利用 XML 驗證之網站安全防護架構

陳彥錚
暨南國際大學資訊管理學系

林錦雲
暨南國際大學資訊管理學系

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 文件作為網站應用程式的安全政策描述語言，用以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會將輸入資料轉換為 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。與先前相關研究比較，本論文所提網站安全防護機制，使用標準的 XML Schema 作為網站安全政策描述語言，容易學習且無需複雜的編譯器。此外，此安全防護機制不必改變網路組態及現有網站應用程式，均優於以往的作法，是一個簡易又有效的網站安全防護機制。

關鍵字：電子商務安全、資料隱碼攻擊、XML Schema、輸入驗證

A Framework for Securing Web Applications by XML Validation

Yen-Cheng Chen
Department of Information Management, National Chi Nan University

Chin-Yun Lin
Department of Information Management, National Chi Nan University

Abstract

Many previous studies on web security focus on the data confidentiality issue. However, confidential data in web applications may be revealed even that security mechanisms like SSL or SET are adopted in web sites. This is because there exist potential security vulnerabilities in web applications themselves. Most of these vulnerabilities are caused by the lack of solid input validations for protecting web applications. SQL injection is a typical example of attacks based on the vulnerabilities. Cross-site Scripting (XSS), price changing attack, and poisoned cookie are other known security threats of web applications.

It is a challenge to develop a unified method to validate web inputs for all web applications. In this paper, we propose a framework for protecting web applications based on the XML validation technology. We use the standard XML schema as a security policy description language (SPDL). Developers can use XML schema to specify the properties of web inputs. In the proposed framework, located between the web server and web applications, web inputs are first encapsulated in an XML document generated on the fly. Then, the XML document is validated by using XML schema. If no errors are found after the XML validation, the web inputs are valid for web applications. Hence, web applications can be protected effectively. Compared with previous approaches, our framework uses the standardized XML schema as the SPDL for web applications. Therefore, no any particular compiler is required. In addition, no any network configuration is needed in our framework. Legacy web applications can also be protected without any modifications. In summary, our framework provides a simpler and more effective mechanism for securing web applications.

Keywords: E-Commerce Security, SQL injection, XML Schema, Input validation.

壹、緒論

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，以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客戶端傳回的資料，並沒有從安全的角度嚴謹地驗證網站輸入的資料，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，對電腦資料庫中必須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(OWSAP 2003)指出，首要的網站安全漏洞便是網頁傳遞未經驗證的資訊(Un-validated Parameters)，此報告同時也指出對網頁輸入缺乏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) (Scott & Sharp 2002)、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程式安全的研究仍然不多。究其原因，主要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應用程式安全的的輸入驗證程式也勢必不同。在這樣面對眾多異質的網站應用程式，如何提供一個具一致性且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，我們首先需要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 (Scott & Sharp 2002)首先提出一個使用可延伸性標示語言(Extensible Markup Language，簡稱 XML)所設計的網頁安全政策描述語言 SPDL(Security Policy Description Language)，網站應用程式開發者可依據 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 並非標準的標示語言，網站應用程式開發者必須熟悉其特殊語法，在使用上相當不便。此外，使用 SPDL 語言尚須一個專屬的編譯器，此編譯器之開發也是一相當龐大的工

程，目前並沒有存在這樣的編譯器工具供網站應用程式開發者使用。因此，現階段採行 SPDL 的網站安全防護機制，僅止於學術性研究，實用性不高。本論文基於使用 SPDL 之不便與開發專屬編譯器的困難，進一步研究更為可行的網站安全防護機制。

針對資料之驗證，在 XML 標準中，XML 文件正確性可利用 XML Schema 進行驗證，雖然一般網站輸入的資料並非 XML 文件格式，我們發現 XML Schema 可作為網頁安全政策描述語言取代原有 SPDL 所扮演的角色。網站應用程式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，我們提出使用 XML Schema 作為安全政策的描述語言之另一重大好處是不需要使用任何複雜的政策編譯器，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要求之前提，任何網站開發者都可以很容易地利用我們所提的網站安全防護機制進行有效的網站安全防護。

