利用 XML 驗證之電子商務網站安全防護架構

陳彥錚，國立暨南國際大學資訊管理學系

林上傑，國立暨南國際大學資訊管理學系

范恭達，國立暨南國際大學資訊管理學系

林錦雲，國立暨南國際大學資訊管理研究所

摘要

過去電子商務安全研究多注重資料通訊的私密性，然而許多電子商務網站即使採用 SSL 或 SET 電子安全交易機制，交易安全資料被竊取或篡改的情形仍時有所聞，主要原因不在於加密機制不夠安全，而是電子商務網站應用程式本身的安全漏洞所致。這些漏洞多由於網站應用程式並沒有從安全的角度嚴謹地驗證網站輸入資料，使得惡意攻擊者能趁虛而入，竊取或篡改交易資料。資料隱碼攻擊為其典型的例子，類似的攻擊尚包括跨網站命令稿、更改標價攻擊、以及毒餅乾等。

每個網站應用程式設計目的不盡相同，很難使用一致的輸入檢查程式避免上述各式攻擊。本論文提出一個利用 XML Schema 驗證技術的網站安全防護架構，網站開發者只需使用標準的 XML Schema 文件作為網站安全政策描述語言，描述網頁輸入資料的屬性，此防護機制便能自動對輸入資料進行驗證。位於 Web 伺服器與應用程式之間的防護機制將輸入資料轉換為 XML 文件，然後利用 XML 程式本身的驗證功能判斷有無應用層級的安全攻擊。我們並開發一個簡便的 XML Schema 文件產生器，方便網站開發者產生 XML Schema 文件，因此本論文所提網站安全防護機制，提供網站開發者在網站安全上一個有效又方便的保障。

關鍵字：電子商務安全、資料隱碼攻擊、XML Schema、輸入驗證

壹、簡介

電子商務安全是電子商務成功的關鍵因素，過去許多電子商務安全相關研究多注重資料通訊的私密性，避免客戶信用卡號碼或電子商務交易資料在網路上傳遞時被竊取，因此目前許多電子商務網站都採用 SSL (Secure Socket Layer)或 SET (Secure Electronic Transaction)電子安全交易機制，以確保電子交易的安全。儘管如此，電子商務網站安全資料被竊取或篡改的情形卻仍無法避免，究其原因，並不在於所採用的電子交易機制不夠安全，而是存在於電子商務網站應用程式本身的安全漏洞所致。這些漏洞多由於網站應用程式過於信任由用戶端傳回的資料，並沒有從安全的角度嚴謹地驗證網站輸入資料，使得惡意攻擊者能趁虛而入，竊取或篡改交易資料。以 2002 年 4 月在國內受到高度重視的資料隱碼攻擊(SQL Injection)為例，即是網站應用程式未做好輸入檢查工作，使得惡意攻擊者可循正常的表單輸入安插不正當的資料庫查詢指令所引起。攻擊者可以透過此應用層的安全漏洞，藉由伺服器執行所夾帶的資料庫查詢指令，輕易地入侵銀行、電子商務網站與國家政府機關的資料庫系統，對電腦資料庫中必需保密的內容帶來極大的威脅。根據 2003 年年初的「開放網路應用安全計畫」（OWASP, Open Web Applications Security Project）報告[8]指出，首要的網站安全漏洞為便是網頁傳遞未經驗證的資訊(Unvalidated Parameters)，此報告同時也指出因未對網頁輸入未做嚴謹驗證所造成的安全漏洞尚有隱碼攻擊、跨網站的命令稿(Cross-Site Scripting)、以及緩衝溢位(Buffer Overflow)。另外，在電子商務上利用改變表單隱藏欄位資料的更改標價攻擊(Price-Change Attack) [11]、以及改變連線狀態資訊的毒餅乾(Poisoned Cookie)攻擊也都是利用網站應用程式本身安全漏洞所造成的。

儘管以上這些安全漏洞陸續被揭發，網站安全議題日益受到重視，然而探討如何提供有效的防護機制保障網站應用安全的研究仍然不多。這可能是因為每個網站應用程式之設計各有其邏輯，網頁間傳遞資料目的不盡相同，每個傳遞參數之型態與長度限制也有所差別，因此針對每一個網頁所設計用來保護網站安全的的輸入驗證程式勢必不同。在這樣面對眾多異質的網站應用程式，如何提供一個有效的網站安全防護機制是一個值得研究的課題。在探討此課題時，我們首先須知道各網頁傳遞資料的相關屬性，方能據此提供正確的安全防護功能。Scott 與 Sharp [11]首先提出一個使用 XML 所設計的網頁安全政策描述語言(Security Policy Description Language, SPDL)，網站應用開發者可依據 SPDL 規格使用 XML 撰寫一份 SPDL 文件，描述網頁傳遞資料相關屬性與限制。SPDL 文件必須經過

