WIPS：A Practical Intrusion Prevention System* for Web Applications

Jui-Wen Chen Bo-Chao Cheng Ming-Ni Chuang
Information Networking Security and Assurance LAB
Department of Communications Engineering
National Chung Cheng University
g92430034@comm.ccu.edu.tw bcheng@ccu.edu.tw mini@insa.comm.ccu.edu.tw

摘要

近來 Web 應用的蓬勃發展，衍生出許多 Web 應用安全的問題。國際產業研究公司 Gartner Group 也提到在所有攻擊事件中，有百分之七十五是發生在應用層（OSI Application layer），並且四分之三的商業網站是有漏洞的，但是傳統的網路安全設備(例如入侵偵測系統以及防火牆)並不能有效的防止應用層的攻擊。有鑑於此，本論文延伸有限狀態機（finite state machine）的原理及整合 stateful session 檢測機制，提出 Web 入侵防禦系統（WIPS）來解決 Web 應用所造成的安全問題。Web 入侵防禦系統並結合正面表述（positive approach）與負面表述（negative approach）的優點防止 Web 攻擊的產生。本系統已完成設計且將其系統實現在 Intel 網路處理器搭配 MontaVista Linux 的開發平台上，透過功能性與效能性的實際量測可以證明 Web 入侵防禦系統可以有效且快速的阻擋 Web 攻擊，建立一個高安全性的 Web 應用環境來保障企業以及合法使用者的財產安全。

關鍵詞：網路應用安全、入侵偵測系統、有限狀態機、網路處理器

Abstract

Web application portal with the single sign on (SSO) feature provides an integrated E-Business solution such that web application becomes an essential building block for business operations. Gartner Group report indicates that 75% of malicious attacks targeting the application layer and three out of four business Web sites are vulnerable to Application-level attacks. Therefore, the traditional security devices (such as firewall and intrusion detection system) are not able to protect web-based applications any more. Implementing a solid web application security protection shield is top-of-mind of security researchers. Extending the finite state machine theory and coupling with stateful session inspection, we propose Web Intrusion Prevention System (WIPS) to solve web application security issues listed in the OWASP Top Ten project. WIPS works as the last defense line to separate web browsers and web servers by examining network traffic, maintaining every session's state information and allowing only specific web behaviors defined by web finite state machine to pass through. With embedded Snort capability, WIPS also provides negative security models to resist the lower layer attacks. A WIPS prototype has been implemented on Intel Network Processor (IXP425) running with MontaVista Linux. In our study, the functionality and performance has been assessed to show WIPS providing a key answer for advancing the state-of-the-art in web application security in a realistic environment.

Keywords: Web application security, intrusion prevention system, finite state machine, network processor

1. 相關研究與討論

因為 Web 應用的蓬勃發展，越來越多網安人員開始注意其衍生的應用安全問題[5][8]。根據市場調查公司 Gartner 在網路安全分析報告中指出，有百分之七十五的攻擊事件是發生在應用層的攻擊，這些資訊安全事件顯示:現在攻擊的方式也從原本的 OSI 底層（如、網路層及傳輸層）已經轉移到應用層（Application Layer）發展。Gartner 的網路安全研究副總裁 Richard Stiennon 也提出利用 Web Application 漏洞來攻擊並不困難，大概比寫一隻病毒要來的簡單，而且，攻擊者可以利用一些簡單的工具(如、telnet 或 netcat)便可產生 HTTP request 對 Web Application 作一系列攻擊。例如: 2003 年 11 月臺灣花旗銀行發生一件重大的資訊安全事件，由於程式設計人員的疏失，客戶只要上網更改 URL 參數就可以瀏覽其他客戶網路申請信用卡業務的個人資料。再者，根據 Gartner 報告更指出:有四分之三的商業網站在安全上是有漏洞的。

針對Web應用程式安全的重要性，學界和產業界也開始這方面的研究[1]，大致上有四種方向：(1). 原始碼檢查: 利用原始碼檢查工具(如Flawfinder, RATS及ITS4)對所撰寫的網路應用程式做檢查，檢

* This work was sponsored by CCL/ITRI grant T1-94025-3.