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，將客戶端所傳遞的資料在進入網站後但在傳至應用程式前先進行驗證。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容易建置許多。結合本論文所提 XML Schema 驗證以及新的網站安全防護架構，我們提出一個具一致性且有效的網站安全防護機制，使得具應用層安全保護能力的網站應用程式得以容易實現。

貳、相關文獻探討

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) 資料隱碼攻擊(SQL Injection)

資料隱碼攻擊(Neff 2002; 陳培德、賴溪松 2002; 鈺松國際 2002; 臺灣電腦網路危機處理暨協調中心 2002; 臺灣微軟 2002)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在輸入資料驗證上的漏洞，利用正常的網頁表單輸入管道直接夾帶惡意指令而仍能穿透防火牆與其他安全防護，進入資料庫中執行惡意查詢指令，進行資料竊取或破壞之惡意攻擊。這個發生在網站伺服器內應用程式的安全漏洞，無論使用 ASP、JSP、PHP、CGI 或 Perl 所撰寫的網頁程式，只要有提供表單輸入介面，並連結到 MS-SQL、MySQL、Oracle、Sybase 或 DB2 等任何支援 SQL 查詢語言的資料庫，均有可能遭受資料隱碼攻擊。

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 1。

表 1：資料隱碼過濾檢查之格式與過濾原因

過濾格式	過濾原因
‘ ’	讓先前正常字串語法結束，以便夾帶後續指令
Select, Create, Update, Insert, Drop, Union, …	執行 SQL 指令，進行惡意攻擊
;	結束先前 SQL 語法，並於後加入惡意的語法
--	不會造成因 SQL 語法錯誤而無法進行攻擊

(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(Cgisecurity 2002; iDEFENSE 2002; Microsoft 2002; Neff 2002)是發生於客戶端的安全漏洞。當使用者瀏覽或連結以為可以信賴但實已遭受攻擊者刻意設計的惡意網頁，瀏覽器下載該網頁後，自動執行嵌於其中的惡意 Script 命令稿(即以<Script>標籤起首的程式碼)，導致使用者資料被竊或使用者電腦遭受惡意攻擊。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，其存於 Cookie 中的個人資訊將被送至攻擊者

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電腦內之資料。

攻擊者通常利用正常的表單輸入方式，將惡意的跨網站命令置於公眾網站以達到入侵的目的，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，安全漏洞便浮現。此類攻擊影響範圍十分廣泛，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均會遭受此類攻擊。在防制方式上，伺服器端之網站應用程式必須針對跨網站命令稿可能出現字元進行過濾，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 2 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 2：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(Scott & Sharp 2002)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客戶端電腦，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程序，便可以極低的價格購買商品。由於 HTTP 通信協定的無狀態(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。防止更改標價攻擊斧底抽薪的方法便是避免使用表單隱藏欄位，改以伺服器端 Session 物件儲存標價等相關資訊。然而，現今仍有許多程式開發者偏好使用表單隱藏欄位，另外對於一些使用傳統程式語言開發的 CGI 程式，並沒有內建的 Session 物件可供運用，因此，研究如何避免更改標價攻擊仍有其必要性。這類更改標價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(Krawczyk, Bellare, & Canetti 1997)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> (Brabrand *et al*. 2002)、AppShield (Sanctum 2002)、與 SWAP(Scott & Sharp 2002)。<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全威脅。David Scott 及 Richard Sharp (Scott & Sharp 2002)首先提出一個用來描述網頁安全政策的 XML 語言─SPDL (Security Policy Description Language)，一份 SPDL 文件描述一個網頁表單輸入資料之相關限制，包括每一輸入欄位之資料型態、資料範圍、及資料格式，並可藉由過濾特殊字元來確保網站應用程式之安全性。SPDL 已被運用在劍橋大學的網站安全研究計劃：SWAP(Secure Web Application Project) (Scott & Sharp Nov. 2002)。

