可動態洞悉、管理內部網路安全之研究與實作

楊有信 王俊鑫

中華大學資訊工程學系

prophet@chu.edu.tw ; chwang@chu.edu.tw

摘要

本篇論文主要研究如何動態監控內部網路(intranet)的資料傳輸，我們實作一個有別於傳統的 layer 2 switch 上的 mirror port 的監控埠，我們稱之為 M-mirror port。M-mirror port 可動態調整對一張到多張網路卡同時進行 mirror。藉由此技術，網管人員可以不再受限於傳統的交換器 mirror port，只能監看單一的 port 的資料來源，可動態同時的監看內部網路的資料，而為了有效的及時抑止異常的資料流量，我們亦在系統中實作二到七層的防火牆機制，除了傳統針對 MAC、IP 位址及 Port Number 之外，亦可對第七層的通訊協定做控管。為方便網路管理者使用，我們實作網頁式(Web)的管理介面，讓網路管理者可藉由瀏覽器，動態的調整欲觀察的任一 port 或數個 ports，及藉由防火牆的工具，及時的有效控管異常的網路傳輸。

關鍵詞： 內部網路, mirror port

1. 前言

隨著網際網路的普及與便利，全球上網人數不斷攀高，平均使用網路的時間也不斷上升，但伴隨而來的資訊保密與網路安全的問題卻層出不窮。當資料因網路傳輸被不當竊取時，資料加密技術的應用，可以讓資料多一層防護，但仍有被解密的疑慮。最好的資料防護，應是防止資料被不當的竊取，因此防火牆的軟硬體工具，成為企業網路不可或缺的第一道防線，但網路的攻擊的形式日新月異，許多的攻擊封包，對防火牆視若無睹，如阻斷服務式的攻擊(DoS, DDoS[1])、蠕蟲式的病毒，既利用大量的網路服務要求來阻斷伺服器的原有的服務。也因此，入侵偵測與入侵防禦偵測系統(IDS/IDP)運應而生，可以有效的阻擋大部份的惡意封包，但仍有漏網之魚，這也造成內部網路安全的隱憂。事實上，容易被忽視的內部網路安全，更是值得我們關注。

因此，在本篇論文，我們將實作一個可動態洞悉、管理內部網路安全之系統。第一章節主要是說明內部網路安全的疑慮及我們所提出的系統架構。第二章節部分主要探討如何實作出整個系統。第三章節則是一些實作方面的圖形界面，展示系統的運作狀態。最後章節則是對整篇論文做一個結論以及未來邁進的目標。

1.1 內部網路安全的疑慮

大部分的網路架構都習慣將防火牆擺在進入網路的開始點，與閘道器整合在一起，管理進出網路的封包，如下圖 1。

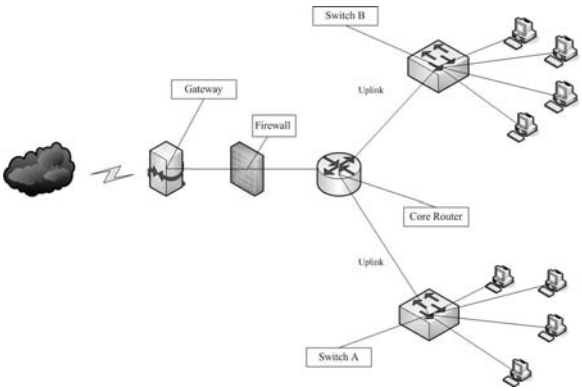

圖 1. 一般網路架構

---

*本論文研究為國科會計畫之研究成果, 計畫編號 NSC 94-2213-E-216-017-

在圖 1 中，為一個較常見到的網路架構的例子。圖中最左方表示網際網路，接著閘道器、防火牆、核心的路由器，再來是分開的子網路等等。這是一個再普通不過的網路架構，我們將藉這個例子來說明傳統的網路安全有哪些盲點與觀察網路資料傳輸的難易度。