專屬的編譯器解譯，方能讓採用 SPDL 的網站安全防護機制了解如何進行輸入驗證工作。由於 SPDL 不易使用且編譯器開發不易，本論文提出一個不需使用 SPDL 與編譯器的網站安全防護機制，我們發現在 XML 標準中用來驗證 XML 文件正確性的 XML Schema 可以取代 SPDL 作為網頁安全政策描述語言，網站開發者先想像網頁輸入資料是置於一個特殊的 XML 文件中，再針對這個刻意製造出來的 XML 文件設計對應的 XML Schema 文件，此 XML Schema 文件即成為一份網頁安全政策描述文件。由於 XML Schema 已成為 XML 標準中不可或缺的成員，越來越多的人熟悉此語言，因此使用 XML Schema 作為安全政策的描述語言遠比 SPDL 容易被大眾接受。此外，我們發現使用 XML Schema 的另一更重大的好處是不需要使用任何政策編譯器，我們只需將網頁傳遞資料轉換成一份簡短的 XML 文件，再載入對應的 XML Schema 文件，便可利用一般的 XML 處理程式所提供的 XML 驗證功能，直接判斷輸入資料中是否包含具安全威脅的資料。因此，在無需任何複雜編譯器的要求前提，任何網站開發者都可以很容易地利用我們所提的方式進行簡單的網站安全防護。

過去相關的網站安全防護研究，除了研究網站安全政策描述語言外，尚探討網站安全防護架構，現有的網站安全防護架構有採用應用層防火牆的安全閘道器架構，也有直接產生程式碼嵌入於網站應用程式的作法。針對此議題，我們提出利用網站伺服器過濾轉送的安全防護架構，把利用 XML Schema 的網站安全驗證程式置於網站伺服程式與應用程式之間，利用網站伺服器本身所提供的過濾或轉送功能，將網頁的傳遞資料在進入網站後但在傳送至應用程式之前進行驗證。選擇此位置進行驗證，無需於網路上進行任何封包轉送與導向，也無需更改原應用程式，置於同一網站之所有應用程式也可共用同一安全驗證程式，與之前相關研究比較，我們所提的網站安全架構建置比較容易。

貳、文獻探討

本章節我們將探討在網站應用層級常見的安全攻擊方式，以期了解安全防護之道，另外，網站應用程式安全防護的相關研究也於本章節介紹。

一、網站應用層級之安全威脅

常見於網站應用程式上的安全攻擊主要有資料隱碼攻擊、跨網站命令稿、更改標價攻擊三種，茲分述如下。

(一)、資料隱碼（SQL Injection）

資料隱碼攻擊[7, 14-18]是起因於網站應用程式未做好輸入檢查的工作所引發。其原理是攻擊者在不按照表單應輸入的資料格式輸入資料，取而代之的是在網站查詢的表單中，夾帶了惡意的 SQL 查詢指令，由於現有資料加密的保護方式只對資料的傳送過程負責資料安全，對於接受加密的內容並沒有探知能力，這說明目前電子商務網站在安全防護上的盲點，因此攻擊者可以利用應用程式沒有做好輸入檢查的漏洞，利用正常的查詢管道輸入惡意指令仍能穿透防火牆與其他安全防護，在資料庫中執行惡意查詢指令，並取得資料庫使用權限，進而入侵資料庫系統進行資料偷取與破壞。這個發生在網站伺服器端的安全漏洞，只要是以 ASP、JSP、PHP、CGI 或 Perl 撰寫以提供表單輸入介面為操作方式的網站，或是連結到 MS-SQL、MySQL、Oracle、Sybase 或 DB2 的任何支援 SQL 查詢語言的資料庫的網站應用程式，都有可能遭到此種攻擊。

資料隱碼攻擊可能利用的手段包括：剪接語法、利用錯誤訊息、使用具破壞力的語法及使用進階的延伸預存程序等四種。資料隱碼有其危險性，但只要在每一網站程式輸入資料做好檢查工作，便可杜絕。為防範攻擊者於表單輸入影響後端 SQL 指令執行的資料，我們必須在伺服器端對於字串的輸入加以過濾（輸入檢查），相關應過濾的格式整理於表一。

表一　資料隱碼過濾檢查之格式與過濾原因

過濾格式	過濾原因
‘ ’	單引號先讓原先正常的語法結束
Select, Create, Update, Insert, Drop, Union, …	防止 SQL 語法惡意的被執行
;	惡意的語法以分號起首
--	不造成 syntax error 結束惡意語法

(二)、跨網站命令稿（Cross-Site Scripting）

不同於資料隱碼攻擊發生在伺服器端的安全漏洞，跨網站命令稿[3, 4, 6, 7]是發生在用戶端的安全漏洞。當使用者瀏覽或連結以為可以信賴，但是已遭到攻擊者刻意設計的惡意網頁後，當瀏覽器下載該頁面後，嵌在其中的 Script 命令稿（即以<Script>標籤起首的程式碼，例如攻擊者於網站的留言版植入以下 Script 命令稿：

<Script>

window.location="http://www.hacker.com/steal.cgi?ck="+document.cookie;

</Script>

當另一使用者瀏覽含此命令稿的網頁時，其 Cookie 資訊已被送至攻擊者所架設的網站(www.hacker.com)，攻擊者在搜集使用者存於 Cookie 的私密資料或權限之後，便可進一步進行安全入侵。另外，使用類似的手法也可讓攻擊者直接存取客戶端的資料。

攻擊者通常利用表單輸入時放置惡意的跨網站命令稿達到入侵的目的，因此只要網站應用程式對於網頁輸入資料之檢查沒有考慮到此類嵌入命令稿攻擊，安全漏洞便出現。此類攻擊影響範圍也十分廣泛，微軟的 IE 瀏覽器內含跨網站命令稿的安全漏洞，同時 JavaScript、Java Applet 或者 ActiveX 控制項都可能被利用，因此未經安全保護的瀏覽器等均會受到攻擊。

