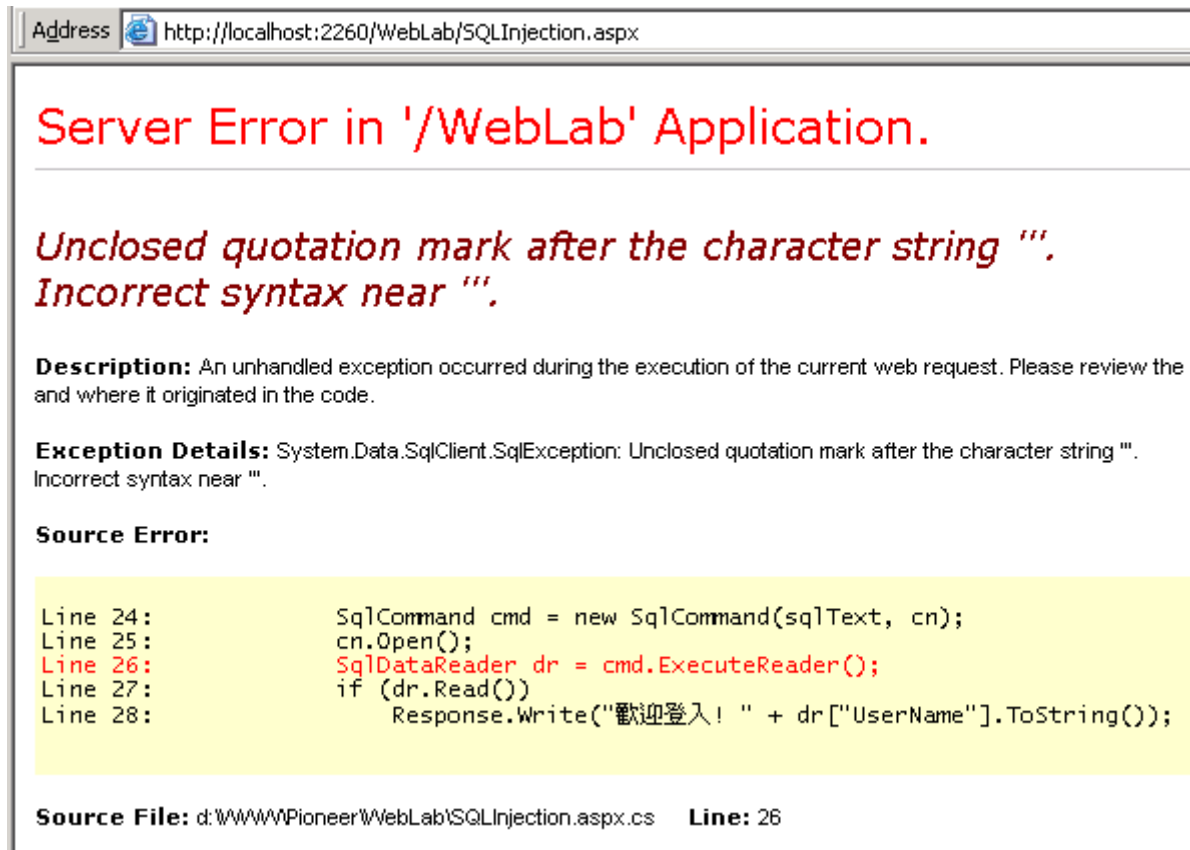


圖 1 明顯的 SQL Injection 漏洞的訊息

那麼，上面這段程式碼，應該怎麼寫才對呢？比較好的做法，是利用 SqlParameter 物件來處理外部輸入的參數(如程式 2)。如此，我們就能隔絕 SQL 指令被入侵的威脅。如果覺得用 SqlParameter 物件太麻煩，最起碼的一件事是，在組合使用者輸入的字串時，記得要將其中的單引號置換成兩個單引號(txtPWD.Replace("'", ""))，但是如果輸入的數字時，則又要做不一樣的檢查與處理，因此使用 SqlParameter 物件算是一勞永逸的做法，如果同樣的 SQL 查詢會重覆多次，還可以享受 SQL 指令編譯一次，使用多次的效能提升。