當網路病毒從內部網路竄起，攻擊相同的子網路(Subnet)或是跨子網路的電腦或是網路設備，導致伺服器無法正常的提供服務或網路頻寬受到嚴重影響，我們如何去發覺異常封包的來源並提出有效的解決方案？雖然在網路的出入端口設有防火牆，但卻無法發揮作用。原因在於防火牆只會對出入網路端口的封包做掃瞄，然後加以制止。但在相同的子網內，封包的傳遞不會經過閘道器的，當然也不會經過防火牆，自然也無法阻止內部電腦癱瘓網路的惡意行為。

要如何有效率的找到正在癱瘓網路的網路節點，實數不易。傳統的方法可以藉助 switch 上 mirror port 的功能，將欲觀察的網路節點的交通流量導到另一個 port 來觀看，再藉由後端的分析工具來檢視收集到的封包內容是否為惡意的攻擊封包，若為異常的封包，只能對該 port 進行封鎖的動作，這也迫使正常的封包無法傳輸，如此使用相同的方法繼續藉由 mirror 檢察每一個連結在 switch 上的網路節點。如果 switch 上有 N 台網路節點，那麼尋找的時間就是，是相當費時費力的工作。

再者，如果網管人員想要觀察同一網段內（沒有跨路由器）網路使用者彼此的網路行為，又該如何達到？傳統交換器(Switch)的 mirror port 只能 mirror 一個 port，若 mirror port 的對象為 switch 上對外的傳輸埠(既 uplink port)，僅能觀察網路使用者對外的網路封包的進出，但當網路使用者的傳輸對像連結在同一個 switch 時，傳輸的封包並不會經過 uplink port，因此傳統的 mirror port 無法達到觀察同一網段內網路使用者彼此的網路行為，雖然可以用 HUB 連結同一網段的網路節點來達到這樣的需求，但卻大幅降低網路效能。

1.2 系統架構

為了滿足前一小節上述的情形，將對傳統 mirror port 做加強的機制—允許同時 mirror 一張到多張網路卡上的封包，我們稱為 M-mirror port 並實作在 pc-based 的 switch 上。

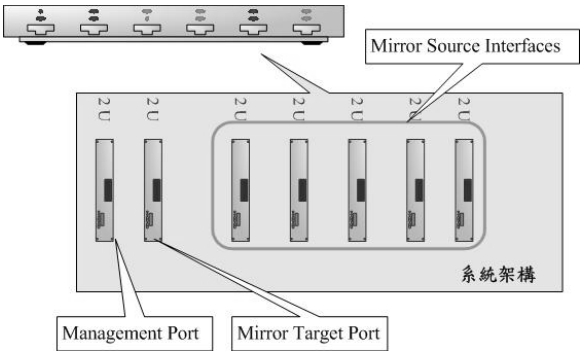

圖 2　系統架構

圖 2 為系統的一個概觀架構。在 PC 上裝上許多網路卡來模擬 Switch 的功能，主要把網路卡拆成三個群組，分別為 Mirror Source Interfaces, Mirror Port Interface, Management Port interface。

Mirror source interfaces 模擬 switch 上一般的 port(s)，做為封包傳遞交換之用。Mirror target interface 則是接收 mirror 來的封包。Management Port 則是用來管理整個系統的管理介面，所有管理封包都得經過這張介面卡進入到系統。為了滿足頻寬的要求，Mirror target interface 的頻寬需要大於 Mirror source interfaces 加起來的頻寬，這是為了確保 mirror 來的封包不被 mirror target interface 丟棄。在實作上，我們以一張 Giga bits 的網路卡來實作 mirror target interface，而其它的 interfaces 為 100M bps.

整個系統是在 RedHat Linux 2.4 上開發，利用 Linux 強而有力的網路系統，開發出新的 kernel module，來達到系統的要求。而實驗平台則是一台 PC，上面有六張網路卡來模擬 Switch 平台。

2. 系統實作

在本章節主要探討如何實作新一代 Mirror 技術，二至七層防火牆，動態建構 VLAN 等。同時也將介紹 Linux 網路底層技術 — Netfilter[2]。利用 Netfilter 的 API 來達到封包的控制，如封包通過或是攔截等動作。藉由模仿 Linux 核心的實作方法，我們亦把這些防火牆功能及 Mirror 功能寫成模組，可以自由的掛載及卸除。

