以關聯規則法分析攻擊探測程式之緩衝區溢位防治研究

李常友 陳嘉玫 鄭炳強 趙善中 林孝忠
國立中山大學資訊管理系
d934020003@student.nsysu.edu.tw

摘要

隨著軟體應用程式與網際網路的發展，其所伴隨而來的安全問題也日益嚴重。緩衝區溢位是軟體寫作上不可避免的問題，根據每年安全通報資料顯示，許多安全上的漏洞源自於緩衝區溢位，並成為駭客入侵攻擊的重大來源。軟體應用程式的一般使用者通常只能依靠軟體廠商發布的更新程式來防止因緩衝區溢位造成的攻擊，於是在尚未使用更新程式前，如何避免軟體遭受到緩衝區溢位攻擊、儘量延長軟體使用的安全時效是防治緩衝區溢位的重點。藉由搜集、分析駭客使用的攻擊探測碼可建立起整體緩衝區溢位攻擊手法的模式，並可將此模式作為未來防治緩衝區溢位攻擊的基礎。

關聯法則能夠發掘未知事物間的關聯性，因此可以協助建立緩衝區溢位攻擊間共有的特徵模式。本研究應用關聯法則以建立緩衝區溢位攻擊模式，找出攻擊探測碼裡系統呼叫間的關係，實驗並建立可區分攻擊行為與正常行為的系統呼叫規則。根據關聯法則建立的規則能夠正確地偵測出緩衝區溢位攻擊，同時在誤判率也有很好的表現，進而能夠協助建立偵測到攻擊後的防範措施，降低系統遭受緩衝區溢位攻擊的嚴重性。

關鍵詞：緩衝區溢位、攻擊探測碼、關聯法則、系統呼叫。

1.前言

緩衝區溢位(Buffer overflow)為由來已久的一個著名程式撰寫上的弱點。緩衝區溢位源自於 C 語言天生上的缺陷，因為 C 語言允許程式設計師可以自由地存取使用記憶體來撰寫程式，雖然對程式寫作上來說非常便利卻也有相對的危險，因為有可能在非預期狀況下對記憶體的不當使用而產生意料之外的程式執行結果。緩衝區溢位就是一種不當使用記憶體的結果。所謂 Buffer，一般是指一段連續的記憶體區塊，用來存放程式裡宣告的變數等資料，而緩衝區溢位為當變數輸入的資料超出宣告時預定分配的記憶體空間，使得部分資料溢出原有的位址，造成強迫去破壞更改目前其他記憶體區塊上的資料，進而導致程式執行流程的轉向或當機。

緩衝區溢位會使得程式的返回位址遭受破壞進而被取代，取代後的位址並不是合法的記憶體位址，因此使得程式無法正常返回而造成當機。然而，更嚴重的情況則為在緩衝區溢位後，以正確的記憶體位址取代原始的返回位址，因為返回位址的變更而致使程式的流程改變，攻擊者可以事先植入攻擊程式在記憶體的某處，再以緩衝區溢位手法將攻擊程式的記憶體位址取代掉原始的返回位址，進而誘導程式流程轉向執行攻擊者的程式，以達成攻擊的目的。通常程式都以系統管理者權限執行，因此攻擊者能利用緩衝區溢位以取得系統管理者的執行權限，進而輕易地在受害系統上為所欲為，故緩衝區溢位攻擊的威力是相當強大的。

在撰寫程式時，只要對於輸入資料檢查不夠完善就有可能產生潛在性緩衝區溢位的風險，因此緩衝區溢位常常成為應用軟體弱點來源之一，不論何種類型、平台的應用程式皆有存在緩衝區溢位的可能性。

近代相關緩衝區溢位的弱點不但數量多，且一直占有相當大的弱點比例，因此緩衝區溢位的問題時至今日依然尚未獲得有效的解決，且過去許多造成重大影響的著名蠕蟲，如 Code Red、SQL Slammer、Blaster 等都是以緩衝區溢位為基礎的攻擊方式。雖然相關的研究已行之有年，但到目前仍然是值得發展與探討的領域。