查所撰寫的網路應用程式是否含有潛在的程式撰寫錯誤。 (2). 漏洞掃描工具: 當網路應用程式撰寫完成並且安裝至Server運作後，使用漏洞掃描工具(如Nikto, Stealth及Paros)對所安裝的網路應用程式做弱點掃描，協助修改或設定所轉寫的應用程式。(3). Web Application Based 入侵偵測系統: 結合web server 紀錄檔(如syslog)與整合web server本身(如 Mod_security—http://www.modsecurity.org 與URL scanner)，此類型工具運作在Web Server上，分析Web 應用程式是否遭受攻擊。(4). Web應用安全閘道器(Web Application Security Gateway, WASG): 架設WASG於公司所架設之網站與外部網路之間，監控Client和Server之間的封包訊息，可以及時發現攻擊並且將其阻擋下來。因為可適用各種web server、有效處理入侵預防(intrusion prevention) 工作並不影響web server本身效能，所以此種方法將是未來之趨勢。

Giovanni Vigna在2003年提出WebSTAT架構[3]針對Web攻擊行為做Stateful的偵測， WebSTAT是沿襲STAT （State-Transition Analysis Technique）[6]架構，使用state-transition diagram的方式將Web相關攻擊模組化並且放置在攻擊行為資料庫中，藉由三種不同的資訊（Server logs, Operating system-level event以及network-level event）的分析來判斷state轉移狀態，若超過所設定的threshold則認定是攻擊行為。雖然此系統能透過Server端不同的應用程式所產生的event來幫助偵測攻擊，然而入侵偵測系統雖然可以偵測出攻擊，但是卻無法防禦攻擊的發生，當偵測出攻擊行為時，Web應用程式已經遭受攻擊甚至產生無發挽救的傷害。所以無法有效抵擋Web攻擊。

基於上述研究成果，我們可以發現上述解決方案無法十全十美地解決OWASP所公佈的Web Application十大漏洞，其瓶頸與原因探討如下：

1. 效能：使用Application Integrated Based入侵偵測系統方式解決Web應用安全問題時，會使Server處理效能減低，因為入侵偵測系統必須搭配其他應用程式或者和Web Server相互運作。此時入侵偵測系統會額外佔用Web Server和Web應用程式的系統資源，會降低Server端的執行效率。
2. Negative Approach檢測方式：系統使用Negative Approach檢測方式雖然擁有不易誤判的偵測特性，但是依賴此檢測方式並無法有效偵測未知的攻擊與降低管理複雜度。
3. 無預防機制：當傳統 IDS 偵測出攻擊時卻無法及時作出有效的防禦機制以至於系統已經造成無法挽救的傷害。

本篇論文以上述原因為出發點，並且延伸有限狀態機（finite state machine）的原理及整合 stateful session 檢測機制，提出 Web Finite State Machine (WFSM) 來解決上述問題，並依此實現一個 Web Intrusion Prevention System (WIPS)達到 stateful web content-aware inspection 與入侵防禦功能。並將架構實作於網路處理器(Intel IXP425)上，建立一個高安全的 WASG 來保護 Web-based 的網路應用與服務。本文架構如下：首先介紹 Web 應用安全有哪些相關的研究以及瓶頸，接下來介紹 Web 入侵防禦系統的架構以及運作原理，再介紹將 Web 入侵防禦系統實現在 Intel IXP425 網路處理器開發平台的測試結果，最後歸納結論。

2.Web 入侵防禦系統架構（WIPS）

WIP 由兩個子系統所組成，其系統架構如圖 1 所示，分別是 Web Stateful 入侵防禦引擎（Web Stateful Intrusion Prevention Engine）以及最受歡迎的開放原始碼系統 Snort。