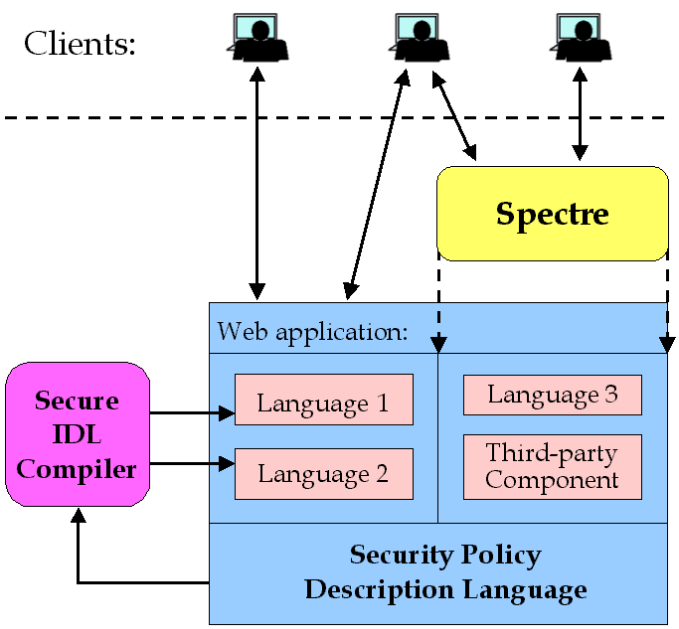

圖 1：SWAP 系統架構圖

如圖 1 所示，SWAP 使用兩種網站安全防護架構，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內，安全閘道器介於客戶端與網站之間，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之傳遞必須先導向至此安全閘道器以便進行安全驗證工作。網站應用程式開發者欲採用 Spectre 安全防護架構，必須先熟悉複雜的 SPDL 語法，方能正確地描述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，並不容易使用，也因此 Spectre 還包括一個 SPDL 編寫工具，提供簡易的 Web 介面方便網站應用程式開發者產生與修改 SPDL 文件。SWAP 所採行的另一個網站安全防護架構是直接產生安全驗證程式碼，將之嵌入於網站應用程式中，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 在使用上的困難是無法避免的，而最大的挑戰是如何撰寫 SPDL 文件解譯器以及 IDL 編譯器，兩者之程式開發並不容易，目前也並沒有商用的 SPDL 文件解譯器以及 IDL 編譯器可供使用。因此，事實上目前的網站應用程式開發者並無法使用 SWAP 所建議的方式防護網站的安全。此外，在實務上使用 Spectre 的安全閘道架構時，還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。另一方面，利用 IDL 編譯器產生驗證程式碼嵌入網站應用程式的方式，除了限制程式語言的使用，對於舊有已存在的系統也並不適用。

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(W3C 2003)。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，以避免資料隱碼攻擊、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。由於 XML Schema 已被廣泛採用，各種用來開發 XML 應用程式之程式語言均有提供 XML 剖析器程式，支援 XML Schema 驗證功能，我們若能使用 XML Schema 作為網頁安全政策描述語言，XML Schema 的高接受度以及驗證程式容易取得，將可大幅簡化網站應用程式的安全防護。然而如前所述，XML Schema 驗證的對象是 XML 文件，為能使用 XML Schema 驗證網頁輸入，我們首先提出一個自動將網頁輸入轉換成 XML 文件的機制，此機制能夠依據網頁輸入欄位資料產生一個暫用的 XML 文件，為執行效率上考量，此 XML 文件無需以真實的檔案方式存在於網站。而從客戶端傳來的網頁輸入資料將被包含於此 XML 文件的 XML 標籤之中，我們依據原網頁輸入資料的相關限制，並依此 XML 文件架構撰寫對應的 XML Schema 文件，此 XML Schema 文件即為原網頁輸入的安全政策描述文件。因此，藉由所提 XML 文件轉換機制，XML Schema 便可作為網頁安全政策描述語言。茲將 XML 文件轉換機制以及 XML Schema 設計方式分述於後。