在防制方式上，伺服器端需做好輸入的驗證，同時將一些危險或特殊的符號替換成相對應的 ASCII 字元碼，表二列出為防範跨網站命令稿應於表單輸入時進行過濾的資料格式。

表二 跨網站命令稿過濾檢查之格式與過濾原因

過濾格式	過濾原因
< >	標籤的起首 標籤的結尾
雙引號、&、 ？、空格或 tab	網址中夾帶入參數（值）
/	有導向惡意網址之可能
%	編碼標示 例 1: %68%65%6C%6C%6F 即 hello 例 2: %3CScript%3E 即<Script>
Non-ASCII 的資料	不符合 ISO8859-1 字集

(三)、更改標價攻擊

更改標價攻擊[11]是一個非常容易攻擊的方式，攻擊者將電子商務網站上最

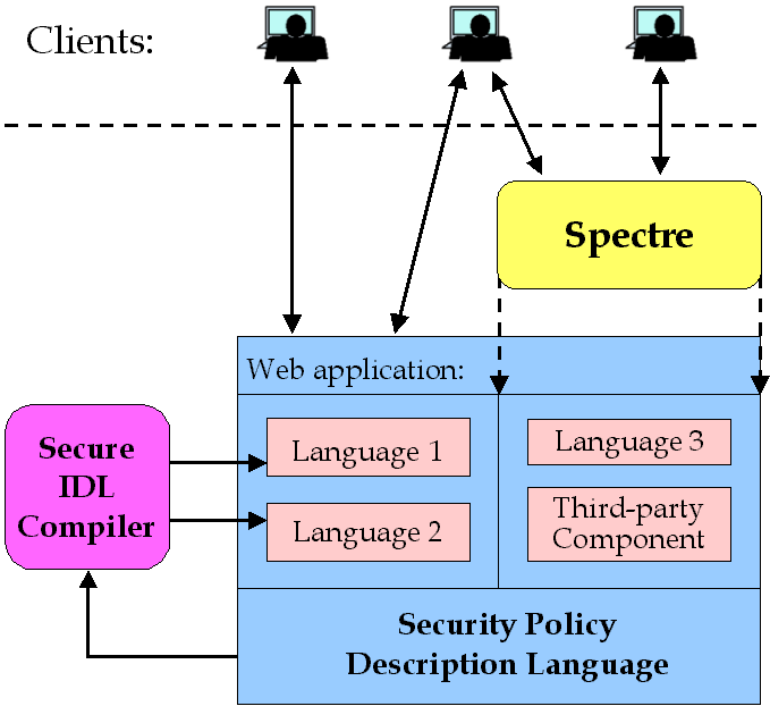
後結帳前的網頁儲存於用戶端電腦，再利用簡單的文字編輯工具將隱藏於結帳表單中的價格欄位改成較低的值，然後重新開啟網頁進行最後結帳的程序，如此便可以很低的價格購買商品。由於 HTTP 通信協定無狀態(Stateless)特性，許多網站應用程式會利用 HTML 表單所提供的隱藏輸入控制項記錄先前狀態資訊，以便將狀態資訊傳遞至下一個網頁。電子商務網站上許多商品雖有不同的商品說明展示網頁，但通常會共用同一個結帳程式，有些程式開發者便利用表單隱藏欄位來記錄來自不同網頁的商品之價格，以便讓同一結帳程式知道各商品之價格，也因此讓攻擊者有機可乘。這種攻擊可視為一種資料完整性(Data Integrity)破壞攻擊，因為攻擊者將先前從網站傳來之資料經竄改後再送回網站，使得網站收到之資料非原來資料，導致資料完整性遭破壞。先前所提的毒餅乾也是竄改狀態資訊導致資料完整性遭破壞的攻擊。資料完整性可利用加入額外的訊息驗證碼（MAC, Message Authentication Code）來保護，我們將於第參章節描述如何利用 HMAC[5] 保護網頁內容完整性來避免更改標價與毒餅乾攻擊。

二、網站應用程式安全相關研究

目前利用輸入驗證防護網站應用程式安全之相關研究包括<bigwig> [2], AppShield [10], 與 SWAP[11, 12] 。<bigwig>為一特殊的高階程式語言，專為發展 Web 服務而設計，因此無法廣泛應用於各網站應用程式安全之防護。而 AppShield 雖然沒有語言限制，然而缺乏一個可供網站開發者設定的安全政策描述語言，只能執行固定預設的表單驗證工作，不足以應付各式的安威脅。David Scott 及 Richard Sharp [11]首先發展了一套以以用來描述安全政策的 XML 語言（SPDL, Security Policy Description Language），網站開發者可使用 SPDL 來描述網頁輸入資料相關限制與安全需求。SPDL 已被運用在劍橋大學的 SWAP（Secure Web Application Project）計劃[12]。

SWAP 使用兩種網站安全防護架構（參考圖一），一種是使用應用層防火牆(Application-Level Firewall)的安全防護架構，稱為 Spectre；另一種則是使用介面定義語言編譯器直接產生程式碼於應用程式中。Spectre 包括四個元件：SPDL、政策編譯器(Policy Compiler)、安全閘道器(Security Gateway)、及 SPDL 編寫工具。政策編譯器解譯以 SPDL 撰寫的 XML 文件內容，產生對應的驗證程式來檢查網頁傳遞資料是否違背安全政策。這些以 SPDL 撰寫的 XML 文件及政策編譯器置於一安全閘道器，安全閘道器介於用戶端與網站之間，多個網站可以共用一個安全閘道器，所有連至這些網站的資料必須先導向至此安全閘道器以便進行安全驗證工作。由於採用 Spectre 時，網站開發者必須先熟悉複雜的 SPDL 語法方