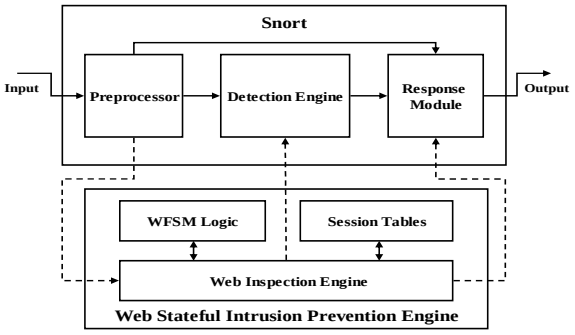

圖 1 Web 入侵防禦系統架構

在Snort子系統中已具備處理封包的基本能力: 可以將截取的封包儲存於其資料結構中，並進行分析工作，然後再經由Response Module做完善的回應處理。因此，在整合的架構中我們只要將HTTP封包傳送給Web Stateful入侵防禦引擎做深層分析(deep layer 7 inspection)，分析完之後再傳遞給Snort Detection Engine做Negative Approach攻擊檢測。若Web Stateful入侵防禦引擎分析封包的結果為不合法時，則呼叫Response Module做進一步的處理。

在此整合架構，WIPS 提供了 OSI 七層的全盤防護，在本章內容接下來的部份，我們將介紹 Web Stateful 入侵防禦引擎的基本架構以及運作原理，並且透過攻擊範例來驗證為何 Web Stateful 入侵防禦引擎能有效的防禦 Web 攻擊。

2.1 Web Stateful 入侵防禦引擎

Web Stateful 入侵防禦引擎是一套結合有限狀態機概念提供 Web 流量深層檢測能力的系統架構，並且使用 Positive Approach 檢測方式將不合法的 HTTP Request 阻擋下來。Web Stateful 入侵防禦引擎架構如**圖 1**所示，主要由三個元件所組成，分別是 Web 有限狀態機（Web Finite State Machine）、Session Tables 以及 Web 入侵行為檢測引擎（Web Inspection Engine）。

(1) **Web 有限狀態機（WFSM）**

在 Web 流量中顯示出四個值得關注的特性：1.HTTP 協定 Stateless 的特性讓使用者透過超連結（Hyperlink）瀏覽網頁的順序並不會相同。2.因為 Session 管理機制的產生，讓 HTTP 協定達到 Stateful 的特性。所以使用者使用 Application 的行為可能會橫跨 Stateless 以及 Stateful 階段，並且在不同的階段建立多條 TCP 連線。3.一個網頁可能包含許多物件（Object）供使用者瀏覽。4.若 Web 應用程式使用 Session 管理的機制，所使用的識別 ID 可能被包含在 URL、hidden field 或者 cookie 中。我們提出 Web 有限狀態機（Web Finite State Machine, WFSM）架構，一個具有 Web Session Stataful 檢測能力的三層決策有限狀態機，期望能夠有效保護 Web 應用安全。為了正規地描述 WFSM 運作原理，我們定義了以下項目：