緩衝區溢位問題的根源發生在原始程式有心或無意地撰寫失當，沒有適當地檢查記憶體的緩衝區是否可能造成溢位，因此理論上要真正解決與防治其實很簡單與直覺，只要比對原始程式裡所有資料的大小與輸入時的大小，藉此檢查是否有無超越界限的情況發生，並修正原始程式，此也為最根治的方式。但是此檢查修正原始程式卻是非常不實際的作法，有如下兩點理由：

(1).檢查工作的困難：在開發程式時，程式開發人員不可能隨時都將程式檢查地面面俱到，因為只要程式一多，檢查的工作自然會益發困難。雖然目前已有許多研究發展出自動化工具以協助檢查的效率，但這些工具也不是完全可靠，總有遺漏的地方。加上有心的攻擊者會使用其他更高階、聰明的方式來繞過檢查。

(2).原始碼取得困難：檢查程式原始碼必需先有權取得原始碼才行，對於一般購買軟體應用程式的使用者來說不可能取得軟體原始碼，且就算能取得軟體原始碼，使用者也不可能自己作檢查的工作。

就某種程度上而言，必須承認緩衝區溢位攻擊一直都存在，實際上不可能完全消除或避免，特別是對於軟體程式的使用者而言，對軟體裡潛在的緩衝區溢位弱點之解決方式只能依賴廠商所發布的更新程式來更新以防範。

本研究的目的在於以軟體程式之使用者角度為出發點，在開發廠商的更新程式公布之前，儘量延長系統安全的時間與保持系統安全的狀態，以降低被緩衝區溢位攻擊的可能性，同時避免遭受到零日攻擊(zero-day attack)的威脅。切入的方向在於不管使用者擁有何種軟體程式或運行何種服務，只針對分析系統上的運作狀況為主，以觀察系統的作業行為來判斷是否有緩衝區溢位攻擊存在，以保護廠商尚未發布更新或尚未使用更新之前的電腦系統。

接下來，第二章為相關緩衝區溢位問題的文獻探討，第三章為本研究的方法與步驟，第四章為實驗結果與評估，第五章為結論。

2.文獻探討

緩衝區溢位攻擊有許多不同的特徵，根據Cowan [5]所提出的整理，可以將緩衝區溢位攻擊歸納按照下列條件來分類：

(1).執行攻擊碼的方式：攻擊者如何使受害系統的記憶體載入並執行緩衝區溢位的攻擊碼，可分為下列兩種方法。

(i).植入式：許多應用程式或服務會要求使用者輸入適當的字串以執行，攻擊者便假裝正常使用者但卻輸入惡意的字串以讓程式執行。

(ii).直接使用受害主機上的程式碼：在產生緩衝區溢位後，直接呼叫執行在受害主機上現有的程式及函式庫。

(2).發生溢出情形的區域：以記憶體所配置的區域來分，Buffer Overflow 可能發生在下列三種區域。

(i).Stack 區域：Stack 為宣告時期各副程式之區域變數預先在記憶體內所分配到的區域。也稱為”Stack Smashing”。

(ii).Heap 區域：程式執行時期變數動態地在記憶體內所分配到的區域，例如使用 C 語言”malloc”函式所分配的記憶體區域。

(iii).擺放靜態資料的區域：程式初始化時資料存在記憶體內的區域。

(3).被溢位所破壞的程式狀態：

(i).Activation Records：副程式執行完畢後的返回位址，最典型的緩衝區溢位擊就是覆寫破壞此種狀態的值。

(ii).Function Pointers：指向副程式或函式的指標，攻擊者藉由溢位後更改此種指標的內容，使得以後函式再度被呼叫時，便能呼叫攻擊者所植入的程式碼。

