以合理輸入值驗證為基礎之 預存程序組合型 SQL 指令植入式攻擊防禦

陳威霖　　田筱榮
中原大學資訊工程系
W.L.Chen@cycu.org.tw　tyan@ice.cycu.edu.tw

摘要

SQL 指令植入式攻擊(SQL Injection Attack)可能導致與網路服務相連的資料庫系統遭受破壞或是資料庫內的資料遭到竊取。過去的研究多以防禦 SQL 指令植入式攻擊為目的，然而對於以預存程序(Stored Procedure)為對象的攻擊較缺乏防禦做法，在本研究中我們將防禦範圍擴展到能處理發生於預存程序內的組合型 SQL 指令植入式攻擊。在靜態分析階段，除了分析網頁原始碼之外，並針對資料庫內的預存程序內容做分析，找出使用者輸入參數與程序內 SQL 指令之間的關係，在動態分析階段，使用者進行參數輸入時能夠檢查參數代入預存程序內與 SQL 指令結合後是否產生 SQL 指令植入式攻擊，將其阻擋並拒絕請求。藉由我們的防禦機制，使用者和網頁程式設計師可以不需知道防禦程式如何運作狀態下即可達到防禦效果；系統管理者只需將防禦程式掛載於網頁伺服器端即可達到防禦功能。

關鍵詞：網頁應用程式安全、SQL 指令植入式攻擊、輸入驗證、預存程序

Abstract

SQL injection attacks may lead to data theft, content destruction, even database crash. Many defense mechanisms have been proposed to resolve explicit SQL injection attacks, where the attacks take advantage of the web application programs. However, defense against SQL injection attacks aiming at stored procedures provided by supporting database system have not been successfully addressed. In this study, we extend the input legitimacy validation method to SQL injection attack on stored procedures. Besides the web application program, the statics analysis is also performed on the stored procedures provided by database system to obtain knowledges on the relationships between user input parameters and the stored procedures. During on-line dynamic analysis phase, input parameters are verified according to the previously derived knowledges. If a potential SQL injection attack is detected, the user request will be rejected before forwarding to the web application programs. The proposed enhancement to the original method provides a more comprehensive defense on SQL injection attacks. Our method is transparent to users and web application program designers. It provides effective defense with no need to rewrite application programs.

Keywords：Web application security, SQL injection attacks, input validation, Stored Procedure

1. 簡介

在科技不斷進步的時代，網際網路成為人們生活中不可或缺的一環，對於網站技術而言，大部分都是使用動態網頁，例如：PHP(Hypertext Preprocessor)、JSP(JavaServer Pages)、ASP(Active Server Pages)，搭配資料庫系統來運作，伴隨著網路安全威脅事件日漸增加的趨勢，假如這些網站沒有受到相當程度的防護，往往會成為駭客們覬覦的對象，例如：SQL 指令植入式攻擊，造成系統檔案損毀或是個人資料外洩。

根據 OWASP Top Ten Project[3]提到 2010 年至 2013 年十大安全弱點中 Injection 排名高居第一，SQL 指令植入式攻擊包含在 Injection 之中，利用 SQL 語法的特性去組合輸入的字串，與網頁程式中的 SQL 指令組合出含有惡意的請求，造成網頁伺服器和資料庫系統遭受破壞或是給予預期之外回應，導致資訊安全受到嚴重的威脅。

因為 SQL 語法能夠允許多種不同指令層層組合，所以 SQL 指令植入式攻擊種類也非常多樣化，從輸入驗證(Input Validation)的角度來探討 SQL 指令植入式攻擊方式，可以分類為恆值邏輯(Tautologies)、註解化(Commentization)、嵌入查詢(Embedded queries)、預存程序(Stored procedures)四大類型。

SQL 指令植入式攻擊通常是程式設計師的疏失或是經驗不足所造成的漏洞，如果要修補程式漏洞，需要重新檢視程式，從中找出有問題的程式碼，但此種方法需大費周章且不切實際，所以將重點擺放於網頁應用程式開發完成上線運作狀態，使用者送出請求後網頁伺服器接收參數時針對使用者輸入的合理性驗證，我們主要目的是設計一個不需更動網頁應用程式原始碼的防禦機制，而且延續[1][2]採用輸入驗證(Input Validation)的方式來防禦 SQL 指令植入式攻擊，驗證方式並非單純的對參數做特徵過濾，而是在離線狀態時對網頁原始碼和資料庫內容做分析，接著上線運作階段在網頁伺服器接收請求時利用離線狀態分析所得之資訊來檢查使用者輸入的請求參數是否構成 SQL 指令植入式攻擊，因此具有容許非典型輸入值的可能。除了延續[1][2]的作法，我們將防禦範圍擴展到能夠處理發