以下的子章節將分別介紹 Netfilter、Socket buffer[3]、動態 VLAN、防火牆以及 Mirror 等實作方式。

2.1 Netfilter 簡介

在 Linux kernel 2.4 之後，netfilter 正式加入 linux 核心底層的工作。成功的取代 ipnatctl 成為 linux 核心的新三代 firewall module。在這樣的架構之下，也產生了上層的防火牆機制 iptables。完成 Linux kernel 2.4 之後的防火牆雛形。雖然 Netfilter 有眾多好處，但我們不在此做討論。只看它的運作架構及封包經過的行為控制。

Netfilter 把封包經過網路卡到核心分成五的狀態(hook point)，如下:

- NF_IP_PRE_ROUTING(#1)
- NF_IP_LOCAL_IN(#2)
- NF_IP_FORWARD(#3)
- NF_IP_LOCAL_OUT(#4)
- NF_IP_POST_ROUTING(#5)

PRE/POST ROUTING 主要定義封包到達網路卡時及丟出網路卡前的狀態；LOCAL IN/OUT 則是經過進入核心處理或是核心處理完的狀態；FORWARD 則是封包在網路卡間做 forwarding 的狀態。而整體的狀態圖如圖 3.所示。

決定了封包經過的狀態，當然也要對其狀態點做控制。在此，Netfilter 也提供了每一個狀態點五種決定封包運命的機制。分別是 NF_DROP、NF_ACCEPT、NF_STOLEN、NF_QUEUE、NF_REPEAT。藉由這些機制就可以達到封包在系統上的控制，如通過、丟棄等。而在此只用了 NF_DROP 跟 NF_ACCEPT 來建構整個系統需求。

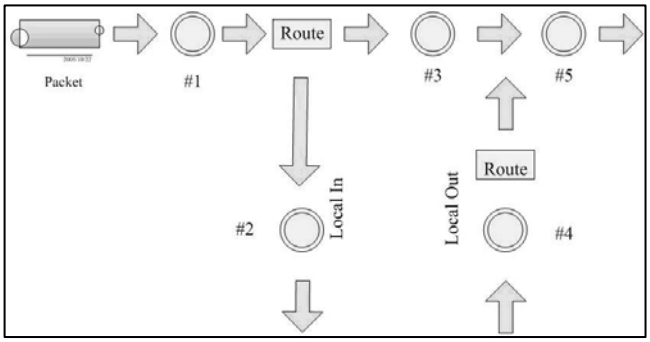

圖 3、 Netfilter 狀態圖

2.2 Socket Buffer 簡介

在 Linux 系統裡，每當有封包經過，核心就會去產生一個資料結構去儲存這個封包所帶的資訊，而這樣的資料結構稱為 socket buffer。

Socket buffer 儲存的資訊從 Layer 2 到 Layer 7 都有，這其中當然有我們最關心的 IP 位址及通訊協定等。每個 socket buffer 都會用 doubly linked-list 串起來形成一個 socket buffer list，如圖 4.所示。

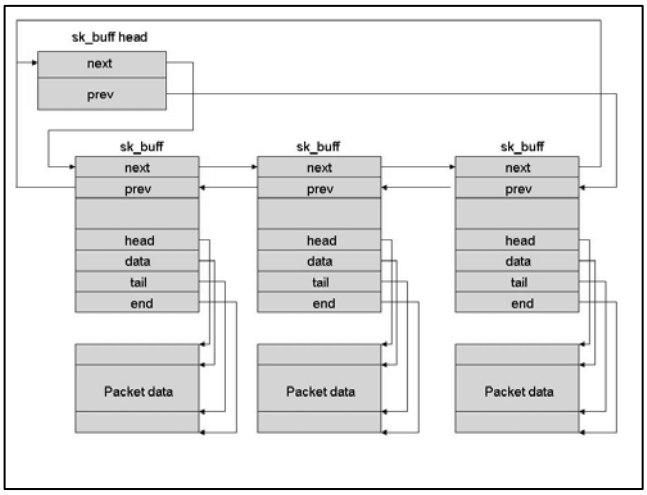

圖 4、 socket buffer list

每個 socket buffer 也包含著一些 Linux 的網路資料結構，如 IP header、tcp header、udp heaer 等，所以瞭解 socket buffer 也就可以瞭解 Linux 網路資料結構及它的實作方式，是一個不錯的學習 Linux 網路架構的切入點。