(iii).其它類型的資料：有些特別的緩衝區溢位攻擊所溢位的對象不限於上述兩種，有可能是針對記憶體內某些鄰近的變數及資料結構等等。

緩衝區溢位相關的研究已行之有年，防治方式一為針對執行時期的檢查方式，如表 1 所示；其二為檢查記憶體界限的方式，如表 2 所示；其三為靜態分析方式，如表 3 所示；其四為專門針對輸入資料的方式，如表 4 所示；最後則為其他方式，諸如正確的程式寫作方式、使用安全的程式語言。

表 1 針對執行時期的檢查方式

論文名稱或技術	內容簡述
StackGuard[4]	在返回位址旁的記憶體位址安插一個 “canary word” 的數值當檢查碼。與程式執行前的值一致，代表程式正常，否則代表產生緩衝區溢位。
StackShield[13]	作法與 StackGuard 類似，只是在副程式呼叫前先將返回位址備份到溢位無法到達的記憶體區塊，副程式結束前抓出之前備份的值與目前的返回位址比對是否一致。
Libsafe[2]	將程式中容易發生緩衝區溢位的相關字串處理函式，如 strcpy、gets、scanf、sprintf 等，置換成具備安全檢查能力的函式，可檢查輸入的字串是否有超出原有的大小，有則結束程式，無則呼叫原有的函式執行。

表 2 檢查記憶體界限的方式

論文名稱或技術	內容簡述
Backwards-compatible bounds checking for arrays and pointers in C programs[9]	為對 GNU C 編譯器的延伸技術，維護一張記錄所有有效資訊的表格，包括資料的基底位址、大小，heap、stack 的插入與刪除的資訊。檢查方式為界限的檢查以及對此表格的檢查。
Purify: Fast detection for memory leaks and access errors[8]	Purify 為商用的對執行時期記憶體存取的除錯工具。優點為將檢查的程式碼直接插入程式的目的碼而不用存取程式的原始碼。可檢查所有的記憶體存取，陣列的檢查方式為對 malloc 所傳回的記憶體區塊兩端作記號，但缺乏原始程式的可用型態或範圍的資訊。

表 3 靜態分析方式

論文名稱或技術	內容簡述
A first step towards automated detection of buffer overrun vulnerabilities[7]	用一對數字構成的範圍來代表一個字串 buffer (上限與下限),以表示此字串分配空間的大小與長度，限制為在操作字串前需先定義，藉由檢查原始程式內每個字串的 buffer 所參考分配到的大小是否最少會大於數字的最大長度。
Statically detecting likely buffer overflow vulnerabilities[6]	使用語意化的註解來偵測程式中潛在的緩衝區溢位，可保護有註解的函式，但程式設計師在撰寫程式時必須提供註解。

表 4 針對輸入資料的檢查方式

論文名稱或技術	內容簡述
Buttercup: On Network-based Detection of Polymorphic Buffer Overflow Vulnerabilities[3]	提出可對付多型態攻擊的辦法：Buttercup。觀察 input 並找出攻擊程式可能溢位到的記憶位址。先定出可能溢位的上限與下限位址，在上限與下限之間再找出可能發生溢位的記憶體範圍。
Network-Based Buffer Overflow Detection by Exploit Code Analysis [14]	Buffer Overflow 攻擊封包通常夾帶呼叫重要的 system call，因此在 input 封包內檢查是否有呼叫 system call 必需的中斷指令存在，並寫成模組以加強 Snort 的偵測能力。

以上文獻中所提出的防治方式，各有不同的應用層面，有些是針對不同的攻擊型態而有不同的方式，但卻較少有研究可針對整體攻擊行為模式的建立，因此本研究著重於建立整體攻擊行為模式，廣泛地搜集、分析舊有緩衝區溢位攻擊的特徵，以做為防範新式緩衝區溢位攻擊的基礎。接下來第三章將詳述研究方法與步驟。