程式 2

```
cn.Open();
string sqlText = "SELECT UserName FROM tblAccount WHERE Account=@account AND
Password=@password";
SqlCommand cmd = new SqlCommand(sqlText, cn);
cmd.Parameters.Add("@account", SqlDbType.NVarChar).Value = txtUID.Text;
cmd.Parameters.Add("@password", SqlDbType.NVarChar).Value = txtPWD.Text;
SqlDataReader dr = cmd.ExecuteReader();
```

防駭守則:

1. 避免將使用者輸入的任何資料直接組裝成爲 SQL 指令的一部分，應使用 SqlParameter 物件來處理參數。
2. 在正式營運環境下，記得將 web.config 的 customErrors 設爲 On，避免程式出錯時，透露出內部程式碼的相關資訊。
3. 連線資料庫時，避免使用 sa 帳號作爲一般的存取之用。最好建立 Web 存取的專屬帳號，只賦與必要的權限，可以減少遭攻擊時的損害範圍。

Cross-Site-Scripting, XSS

相形之下，XSS 的名氣不如 SQL Injection 來得響亮，但其威脅性卻不容小覷。

所謂 XSS 的定義是，使用者在輸入到網站的文字內容中，夾帶了 Javascript 程式碼。網站顯示這段惡意的內容時，一併執行了殖入的程式碼，而對後續瀏覽這個網頁的使用者造成威脅。雖然我們知道在網頁中 Javascript 無法存取本機的檔案等資源，但它藏於無形，透過精巧的設計，還是可以做出許多可怕的事。例如:

- 透過 DOM 動態更改網頁上的文字、圖檔、連結等
- 將使用者偷偷導向另一個特意仿造的網站，由假網站設法套問姓名、電話、信用卡號等機密資訊
- 讀取該網站儲存在使用者機器上的 Cookie 資料，其中可能隱含機密資訊(姓名、線上購物記錄等等)，取得後送至特定的資訊蒐集網站
- 利用 DOM 物件，取得使用者個人專屬的資訊(帳號名稱、遊戲點數餘額...)送至特定的資訊蒐集網站
- 將 HTML Form Submit 的目的網頁導至惡意網頁，截取使用者原本要送至該網站的機密資訊
- 偽造使用者登入視窗，騙取使用者的帳號、密碼

從 ASP.NET 1.1 起，爲了防範這類的資安風險，ASP.NET 預設會禁止使用者在 INPUT、TEXTAREA 中輸入 <script> 其至任何 HTML Tag。當使用者在 TextBox 中輸入 HTML Tag 時，會出現如圖 2 的錯誤訊息。但有時開發人員爲了讓使用者可以在文字內容中加入 <i> 等等 HTML Tag 提供較豐富的文字格式設定，或許會將網頁或網站的 ValidateRequest 屬性設爲 false。在放行 HTML Tag 的同時，也讓挾帶其中的 Javascript 一併闖關成功了。圖 3 是一個簡單的網頁，上方有按一個 TextBox1，按下 Button1 時，下方的 Label1.Text=TextBox1.Text，停用 ValidateRequest

後，使用者就可在你的網頁上寫作 Javascript 程式了。圖 3 中我們只是簡單地改了 Button1 的按鈕名稱，再 alert 一個訊息，但駭客可不會這麼客氣。