- **Ω**： HTTP Request，根據 HTTP /1.1 [9]以及 cookie [7]的定義，我們可以從 Client 所送出的訊息中取得 HTTP Request。HTTP Request（Ω）包含 http method、request-URI、request header fields、cookie 名稱與內容以及自定的欄位。假設 h_i = <field-name, filed-value>表示 HTTP Request 第 *i* 個欄位的名稱以及內容，則 $\Omega = \bigcup_{i=1}^{i=n}\{h_i\}$。
- **μ**：Client 所送出的 URL request。
- **μ_r**：當 Server 接收到 μ 時，所送出的 Response 訊息。
- **h_u**：代表 HTTP Request 中的 Request-URI，h_u = <request_URI, μ> where $h_u \in \Omega$。
- **O**：Access object list (AOL)，代表在相同 State 中所有允許存取物件以及對應的 Server Response 行為的集合，假設 a_i = <request_uri, server_response> 為 AOL 第 *i* 個物件的名稱以及相對應的 Server Response 行為，則 $O = \bigcup_{i=1}^{i=n}\{a_i\}$。
- **φ（Ω, O）**：為一布林運算函數，若回傳值為真（True）則代表 μ = a. request_uri where $h_u \in \Omega$ and a ∈O。
- **γ（Ω, O）**：為一布林運算函數，若回傳值為真（True）則代表 μ_r = a.server_response where $h_u \in \Omega$ and a ∈O。
- **ω（Ω, O）**：為一布林運算函數，若回傳值為真（True）則代表 μ = a.request_uri 以及 μ_r = a.server_response where $h_u \in \Omega$ and a ∈O。
- **χ**：Client 送出的 URL Request 造成合法 inter-state transition 的條件，$h_\chi \notin O$。
- ***t***：TCP 連接的 time out 時間參數，也適用於 State time out。
- ***T***：Phase time out 時間參數。
- **Γ**：Phase check list，用來定義在同一 Phase 內 HTTP Request 只允許或不被接受的資訊。$\Gamma = \bigcup_{i=1}^{i=n}\{c_i\}$ where c_i = < condition name, condition value>；condition name 可以跟任何資訊相關例如 session id、user id、referer、參數長度、timeout *T*、http method 以及使用者自訂的參數。
- **ψ**：State Check list，內容類似於 Phase check list，用來定義在同一 State 內 HTTP Request 只允許或不被接受的資訊。
- **κ（h,Y）**：為一布林運算函數，若回傳值為真（True）則代表 h 符合 Y 的條件，若為假（False）則不符合。
- **K（Ω, Y）**：為一布林運算函數，若回傳值為真（True）則代表 $\Pi_{i=1}^{i=n}\kappa(h_i, Y) = true$ where $h_i \in \Omega$。
- **ρ**：Entry Action，用來定義 State transition 到達目的 State 時必須預先處理的動作，例如將封包 Pass、Deny、Drop 的處理動作或者是其他使用者自訂的行為。
- **σ**：Exit Action，內容類似於 Entry Action，用來定義 State 轉移前必須處理的動作。

為了達到健全的檢測機制，WFSM使用三層有限狀態機的機制達到Session狀態的維持以及驗證Client Request的Server Response行為是否符合預期。三層有限狀態機的機制是由Intra-WFSM、Inter-WFSM以及Global-WFSM所組成，架構如圖 2 所示，詳細運作原理將在以下內容介紹。

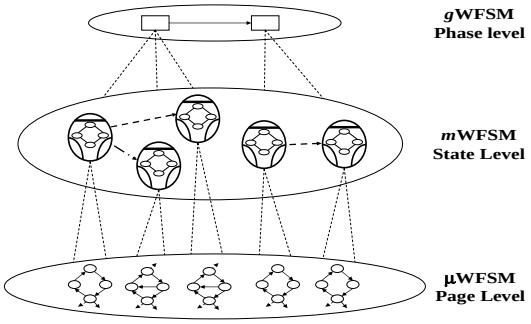

圖 2 WFSM 架構圖

■ Intra-WFSM（μWFSM）

μWFSM 是一個簡單且具有生產者-消費者（producer-consumer）關係的狀態機，在每一個 WFSM 所定義的 State 中皆具有此狀態機對 Client Request 以及 Server Response 作判斷。μWFSM 狀態圖如圖 3所示。就單純的 Web 流量 Stateless 處理行為而言，可以被 μWFSM 清楚表示之。