3.攻擊探測碼的分析方法

在明確描述研究問題之前，必須先解釋什麼是 exploit code、shell code、與 system call。所謂的 exploit code(攻擊探測碼)是指有心人士，針對其所發現的特定軟體弱點，精心撰寫而成的完整攻擊程式；此攻擊程式會探測並利用個別的弱點來達成攻擊者欲達成的目的，包括在受害系統上執行任意程式或使系統失效等。shell code 指的是一些十六進位可執行的機器指令碼，通常在 exploit code 裡都會出現不少的 shell code，為是攻擊探測碼的核心，功用為攻擊者拿來攻陷遠端系統及在遠端系統上執行的惡意指令。其中有很多是專門針對緩衝區溢位弱點所開發的攻擊探測碼。通常在緩衝區溢位攻擊探測程式的 shell code 裡，除了夾帶了用來溢位的字串外，還包含了攻擊者所欲執行的重要指令，這些指令大多都是在呼叫重要的系統函式(system call)，以達成攻擊者的目的。

以 Linux 平台為例，所有的 system call 都有一個獨一無二的編號以供識別，而呼叫 system call 的步驟必須先將 CPU 內暫存器的值設定為欲呼叫的 system call 的編號，再發出一個中斷訊號 (interrupt)，接著 CPU 呼叫剛剛暫存器內所設定編號的 system call。以呼叫"execve" system call 為說明，列表整理出對應的指令，如表 5 所示。

表 5 呼叫 execve 的步驟順序與指令

動作	指令	機器碼
將暫存器設定為欲呼叫 system call 的編號	mov $11, %ax	b0 0b
中斷訊號	int 80	cd 80

如表 5 所示，execve 的編號為 11，將暫存器 ax 設定為 11 再發出中斷訊號即呼叫 execve，其所對應的機器碼為「b0 0b cd 80」。因此若在 shell code 裡發現「\xb0\x0b\xcd\x80」的字串就代表這個 shell code 會呼叫"execve"system call。

在 "Network-Based Buffer Overflow Detection by Exploit Code Analysis"[14]提出的方式也是以上述系統呼叫方式為基礎，搜尋網路封包內含有「\xcd\x80」的字樣，將單一封包內出現此 2 次 system call 的呼叫行為視為攻擊，並根據呼叫 system call 的重要性來發出警報提醒可能的緩衝區溢位攻擊。雖然能即時找出潛在的緩衝區溢位攻擊，但不是很精確，因為正常的系統行為也會呼叫這些重要的 system call，故此方式較難以區分攻擊行為與正常行為，難免會因誤判率過高而將正常行為也視作攻擊行為。故本研究以此搜尋 system call 的方式為基礎並再加以深化，期望建立出有別於正常行為的攻擊行為模式來偵測緩衝區溢位攻擊，使偵測能夠更為精確。

為了建立攻擊模式，先一一找出 shell code 裡所呼叫的 system call 之情形並記錄，接著根據呼叫情形的記錄更進一步建立起它們之間共有的模式，歸納出攻擊程式在呼叫 system call 時的規則，接著利用規則來偵測緩衝區溢位攻擊。本研究採用關聯法則(Association Rule)建立攻擊規則。