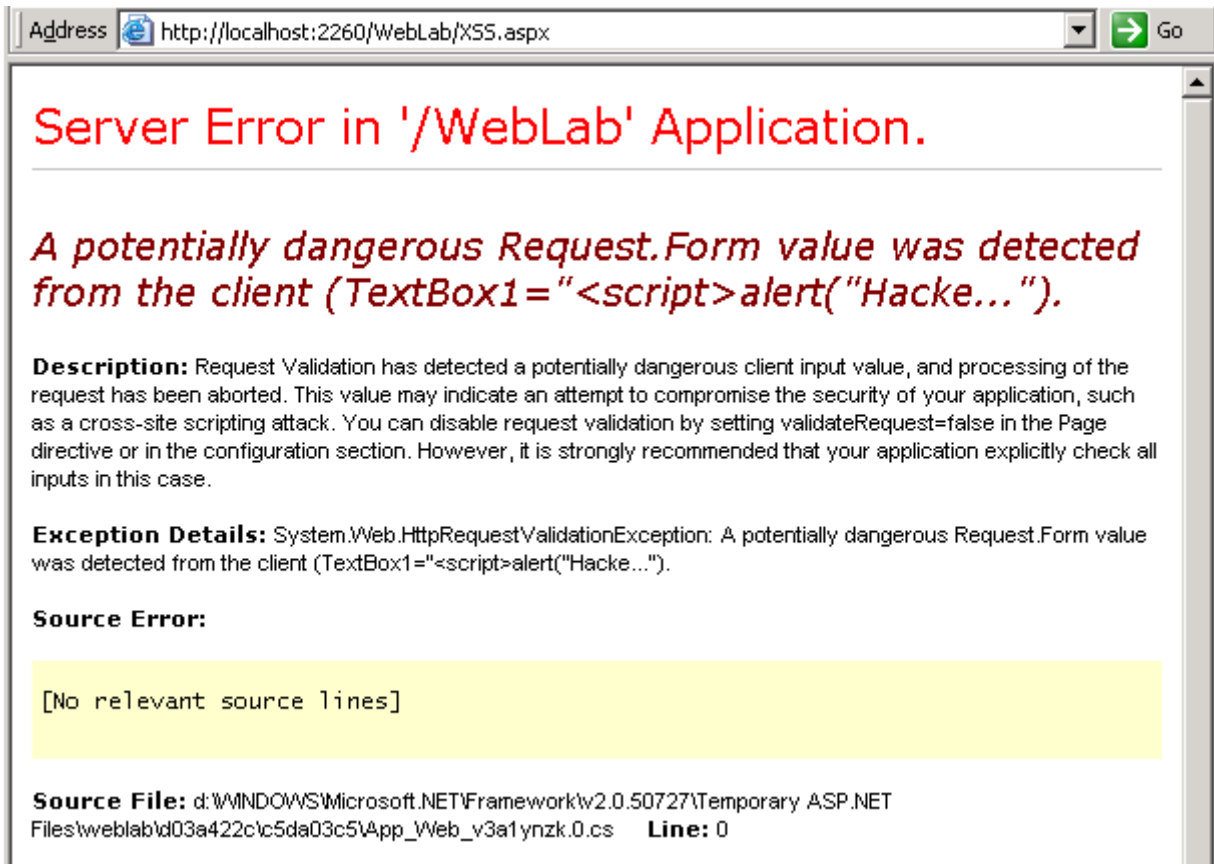


圖 2 ASP.NET 預設防止 XSS 的安全機制

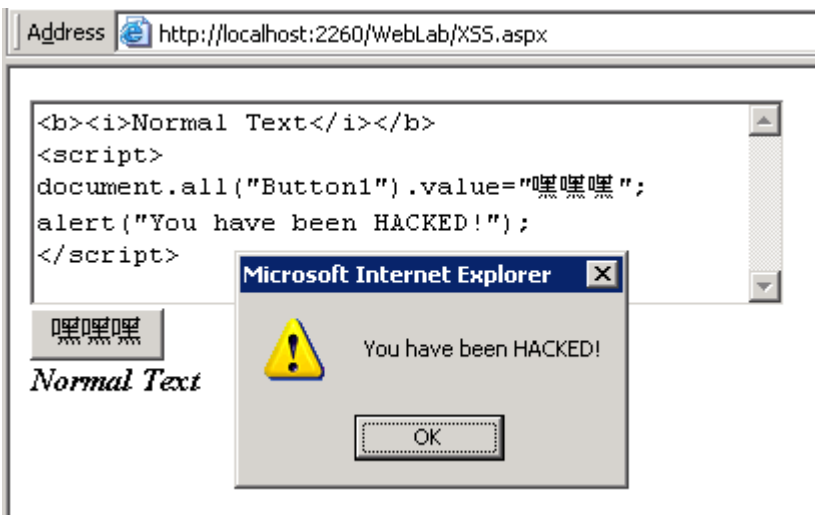


圖 3 XSS 攻擊示範

要防範 XSS 攻擊很簡單，如果文字內容不打算接受 HTML Tag，保持 ValidateRequest 為 true 就好了。若你覺得讓使用者莫名其妙因為輸入 HTML Tag 而發生網頁錯誤有點不 Friendly，可以考量在前端加以阻擋或轉換(網路上可以找到不少 Javascript 版的 HtmlEncode 函數[參 2])，或者將 ValidateRequest 參數打開，然後在 Server-Side 做 HtmlEncode。不過有一種狀況是，如果 TextBox 的內容屬於 RichText 性質，允許<i>等格式相關的 Tag，

則禁用 HTML 是行不通的。一般 RichText Editor 之類的元件會排除具有危險性的 HTML Tag，而危險的 Tag 可不只 <script>而已，不要忘了 <applet>、<object>等也可以拿來做文章，甚至連 在稍作加工後也具有殺傷力，真的要開放 HTML 格式，要更謹慎地處理。微軟有一篇很詳盡的 XSS 防範文件可以參考[參 3]。

防駭守則:

1. 輸入文字時若不接受 HTML Tag，應保持 Page ValidateRequest 屬性於啟用狀態。
2. 要允許使用者輸入 <...> 的文字時，可於前端以 Javascript 進行 HTML Encode 或停用 ValidateRequest 後於 Server-Side 處理。
