

*SQL*資料隱碼攻擊(*SQL Injection*) 檢測與防護

行政院國家資通安全會報
技術服務中心

Part 1: *SQL Injection* 簡介

什麼是SQL?

- *SQL* 全稱是結構化查詢語言(*Structured Query Language*)，是用於資料庫中的標準數據查詢語言。
- *SQL* 語言功能大致上可分為4個部分：
 - 數據查詢語言 (*SELECT* 語句)
 - 數據操縱語言 (如 *INSERT*, *UPDATE*, *DELETE* 等語句)
 - 數據定義語言 (如 *CREATE*, *DROP* 等語句)
 - 數據控制語言 (如 *COMMIT*, *ROLLBACK* 等語句)
- 各種通行的資料庫系統在實作時都對 *SQL* 規範作了某些編改和擴充 (如：*SQL Server* 系列資料庫所用的 *Transact-SQL* 以及 *Oracle* 資料庫所使用的 *PL-SQL*)。所以實際上，不同資料庫系統之間的 *SQL* 語言不能完全相互通用。

何謂 *SQL Injection*?

- *SQL Injection* 又稱為 *SQL* 資料隱碼攻擊、隱碼攻擊、*SQL* 注入等，是發生於應用程式層的安全漏洞。
- *SQL Injection* 利用現有應用程式未妥善過濾與處理輸入之資料，便傳遞給後端資料庫伺服器執行，使得有心人士可將 *SQL* 指令夾帶於資料中，那麼這些夾帶進去的指令就會被資料庫伺服器誤認為是正常的 *SQL* 指令而執行。

*SQL Injection*影響範圍及危害

- 只要應用程式提供資料輸入介面且將使用者輸入之資料送至資料庫進行處理，即可能遭受*SQL Injection*的威脅，最常見符合上述條件的即為網頁應用程式。
- *SQL Injection*可能造成的危害包含：資料庫資料遭竄改、網頁遭竄改，甚至遭植入惡意程式！

Part 2: *SQL Injection* 攻擊範例



- 以下說明為教學講解用途，嚴禁測試他人網站系統，否則造成任何法律糾紛請自行負責。

使用者身分驗證

- 一般網站進行使用者輸入帳號密碼驗證的SQL語法（以ASP為例）：

```
If Request("Username")<>"" And Request("Password")<>"" Then
```

```
Dim cnn,rec,strSQL
```

```
Set cnn=Server.CreateObject("ADODB.Connection")
```

```
With cnn
```

```
.ConnectionString=Application("Conn")
```

```
.Open
```

```
strSQL="SELECT * FROM tblUser WHERE Username='" & Request("Username") & "' AND  
Password='" & Request("Password") & "'" '利用使用者輸入的資料來組合 SQL 語法
```

```
Set rec=.Execute(strSQL) '直接交給資料庫執行，這是最危險的地方
```

```
End With
```

```
If NOT rec.EOF Then
```

```
Session("Username")=Request("Username")
```

```
Response.Write "歡迎登入 " & Request("Username")
```

```
Else
```

```
Response.Write "帳號/密碼輸入錯誤"
```

```
End If
```

```
.....
```


使用者身分驗證

- 一般網站進行使用者輸入帳號密碼驗證的 SQL 語法：

```
strSQL="SELECT * FROM tblUser WHERE  
UserName='" & Request("Username") & "'  
AND Password='" & Request("Password") & "'"
```

管理者登入

Username:

Password:

登入

利用任何已知的使用者名稱登入

- 若攻擊者已知系統中已有一個Admin的管理者帳號，則於帳號欄位輸入Admin'--，密碼欄位隨便輸入即可，對於會被執行的SQL語法沒有什麼影響。
- 則strSQL將變成：
*SELECT * FROM tblUser WHERE Username='Admin'--' AND Password='xx'*
註：--符號後的敘述都會被資料庫系統當作註解
- 故對資料庫而言，所執行的SQL語法將為：
*SELECT * FROM tblUser WHERE UserName='Admin'*
- 若有Admin存在，則這個SQL查詢語法就傳回該記錄的所有欄位內容，按照範例程式碼的撰寫方式，駭客即可輕易以Admin身分進入。

用未知的使用者名稱登入

- 若沒有已知的帳號，可以於使用者名稱欄位輸入 '*or 1=1--*，而密碼欄位隨便輸入即可，對於會被執行的SQL語法沒有影響。
- 則*strSQL*將變成：
*SELECT * FROM tblUser WHERE Username=" or 1=1-- AND Password='xx'*
- 故對資料庫而言，所執行的SQL語法將為：
*SELECT * FROM tblUser WHERE Username=" or 1=1*
- 因為加上的 *or 1=1*，使得WHERE判斷式結果必定為真，因此回傳的 *Recordset* 物件包含了全部的會員記錄，按照範例程式碼的撰寫方式，則駭客即可用*tblUser*中，第1筆資料的使用者身分登入。