能正確的描述於網頁中所傳遞資料的相關屬性，雖然 SPDL 為一 XML 語言，但並非一個廣被採用的語言，且不容易使用，因此 Spectre 尚包括一個 SPDL 編寫工具，提供一個簡易的 Web 介面方便網站開發者產生與修改 SPDL 文件。



圖一　SWAP 系統架構圖

SWAP 的另一個網站安全防護架構是直接產生安全驗證程式碼嵌入於網站應用程式之中，由於此作法最明顯的缺點是程式語言相依性，為此 SWAP 採用支援多語言的介面定義語言編譯器（IDL Compiler, Interface Definition Language Compiler），該 IDL 編譯器解譯 SPDL 文件並依據網站程式所採用程式語言產生對應的程式碼，直接嵌入網站應用程式碼內。雖然此架構本身沒有程式語言相依性問題，然而 IDL 編譯器本身仍是程式語言相依性，目前該 IDL 編譯器只支援 Ocam 一種程式語言，限制了此架構的應用範圍。

我們可以發現，無論採用 SWAP 的任一種架構，SPDL 之使用以及用來解譯 SPDL 文件的政策編譯器與 IDL 編譯器之開發，均不是件容易的工作。此外，使用 Spectre 的安全閘道架構，需要網路管理人員協助，方能將連至網站的封包事先導向至安全閘道器。而利用 IDL 編譯器產生驗證程式碼嵌入網站應用程式的方式，限制了程式語言的使用，且不適用於舊有已存在的系統。

參、以 XML Schema 作為安全政策描述語言

顧名思義，XML 為一可以讓使用者自行擴充定義的標籤語言，W3C 組織為了讓使用者能夠驗證一份 XML 文件是否符合由使用者自行定義的文件架構與標籤規則，先後發展了 DTD 與 XML Schema 標準[13]。由於 XML Schema 功能較多，本身也為一種 XML 語言，目前已漸取代 DTD 成為 XML 文件最常使用的驗證機制。XML Schema 除了可讓使用者定義一份 XML 文件的架構外，更可對文件中的元件與屬性設定資料型態，包括超過 44 種的內建資料型態，也可讓使用者自行定義複雜的資料型態。使用者可以進一步限制資料的範圍、長度、甚至利用正規表示法(Regular Expression)來限制資料的式樣(pattern)。也就是說，我們可以利用 XML Schema 強大的資料型態設定功能完整地規範 XML 文件內容可接受的資料，然後經由 XML 剖析器便可判斷一份 XML 文件是否合乎 XML Schema 規定。由於現今的安全政策描述語言主要功能也是用來描述如何限制網頁輸入資料的資料型態、長度、及允許字元，來避免資料隱碼攻擊、跨網站命令稿、及緩衝器溢位等攻擊。此特性非常類似 XML Schema 限制 XML 文件資料內容的功能，主要的差別在於安全政策描述語言驗證的對象是網頁輸入，而 XML Schema 驗證的對象是 XML 文件。為使 XML Schema 也可用來驗證網頁的輸入，我們首先設計一個自動將網頁輸入轉換成 XML 文件的機制，為轉換後的 XML 文件所訂定的 XML Schema 文件，即是原網頁輸入的安全政策描述語言。

一、將網頁輸入轉換成 XML 文件

網站應用程式從用戶端所獲得的輸入資料可來自於表單輸入、網址參數、以及狀態餅乾(Cookie)，無論使用哪一種形式的輸入方式，所有的輸入的資料都是以”名稱=值” (*name=value*) 成對的參數方式出現，例如：網址 http://www.ebuy.com/list.cgi?music=POP&page=3 包括兩對輸入參數：music=POP 及 page=3，這種網頁輸入的特性可以方便網站應用程式取得對應的輸入參數資料，我們也利用此特性訂定將網頁輸入轉成 XML 文件的規則。我們首先將網頁輸入資料依輸入方式分別置入如圖二所示 XML 文件架構對應的<Form>(表單輸入)、<QueryString>(網址參數)、以及<Cookie>(狀態餅乾)元件內。為符合 XML 文件規定，每對”名稱=值”的輸入參數改以”<名稱>值</名稱>”儲存，例如 music=POP 改成<music>POP</music>。

```
<?xml version=”1.0” encoding="Big5"?>
<form
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
  xsi:noNamespaceSchemaLocation="framework.xsd">
<Http>
   <Form>
      <price MAC=”Y”>0577</price>
      <amount>3</amount>
      <isMember>Y</isMember>
      <mac>3a53fe1d995a23</mac>
      <time>13eaf49b</time>
   </Form>
   <QueryString>
      <music>POP</music>
      <page>3</page>
   </QueryString>
   <Cookie>
      <sessionID>8988991225</sessionID>
   </Cookie>
</Http>
```