在眾多規則中，不是每條規則都是有用的，在關聯法則裡有三項指標可供判斷那些規則是有用的[1]：

(1).信心水準 (Confidence)：指的是這條規則的準確度有多少，信心水準越高的話，代表這條規則的準確度也越高，越有參考的價值。

(2).支援 (Support)：雖然信心水準越高代表準確度越高，但也不代表是真正有用的規則，支持的案例數量太少，代表該條規則沒什麼參考價值。支援指

的是符合該條規則的案例數量。
(3).重要性 (Importance)：此值大於零且越大，代表此規則越顯著。反之，若小於零，則代表該規則的效用不大。

本研究的系統架構，主要共有兩個階段，第一階段為比對攻擊行為階段，如圖 1 所示；第二階段為比對正常行為階段，如圖 2 所示。

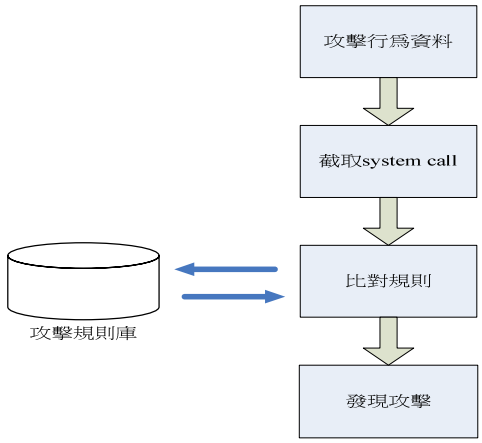

圖 1 攻擊行為的系統架構

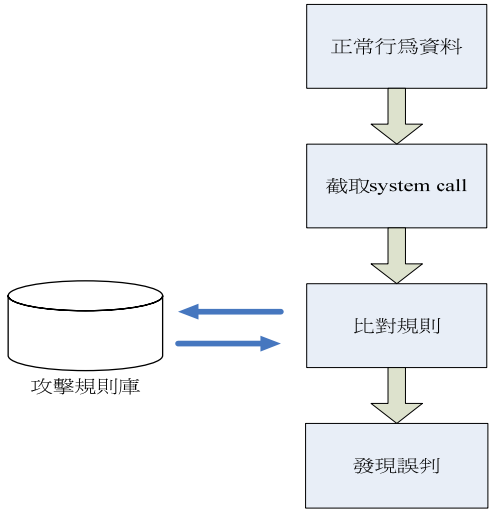

圖 2 正常行為的系統架構

本研究共設計「反組譯模組」、「截取 system call 模組」、「規則庫建立模組」、「比對攻擊行為模組」、「比對正常行為模組」等 5 大模組來進行實驗與模擬，各模組詳述如下：

3.1 反組譯模組

在建立起可供比對攻擊與正常行為的規則庫之前，必須先搜集攻擊探測碼來當作規則庫的訓練資料，由於本研究是針對 Linux 平台上的防治，所以搜集了網路上近幾年在 Linux 上已公布的攻擊探測程式，這些攻擊探測程式不限定任何特定的服務或應用程式，較為廣泛的搜集是期望能建立起較完整的攻擊模式。搜集完成後，必須將這些攻擊探測程式裡所含的 shell codes 給截取出來，但在截取之前，必須先將這些 shell codes 反組譯成組合語言後才能便於「截取 system call 模組」來截取 system call。

3.2 截取 system call 模組

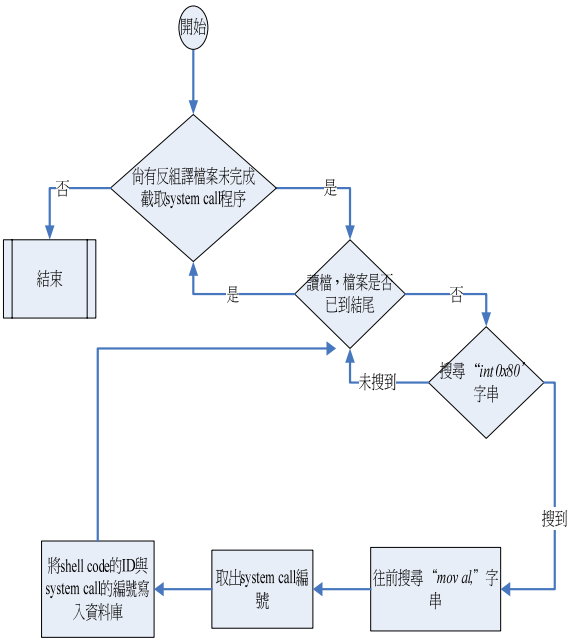

圖 3 截取 system call 的演算法流程

擷取 system call 的整個流程如圖 3 所示。在所有搜集之 shell codes 的 system call 截取完畢後，最後產生的資料庫可供「規則庫建立模組」來建立供比對攻擊行為與正常行為的規則庫。