一、將網頁輸入轉換成 XML 文件

網站應用程式從客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 2 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>。我們可以發現，網頁輸入轉成 XML 文件之規則非常直覺簡單，通常網頁輸入參數也不多，因此轉換的工作可以很快完成。此外，由網頁輸入所轉換的 XML 文件，只是為了讓後續利用 XML Schema 的驗證工作得以進行，且所產生供驗證的 XML 文件通常不大，因此在系統實作上，XML 文件可以即時(On-The-Fly)方式產生並供 XML 驗證程式使用，而無需以檔案方式儲存於網站上，可避免因檔案存取影響網站程式執行的效率。

```
<?xml version=”1.0” encoding="Big5"?>
<Http
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xsi:noNamespaceSchemaLocation="framework.xsd">
    <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 2：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”/>
```

雖然 XML Schema 提供內建的資料型態供使用者選用，我們使用上述自訂的資料型態，一方面可以在自訂的資料型態加上與網站安全相關的限制，另一方面也可簡化 XML Schema 文件之設計，使其具一致性，亦即每一參數之名稱為“*name*”，必然對應一自訂資料類型 “*name*Type”，而所有對此參數的相關輸入限制，均置於對應的資料類型中定義。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 描述如下：

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

對輸入資料字元數目加以限制，除了後端資料庫處理之需要，對網站安全而言，可以避免緩衝器溢位所造成的安全漏洞。

(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 描述如下：

```
<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 描述如下：

```
<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

XML Schema 之一強大功能在於其支援正規表示式(Regular Expression)，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}”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 描述如下：

```
<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 描述如下：