圖二 作為安全政策描述語言的 XML Schema **文件架構**

二、產生 XML Schema 文件

在訂定網頁輸入轉成 XML **文件的規則後，我們便可以針對轉換後的** XML **文件設計對應的** XML Schema **文件，以規範此** XML **文件的架構以及每一輸入參數值的相關限制。首先我們對於每一對輸入參數** *name=value*，**產生以下的** XML Schema **元件宣告：**

```
<xs:element name=”name” type=”nameType”/>
```

然後針對參數為”*name*”**的資料輸入限制，於資料型別**”*name*Type” **的定義中描述。以下針對不同資料輸入限制，逐一說明如何產生對應的** XML Schema。

(一)、文字長度限制

如果我們想限制某一輸入資料字元數目，我們可以使用簡單資料型別 (SimpleType)，**並使用** restriction **標籤對字串類型進行長度限制，長度限制的表示尚需用到** minLength **與** maxLength **兩個細節描述元素標籤。例如我們想限定密碼**

(參數名稱為 passwd)長度必須介於 8 至 15 個字，其對應的 XML Schema 如下：

```
<xs:simpleType name=″passwdType″>
   <xs:restriction base=″xs:string″>
      <xs:minLength value=″8″/>
      <xs:maxLength value=″15″/>
   </xs:restriction>
</xs:simpleType>
```

(二)、數值範圍限制

輸入資料如果為數值，我們可以限制數值允許範圍。我們使用簡單資料型別(SimpleType)，並使用 restriction 標籤對整數或浮點數類型進行數值範圍限制，數值範圍限制限制的表示尚需選用 minExclusive (>)、maxExclusive(<)、minInclusive (≧)、或 maxInclusive(≦)四個細節描述元素標籤。例如我們限定年齡(參數名稱為 age)為 0 至 100 間的整數，其對應的 XML Schema 如下：

```
<xs:simpleType name=″ageType″>
   <xs:restriction base=″xs:int″>
      <xs:minInclusive value=″0″/>
      <xs:maxInclusive value=″100″/>
   </xs:restriction>
</xs:simpleType>
```

(三)、列舉限制

表單中之 radio, checkbox, 及 select 輸入控制項，都限制了輸入資料只能在預設的清單項目內。我們可以同樣以 restriction 標籤對字串類型進一步列舉限制，列舉限制的表示尚需用到 enumeration 細節描述元素標籤。例如有一 select 控制項(參數名稱為 ccard)，可選的清單項目值為' visa' , ' master' , ' JCB'，其對應的 XML Schema 如下：

```
<xs:simpleType name=″ccardType″>
   <xs:restriction base=″xs:string″>
      <xs:enumeration value=″visa″/>
      <xs:enumeration value=″master″/>
      <xs:enumeration value=″JCB″/>
   </xs:restriction>
</xs:simpleType>
```

(四)、特殊式樣限制

如果我們能夠用正規表示式描述輸入的式樣，我們便可利用 XML Schema 的 pattern 標籤限制文字型別資料。例如我們以"[A-Z]\d{9}"限定正規表示式限制身份證號碼(參數名稱為 ID)輸入資料，其對應的 XML Schema 如下：

```
<xs:simpleType name=″IDType″>
   <xs:restriction base=″xs:string″>
      <xs:pattern value=″[A-Z]\d{9}″/>
   </xs:restriction>
</xs:simpleType>
```

(五)、防止資料隱碼與跨網站命令稿攻擊

除了對資料輸入特定的長度、範圍、式樣限制外，為了防止輸入資料包括資料隱碼與跨網站命令稿攻擊所用到的特殊字元，我們可以使用正規表示式來限制。其對應的 XML Schema 如下：

```
<xs:simpleType name=″nameType″>
   <xs:restriction base=″xs:string″>
      <xs:pattern value=″[^’″;<>%]″/>
   </xs:restriction>
</xs:simpleType>
```

(六)、資料完整性保護

防止更改標價攻擊的方式是不允許用戶端對隱藏欄位的值進行修改，我們可以針對這些不能改變的參數使用訊息驗證碼進行資料完整性保護，在 XML Schema 文件中我們針對要保護的元素額外定義一個 MAC 屬性以指明該元素必

須保護。例如圖二的 price 標籤所對應的 XML Schema 如下(macType 定義沒有列出)：

```
<xs:complexType name="priceType">
<xs:simpleContent>
<xs:extension base="xs:int">
<xs:attribute name="MAC" type="macType" use="required"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
```

肆、網站安全防護架構

由於使用應用層防火牆方式的網站安全閘道架構，必須調整網路相關設定，且容易於安全閘道器造成效能瓶頸，而直接嵌入驗證程式碼在網站應用程式的架構，雖然預期有較佳的效能，卻有程式語言上的限制，也不適用於對舊有系統的安全防護。我們提出一個新的網站安全防護架構，將安全防護程式置於網站伺服程式與應用程式之間，利用網站伺服程式所提供的過濾轉送功能，將網頁輸入資料導入網頁過濾器進行安全過濾，圖三顯示新的網站安全防護架構，此架構包括四個模組，茲分述於后。

一、網頁過濾器

大部份網站伺服器都會提供過濾轉送機制，讓系統開發者能對網頁資料做額外的篩選與處理，此機制也可讓網站伺服器外掛命令稿編譯執行的功能。因此我們可經由網站伺服器的設定，將網頁輸入資料轉送至網頁過濾器。網頁過濾器收到網頁輸入資料後，執行以下步驟工作：