3. 若必須接受夾帶 HTML 的內容並顯示時，應將具有危險性的 Tag 加以置換。直接引用支援 RichText 的控件 (Control) 是更省事的方法。

Cookie

Cookie 是 Web Server 寄放在 Client 端的一小段資料，可以用來保存與使用者個人相關的設定記錄。在大部分的情形下，Cookie 存放資料的功能可以被 ASP.NET 中的 Session、Cache 或直接存放 DB 取代。但是，有一些狀況下，Cookie 也有其獨到的不可取代性。例如:

- 使用網站伺服器陣列(Web Farm)架構時，由於每次連線的伺服器未必相同，使用 Cookie 儲存使用者狀態資料可以免除在伺服器間共享狀態資料的需求。
- 若儲存資料的性質是跟著機器的，則設定長效型的 Cookie(Persistent Cookie，指瀏覽器關閉後仍可保存在硬碟上一段時間)是很方便直覺的做法。
- 當使用者由一個網站轉接到另一個網站時，只要 Cookie 的 Domain 屬性設定妥當，兩個不同的網站間可以共享 Cookie。這在設定單一登入(Single Sign On)機制時，可以拿來儲存使用者已登入的特別註記，省去在第二個網站的登入程序，十分好用。

應用 Cookie 時，要特別認清一個觀念---Cookie 的內容為何是由瀏覽器夾帶在 Request 中主動告知的，因此有心人可以輕易地模仿或偽造 Cookie 內容，另外，長效式的 Cookie 被存放在磁碟後，也可能被惡意網站或程式竊取。因此，要把握以下原則:

1. 儘可能避免在 Cookie 中存放機密性或隱私資料。
2. Cookie 會被夾帶在瀏覽器每次要讀取網頁的 Request 中，因此資料大小應力求簡短，以節省頻寬。
3. 對 Cookie 內容永遠抱持懷疑的態度，不可盡信而輕易授與權限。
4. 使用加密演算及加上時間戳記(Timestamp)，以提高 Cookie 的安全性及防偽性。

以 ASP.NET 內建的表單式認證機制(Form Authentication)為例，也是採取寫入 Cookie 的方式註記登入狀態，而 web.config 中 forms 設定中，即可使用 protection 屬性決定是否要啟用加密保護[參 3]，足見加密已是 Cookie 應用時的基本要求了。

字串加密

我們應盡可能減少將機密性資料寫入媒體的機會，以降低資料外洩的風險，但寫入機密資料的需求沒辦法百分之百避免。在必須將機密資料保存在檔案或其他媒體的場合，對資料加密可提供有效的保護作用。在 ASP.NET 應用中，最常要面對的機密資料就是資料庫的連線字串(Connection String)，除非使用 SQL Server 的 AD 整合式身

份認證(Integrated Security=SSPI); 否則就少不了要在連線字串中指定帳號、密碼。然而,我看過不少的設計,連線字串就大刺刺地寫明在 web.config 中,雖然說 Windows 與 IIS 對 web.config 做了保護,一般人不容易輕易取得。但是,這些檔案有可能因檔案備份等機制的運行,提高了曝光的風險,因此我一向主張連線字串必須要加密才安全。這個概念倒是在業界仍未建立一致的共識,一些 Open Source Project 例如: log4net、Community Server 等也還是用明碼存放連線字串;有些產品如 SQL Reporting Service,連線字串則是加密過才儲存。

有了資料要加密的觀念,接著需要一個可以簡便完成加解密的共用元件。雖然.NET 已內建了 DES、RSA 等等業界標準的加密演算法,安全性絕對足夠,但是要操作這些演算函數,得搞懂 Initial Vector、Key 等密碼學術語,有點小煩。而我們期望的加解密函數最好能簡化成傳入原始字串及通關密語就產生加密後的字串,然後只要提供通關密語及加密字串,就可解密回原始字串。要做好資安,這類方便的加解密函數是不可或缺的。以下就是一個以 DES 演算法為基礎加解密程式範例。

程式3 Cipher.cs

```
using System;
using System.Security.Cryptography;
using System.Text;
using System.IO;

public struct DESKeyPack
{
    public byte[] Key, IV;
    public DESKeyPack(byte[] data)
    {
        Key = new byte[8];
        Buffer.BlockCopy(data, 0, Key, 0, 8);
        IV = new byte[8];
        Buffer.BlockCopy(data, 8, IV, 0, 8);
    }
}

public class Cipher
{
    private static DESKeyPack genKeyPack(string keyString)
    {
        //可以在不同的版本設不同的SALT值
        //則不同版本的程式不能用來解密
        const string salt = "SALT";
        MD5CryptoServiceProvider md5 = new MD5CryptoServiceProvider();
        byte[] data = md5.ComputeHash(
            UTF8Encoding.UTF8.GetBytes(keyString + salt)
        );
    }
}
```

```
md5.Clear();
DESKeyPack dkp = new DESKeyPack(data);
return dkp;
}

public static string Encrypt(string rawString, string keyString) {
    DESKeyPack dkp = genKeyPack(keyString);
    DESCryptoServiceProvider des = new DESCryptoServiceProvider();
    ICryptoTransform trans = des.CreateEncryptor(
        dkp.Key, dkp.IV
    );
    MemoryStream ms = new MemoryStream();
    CryptoStream cs = new CryptoStream(ms, trans, CryptoStreamMode.Write);
    byte[] rawData = UTF8Encoding.UTF8.GetBytes(rawString);
    cs.Write(rawData, 0, rawData.Length);
    cs.Close();
    return Convert.ToBase64String(ms.ToArray());
}

public static string Decrypt(string encString, string keyString)
{
    DESKeyPack dkp = genKeyPack(keyString);
    DESCryptoServiceProvider des = new DESCryptoServiceProvider();
    ICryptoTransform trans = des.CreateDecryptor(
        dkp.Key, dkp.IV
    );
    MemoryStream ms = new MemoryStream();
    CryptoStream cs = new CryptoStream(ms, trans, CryptoStreamMode.Write);
    byte[] rawData = Convert.FromBase64String(encString);
    cs.Write(rawData, 0, rawData.Length);
    cs.Close();
    return UTF8Encoding.UTF8.GetString(ms.ToArray());
}
}
```