3.3 規則庫建立模組

應用關聯法則產生可供比對的規則，在此使用 Apriori[11]演算法來產生規則。Apriori 演算法有幾個重要的參數需要設定[1]，如下所述：
(1).最大集合物件項目數：物件為集合裡的元素，集合物件項目數就是此集合裡的物件數目。
(2).最小支援：此值可以是個數值或百分比。
(3).最小信心水準：此值為一個百分比的機率值。

3.4 比對攻擊行為模組

本實驗以模擬的方式來進行偵測，方式為建置一個測試資料庫來記錄 shell codes 裡所呼叫的 system call 來代替實際執行的 shell codes。先搜尋攻擊探測碼當作測試的待比對資料，並將攻擊探測碼內的 system call 呼叫記錄與規則庫比對，步驟依序如下：
(1).將欲偵測的攻擊程式反組譯。
(2).截取 shell codes 內的 system call：將 shell code 之 ID 與其所呼叫的 system call 內容寫入至一個待測試的資料庫。
(3).與規則庫比對：比對的演算法流程如圖 4 所示。

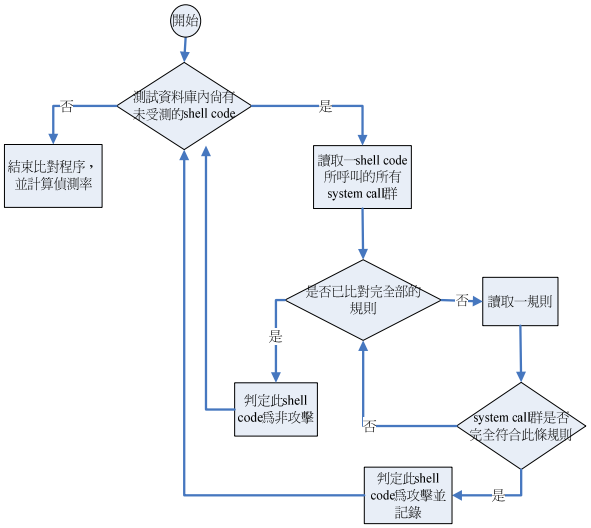

圖 4 比對攻擊的演算法流程

3.5 比對正常行為模組

在將規則庫比對完攻擊行為計算出偵測率後，也必須將規則庫與正常行為比對，以觀察此規則庫的誤判率。為了摸擬此步驟的實驗，可針對正常系統上運行的 system call 著手。在 Linux 系統上，有個方便的指令可協助完成此動作，此指令為”strace”。”strace”可以追蹤系統上所有 system call 的呼叫記錄，在下達此指令後，就能夠記錄某行程所呼叫的所有 system call，並將此追蹤的結果記錄存成一個檔案以方便之後程式演算法的執行。整個流程如圖 5 所示。