其他攻擊指令

- *SQL Injection*攻擊指令，網路上已有許多相關資料，雖然指令種類繁多且隨資料庫種類不同而稍有變化，但原理上均是以此類方式，於資料輸入欄位中夾帶*SQL*指令，欺騙後端資料庫執行。
- 其他攻擊手法介紹可參考：
 - *SQL Injection* (資料隱碼)– 駭客的*SQL*填空遊戲(上)
http://www.microsoft.com/taiwan/sql/SQL_Injection_G1.htm
 - *SQL Injection* (資料隱碼)– 駭客的*SQL*填空遊戲(下)
http://www.microsoft.com/taiwan/sql/SQL_Injection_G2.htm

Part 3: *SQL Injection* 漏洞檢測

SQL Injection漏洞檢測方式

- 檢測方式大致上可分為白箱測試及黑箱測試，說明如下：
 - 白箱檢測(原始碼檢測)
 - 需取得網頁程式原始碼。
 - 檢視原始碼中所有關於資料庫存取的部分，判斷程式是否針對資料進行適當處置後，才送給資料庫管理系统執行。
 - 檢測準確率較黑箱測試高，但若以人工方式進行則非常耗時費力。
 - 可利用原始碼檢測軟體輔助人工檢測，但這類軟體往往非常昂貴，且僅支援特定網站開發語言，而檢測準確率也因產品不同而異。

*SQL Injection*漏洞檢測方式(續)

一 黑箱檢測

- 較白箱測試準確率低，但不需取得原始碼進行檢測，為駭客使用的檢測方式。
- 針對有存取資料庫之網站頁面進行檢測。
- 以特定測試字串，檢測網站應用程式是否有對使用者輸入之資料進行適當處置。
- 必須猜測程式所使用的SQL語法，並視回應訊息調整測試字串。
- 人工檢測耗時費力，同樣可使用軟體輔助人工檢測，但此類黑箱測試軟體誤判率高。

各種檢測方式之優缺點比較

	人工原始碼檢測	軟體原始碼檢測	人工黑箱測試	軟體黑箱測試
效率	低	高	低-中	高
準確	高	低-高 (視檢測技術而定)	低-中	低
備註	需取得程式原始碼		不需原始碼即可檢測	

備註：此表內容為一般狀況下之歸納推論，僅提供作為參考，實際狀況視檢測軟體及網站平台而異。

檢測網頁應用程式是否存有 *SQL Injection*漏洞

- 本簡報主要介紹*SQL Injection*人工黑箱測試技術，使用自動化軟體檢測工具及人工原始碼檢視方式則暫不探討。
- 人工黑箱測試技術主要是檢測網頁程式是否對使用者輸入資料進行適當處置後，方才交由資料庫處理，藉以判定網頁程式是否有*SQL Injection*漏洞。
- 由於各種資料庫管理系统所支援之*SQL*語法及資料庫操作命令均有異同，故實務上難以逐一測試各種資料庫指令是否有過濾。
- 以下將介紹一些普遍適用於大部分資料庫的非破壞性檢測手法，但由於黑箱測試的限制，故即使通過這些檢測手法的網頁，也不代表一定沒有*SQL Injection*漏洞（若要確保沒有*SQL Injection*漏洞需採用白箱檢測）。

檢測範圍

- 進行*SQL Injection*漏洞檢測時需檢查**所有動態網頁**的：
 - 使用者輸入介面
 - 網頁上的表單(*Form*)物件。
 - 可以用來群組其他的網頁物件，進行資料的輸出入。
 - 常見的表單有：使用者登入介面、會員資料填寫頁面、活動報名頁面等...
 - *URL*列參數
 - 通常用以傳遞網頁資料。

使用者輸入介面

請輸入帳號及密碼	
帳號	<input type="text"/>
密碼	<input type="password"/>
<input type="button" value="清除"/>	
<input type="button" value="登入"/>	

- 按下登入的動作，有兩種方法將表單中的輸入欄位資料傳送給網站伺服器：
 - *method=POST*
 - 輸入資料不會顯示在URL列上
 - *method=GET*
 - 輸入資料會顯示在URL列上