生於預存程序內的組合型 SQL 指令植入式攻擊，針對此攻擊將提出修正版的合理輸入值驗證機制。

接下來第二節會介紹 SQL 指令植入式攻擊防禦的相關研究與成果；第三節會說明我們的分析規劃與系統架構；第四節為實作，將會展示我們的系統如何防禦 SQL 指令植入式攻擊，並討論改良後的結果；第五節則是結論和未來工作。

2. 相關研究

SQL 指令植入式攻擊的防禦階段分類，根據（圖 1），應用程式開發完成上線運作狀態，此狀態又可分為四個階段來探討，使用者送出請求階段是屬於瀏覽器端防禦，後面三個皆屬於網頁伺服器接收請求階段、應用程序執行階段、送出 SQL 請求存取資料庫階段，皆屬於伺服器端的防禦。

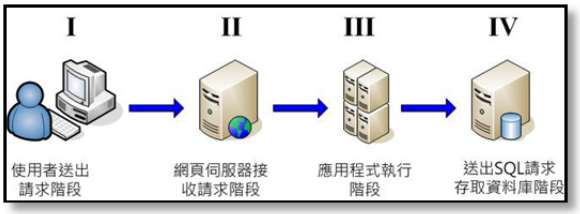

圖 1. SQL 指令植入式攻擊之防禦階段分類

至目前為止，針對 SQL 指令植入式攻擊已有許多方法能夠偵測與防禦，防禦類型通常可分為靜態、動態、動靜態混合式防禦，也因為於方法不同，所以能夠防禦的攻擊種類與特色各有差異，如(表 1)。

表 1. SQL 指令植入式攻擊相關研究比較表

	Taut.	Com.	Embedded	Stored proc.	不需要更動程式碼	上線運作階段
Input Legitimacy Validation[1][2]	○	○	○	X	○	II
ModSecurity[4]	○	○	○	○	○	II
SQL Guard[10]	○	○	○	X	○	III
SQL Check[11]	○	○	○	X	○	III
WASP[12]	○	○	○	○	X	III
SQL-IDS[13]	○	○	○	○	○	IV

Taut.：恒值邏輯　　Com.：註解化
Embedded：嵌入查詢　　Stored proc.：預存程序

輸入合理性驗證 (Input Legitimacy Validation)[1][2] 系統架構分成靜態分析與動態分析兩部分，靜態分析是掃瞄出網頁程式內含有 SQL 語法部分，並產生「參數元素過濾檔」，提供給動態分析來判定網頁伺服器接收到的參數有無符合 SQL 指令植入式攻擊

ModSecurity[4]是屬於網路應用程式防火牆，利用正規表示法(Regular expression)編譯規則集，使用規則制定特徵進行 SQL 指令植入式攻擊過濾。

Buehrer、Weide 和 Sivilotti 提出 SQL Guard 的方法[10]，其做法是在程式執行時會產生語法樹，將程式碼裡的 SQL 指令和語法樹做比對。

Wassermann 和 Su 提出 SQL Check 的方法[11]，其做法是事先產生定義好語法樹，在程式碼裡將 SQL 指令和事先產生定義好的語法樹做比對。

Halfond、Orso 和 Manolios 提出 WASP 的方法[12]，其做法是在網頁程式碼轉換成 bytecode 時，將疑似有弱點的參數加入檢測用的字串作為標記，並立即過濾檢查，但此做法會更動程式碼，有可能會影響原本程式執行的目的和結果。

Kemalis 和 Tzouramanis 提出 SQL-IDS 的方法[13]，其做法是使用特徵比對方法，佈署在應用程式送出 SQL 請求至資料庫之間，使用規則過濾 SQL 指令植入式攻擊，其判斷的準確度會依據特徵制定嚴謹度有所不同。