(1). 取得所有參數資料，產生一份 XML 文件，將 XML 文件傳遞給 XML 剖析器進行輸入驗證。如果驗證成功，繼續執行下一步驟，否則傳送錯誤訊息至用戶端並結束。

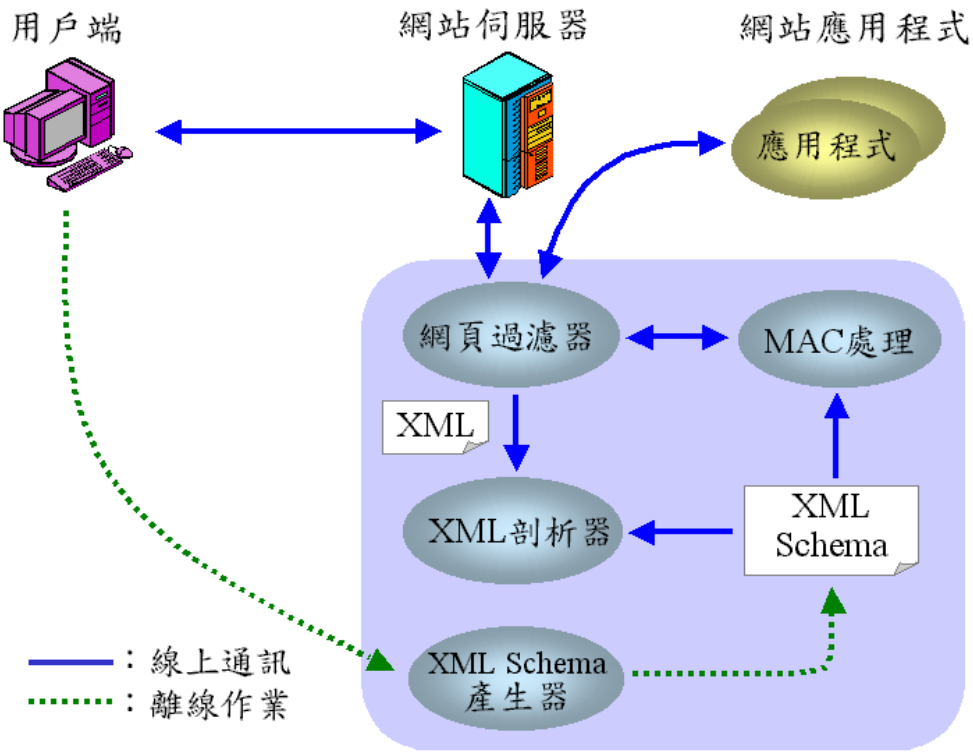
(2). 將輸入資料傳遞給 MAC 處理模組進行資料完整性檢查。如果資料沒遭破壞，繼續執行下一步驟，否則傳送錯誤訊息至用戶端並結束。

(3). 將輸入資料傳遞給網站應用程式。

(4). 接收網站應用程式傳回的 Cookie 及 HTML 資料，擷取 HTML 資料中包含參

數的網址以及表單資料，把 Cookie 及擷取的資料傳給 MAC 處理模組以便產生 MAC 碼與時間戳記。

(5). 接收 MAC 處理模組傳回之 MAC 碼與時戳資料。將 MAC 資料加到 Cookie 及 HTML 文件，傳回用戶端。



圖三　網站安全防護架構

二、XML 剖析器

XML 剖析器(Parser)接收由網頁過濾器送來的 XML 文件，判斷此文件是否符合 XML Schema 規定。由於每個網頁有特定的網頁輸入，因此對應特定的 XML 文件架構與規則，XML 剖析器可以依據網址知道該使用哪一份 XML Schema 文件來驗證此動態產生的 XML 文件。事實上，我們無須自行開發 XML 剖析器，目前各式支援 XML 處理功能的軟體開發工具都有內建 XML 剖析器程式，幾乎所有 XML 程式都會用到 XML 剖析器程式。本論文提出使用 XML Schema 作為網站安全政策描述語言，最大的好處便是不需要自行開發任何政策編譯器，可節省軟體開發成本，同時也可保障較好的程式碼品質。

三、MAC 處理

MAC 處理模組用來保障網頁資料的完整性在我們的架構中，我們採用 HMAC 演算法對需要保護的資料產生訊息驗證碼，惡意攻擊者在篡改受完整性保護的資料時，也必須有能力產生對應的訊息驗證碼才能進行更改標價或毒餅乾攻擊，由於 HMAC 是將受保護資料與秘鑰一起經過單向的雜湊函數（Hash function）所產出的訊息驗證碼，因此，惡意攻擊者很難在不知密鑰的情況下自行產生正確的訊息驗證碼。然而在網頁的操作環境，惡意攻擊者仍有可能經由重播攻擊(Replaying Attack)來更改標價。他可以先在網頁點選單價較便宜的產品，例如軟碟片，看到單價為 150 元以及使用單價 150 元資料與秘鑰合在一起所產生的訊息驗證碼。然後再回去網站點選單價較高的產品，例如定價 5 萬元的筆記型電腦，再修改網頁，將單價設為 150 元、訊息驗證碼設成 150 元所對應的驗證碼，如此便能通過完整性資料檢查。有兩個方法可以用來改善使用 HMAC 所發生的問。第一個方法是將資料再加上時間戳記進行雜湊，使得同樣的資料在不同時間所產生的訊息驗證碼不同，且限制訊息驗證碼只能在戳記時間 5 或 10 鐘以內使用。此方法可以避免惡意攻擊者一直重複使用低標價資料的訊息驗證碼進行攻擊。然而，惡意攻擊者還是可以在訊息驗證碼有效期間內進行相同手法的入侵。第二個方法是不要只對單一參數附加訊息驗證碼。例如把產品編號與售價，甚至與 Cookie 資料一起訊息驗證碼，如此惡意攻擊者便很難使用重播攻擊方式破壞資料完整性。基於以上分析，我們採用(資料+時戳+秘鑰)的 HMAC 方式降低遭受重播攻擊，此外當我們發現使用者在設定 XML Schema 時只標明一個參數需產生 MAC 時，我們以警告的方式提醒使用者可能存在的安全漏洞。