2.3 動態 VLAN 實作

為了可以動態調整 LANs 的大小，實作了可以動態建立 LAN 跟指定哪張網路卡在那個 LAN 底下。只要利用 Open Source 的 bridge-control[3] 來實作這整個功能。在系統上安裝 bridge-control，就可以在 run time 去調整新增/移除一個 broadcast domain 用來模擬 switch 上的 VLAN。當然 switch 上的 VLAN 可以設定 VLAN tag 或是 PVID，在此系統上並沒有提供這樣的實作，只是單純的做 broadcast domain 的切割。

2.4 防火牆實作

為了控管系統上的封包行為，必須要實作一個防火牆的機制。這裡沒有採用 Linux 內建的防火牆-iptables[4]主要是因為想要試著去實作一個可以控管的防火牆，所以藉由 Netfilter 提供的 API 實作一個防火牆的模組來達到監控封包行為。
我們在封包進入及出去網路卡的時候做阻攔，也就是對應到 Netfilter 的 NF_IP_PRE_ROUTING 跟 NF_IP_POST_ROUTING 的狀態點。因為這個系統是模擬 switch 的角色，只關心封包進入及出去的行為，所以在這兩個狀態作封包的控管。

在 socket buffer 介紹裡有提過，封包會對應到一個資料結構叫做 socket buffer，Linux 核心在處裡的時候，會把正在處裡的 socket buffer 的指標傳出來，藉由得到這個指標就可以去找出這個封包的內容，然後去判斷這個封包是合法的或是不合法的，再去決定它的命運。這就是整個防火牆模組的大致流程。

就功能面來看，除了一般的 MAC、IP address、port 及 protocol 之外，也增加了最近防火牆不可或缺的 content filtering，也就是第七層資料的辨認。第七層資料辨認有它一定的難度，主要是因為要決定每一個通訊協定的特徵值然後加以判斷，而每一個通訊協定又不相同，行為模式也不同，所以要怎麼去決定一個通訊協定的特徵值需要有大量的資料統計才能算是一個完整的特徵值。此系統的特徵值是藉由 sniffer 工作 mirror 來的封包做統計而來，藉由特徵值就可以跟第七層的資料做比對，來決定這個封包是屬於那個通訊協定。

以下是一個簡單的 Netfiler 範例程式，主要是丟棄網路卡上收到的所有封包。

```
/* This is the hook function itself */
unsigned int hook_func(
unsigned int hooknum, struct sk_buff **skb,
const struct net_device *in, const struct net_device
*out,
int (*okfn)(struct sk_buff *)) {
        return NF_DROP; /* Drop ALL packets */
}
/* Initialisation routine */
int init_module(){
/* Fill in our hook structure */
nfho.hook = hook_func; /* Handler function */
nfho.hooknum = NF_IP_PRE_ROUTING;
/* First hook for IPv4 */
nfho.pf = PF_INET;
nfho.priority = NF_IP_PRI_FIRST;
/* Make our function first */
nf_register_hook(&nfho);
return 0;
}
/* Cleanup routine */
void cleanup_module(){
nf_unregister_hook(&nfho);
}
```

2.5 M-mirror port 實作

要達到 mirror 多張網路卡的功能並不簡單，實際上也有許多限制。頻寬就是一大致命傷。所以在此定義 mirror source interface 為 100Mbps，mirror target interface 為 1Gbps。

跟防火牆相同，還是藉助了 Netfilter 提供的 API 來實作整個模組。狀態點也是選擇 NF_IP_

PRE_ROUTING 跟 NF_IP_POST_ROUTING。不同的是對 socket buffer 關心的地方不一樣。

Socket buffer 除了帶整個封包的資訊，同時也會紀錄是由那個網路卡收進來的或是要從哪張網路卡出去。藉由網路卡代號的比對，就可以知道這個封包要不要進行 mirror 的動作。一旦比對是符合的，就在核心產生一個新的 socket buffer 並把 socket buffer 的內容複製到新的 socket buffer，然後改變新的 socket buffer 出去的網路卡代號，再丟到 mirror target interface 的 queue 裡，交由 mirror target interface 的 device driver 去丟出，就完成了整個 mirror 的動作。