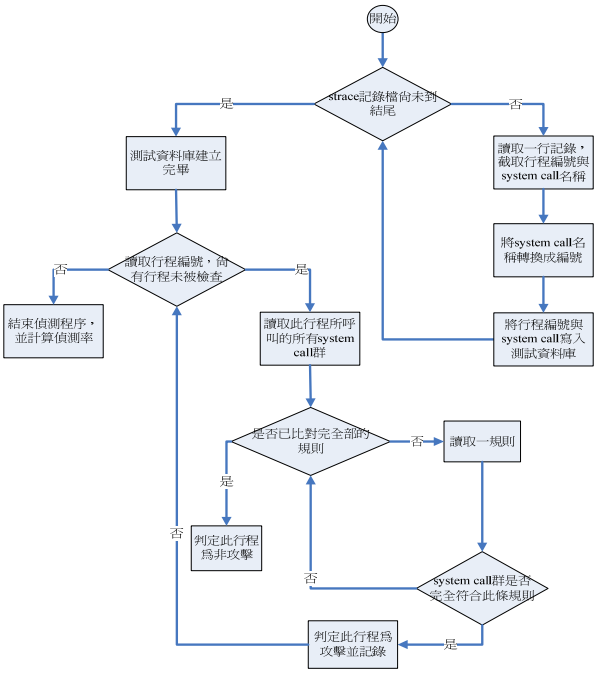

圖 5 誤判率的計算流程

4. 實驗結果與評估

在偵測攻擊實驗方面，建立資料庫時，本研究共搜集並截取 40 支緩衝區溢位的 shell codes 來當作建立規則庫的訓練資料，這 40 支 shell codes 內共包含了 210 條 system calls。為求後續規則的精簡與比對的方便，在此將最大集合物件項目數設為 3，執行關聯法則演算法後，再篩選適合的規則。為減少規則的數量與增加比對的效率及準確性，在此只挑選支援數在 2 以上的集合來當作規則篩選的對象，再依較恰當的條件來做初步的規則篩選，篩選的條件如下：

(1).不選只有一條 system call 的集合。
(2).分別列出集合內，每條 system call 相對於其它 system call 的相關關聯規則參數，。
(3).挑選出所有規則其重要性皆大於 0 的集合。
(4).選出所有規則其信心水準皆大於或等於 0.5 的集合。
(5).以此集合為最終比對的規則單位。

初步挑選規則完畢後，即可開始進行比對攻擊行為的實驗。在此總共搜集了 84 支的 shell codes 測試資料，這些 84 支 shell codes 共包含了 476 條 system calls，接著進行偵測攻擊流程。為了得出最有效與精簡的規則，在此分別採用重要性值為 1 以上、重要性值為 0.3 以上，及重要性值為 0 以上的規則進行實驗，偵測結果如表 6、7、8 所示。

表 6 重要性值為 1 以上的規則偵測結果列表

Shell code 數目	84
發現攻擊數	20
偵測率	23.80%

表 7 重要性值為 0.3 以上的規則偵測結果列表

Shell code 數目	84
發現攻擊數	62
偵測率	73.81%

表 8 重要性值為 0 以上的規則偵測結果列表

Shell code 數目	84
發現攻擊數	62
偵測率	73.81%

由表 6、7、8 可發現，採用重要性在 1 以上的規則其偵測率不甚理想，而採用重要性在 0.3 以上的規則與重要性 0 以上的結果相同，因此發現重要性在 0.3 以上的規則已經達到最佳的實驗結果，以其當作偵測攻擊的基礎。

在比對正常行為方面，本實驗的對象為一台 2.6.17-gentoo-r4 版本的 Linux 系統，此系統上架有重要並常見的服務如網頁服務 www、郵件服務 smtp、檔案傳輸服務 ftp 等等，下達”strace”指令分別對上述服務的行程進行追蹤，持續 7 天的追蹤後停止。再將此三種服務行程分別產生的記錄檔進行偵測比對，實驗結果如表 9 所示：

表 9 初步比對正常行為的結果列表

服務的行程類型	www	smtp	ftp	總合
行程數目	529	386	337	1252
發現攻擊的數目	529	383	169	1081
誤判率(%)	100	99.22	50.14	86.34

由表 9 可發現，規則的誤判率非常高，非正常現象，因此必須篩選掉不合適、無效的規則以降低誤判率，觀察所比對的規則中，發現規則中「open」、「close」、「write」此三條所組成的規則為造成誤判率過高的原因，雖然這些規則都有較高的信心水準與重要性，但並不一定適合用來偵測攻擊，將這些規則刪除後再進行實驗，得出新的結果如表 10 所示：