由於網頁的資料輸入中，部份資料要受保護，其他則為使用者自行輸入的資料，因此 MAC 處理模組在收到從網頁過濾器所送來的資料時，必須至 XML Schema 文件找出具有 MAC 屬性的元素，再針對這些元素對應的參數產生訊息驗證碼，並與一起送來的訊息驗證碼進行比對。同樣地，對於要送回用戶端的 HTML 資料，也要參考 XML Schema 文件才能知道哪些資料要產生訊息驗證碼。

四、XML Schema 產生器

XML Schema 產生器是個供離線使用的輔助性軟體模組，它提供 Web 介面讓應用程式開發者能很容易地填寫或勾選網頁輸入資料的相關屬性，XML Schema 產生器便會自動產生對應的 XML Schema 文件。如圖四所顯示畫面，應用程式開發者針對每一網頁輸入設定欄位名稱、表單輸入型態、資料型態；驗證規則的選擇項包括：資料長度、數字長度、預設值、正規表示式。

圖四 XML Schema **產生器操作畫面**

伍、系統實作與比較

我們根據所提以 XML Schema **驗證技術所建立的網站應用安全防護架構，使用** Java **程式語言開發了一套網站應用安全防護系統。由於採用** Java **技術，我們將主要的網頁過濾程式寫成** Java Servlet **置於** Tomcat **網站伺服器，利用網站伺服器所提供的過濾器（**filter**）的功能將網頁輸出入資料導向此** Servlet **程式。由於我們所選用的網站運作環境只支援** JSP **網站應用程式，目前我們的系統實作只支援** JSP **程式的安全防護。另外在** XML **剖析器方面，我們使用昇陽公司提供的** JAXP 1.2 **開發套件[1]所提供的** XML Schema **驗證功能，程式中需要自行解析** XML Schema **文件的工作則使用** SAX **應用程式介面。而在** MAC **處理模組中，我們實作** HMAC**，使用** Java JDK **本身即有提供** MD5[9]**作為** HMAC **的雜湊函數。**

我們將以上程式在支援 Windows 2000 **的** PC **環境執行，在** PC **本身執行速度不快的硬體限制下，此安全防護系統處理一次網頁輸入安全驗證所花費的時間約在** 50~70 ms **間，平均約為** 60 ms**。我們相信若在較高階的伺服器上執行，應可讓每次處理時間降至** 30 ms **左右。這些數據對於一般網站應用的效能要求而言，都是可以接受的範圍。而此系統實作更重要的成果是證明使用** XML Schema **作為網站安全政策描述語言的網站安全防護架構確實可以防範資料隱碼、跨網站命令稿、緩衝器溢位等各式攻擊，也降低了更改標價及毒餅乾攻擊的可能性。**

表三列出本篇論文所提方法與其他兩種安全架構之比較，從表三首先我們可以看到，由於使用 XML Schema **取代** SPDL**，我們可以直接使用** XML **剖析器而**

無需自行發展政策編譯器。其他在網路設定限制、舊有系統及多語言支援上，本論文所採架構兼具其他兩者之優點。至於系統整體效能評估上，採用安全閘道架構時，多個網站的資料輸出入都需經過安全閘道，容易於安全閘道上造成效能瓶頸。使用 IDL 編譯器的架構則直接產生輸入檢查程式，無需於每次網頁輸入時處理 XML 文件，因此效能最好。在平台相依性上，由於安全閘道完全獨立於網站，因此沒有平台相依性限制，使用 IDL 編譯器則有程式語言的限制，針對每一程式語言我們需要開發專屬的 IDL 編譯器。本論文所提架構需使用網站伺服器的過濾轉送功能，因此與所使用的網站伺服器相關。

表三　網站應用安全防護架構比較

安全架構	安全閘道	使用 IDL 編譯器	本篇論文
安全政策語言	SPDL	SPDL	XML Schema
政策編譯器	需要	需要	XML 剖析器
網路設定	需要	不需要	不需要
支援舊有系統	支援	不支援	支援
支援多語言	支援	不支援	支援
系統效能評估	最差	最好	中等
平台相依性	無	程式語言	網站伺服器

陸、結論與未來方向

傳統的網路安全議題多注重於資料加密、身份認證、以及交易安全，然而網站應用程式的設計不當卻容易造成嚴重的安全漏洞。許多網站應用程式過於信任由用戶端傳回的資料，惡意攻擊者可利用不夠嚴謹的資料驗證漏洞，危害網站應用程式安全，因此，單靠加密的網站安全防護是不夠的，本論文探討網站在應用層次上的安全問題，提出一個利用 XML Schema 的網頁輸入驗證方法，並提出利用網站伺服器過濾功能的網站應用安全防護架構，避免網站應用程式受到安全威脅。