3. 分析規劃與系統架構

3.1 分析規劃

我們主要目的是設計一個不需更動網頁應用程式原始碼的 SQL 指令植入式攻擊防禦系統，並以[1][2]提出的方法為基礎，將輸入合理性驗證檢查延伸至預存程序的輸入參數大到防止預存程序內的組合型 SQL 指令植入式攻擊的目標。

在前人的研究中所提出的防禦機制是將系統架構分成靜態分析和動態分析兩部分，靜態分析是在系統處於離線狀態時事先對網頁程式碼做掃描檢查的動作，找出程式碼內含有 SQL 語法部份，針對 SQL 語法的內容結構產生「參數元素過濾檔」，並在動態分析時利用靜態分析產生的「參數元素過濾檔」來判定伺服器接收自使用者傳來的請求字串是否含有 SQL 指令植入式攻擊。

但此方法無法有效偵測預存程序內的組合型 SQL 指令植入式攻擊，因為使用預存程序在網頁程式碼中是找不到任何 SQL 語句，只能在網頁上找到預存程序的呼叫指令，僅知道欲呼叫之程序名稱以及傳入的參數內容，而不知道程序內 SQL 語句結構，將無法判斷程序傳入之參數所對應的 SQL 欄位名稱是如何組合而成的。如果只單純判斷傳入的參數，將可能會導致嚴重的誤判或漏判情形發生，所以在我們的系統架構上，靜態分析步驟會特別針對資料庫裡的預存程序內容做分析，找出程序內的 SQL 語句搭配傳入的參數做結合，產生一份資訊提供給動態分析步驟做為參考，判斷預存程序內的 SQL 語句是否有構成 SQL 指令植入式攻擊。

3.2 系統架構

這是完整的預存程序系統分析架構(圖 2)，結合了靜態分析與動態分析，接下來將會分別詳細介紹靜態分析以及動態分析的組織架構和運作流程。

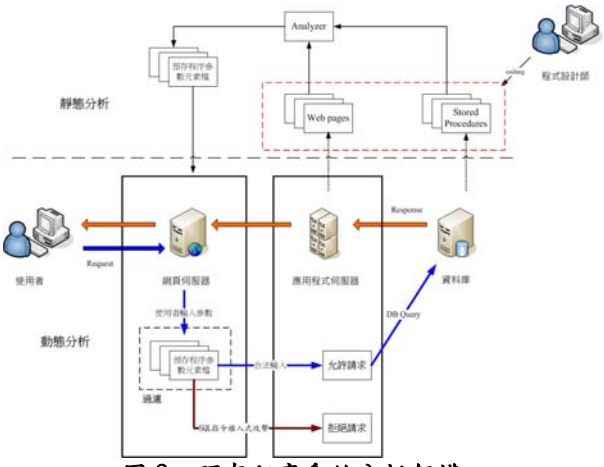

圖 2. 預存程序系統分析架構

3.2.1 靜態分析

靜態分析的架構如(圖 3)，這是使用者對伺服器送出請求，配合網頁應用程式的執行，向資料庫送出 SQL 請求，得到相對應的結果回傳至使用者。在進行一系列的步驟之前，我們會先在離線狀態時做出分析動作。首先我們會針對網頁上掃瞄出有呼叫預存程序的指令，接著在資料庫將預存程序的程式碼匯出成為可分析的檔案，搭配在網頁上欲呼叫的程序名稱和參數進行分析，最後組成一個預存程序參數元素檔，此檔案中所記錄的資訊是為了在動態分析時能夠辨別使用者輸入請求參數的性質，可以判斷使用者輸入的參數型態、長度是否符合，並分辨出單一型參數或是組合型參數。