```
<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客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 2 的 price 標籤所對應的 XML Schema 描述如下(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>
```

以上說明六種使用 XML Schema 限制網頁輸入資料的方式，我們可針對每一輸入

欄位，依據應用程式資料驗證之規則以及所要進行的安全防護，加上相關的 XML Schema 敘述對應的自訂資料型態之中。基本上，如果輸入欄位之資料限制為數值資料、列舉資料、或有嚴格的資料格式規定，只要使用上列(二)~(四)所述之 XML Schema 即可，如有資料完整性保護，再於所自訂的資料型態加上如(六)所述的 MAC 屬性即可。其他一般允許自由格式的字串輸入資料，則必須於自訂資料型態加上如(一)所述的文字長度限制與(五)所述的特殊字元限制。

肆、網站安全防護架構

目前相關研究所提出的網站安全防護架構主要有安全閘道及嵌入驗證程式兩種。使用應用層防火牆方式的網站安全閘道架構，必須藉由網路管理人員介入，調整網路相關設定，確保所有進出網站的封包經過安全閘道，此種作法容易於安全閘道器造成效能瓶頸。另外，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，在程式開發上也需要一個專屬的平台提供程式庫或編譯器，對於現有舊系統的安全防護也無能為力。基於目前兩種架構之可能缺失，我們提出一個新的網站安全防護架構，此架構將負責網站安全防護之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，將網頁輸入資料先行導入網頁過濾器進行安全過濾，如果輸入資料沒有包含危害網站應用程式安全之字元，才將網頁輸入資料傳送至後端的應用程式。圖 3 顯示新的網站安全防護架構，此架構包括四個模組：網頁過濾器、XML 剖析器、MAC 處理模組、以及 XML Schema 產生器，茲分述於後。

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。目前廣被採用的 Apache 網站伺服器提供 Filter 應用程式介面(Apache 2004)，讓程式開發者自行開發特殊需求的網頁過濾功能；微軟公司的 IIS 伺服器程式則提供 ISAPI Filter(Microsoft(1) 2004)及 ISAPI Extension(Microsoft(2) 2004)，同樣支援網頁過濾功能。因此我們可利用網站伺服器本身所提供的過濾機制，將網頁輸入資料先行轉送至網頁過濾器，網頁過濾器為本網站安全防護架構之主要程式，負責偵測危害網站應用程式安全的輸入資料。在取得網頁輸入資料後，網頁過濾器執行以下六個步驟：

(1)取得所有輸入參數資料，依據轉換規則產生一份 XML 文件暫存於記憶體。

(2)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客戶端並結束。

(3)將輸入資料傳遞給 MAC 處理模組進行資料完整性檢查。供 MAC 處理模組檢查資料完整性之輸入資料包括具有 macType 屬性之輸入欄位、MAC 值、及時間戳記。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客戶端並結束。

(4)將輸入資料傳遞給網站應用程式。

(5)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。

(6)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，傳回客戶端。

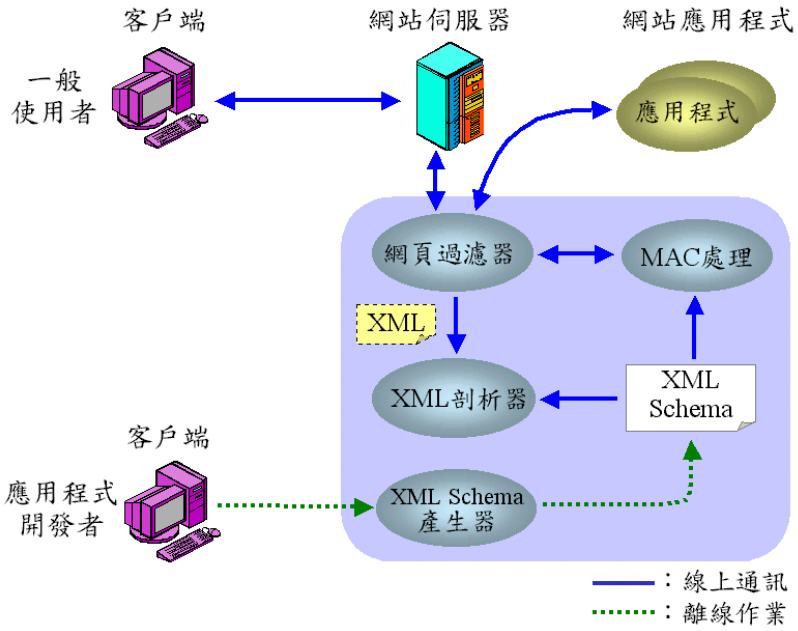

圖 3：網站安全防護架構

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，我們無需自行開發 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。

由於 XML Schema 文件基本上是以檔案方式儲存於網站上，XML 剖析器針對每一網頁輸入必須讀取一次 XML Schema 文件，執行效能可能受到檔案讀取動作所影響。事實上，大部份網站應用程式的輸入表單並不多，亦即所需的 XML Schema 文件並不多，因此，在 XML 剖析器執行效能考量上，我們可以預先將 XML Schema 文件

讀至程式中，以減少檔案讀取次數。昇陽公司所提供的 J2EE 平台，即有提供將所用到的 XML Schema 文件列於字串陣列之功能(Sun, 2004)，無需重覆開啟相同的檔案。另外一個改進執行效能的方式是使用 XML 資料庫，由於 XML Schema 本身也是 XML 文件，可將 XML Schema 文件存於 XML 資料庫，以改進 XML Schema 文件存取效能。

三、MAC 處理模組