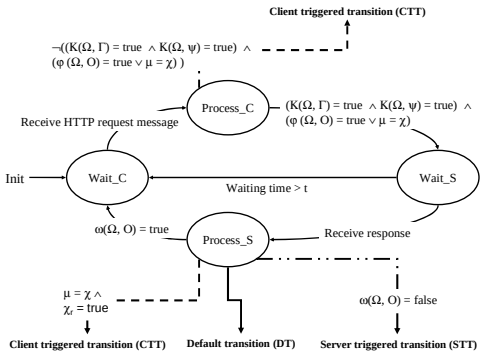

圖 3 μWFSM 狀態圖

基於正面表述(Positive Approach)檢測方式，μWFSM 明確的定義被允許的 Web 應用程式使用行為（包含 Client Request 以及 Server Response 行為），若使用行為不符合其規定則將可疑的封包阻擋下來。在 μWFSM 中，初始狀態 Wait_C 代表等待 Client 送出 HTTP Request 的狀態。當系統收到由 Client 送出的 HTTP Request（Ω），狀態就由原本的 Wait_C 轉換至 Process_C 對 Ω 內容做檢測工作:比對 Phase check list（Γ）、State Check list（ψ）、AOL（Ｏ）與檢視 inter-state transition 條件（χ）。若發現不符合定義行為的內容則經由 client triggered transition (CTT)進入另外一個 State 處理不合法行為。反之若通過條件檢測則將狀態轉換至 Wait_S 狀態等待 Server Response（μ_r），若等待時間超過 timeout 時間 *t* 還未收到 μ_r，狀態將由 Wait_S 回到初始狀態 Wait_C 等待下一個 Ω。若 *t* 時間內收到 μ_r 則狀態將會轉移至 Process_S 狀態做進一步處理，在 Process_S 狀態有四種 transition 狀況可能發生：（1）若 Ω 對 State AOL 所定義的物件作存取，μ_r 與 AOL 所定義的 Server Response 行為相符合則回到 Wait_C 初始狀態等待下一個 Ω。（2）若 $\mu=\chi$ 且 $\mu_r=\chi_r$ 則符合 inter-state transition 條件，Session 狀態經由 CTT 執行 Inter State 轉換至下一個 State。（3）若 μ_r 與 AOL 物件或者是 χ_r 所定義的 Server Response 行為不符合，則經由 Server triggered transition（STT）進入 Error Inter-State 進行錯誤處理程式。(4)若前三種情況都不符合，則經由 Default transition 進入 Default Inter-State 進行預設的處理程式。

■ Inter-WFSM（*m*WFSM）

*m*WFSM定義了Inter-State transition關係。其組成元件如圖 4所示，每一個State包含：State ID（STID）、μWFSM、O、Γ（由gWFSM所定義並繼承之）、Ψ、State type（代表initial state, regular state, error state或者是sink state）、Entry Action、Exit Action以及註解。*m*WFSM是一個由六種元素<α, ε, S, S_0, η, λ>組成的有限狀態機，每一個元素定義如下：

◆ α：Client triggered transition requests以及server triggered transition responses的有限輸入標記

◆ ε：表示警報訊息（Alert）的輸出符號

◆ S：不包含空集合的所有State的集合

◆ S_0：初始狀態（Initial State），為S集合中的一個元素

◆ η：State transition function： $\eta : S \times \alpha \rightarrow S$

◆ λ：Output function： $\lambda : S \rightarrow \varepsilon$

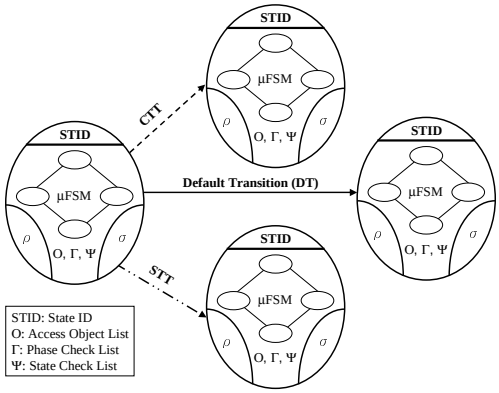

圖 4 *m*WFSM 狀態圖

如果只要達到一般 web-content awareness 及 filter，WFSM 即可以三個 *m*WFSM 完成: (1). CTT Error State: 可以利用定義於 Entry Action (ρ)或 Exit Action (σ)之 functions 處理任何不當 HTTP request。(2). STT Error State: 類似 CTT Error State，用來應付 server 不正常回覆。(3). Nomal State: 在此 state 中嵌入一個 *u*WFSM 來處理一般 stateless client-server 檢視工作。

■ Global-WFSM（*g*WFSM）

*g*WFSM 主要是定義各種不同 phase (如 Browsing Phase、Shopping Cart Phase、Check Out Phase)及 Phase 之間的轉換關係。在同一個 Phase 可包含許多 State，而每一個 phase 都有它獨特的 business logic。關於 Phase Type 一共有兩種: Surfing and Transaction。而因為使用者可未經由認證之下先行瀏覽網站內容，我們將此類狀況定義為 Surfing。反之若 Application 皆使用 Session 管理機制追蹤使用者行為，此時 Phase 狀態就由 Surfing 轉移至 Transaction。如此一來，*g*WFSM 可以能夠因應不同階段的需求使用不同的 Session 資訊內容管理 web Traffics。