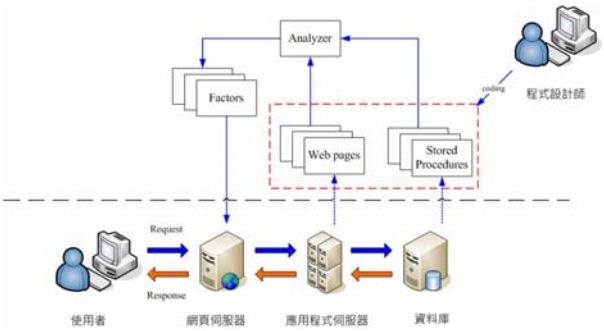

圖 3. 靜態分析架構

接下來以(圖 4)來說明靜態分析的流程和步驟。

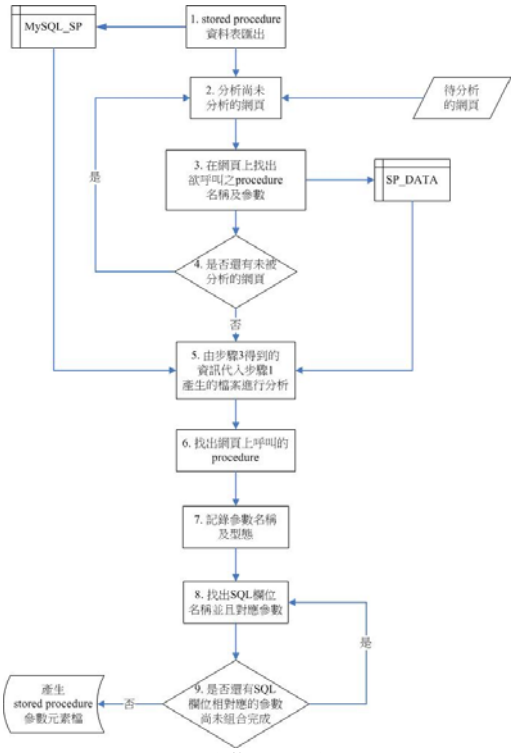

圖 4. 靜態分析流程圖

1. 將預存程序從資料庫匯出，如(圖 5)。

```
DELIMITER //
CREATE DEFINER=`root`@`localhost` PROCEDURE
`sp_test`(IN uname VARCHAR(50), IN pin1
VARCHAR(50), IN pin2 VARCHAR(50), IN pin3
VARCHAR(50))
BEGIN
SET @sql=CONCAT('select * from school where
id =\'', uname, '\' and pwd =\'', pin1, pin2, pin3, '\'
# select student info');
PREPARE stmt FROM @sql;
EXECUTE stmt;
END //
DELIMITER ;
```

圖 5. 預存程序匯出資料表

2. 開始掃瞄網頁原始碼是否有出現預存程序呼叫的指令。
3. 在網頁程式碼中找出欲呼叫之程序名稱及參數並記錄下來，如(圖 6)。

```
sp_test(id, pwd1, pwd2, pwd3)

sp_test1(uname, password)

sp_test2(id)
```

圖 6. 網頁原始碼呼叫預存程序指令

4. 重覆步驟 2、3，直到所有網頁分析完成。
5. 由步驟 3 的資訊代入步驟 1 的檔案進行分析。
6. 找出呼叫的程序名稱進行分析。
7. 記錄程序的參數名稱、型態、長度。
8. 找出 SQL 欄位名稱並且找出相對應的參數。
9. 如果是組合型參數，將其組合完成並且對應給 SQL 欄位名稱。

以上步驟都完成後，將產生一份預存程序參數元素檔如(圖 7)。

```
#sp_test
$uname, VARCHAR, 50
$pin1, VARCHAR, 50
$pin2, VARCHAR, 50
$pin3, VARCHAR, 50
@id, uname
@pwd, pin1, pin2, pin3
```

圖 7. 預存程序參數元素檔

3.2.2 動態分析參數過濾

此階段為上線運作狀態，如(圖 8)，在網頁伺服器端接收使用者輸入的參數，並且參照靜態分析階段產生的預存程序參數元素檔來過濾使用者輸入之參數，防止使用者輸入的參數代入預存程序內與 SQL 語句結合後可能造成 SQL 指令植入式攻擊的問題發生。