URL列參數

- URL範例：
 - <http://www.google.com.tw/search?q=web+Form&hl=zh-TW&hs=l31&lr=&cr=countryTW&client=firefox-a&rls=org.mozilla:zh-TW:official&start=20&sa=N>
- 「？」表示後面接參數名稱及資料，若有多個參數則以「&」隔開。
- 上面的例子表示將「？」後的參數資料送給 <http://www.google.com.tw/> 上的 *search* 程式處理。
- 本範例共傳遞了「*q*」、「*hl*」、「*hs*」、「*lr*」、「*cr*」、「*client*」、「*rls*」、「*start*」、「*sa*」等9個參數，參數的值為「=」後面所接的字串。

檢測原理

- 黑箱檢測原理為輸入SQL語法及特殊符號（如單引號「'」），並依據網頁回傳訊息判斷SQL語法或特殊符號是否直接被送入資料庫處理。
- 較常使用的測試語法有：
 - 檢測數字型別欄位使用「*or 1=1*」搭配「*or 1=2*」（視狀況改用 *and* 取代 *or*）
 - 檢測字串型別欄位使用「'*or '1'='1*」搭配「'*or '1'='2*」（視狀況改用 *and* 取代 *or*）

檢測範例-表單

- 實測案例：



車號	進場日期	違規法條
該車未被拖吊		

- 於上圖車號查詢欄位中填入「'」，回傳頁面為：

'/' 應用程式中發生伺服器錯誤。

Runtime 錯誤

- 回傳訊息為伺服器錯誤，研判「'」已被送入資料庫執行，故合理推測該欄位存在SQL Injection漏洞。

備註：錯誤訊息頁面視網頁伺服器類型或其他狀況而有不同，原則上只要是回傳內部伺服器錯誤的訊息（HTTP Error 500 - Internal Server Error）皆可能代表該頁面存在弱點。

檢測範例-表單

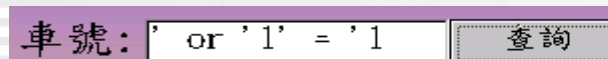
- 若欲再進一步確認該頁面是否存在 *SQL Injection* 漏洞，可於測試欄位中輸入「' or '1'='1」。
- 但由於該欄位有輸入長度限制，僅可填入8個字元，故無法完整輸入測試字串。

車號:	<input type="text" value="' or '1'"/>	<input type="button" value="查詢"/>
車號	進場日期	違規法條
該車未被拖吊		

檢測範例-表單

- 檢視HTML原始碼後發現限制欄位輸入長度功能寫在client端：

```
<span id="lblMesgBox" style="font-family:標楷體;font-size:Medium;height:29px,width:592px,Z-INDEX:
<span id="Label2" style="font-family:標楷體;height:32px,width:184px,Z-INDEX: 110; LEFT: 416px; POSI
<a id="hlMemo" href="Rule_momo.aspx" style="height:32px,width:104px,Z-INDEX: 108; LEFT: 544px; POS
<a id="hlCI" href="http://59.124.206.221/" style="height:32px,width:120px,Z-INDEX: 109; LEFT: 416px; POS
<span id="Label5" style="font-family:標楷體;height:24px,width:232px,Z-INDEX: 107; LEFT: 416px; POSI
<input name="txtPlt_no" type="text" value="" maxlength="8" id="txtPlt_no" style="font-family:標楷體,f
<input type="submit" name="cmdQuery" value="查詢" id="cmdQuery" style="font-family:標楷體;font
src="map1.jpg" height="233" width="320"></FONT>
```
- 將maxLength屬性移除後，將原始碼另存成HTML檔後執行，即可繞過欄位長度限制。



車號:

檢測範例-表單

- 執行「' or '1'='1」，即可檢視所有資料：

車號:	<input type="text" value="' or '1'='1"/>	<input type="button" value="查詢"/>
車號	進場日期	違規法條
██████	960415	5610101
██████	960629	5610402
██████	960504	5610402
██████	960301	3510101
██████	960710	3510101
██████	960830	5610101
██████	960612	5610101

- 反之輸入「' or '1'='2」執行，則看不到任何資料，故可判定，此頁面有SQL Injection漏洞。

車號:	<input type="text" value="' or '1'='2"/>	<input type="button" value="查詢"/>
車號	進場日期	違規法條
該車未被拖吊		