(2) Web 入侵行為檢測引擎（Web Inspection Engine - WIE）

Web 入侵行為檢測引擎為 Web Stateful 入侵防禦引擎的核心部份。當系統管理者將 WFSM 設定完成可使用 WFSM Importer 將 WFSM Profile 讀入後，Web 入侵行為檢測引擎就能夠搭配 Surfing 以及 Transaction 型態執行 Web Stateful 的入侵防禦。Web 入侵行為檢測引擎演算法分成兩個部分，分別處理系統所接收到的 Client Request 以及 Server Response 行為。Web Inspection Engine Client Request 處理演算法如圖 5所示。

```
c_mid = current mFSM id;
n_mid = next mFSM id;
c_gid = current gFSM id;
n_gid = next gFSM id;
For each client packet CP {
        Ω = extractRequest (CP);
        stn_id = lookUpIDClientPacket(CP, &c_mid, &c_gid);
        if (stn_id < 0) {
                insertSessionTable(CP, &c_mid, &c_gid);
        }
        Γ = getPhaseCheckList (c_gid);
        Ψ = getStateCheckList (c_mid);
        O = getAccessObjectList (c_mid);
        processClientRequest(Ω, Γ, Ψ, O, c_mid, &n_mid, c_gid, &n_gid);
        if (c_mid != n_mid) {
                exitAction(c_mid);
                trasnition Action (c_mid, n_mid, c_gid, n_gid);
                entryAction(n_mid);
        }
        else {
                forwardPacket (CP);
        }
        updateSessionTable (stn_id, CP, n_mid, n_gid);
}
```

圖 5 WIE Client Request 處理演算法

Web Inspection Engine Server Response 處理演算法如圖 6所示，當系統收到由 Server 端所回傳的 Response 封包後，查詢封包資訊是相關於哪一條 Session，若找不出相對應的 Session 資訊則將封包 Drop，若找出相對應的 Session 資訊系統則呼叫 processServerResponse 函式進行判斷，若符合 State transition 條件則進行相關封包處理。若判斷結果為合法 AOL 存取行為，則 forward packet 並且將 μWFSM 狀態返回至 Wait_C 狀態。

```
c_mid = current mFSM id;
n_mid = next mFSM id;
c_gid = current gFSM id;
n_gid = next gFSM id;
For each server packet SP {
        stn_id = lookUPIDServerResponse (SP, &c_mid, &c_gid);
        if (stn_id < 0) {
                drop packet;
        }
        else {
                Ω = getClientRequest (stn_id);
                O = getAccessObjectList (c_mid);
                processServerResponse(Ω, O, SP, c_mid, &n_mid, c_gid, &n_gid);
                if (c_mid != n_mid) {
                        exitAction(c_mid);
                        trasnitionAction (c_mid, n_mid, c_gid, n_gid);
                        entryAction(n_mid);
                }
                else {
                        forwardPacket (SP);
                }
        updateSessionTable (stn_id, NULL, n_mid, n_gid);
        }
}
```

圖 6 WIE Server Response 處理演算法

透過 Web 入侵行為檢測引擎結合 Surfing 以及 Transaction 型態以及 WFSM，可以讓 Web Stateful 入侵防禦引擎擁有雙向 Stateful 深層檢測的能力。並且能夠有效的防止 Web 應用攻擊。下一節我們將使用實際的攻擊範例驗證 Web Stateful 入侵防禦引擎是否能夠有效的防止 Web 應用攻擊。

2.2Web Stateful 入侵防禦引擎防禦攻擊範例

WFSM 範例如圖 7所示，在 Transaction 型態我們定義三個 Inter-State，State ID 分別是 2000、2501 以及 2808。三種 Transition 關係也明確定義其狀態轉移的條件。當 State 2000 要轉移至 State 2501 時，必須符合 Intra-state 所判斷的 Server Response 結果以及使用者所送出的 HTTP Request 中 referer 欄位必須是 State 2000 所包含的 URL，若判斷成功則 State 狀態由 2000 轉移至 2501，反之則進入到 Error State 2808。當 State 狀態轉移前後系統分別會呼叫 Entry Action（ρ）以及 Exit Action（σ）執行封包相關的處理動作。