程式 4 函數呼叫範例

```
protected void Page_Load(object sender, EventArgs e)
{
    string rawString="Hello World!";
    string encString = Cipher.Encrypt(rawString, "MyKey");
}
```

```
Response.Write("<br>ENC=" + encString);  
string decString = Cipher.Decrypt(encString, "MyKey");  
Response.Write("<br>DEC=" + decString);  
}
```

顯示結果:

ENC=kGdj4wLUX1NVd5BGB3w/iA==

DEC=Hello World!

針對 web.config/machine.config 中有些設定可能涉及帳號密碼，例如: <identity userName= password= /> <processModel userName= password= /> <sessionState stateConnectionString= sqlConnectionString= />等。如果不希望機密大刺刺地寫在配置檔案裡，微軟提供了 AspNet_setreg.exe 程式，可將帳號密碼改儲存於 Registry 裡，詳細的做法見參考資料 4。更進一步的，從 ASP.NET 2.0 起，web.config 中如 connectionStrings 也可以使用 DAPI、RSA 等方式加密，既方便又有保障，詳細的做法可以參考[參 4/參 5]。

上傳檔案處理

網站系統中，少不了讓使用者上傳檔案的應用，一旦設計不當，也會讓你的網站危機四伏。

上傳檔案可能引發危機的原理跟 SQL Injection、XSS 相同，都來自於“過分信任使用者的輸入資料”!

大部分的 HTTP 上傳程式的寫法，會將上傳的檔案，寫入特定的網站目錄，未來再讓使用者以 URL 方式直接連向這些檔案。這種做法最大的問題在於使用者會上傳何種檔案內容無法預期，若在上傳時未加攔阻檢核，寫入的網站目錄又未做好執行權限制，就可能出大問題。為避免使用者傳個 ASP 或 ASPX 檔案，就把你的網站伺服器當成自己的來使喚，記得一定要移除上傳檔案目錄的 Script 或 EXE 檔執行權限。即使只留下讀取權限，若使用者上傳了夾帶惡意 Client-Script 的 HTML，仍然會變成 XSS 攻擊的大漏洞。根本的解決之道應限制上傳檔案的檔案類型，多數上傳的應用集中在圖檔、附檔，因此可以限制副檔名為 jpg, gif, png, doc, xls, ppt, pdf 等等才接受，其餘有風險的格式，一律要求壓成 ZIP 檔後再上傳。除了在上傳檔案時對檔案類型做限制，還有另外一種解決方式，就是將檔案內容寫入資料庫或網站範圍以外的目錄，當使用者要讀取檔案時，不能直接以 URL 指向該檔案，而需要透過一隻中間程式或 HttpHandler，讀取檔案內容後以 Response.BinaryWrite 方式將結果傳回，如此可完全杜絕使用者上傳的檔案被執行的機會(將 MIME Type 設成 application/octet-stream，則 HTML 檔亦會被下載後才開啓，減少 XSS 攻擊的風險)。只是此種做法，程式較為複雜，且由於使用者取得檔案的過程隔了一手，對效能會略有影響，算是為了資安付出的代價之一吧!