經過上述的動作，就可以在事先定義好的 mirror target interface 聽到 mirror source interfaces 的封包，藉由這些封包就可以讓 MIS 或是系統管理員來瞭解內部的哪些電腦正在做什麼樣的溝通行為。

2.6 封包處理過程

完成 firewall 模組跟 mirror 模組之後，封包在系統的運作流程的順序就十分的重要。因為流程錯誤的話，會影響到 mirror 模組的正常運作，導致有些封包無法被 mirror 到而產生非預期的結果。因此在系統上的模組順序可以藉由一個流程圖來表示，如下圖 5.。

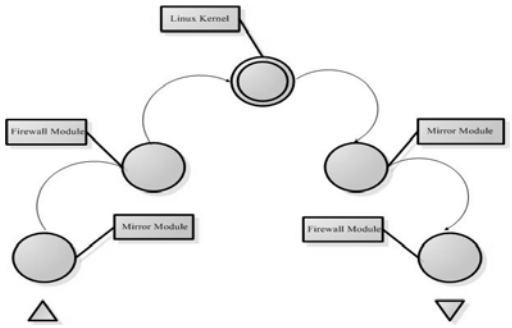

圖 5、系統封包流程

如圖五所示，左邊兩個圓形圖代表在 NF_IP_PREROUTING 狀態點裡兩個系統模組；右邊兩個圓形圖代表在 NF_IP_POSTROUTING 狀態點有兩個系統模組。而中間的圓形圖代表 Linux Kernel。左下角的三角形代表封包進入系統；右下角三角形代表封包離開系統。

在封包進入或出去的時候，必須經過 firewall 模組跟 mirror 模組，但是經過的順序卻一定是 mirror 模組再來才是 firewall 模組。原因是封包如果先經過 firewall 模組，可能會因為系統上的 ACL(Access Control List)的設定，而把封包給丟棄掉，這樣一來就無法 mirror 到被丟棄的封包。所以一定要先經過 mirror 模組在經過 firewall 模組才能保持系統的正常運作。

3. 系統成果展示

在介紹完系統的實作之後，我們亦實作 Web-based 的管理界面，在此章節我們提供一些使用者的圖形界面 UI 來展示實作系統跑得實際情況。從下面的圖形，可以清楚得知系統上封包經過、丟棄以及 mirror 的情形。

System Parameter	Parameter Content
System Name	Project Firewall Server
Management Port IP	140.126.130.53
Management Gateway	140.126.130.254
Firewall Status	Active
Mirror Status	Active
Software Version	Intranet Security v5
Software Designer	You-Sin Yang R.O.C

Load Default (wait 2 minutes)

Please set your browser size to 1024*768

圖 6、系統狀態

Add Bridge Delete Bridge

Bridge Name	Bridge ID	STP Activity	Interfaces	Bridge Settings	STP Status
DefaultBR	8000.0050ba04dd6b	yes	eth0 eth1 eth2	Settings	STP Status

圖 7、Bridge 狀態

Firewall Logs: Mirror Logs List Logs Clear Logs

```
Jan 14 22:33:37 Mirror packet send from 140.126.130.53 to 59.115.228.135 interface=eth4 direction=OUT
Jan 14 22:33:37 Mirror packet send from 140.126.130.53 to 59.115.228.135 interface=eth4 direction=OUT
Jan 14 22:33:37 Mirror packet recv from 59.115.228.135 to 140.126.130.53 interface=eth4 direction=IN
Jan 14 22:33:37 Mirror packet recv from 59.115.228.135 to 140.126.130.53 interface=eth4 direction=IN
Jan 14 22:33:37 Mirror packet send from 140.126.130.53 to 59.115.228.135 interface=eth4 direction=OUT
Jan 14 22:33:38 Mirror packet recv from 140.126.130.46 to 140.126.255.255 interface=eth4 direction=IN
Jan 14 22:33:38 Mirror packet recv from 140.126.130.46 to 140.126.255.255 interface=eth4 direction=IN
Jan 14 22:33:38 Mirror packet recv from 140.126.130.46 to 140.126.255.255 interface=eth4 direction=IN
Jan 14 22:33:38 Mirror packet recv from 59.115.228.135 to 140.126.130.53 interface=eth4 direction=IN
```