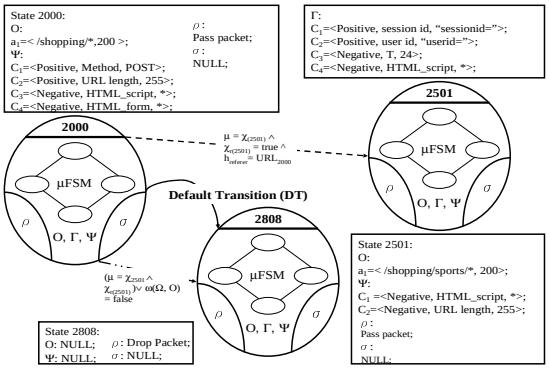

圖 7 WFSM 範例

當系統收到由Client端所送出的HTTP Request除了符合狀態轉移的條件之外，還必須符合Phase checklist以及State checklist所設定的檢查項目，如不符合Phase以及State checklist的檢測，會將其認定為不合法的使用者行為。在範例中的Phase checklist以及State checklist中定義了符合使用環境的檢查項目，詳細定義內容如圖 8所示。

Γ	C_1： 此階段使用Session ID的Session管理模式，使用者送出的HTTP Request會包含Session ID的訊息，此訊息出現在"sessionid="字串之後。
	C_2： 使用者送出的HTTP Request也會包含User ID的訊息，此訊息出現在"userid="字串之後。
	C_3： 使用timeout機制，時間為24小時。
	C_4： 不允許使用者送出的HTTP Request中包含HTML Script標籤。
Ψ_{2000}	C_1： 只允許使用HTTP POST Method。
	C_2： 最大URL長度為255 Bytes。
	C_3： 不允許使用者送出的HTTP Request中包含HTML Script標籤。
	C_4： 不允許使用者送出的HTTP Request中包含HTML Form標籤。
Ψ_{2501}	C_1： 不允許使用者送出的HTTP Request中包含HTML Script標籤。
	C_2： 最大URL長度為255 Bytes。

圖 8 Phase checklist 與 State Checklist 範例

3. Web 入侵防禦系統整合 Snort 並且實現在 IXP425 開發平台

在本章節我們將介紹 Web 入侵防禦系統實現在 Intel IXP425 開發平台的系統量測結果。

3.1 WIPS 在 IXP425 開發平台的量測

WIPS測試環境如圖 9所示，WIPS在IXP425開發平台的量測由功能性與效能測試兩個部份所組成，各測試內容如下所示。

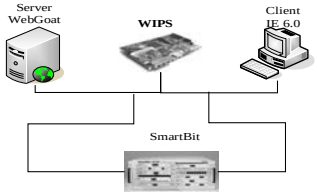

圖 9 WIPS 測試環境

- **WIPS功能性測試**

在WIPS功能性的量測部分，我們使用OWASP組織所發佈的WebGoat軟體，將其安裝在Server機器裡，在透過Client遠端登入Server機器內進行驗證WIPS是否能夠將攻擊有效阻擋下來。驗證結果與Snort Inline比較如圖 10所示。

OWASP Top Ten	WIPS	Snort Inline
A1_Unvalidated Input	O	O
A2_Broken Access Control	O	×
A3_Broken Authentication and Session Management	O	×
A4_Cross Site Scripting (XSS) Flaws	O	×
A5_Buffer Overflows	O	O
A6_Injection Flaws	O	×
A7_Improper Error Handling	O	×
A8_Insecure Storage	×	×
A9_Denial of Service	O	O
A10_Insecure Configuration Management	O	×

圖 10 WIPS 與 Snort Inline 防禦功能分析

WIPS因為沒有檢查加密儲存機制所以無法解決此類攻擊問題（A8_Insecure Storage），至於其他類的攻擊，WIPS可以達到一定程度的防禦功效。至於Snort Inline，只能在其中的三類提供解決方案，由此證明WIPS比未經整合後的Snort系統更能防止Web應用的攻擊。