表 10 最終比對正常行為的結果列表

服務的行程類別	www	smtp	ftp	總合
行程數目	529	386	337	1252
發現攻擊的數目	0	0	0	0
誤判率(%)	0	0	0	0

由表 10 可知總誤判率已降至 0%，應屬相當理想且合理的誤判率，同時這些規則再用來進行比對攻擊的實驗，實驗結果如同表 7 所示，故這些規則已符合本實驗的期待。

5. 結論

本研究的貢獻在於應用關聯法則於網路安全相關研究，以系統呼叫的 system call 為基礎，搜集並分析攻擊探測程式呼叫 system call 的模式。其以關聯法則演算法建立可區分正常行為與攻擊行為的規則，偵測緩衝區溢位攻擊，並根據系統呼叫(system call)的情形判斷是否有無攻擊行為發生，進而保護系統在尚未完全被攻陷時，能及早發現潛在的攻擊，並利於進行相關的防範措施。若發現攻擊行為產生時，能夠及時中斷相關遭受攻擊的行程或連線，期望能在軟體程式的更新尚未發佈前或尚未更新軟體程式前，降低緩衝區溢位攻擊的嚴重性。

在未來發展方面，有幾個面向可當作未來改善方向：

(1).本研究所進行的比對僅為非線上的模擬方式，未來可將系統發展成即時的入侵偵測系統以求有更快的反應時間。

(2).未來可實驗更多其他屬於系統的正常行為以求更精確的結果。

(3).未來可以建立系統正常行為的規則來當作偵測攻擊的模式。

誌謝

資通安全人才培育計畫-人才培育
NSC 96-2219-E-006-009.

參考文獻

[1] 尹相志。SQL Server 2005 資料採礦聖經。學貫。2005。ISBN：9867198395。

[2] A. Baratloo, T. Tsai, and N. Singh. “Libsafe: Protecting Critical Elements of Stacks”, Technical report, Avaya Labs, 1999.

[3] A. Pasupulati, J. Coit, K. Levitt. S. F. Wu. “Buttercup: On Network-based Detection of Polymorphic Buffer Overflow Vulnerabilities”, In IEEE/IFIP Network Operation and Management Symposium, May 2004.

[4] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang. “StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks”, Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, January 26-29, 1998.

[5] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole. “Buffer Overflows: Attacks and Defenses for the vulnerability of the Decade”, DARPA Information Survivability Conference and Exposition 2000 Proceedings.

[6] D.Larochelle, D.Evans, “Statically detecting likely buffer overflow vulnerabilities”, Proceedings of the 10th USENIX Security Symposium. USENIX: Washington, DC, 2001; 177-189

[7] D. Wagner, J.S. Foster, E.A. Brewer, A. Aiken, “A first step towards automated detection of buffer overrun vulnerabilities”, Network and Distributed System Security Symposium, San Diego, CA, February 2000; 3-17

[8] R. Hastings, B. Joyce, “Purify: Fast detection of memory leaks and access errors”, In Proceedings of the Usenix Winter 1992 Technical Conference, pages 125 138, Berkeley, CA, USA, Jan. 1991. Usenix Association.

[9] R. Jones, P. Kelly, “Backwards-compatible bounds checking for arrays and pointers in C programs”, Proceedings of the Third International Workshop on Automatic Debugging, Sweden, May 1997. Linkoeping University Electronic Press,13-26.

[10] R. Agrawal, and R. Srikant, “Fast algorithms for mining association rules in large database”, Technical Report FJ9839, IBM Almaden Research Center, San Jose, CA, Jun. 1994.

[11] R. Agrawal, and R. Srikant. “Fast algorithms for mining association rules”, In Proc. 1994 Int. Conf. Very Large Databases(VLDB’94), Sep. 1994

[12] R. Agrawal, T. Imielinski, and A. Swami., “Mining association rules between sets of items in large databases”, Proceedings of the ACM SIGMOD Conference on Management of data, p.p. 207-216, May 1993.

[13] StackShield, http://www.angelfire.com/sk/stackshield/

[14] S. Andersson, A. Clark, G. Mohay, “Network-Based Buffer Overflow Detection by Exploit Code Analysis”, Proceedings of AusCERT Asia Pacific Information Technology, 2004.