檢測範例-URL列參數

- 觀察一原始網頁URL：<http://127.0.0.1/ASP/SQLInji.asp?txtCustID=1234>



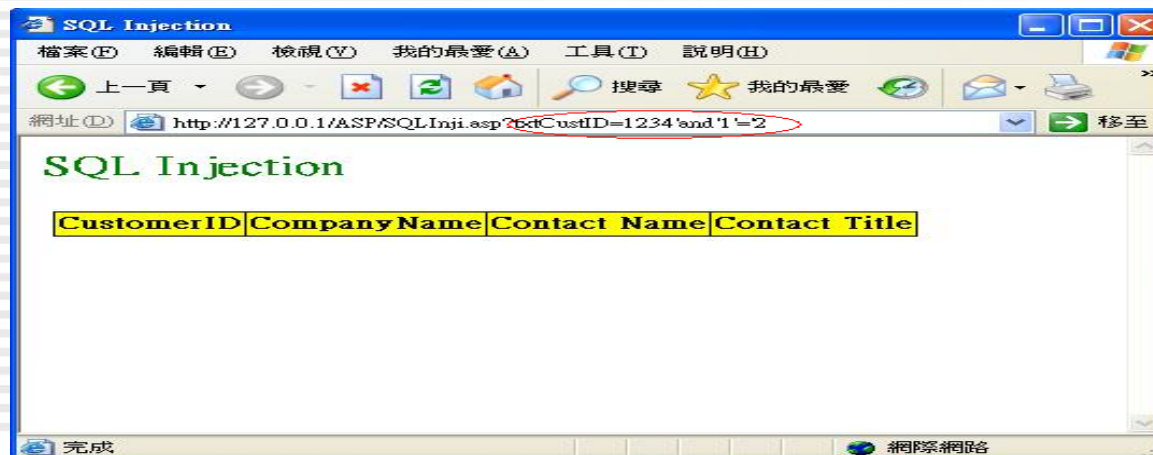
- 得知SQLInji.asp程式接受一個傳入參數txtCustID，目前傳入值為1234

檢測範例-URL列參數

- 於網址列輸入：*http://.../SQLInji.asp?txtCustID=1234' and '1'='1*



- 於網址列輸入：*http://.../SQLInji.asp?txtCustID=1234' and '1'='2*



檢測範例-URL列參數

- 由「*' and '1'='1*」及「*' and '1'='2*」測試結果可得知，測試語法已送入資料庫執行，導致回傳頁面不同，故合理推測此頁面有*SQL Injection*漏洞。
- 在這個案例中為什麼要改用*and*而非用*or*？

其他檢測技巧

- 以其他編碼方式進行檢測
 - 如：以「%27」（'的URL編碼）取代「'」進行檢測，規避網頁程式過濾「'」的情形。
 - Ex：
`http://.../SQLInji.asp?txtCustID=1234%27 and %271%27=%271`
- 大小寫混用
 - 如：將select改成SeLeCt，規避網頁程式過濾select的情形。
- 使用破壞性檢測
 - 如停止資料庫運作、刪除資料表格等...此類檢測方式準確度更高，但將對受測系統造成實質上的破壞。

Part 4: *SQL Injection*防護

SQL Injection防護

- 加強程式參數（包含所有表單及網址列參數）的安全檢查機制：
 - 取代或過濾下列符號「'」、「"」、「%」、「;」、「*」、「#」、「&」、「+」、「-」、「<」、「>」、「xp_」。
 - 當參數資料具有特定長度性質或可預估大約長度時（如：身分證字號、資料編號、姓名、住址等...），則應限制長度。
 - 若參數資料預估應均為數字符號（0~9），則應檢查其中是否包含非數字字元。
 - 以上過濾條件程式均必須寫在伺服器端，不可僅寫在客戶端（*client side script*）。

*SQL Injection*防護(續)

- 做好正確錯誤處理：
以程式攔截下任何作業系統或資料庫管理系統所可能產生的錯誤訊息，改以統一且明確的訊息來指導使用者，以免惡意使用者由系統錯誤訊息中獲取過多資訊。
- 透過預儲程序(*stored-procedure*)來存取或查詢資料庫。

*SQL Injection*防護(續)

- 加強資料庫帳號與權限管理：
讓網站或軟體以非系統管理者的帳號連結資料庫，且對每個資料庫設定一組個別的帳號與強健的密碼，限制這組帳號僅能對該資料庫有讀寫權限。當面臨*SQL Injection*的侵入時，分權管理能夠限制損害的範圍，降低損失。

*SQL Injection*防護(續)

- 刪除多餘的資料表：
資料庫管理系統安裝後會預設安裝幾個資料表，這些資料表是供程式開發、範例資料庫或暫存使用（如*pub*及*Northwind*等...），請在網站上線後移除，因為這些資料表架構是公開的，易遭駭客利用。