圖 8、Mirror 封包的 Log

Firewall Logs: All Packets List Logs Clear Logs

```
Jan 14 22:33:37 accept packet recv from 59.115.228.135 sport 1108 to 140.126.130.53 dport 80
Jan 14 22:33:37 accept packet recv from 59.115.228.135 sport 1108 to 140.126.130.53 dport 80
Jan 14 22:33:37 accept packet recv from 140.126.130.46 sport 137 to 140.126.255.255 dport 137
Jan 14 22:33:37 accept packet send from 140.126.130.53 sport 80 to 59.115.228.135 dport 1108
Jan 14 22:33:37 accept packet send from 140.126.130.53 sport 80 to 59.115.228.135 dport 1108
Jan 14 22:33:37 accept packet send from 140.126.130.53 sport 80 to 59.115.228.135 dport 1108
```

圖 9、Firewall 模組通過的封包 Log

Firewall Logs: Dropped Packets List Logs Clear Logs

```
Jan 14 22:34:52 drop packet send from 192.168.0.254 sport 2048 to 192.168.0.254 dport 63217
Jan 14 22:34:53 drop packet send from 192.168.0.254 sport 2048 to 192.168.0.254 dport 51122
Jan 14 22:34:54 drop packet send from 192.168.0.254 sport 2048 to 192.168.0.254 dport 48049
Jan 14 22:35:56 drop packet recv from 140.126.21.8 sport 2048 to 140.126.130.53 dport 50365
Jan 14 22:35:57 drop packet recv from 140.126.21.8 sport 2048 to 140.126.130.53 dport 59835
Jan 14 22:35:58 drop packet recv from 140.126.21.8 sport 2048 to 140.126.130.53 dport 21179
Jan 14 22:35:59 drop packet recv from 140.126.21.8 sport 2048 to 140.126.130.53 dport 41659
```

圖 10、Firewall 模組丟棄封包 Log

4. 結論

資訊越發達，人的行為就越難預測。或許就坐在對面的人，也不知道他在利用電腦做些什麼事。而這整個系統就是希望能提供及時完整的資訊去瞭解使用者之間的網路行為，而又不會讓使用者察覺到。當然，這牽扯到許多法律、道德相關的問題，在此並不討論這些問題。純粹提供一個管理者簡單又方便的管理方式去管理它的網路。站在管理者的角度，他可能不知道公司的資料已經被洩漏出去了，他可能不知道中毒的是哪一台電腦而煩惱怎麼去解決，所以這類的網路監控系統就會越來越有它的價值，幫助管理著快速掌握問題發生的原因。

然而這個系統當然也不是百分之百的好，也需要改進的空間。當初的測試平台是在 Linux 2.4 的 kernel 上，由於已經過舊所以需要把整個系統搬移到 2.6 的 kernel。也希望把整個 mirror 的功能寫成 iptables 的 target 模組，原因是因為 iptables 跟 Netfilter 是一脈相傳的，寫在 iptables 上效率會更好，也更模組化，也更符合 Linux 當初的設計精神。而在第七層資料的比對上，應該要改成 regular expression 的比對，這樣才能更精準的去比對封包。

參考文獻

[1] Rocky K. C. Chang. “Defending against flooding-based distributed denial-of-service attacks,” *IEEE Communication Magazine*, October 2002, pp. 42-51.

[2] Harald Welte, Jozsef Kadlecsik, Martin Josefsson, Patrick McHardy, and Yasuyuki Kozakai, The software of the packet filtering framework inside the Linux 2.4.x and 2.6.x kernel series. http://www.iptables.org/index.html

[3] http://linux-net.osdl.org/index.php/Bridge.

[4]http://www.iptables.org/projects/iptables/index.html