MAC 處理模組用來保障網頁資料的完整性，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 元資料與密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 所發生的問題。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 分鐘以內使用。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，以及 Cookie 資料一起產生訊息驗證碼，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 碼時，我們以警告的方式提醒使用者可能存在的安全漏洞，令其加入適當的欄位來保護資料完整性。

由於網頁的資料輸入中，部份資料需要受保護，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，對於要送回客戶端的 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。當網頁表單或網址中有需保護之資料時，MAC 處理模組先依據目前時間產生一時戳，然後使用 HMAC 產生 MAC 碼，並於表單或網址中自動加上兩個輸入參數：mac 及 timestamp，分別存放所產生的 MAC 碼及時戳，以便將之送至客戶端。屆時由客戶端傳回資料時，網頁過濾器將輸入資料以及所附帶的 mac 及 timestamp 參數值傳給 MAC 處理模組，MAC 處理模組收到資料後，首先判斷傳回的 timestamp 參數值距目前時間是否已超過 5 分鐘，若未逾時，MAC 處理模組將需受保護之輸入資料、timestamp 值、以及密鑰產生 MAC 碼，並與傳回的 mac 參數值比較，判斷資料是否遭受竄改。藉由 MAC 處理模組負責產生與驗

證 MAC 碼，網頁資料的完整性得以受到保護，因此可避免更改標價攻擊及其他藉由擅自更改網址以竊取資料之惡意行為。

使用 HMAC 產生訊息驗證碼時需要一把密鑰，傳統使用 HMAC 的方式是送方使用密鑰與欲傳送訊息一起產生訊息驗證碼，收方收到訊息與訊息驗證碼，再使用密鑰與所收到訊息產生訊息驗證碼並與所收到的訊息驗證碼進行比對，以確保訊息忠實地由送方產生並完整地送至收方，使用 HMAC 做此用途時，送方與收方均必須事先知道密鑰。在本論文所提網站安全防護架構中，我們使用 HMAC 之目的是用來保護從網站傳至用戶端一些不可被變動的資料，例如隱藏欄位與網址中的參數，以防止這些資料被使用者惡意竄改後再傳回網站達到安全攻擊之目的。針對這些需保護的資料，MAC 處理模組使用一把密鑰針對不可被變動的資料產生對應的訊息驗證碼，此訊息驗證碼將隨著資料一起送至用戶端，當用戶端隨後把這些不能變動的資料及訊息驗證碼再傳回網站時，MAC 處理模組使用同一把密鑰對收到的資料產生訊息驗證碼，再與所收到的訊息驗證碼進行比對，便可得知資料是否被竄改。由於訊息驗證碼之產生與驗證都在 MAC 處理模組執行，只有 MAC 處理模組需要知道 HMAC 的密鑰，因此，此網站安全防護架構並不存在 HMAC 密鑰傳遞問題。

四、XML Schema 產生器

每一網頁輸入需要一個 XML Schema 文件描述輸入限制，雖然 XML Schema 已廣被接受，對網站應用程式開發者而言，為每一網頁輸入準備與維護一個 XML Schema 文件仍是一個工作負擔，為使本網站安全防護架構更方便使用，我們設計一個 XML Schema 產生器，讓網站應用程式開發者經由簡單的 Web 介面操作即能產生一份 XML Schema 文件。XML Schema 產生器是個供離線使用的輔助性軟體模組，其使用方式有三種。

(1) 輸入網站應用程式網址，由 XML Schema 產生器自動解析 HTML 內容，找出需受保護之輸入欄位，供網站應用程式開發者利用 Web 介面進一步設定。
(2) 由網站應用程式開發者利用 Web 介面自行設定欲受保護之網址、輸入欄位、及相關安全限制，新增一 XML Schema 文件至網站上。
(3) 開啟已存在的 XML Schema 文件，利用 Web 介面進行修改或刪除。

網站應用程式開發者使用 XML Schema 產生器，無需熟悉 XML Schema 語法，只要依據網頁輸入參數之相關限制，填寫或勾選網頁輸入資料的相關屬性，XML Schema 產生器便能自動產生 XML Schema 文件。而每當網頁輸入限制有所變更時，只要利用 XML Schema 產生器修改 XML Schema 文件即可。比傳統網站應用程式必須修改原始程式中有關輸入安全檢查之程式碼，方便許多。而使用 IDL 編譯器則必須重新編譯原應用程式才能將新的驗證程式嵌入。另外，與使用安全閘道的網站安全防護架構比較，雖然其使用的 SPDL 文件更新時，不需變動原始程式，但網站應用程式開發者必須將更新的 SPDL 文件傳送至安全閘道上。因此，使用 XML Schema 產生器，網站應用程式開發者可以容易管理與維護網站安全防護架構所需的 XML Schema 文件。圖 4 顯示

網站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。

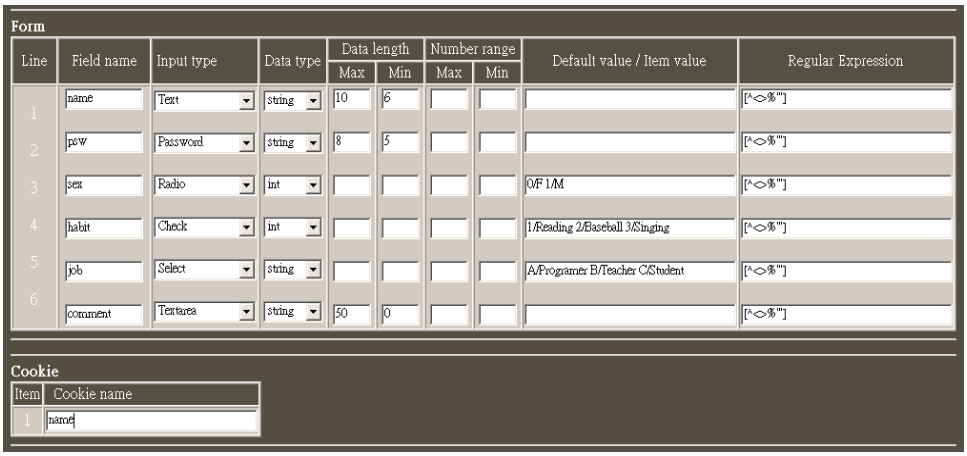

圖 4：XML Schema 產生器操作畫面

伍、系統實作與比較