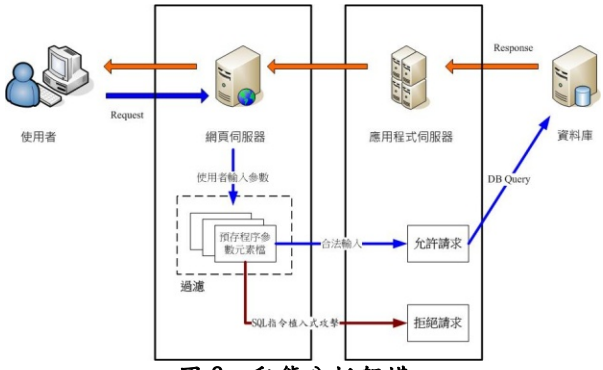

圖 8. 動態分析架構

動態分析的參數過濾流程如(圖 9)，首先取得使用者輸入參數，接著依照靜態分析階段產生的預存程序參數元素檔針對組合型參數做參數結合動作，然後進行參數型態檢查、字串檢查，確認參數代入預存程序內是否形成 SQL 指令植入式攻擊。

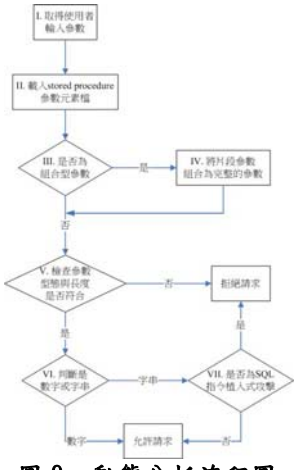

圖 9. 動態分析流程圖

I. 取得使用者在網頁上輸入的參數。

II. 載入靜態分析時產生的預存程序參數元素檔做為判斷之依據。

III. 根據預存程序參數元素檔內容判斷使用者輸入的參數是否為組合型參數。

IV. 如果是組合型參數，如(圖 10)，將片段參數組合為完整參數。

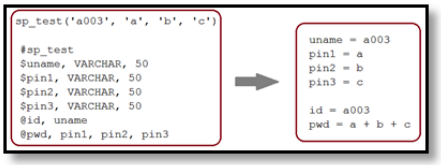

圖 10. 預存程序組合型參數

V. 檢查參數的型態以及長度是否有符合預存程序參數元素檔內記錄之標準，如果不符合便拒絕使用者輸入的請求。

VI. 判斷參數是字串還是數字，因為字串可能含有 SQL 指令植入式攻擊，必須進一步檢查，如果是數字則直接允許請求。

VII. 如果參數是字串的話則進入字串檢查流程，如(圖 11)，判斷出參數字串中含有 SQL 指令植入式攻擊特徵，將拒絕請求，若無 SQL 指令植入式攻擊特徵，將允許請求。

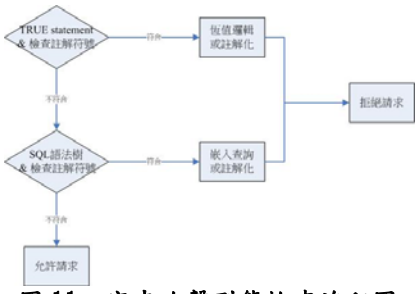

圖 11. 字串攻擊型態檢查流程圖

(1) 判斷字串中是否包含運算子和註解符號：

SELECT student FROM test.school WHERE id='a001' AND pwd='' **OR 1=1 --**;

若運算子存在且運算式為恆等式，例如：1=1、0<1，且運算式前有出現邏輯運算子，例如：AND、OR，則判定使用者輸入的參數為恆值邏輯的 SQL 指令植入式攻擊。

若 SQL 語句中有出現註解符號，例如：--、#，將判定使用者輸入的參數為註解化的 SQL 指令植入式攻擊。

(2) 檢查 SQL 語法中是否有嵌入第二段 SQL 語句：

SELECT student FROM test.school WHERE id='' **UNION SELECT math FROM test.score WHERE id='a001' --** ' AND pwd='111';