與過去相關研究比較，本論文無論在輸入驗證方式與防護架構設計上，都有較佳的表現，網站開發者只要熟悉標準的 XML Schema 規格便能很容易地訂定

網頁輸入的驗證規則，或是經由我們所提供 Web 介面操作的 XML Schema 產生器自動產生 XML Schema 文件。而使用本論文所提的安全防護架構建置安全防護系統時，也無須建置防火牆、開發編譯器、以及限制特殊的程式語言。這些優點有效降低了網站應用安全防護系統開發的成本與建置的複雜度。

我們利用嚴謹的網頁輸入驗證有效解決了資料隱碼、跨網站命令稿、及緩衝器溢位等問題，對於利用破壞資料完整性的更改標價及毒餅乾攻擊，我們所使用加上時戳的訊息驗證碼機制可以降低攻擊的可能性，但仍不能完全解決此安全問題。我們認為問題的根本在於表單的隱藏欄位本身的不安全性，將重要的資料放於此欄位，等於明白告訴惡意攻擊者如何進行安全入侵。因此除了避免使用隱藏欄位外，我們可以在網頁過濾器在傳回 HTML 文件之前，將隱藏欄位資料取出並加密，放在 Cookie 或 Session 變數中，使惡意攻擊者無法得知一個表單所包含的隱藏欄位。此外，我們在系統的開發過程發現無論使用 SPDL 或 XML Schema 作為安全政策語言，一個運作的安全防護系統針對每一次的網頁輸入都必須進行一次檔案存取動作，這可能是系統效能的瓶頸所在。未來在實作上，我們將探討如何將最近存取過的 XML Schema 以物件方式暫存在記憶體，藉由減少檔案存取次數來改進系統效能。

最近由於 XML 技術的快速發展，許多企業開始使用 Web 服務做為 B2B 電子商務平台，雖然 Web 服務通訊多發生在企業內部，不易被攻擊者查覺，然而，Web 服務可能被應用在企業重要的系統，因此 Web 服務的安全性值得進一步探討，可能的議題包括 Web 服務是否有資料隱碼攻擊的威脅，以及可不可能於 Web 服務訊息資料置入另一個 SOAP 訊息或 XML 文件來達到安全入侵的目的。

參考文獻

[1] Armstrong, Eric, *et al*., The Java Web Service Tutorial, http://java.sun.com/webservices/docs/1.1/tutorial/doc/, Sep. 2003.

[2] Brabrand, Claus, Møller, Anders, and Schwartzbach, Michael I., The <bigwig> Project, ACM Transactions on Internet Technology, Vol. 2, No. 2, May 2002, pp. 79-114.

[3] Cgisecurity, The Cross site Scripting Faq, http://www.cgisecurity.com/articles/xss-faq.shtml.

[4] iDEFENSE, Evolution of Cross-site Scripting attacks, http://www.idefense.com/XSS.html, May 2002.

[5] Krawczyk, H., Bellare, M., and Canetti, R., HMAC: Keyed-Hashing for Message

Authentication, Internet Request For Comments 2104, Feb. 1997.
[6] Microsoft Knowledge Base, HOWTO: Prevent Cross-Site Scripting Security Issues, http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q252985.
[7] Neff, Patrice, Web Application Security, http://www.patrice.ch/en/computer/web/articles/2002/jul_18, July 2002.
[8] OWASP, The Ten Most Critical Web Application Security Vulnerabilities, http://unc.dl.sourceforge.net/sourceforge/owasp/OWASPWebApplicationSecurityTopTen-Version1.pdf, January 13, 2003.
[9] Rivest, R., The MD5 Message Digest Algorithm, Internet Request For Comments 1321, April 1992.
[10] Sanctum, AppShield 4.0 White Paper, http://www.sanctuminc.com/pdf/AppShield_40_WhitePaperFINAL.pdf.
[11] Scott, D., and Sharp, R., Abstracting Application-Level Web Security, Proc. 11th Int' l World Wide Web Conf., ACM Press, New York, May 2002, pp.396-407.
[12] Scott, D., and Sharp, R., Developing Secure Web Applications, IEEE Internet Computing, November/December 2002,Vol.6, Issue: 6, pp.38-45.
[13] W3C , XML Schema, http://www.w3.org/XML/Schema
[14] **陳培德、賴溪松，資料隱碼**（SQL Injection）**原理與防範，**Communication of the CCISA, Vol.9, No.1, Dec. 2002, pp.37-44.
[15] **鈺松國際，**SQL Injection**攻擊法與安全程式，**Communications of the CCISA, 2002**年，**6**月，第**8**卷，第**3**期，頁**4-7。
[16] **臺灣電腦網路危機處理暨協調中心，**SQL Injection**，**http://www.cert.org.tw/news/020424.php。
[17] **臺灣徵軟公司，**『**資料隱碼**』SQL Injection**的源由與防範之道，**http://www.microsoft.com/taiwan/sql/sql_injection.htm, 2002**年，**6**月**。
[18] **臺灣徵軟公司，** SQL Injection (**資料隱碼**) – **駭客的** SQL**填空遊戲，**http://www.microsoft.com/taiwan/sql/SQL_Injection_G1.htm, 2002**年，**6**月**