為驗證本論文所提使用 XML Schema 驗證技術之網站應用安全防護架構，我們使用跨平台程式語言 Java 開發一套網站應用安全防護系統。此網站應用安全防護系統建置於 Tomcat 網站伺服程式上，Tomcat 本身也是一個完全由 Java 所寫的網站伺服程式，同樣支援跨平台功能。因此，此安全防護系統可與 Tomcat 一起安裝於各式作業系統平台。Tomcat 廣受採用是因其支援使用 Java 所寫之 JSP 或 Servlet 網站程式，也因此本系統實作目前只支援 JSP 及 Servlet 網站程式的安全防護。針對使用其他的程式語言如 ASP、PHP，或一般 CGI 程式之安全防護，網站安全系統開發者均可於 Apache 網站伺服器實作 Filter 或 IIS 網站伺服器實作 ISAPI Filter 或 ISAPI Extensions，提供支援多語言的網站應用安全防護系統。

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。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(Armstrong 2003)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(Rivest 1992)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。此外，我們並以實驗證明本論文所提網站安全防護架構確實能夠保護網站應用程式安全；我們先針對資料隱碼、跨網站命令稿、緩衝器溢位、以及更改標價攻擊，撰寫未做任何輸入驗證之範例程式。在未啟動網站安全過濾機制情況下，我們以實際的測試個案(test cases)對這些程式進行測試，確定網站能夠被這些測試個案成功入侵。然後，我們為這些程式利用 XML Schema 產生器產生對應的 XML Schema 文件並存於網站，再啟動網站安全過濾機制。當我們再重新將相同的測試個案輸入至這些範例程式，有關資料隱碼、跨網站命令稿、緩衝器溢位的測試均無法通過 XML Schema 驗證。而針對更改標價攻擊，更改隱藏的價格傳回網站並無法通過 MAC 驗證，而將五分鐘以前的資料傳回網站，也會因 MAC 碼逾時遭受網站拒絕。這些測試都顯示本論文所提的網站安全防護架構確實能夠保護網站安全。

表 3 列出本篇論文所提方法與其他兩種安全架構之比較，從表 3 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，一個網頁輸入過程包括客戶端與安全閘道建立 HTTP 連結、讀取 SPDL 文件進行解析、執行輸入過濾、以及安全閘道與網站伺服器建立 HTTP 連結；由於每一網頁輸入過程包含兩個 HTTP 連結通訊及一次 SPDL 解析，預期執行效能較差，此外多個網站的資料輸出入都需經過安全閘道，也容易於安全閘道上造成效能瓶頸。本論文採用安全機制，客戶端只需與網站伺服器建立一次 HTTP 連結，網頁過濾器與網站伺服器緊密結合，並採用技術成熟的 XML 剖析器，因此輸入過濾工作可以很快完成，在架構上也沒有安全閘道的效能瓶頸，因此整體效能優於安全閘道。而使用 IDL 編譯器的架構，由於輸入檢查程式已內嵌於網站應用程式，無需於每次網頁輸入時處理 XML 文件，因此預期效能最好。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，目前網站伺服器並沒有標準的擴充介面提供過濾轉送，在 Apache 伺服器需使用其提供的 Filter 設定及應用程式介面，而在 IIS 伺服器則需使用 ISAPI Filter 或 ISAPI Extensions，這是本論文所提網站安全防護架構因有平台相依性限制而在實作上較為不便之處。不過，網站伺服器的過濾機制已成為網站伺服器提供擴充功能必要的元件，本論文所提使用網站伺服器過濾機制的網站安全防護架構即是一個典型而重要的應用。

表 3：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客戶端傳回的資料，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，我們可以了解只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 產生器產生及更新 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，也無需建置防火牆、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。對於有大量使用者的網站，在實作上我們可以將最近存取過的 XML Schema 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. Apache Software Foundation, “Filters - Apache HTTP Server,” http://httpd.apache.org/docs-2.1/filter.html, Nov. 2004
2. Armstrong, Eric, *et al*., “The Java Web Service Tutorial,” http://java.sun.com/webservices/docs/1.1/tutorial/doc/, Sep. 2003