我們使用 SQL 語法樹來輔助判斷，如(圖 12)，藉由語法樹架構來分析每一段字串，判別是否含有嵌入查詢的 SQL 指令植入式攻擊。

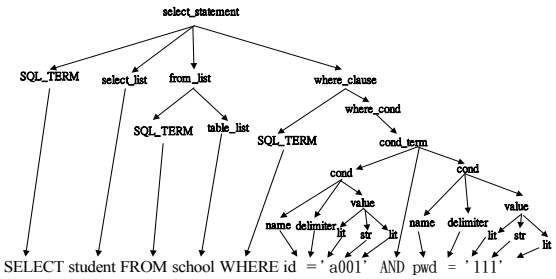

圖 12. SQL 語法樹

4. 實作與討論

為了驗證我們的方法能夠抵擋在預存程序內發生的 SQL 指令植入式攻擊。在 4.2 節我們將會示範展示實作的系統如何防禦 SQL 指令植入式攻擊。

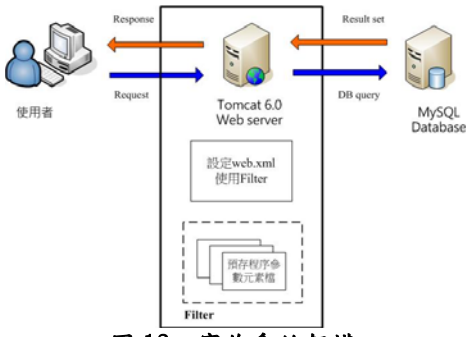

圖 13. 實作系統架構

4.1 實作環境

本實驗所使用的網頁伺服器為 Tomcat6.0，網頁應用程式使用 JSP 做編輯，資料庫伺服器則是使用 MySQL Server 5.1，並安裝於相容的作業系統(Windows 7 64-bit)。另外我們在 Tomcat 中的 web.xml 設定所有 JSP 及 servlet，全部套用 web.xml 中所制定的<filter>。

4.2 實作與防禦

我們實作的登入系統之中，SQL 語句是寫在預存程序內部，並非在網頁原始碼中，在網頁原始碼只看的到程序的呼叫指令以及傳入程序的參數，如(圖 14)，是正常輸入帳號密碼登入系統所得到的資訊，密碼部份是利用 3 組參數所組合起來的完整字串對應給程序內的 SQL 欄位。

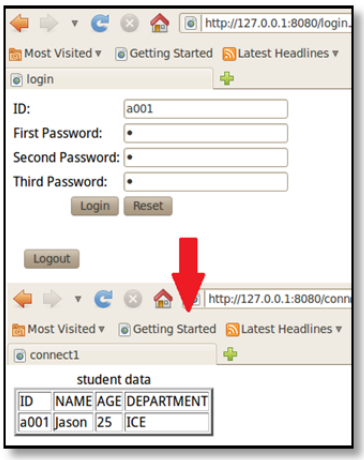

圖 14. 實作範例(正常登入)

接著我們將含有 SQL 指令植入式攻擊的惡意字串(a' OR 1=1#)拆散，分別寫在密碼部份的 3 組參數做為請求，如(圖 15)，即可成功登入系統，且將資料庫內所以資料全部顯示出來。

圖 15. 實作範例(SQL 指令植入式攻擊)

若在網頁伺服器 Tomcat 中的 web.xml 掛載 Filter，啟動防禦系統，將會在靜態分析時蒐集資訊產生一份「SP_FACTOR.txt」預存程序參數元素檔，如(圖 16)，並且在使用者輸入參數送出請求後，依照「SP_FACTOR.txt」的內容分析使用者輸入參數，假如判斷內容含有 SQL 指令植入式攻擊，將其阻擋拒絕請求，並將網頁導向錯誤訊息畫面，如(圖 17)。

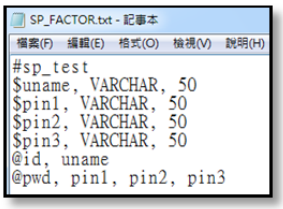

圖 16. sp_test 的參數元素檔內容

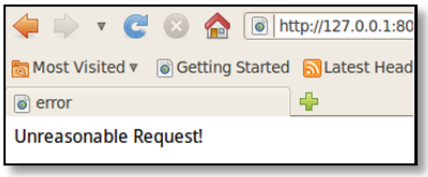

圖 17. 實作範例(成功防堵 SQL 指令植入式攻擊)

4.3 討論

實作中我們發現，在靜態分析時對預存程序的傳遞參數以及內部的 SQL 語句徹底分析，例如：判斷對應給 SQL 欄位的參數是單一型還是組合型，若是組合型參數，再判斷參數的排列順序為何。有了充足的資訊，在動態分析時即可有效阻擋預存程序內部發生的 SQL 指令植入式攻擊。另外我們延續[1][2]的特點，不需要更動網頁應用程式原始碼，只需在網頁伺服器 Tomcat 的 web.xml 中設定好需要掛載的防禦系統(Filter)，即可達到防禦效果。