- **WIPS的效能測試**

在WIPS的效能測試中項目中，我們選擇兩項測試標準測試WIPS的效能，一項是Throughput測試，另外一項測試為Latency測試。系統中的WFSM Logic採用圖 7範例，測試過程中傳送合法的HTTP封包進行量測，由於WIPS是由SwBridgeCodelet原始碼修改而成，故以SwBridgeCodelet為Baseline，量測WIPS之效能。

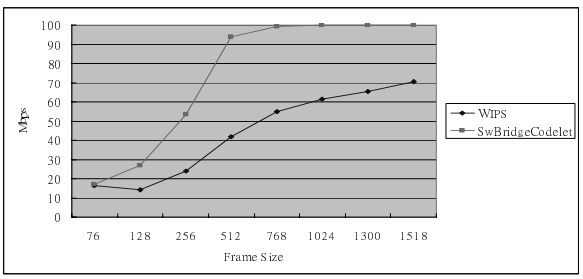

圖 11 WIPS 與 SwBridgeCodelet Throughput 測試

WIPS 與 SwBridgeCodelet Latency 比較如圖所示，在圖中我們可以觀察出 WIPS 的 Latency 並不會落後 SwBridgeCodelet 太多，所以 WIPS 封包延遲時間算是在可接受的範圍之內。

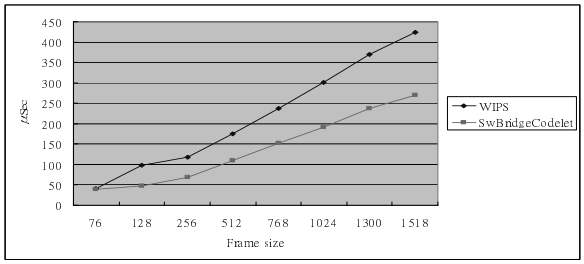

圖 12 WIPS 與 SwBridgeCodelet Latency 比較

4.結論

在本篇論文中我們提出了Web入侵防禦系統架構，以Web Session管理機制為基礎，搭配有限狀態機的（Finite State Machine）的概念提供Web入侵防禦機制。透過Web Finite State Machine的定義，讓Web入侵防禦系統可以提供Web Stateful的深層檢測能力，更結合Positive Approach以及Negative Approach檢測方式讓Web入侵防禦系統降低攻擊誤判率，增加系統可靠度。我們將WIPS功能實現於IXP425網路處理器開發平台上，並驗證其可行性。所以，WIPS提供一個完整的Web應用安全解決方案。

參考文獻

[1] B. Violino, "Decoding Application Security",May 2004. at:http://www.csoonline.com/read/050104/application.html

[2] G. Zuchlinski, “The Anatomy of Cross Site Scripting”,November 2005 .at:http://www.net-security.org/dl/articles/xss_anatomy.pdf

[3] G. Vigna, W. Robertson, V. Kher and R. A. Kemmerer, “A Stateful Intrusion Detection System for World-Wide Web Servers”, ACSAC, Dec 2003.

[4] G. Ollmann, “HTML Code Injection and Cross-site scripting”, at:http://www.technicalinfo.net/papers/CSS.html

[5] G. Ollmann, “Web Based Session Management", at:http://www.technicalinfo.net/papers/WebBasedSessionManagement.html

[6] K. Ilgun, R.A. Kemmerer, and P.A. Porras, “State Transition Analysis: A Rule-Based Intrusion Detection System”, IEEE Transactions on Software Engineering, 21(3):181–199, March 1995.

[7] D. Kristol, and L. Montulli, “HTTP State Management Mechanism”, RFC 2965, October 2000.

[8] OWASP, “The Ten Most Critical Web Application Security Vulnerabilities 2004 Update”, January 27th, 2004

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP 1.1. RFC 2616”, June 1999.

[10] H. Lewis and C. Papadimitriou, "Elements of the Theory of Computation", Prentice Hall; 2nd edition, August 7, 1997.