防駭守則:

1. 針對用途限制使用者可上傳的檔案類型
2. 上傳檔案儲存的網站目錄限定只開放讀取，不允許 Script 或 EXE 檔執行(但仍有 XSS 攻擊的風險)
3. 考慮透過下載程式或 HttpHandler 間接取檔

ASP.NET 安全提示

MSDN patterns & practices 中有一個專門的章節在談 .NET 的資訊安全[參 7]，是本份量十足的原文大作，可說是 .NET 安全的百科全書，建議有空花點時間瀏覽一下。以下則整理一些關於 ASP.NET 安全相關的建議事項:

1. ASP.NET 預設的執行身份會用 ASPNET(Windows XP/2000)或 Network Service(Windows 2003)等特定帳號，這類帳號的權限都設得很低，以降低程式失控或被駭時的風險。如果因特殊需求要透過 web.config 的<identity>另外指定執行身份，切記要把握權限愈少愈好的原則，尤其要避免設成 SYSTEM 或 Administrator 身份。設定 ASP.NET 執行帳號的注意事項可以參考 MSDN 文件。[參 8]
2. 在正式環境中，記得將設定<customErrors>為 RemoteOnly 或 On，可以避免發生錯誤時透露了不必要的程式細節，成為指引駭客攻擊的線索。<compilation>的 debug 屬性及<trace>的 enabled 屬性也請設為 false，一樣可以避免不必要的程式資訊洩漏，同時對效能也有幫助。[參 9]
3. ASP.NET 2.0 內建了完整的 Authentication、Membership、Role 機制，應用簡便，也具有一定的安全性。甚至直接在 Code 中以 PrincipalPermissionAttributes 宣告即可套用，完全不必寫 Code[參 10]，不妨多加利用。
4. 不要相信任何做在 Client 端的安全機制，有心人可以輕易破解。所以重要的表單欄位驗證(Form Validation)應該在前後端各檢核一次，使用 ASP.NET 既有的 Validator 是不錯的選擇。
5. 預設的 ViewState 內容只是編碼，並未加密，所以不能排除它被篡改的可能性。我們可以透過 enableViewStateMac 屬性為它加上 Hash 仿偽，也可以設定 viewStateEncryptionMode 對 ViewState 內容加密。在伺服器陣列(Web Farm)上要加密 ViewState 時，記得要統一設定 validateKey。最後，MSDN 上還有一份 ASP.NET 安全檢核清單，頗具參考價值。[參 11]

結語

與駭客的對抗是場永不休止的戰爭，不斷有新的攻擊花招誕生，也持續有新的防禦機制問世。身為防守方的網站開發人員，永遠保持高度的警覺，並時時留意資訊相關的訊息，在這場正邪大戰中存活的不二法門。資安相關的議題十分廣泛，本文只提供了一些指引，說明基礎的觀念，文中所列的參考資料都是值得進一步探索的入口，希望對大家有所助益。

參考資料

1. 採用 SQL 隱碼攻擊的駭客入侵新聞
<http://www.ettoday.com/2006/01/19/91-1896211.htm>
<http://www.ettoday.com/2002/04/22/339-1293697.htm>
<http://forum.icst.org.tw/phpBB2/viewtopic.php?t=4319>
<http://www.informationsecurity.com.tw/news/view.asp?nid=2483>
2. Javascript 版 HtmlEncode 函數
http://lunarmedia.com/blogs/lunarmedia_blog/archive/2006/10/23/120405.aspx
<http://www.tumuski.com/code/htmlEncode.php>
3. How To: Prevent Cross-Site Scripting in ASP.NET
<http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000004.asp>
4. Web.config 中關於表單認證 Cookie 的加密設定
[http://msdn2.microsoft.com/en-us/library/1d3t3c61\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/1d3t3c61(VS.80).aspx)
5. 如何使用 ASP.NET 公用程式加密認證及工作階段狀態的連接字串

<http://support.microsoft.com/kb/329290/zh-tw>

6. How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI

<http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000005.asp>

7. How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA

<http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000006.asp>