5. 結論與未來工作

SQL 指令植入式攻擊是一項已經存在許久的攻擊手法與威脅，但此問題始終一直發生，而且攻擊排名統計居高不下。防禦方面，在應用程式開發階段，除了本身加強程式撰寫的能力，減少程式漏洞產生以外，在應用程式上線運作時，網頁伺服器端加上一道防禦系統以防堵 SQL 指令植入式攻擊也是一個完善的解決方法。

我們延續[1][2]的研究特點，在離線狀態時先做靜態分析，針對資料庫內的預存程序內容產生一份預存程序參數元素檔提供給動態分析時參考，在動態分析時，網頁伺服器接收參數，並根據靜態分析產生的預存程序參數元素檔進行分析，檢查參數代入程序內與 SQL 語句結合後是否會產生 SQL 指令植入式攻擊。我們的防禦機制與[1][2]同樣不限制網頁應用程式撰寫語言，以及網頁伺服器選擇，但目前實作的防禦系統僅能在 Tomcat 網頁伺服器上執行，以及使用 JSP 撰寫的網頁應用程式做檢查。

未來我們將針對不同的網頁應用程式上做偵測防禦，例如：PHP、ASP，在不同的網頁伺服器上掛載我們的防禦系統，例如：Apache、IIS。另外，希望能在靜態分析掃瞄預存程序內容時，加入檢查會受程序流程影響而改變的 SQL 指令，例如：條件式(if、else)、迴圈(while、for)…等，並且在動態分析時依據影響程序流程的條件式做檢查，正確找出要執行的 SQL 指令。

參考文獻

[1]. 吳靜茹，「以合理輸入值驗證為基礎之 SQL 指令植入式攻擊防禦」，中原大學研究所，碩士論文，中華民國九十八年七月。

[2]. 李安娜，「以合理輸入值驗證為基礎之組合型 SQL 指令植入式攻擊防禦」，中原大學研究所，碩士論文，中華民國一○○年七月。

[3]. OWASP Top Ten Project https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[4]. ModSecurity Open Source Web Application Firewall http://www.modsecurity.org/

[5]. Wu X.R. and Chan P.P.K, "SQL injection attacks detection in adversarial environments by k-centers", *IEEE Machine Learning and Cybernetics (ICMLC), 2012 International Conference on (Volume:1), Page406 – 410, 15-17 July 2012*

[6]. Sruthy M., Manesh T., and Varghese P., "SQLStor: Blockage of stored procedure SQL injection attack using dynamic query structure validation." *Intelligent Systems Design and Applications (ISDA), 2012 12th International Conference on*. IEEE, 2012, *Page 240 – 245, 27-29 Nov. 2012*

[7]. Moosa A., "Artificial Neural Network based Web Application Firewall for SQL Injection", *World Academy of Science, Engineering & Technology; Vol. 64, p12, Apr 2010.*

[8]. Ezumalai R., Aghila G., "Combinatorial Approach for Preventing SQL Injection Attacks", *Advance Computing Conference, 2009. IACC 2009. IEEE International, 6-7 March 2009, India*

[9]. Khochare N., Chalurkar S. and Meshram B.B., "Web Application Vulnerabilities Detection Techniques Survey", *IJCSNS International Journal of Computer Science and Network Security, VOL.13 No.6, June 2013*

[10]. Buehrer G.T., Weide B.W., and Sivilotti P.A., "Using parse tree validation to prevent SQL injection attacks." , *International Workshop on Software Engineering and Middleware (SEM) at Joint FSE and ESEC*, pages 106–113, Sept. 2005.

[11]. Wassermann G. and Su Z., "The Essence of Command Injection Attacks in Web Applications", *33rd Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, pages 372-382, Jan. 2006.

[12]. Halfond W., Orso A. and Manolios P., "WASP: Protecting Web applications using positive tainting and syntax-aware evaluation." *Software Engineering, IEEE Transactions on 34.1 (2008): 65-81.*

[13]. Kemalis K. and Tzouramanis T., "SQL-IDS: a specification-based approach for SQL-injection detection." *2008 ACM symposium on Applied computing, pages 2153-2158, 2008.TSE, 34(1):65, 2008*