3. Brabrand, Claus, Møller, Anders, and Schwartzbach, Michael I., “The <bigwig> Project,” *ACM Transactions on Internet Technology* (2:2) May 2002, pp: 79-114
4. Cgisecurity, “The Cross site Scripting Faq,” http://www.cgisecurity.com/articles/xss-faq.shtml, 2002
5. iDEFENSE, “Evolution of Cross-Site Scripting Attacks,” http://www.idefense.com/XSS.html, May 2002
6. Krawczyk, H., Bellare, M., and Canetti, R., “HMAC: Keyed-Hashing for Message Authentication,” *Internet Request For Comments 2104*, Feb. 1997
7. Microsoft Knowledge Base, “HOWTO: Prevent Cross-Site Scripting Security Issues,” http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q252985, 2002
8. Microsoft Corporation (1), “ISAPI Extensions,” http://msdn.microsoft.com/library/en-us/iissdk/iis/isapiextensions.asp, April 2004
9. Microsoft Corporation (2), “ISAPI Filters,” http://msdn.microsoft.com/library/en-us/iissdk/iis/isapifilters.asp, April 2004
10. Neff, Patrice, “Web Application Security,” http://www.patrice.ch/en/computer/web/articles/2002/jul_18, July 2002
11. OWASP, “The Ten Most Critical Web Application Security Vulnerabilities,” http://unc.dl.sourceforge.net/sourceforge/owasp/OWASPWebApplicationSecurityTopTen-Version1.pdf, January 13, 2003
12. Rivest, R., “The MD5 Message Digest Algorithm,” *Internet Request For Comments 1321*, April 1992
13. Sanctum, “AppShield 4.0 White Paper,” http://www.sanctuminc.com/pdf/AppShield_40_WhitePaperFINAL.pdf, 2002
14. 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
15. Scott, D., and Sharp, R., “Developing Secure Web Applications,” *IEEE Internet*

Computing (6:6) 2002, pp:38-45
16. Sun Microsystems, “Validating with XML Schema,” http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JAXPDOM8.html, Aug. 2004
17. W3C, “XML Schema,” http://www.w3.org/XML/Schema, 2003
18. 陳培德、賴溪松，2002，『資料隱碼(SQL Injection)原理與防範』，*Communication of the CCISA*，第九卷・第一期，37~44頁。
19. 鈺松國際，2002，『SQL Injection攻擊法與安全程式』，*Communications of the CCISA*，第八卷・第三期，4~7頁。
20. 臺灣電腦網路危機處理暨協調中心，2002，『SQL Injection』，http://www.cert.org.tw/news/020424.php。
21. 臺灣微軟公司，2002，『資料隱碼SQL Injection的源由與防範之道』，http://www.microsoft.com/taiwan/sql/sql_injection.htm。
22. 臺灣微軟公司，2002，『SQL Injection (資料隱碼)–駭客的 SQL填空遊戲』，http://www.microsoft.com/taiwan/sql/SQL_Injection_G1.htm。