NetBSD 核心網路安全模組：IPFilter 及 IPsec

蔡孟甫 曹世強 林盈達

國立交通大學資訊科學系

新竹市大學路 1001 號

TEL：（03）5712121 EXT. 56667

E-MAIL：{mftsai, weafon, ydlin}@cis.nctu.edu.tw

摘要

隨著網路的普及及使用者的增加，網路安全的議題便越顯得重要。要解決安全問題，首先想到就是防火牆。在眾多防火牆軟體套件中，根據我們的深入觀察, IPFilter 內建 NAT，並且使用快取與雜湊技術來處理許多封包分類等工作，有效提升處理效率. 因此獲選為 NetBSD 內建防火牆套件。然而有了防火牆還不夠，因為封包一出了防火牆還是不安全的，還需要一個替封包加密的套件。KAME 的 IPsec 便是被 NetBSD 選來負責這項工作的內建核心模組。他實做了複雜 IPsec 相關協定架構，包含 AH、ESP、與 IKE，IPcomp 等，支援非常完整的 IPsec 協定。本篇將透過分析程式碼的方式描述這兩套模組的重要資料結構和資料處理流程。

關鍵字：IPFilter，IPsec，Security Policy，Security Association

1. 簡介

隨著網際網路的普及，網路安全的重要性也越顯重要，而架設防火牆便是最基本的防範措施。防火牆所提供的封包過濾功能，形成一道與網際網路之間的安全守護者，避免我們的電腦受到不必要的攻擊。然而有了防火牆還是不夠的，由於 IP 本身並不是安全的，因此封包出了防火牆後還是容易遭到竊聽與修改，有鑑於此，IPsec 定義了如何對封包做加密與認證的動作，保證封包在傳輸期間不會受到修改與竊聽內容。在本文中，我們挑選 IP Filter, KAME 之 IPsec 這兩套 NetBSD[1]內建分別作為防火牆，資料加密、認證的開放程式碼套件來追蹤。首先，我們找出這兩組套件在核心中所在的目錄。IPFilter 的程式碼及核心 TCP/IP

堆疊放在 netinet 目錄下。而由 KAME 計畫所提供的 IPsec 程式碼則放在 netinet6 中。至於 IPsec 所需要的金鑰管理與各種安全政策資料庫管理，則放在 netkey 目錄裡。此外，為能得心應手的追蹤核心模組，我們使用下列三個工具：

GLOBAL：GNU 計畫的一部分，將難讀的程式碼整理成 HTML 網頁，並替相關的部分建立超連結，例如當你看到程式碼中呼叫某個函式的時候，你可以直接點選這個函式，便能超連結到這個函式所在的位置以方便閱讀及追蹤。非常好用，相信你用過一次就會愛上他。

Kernel Profiling：NetBSD 核心本身提供的功能，讓我們了解核心程式之間的呼叫流程(Call Graph)。要使用此功能前，需在核心把它打開。NetBSD 的網頁[3]上有教使用方法。[2]有介紹如何看 Kernel Profiling 產生出來的訊息。

KGDB：有了看 Code 和了解 Call Graph 的工具後，我們需要 Debug 工具。要 debug 一般想到 gdb，可是它無法追蹤核心部份。因此我們使用 KGDB[4]，從遠端除錯核心程式。只是 KGDB 設定很麻煩，且需要兩台電腦，因此我們建議使用 VMWare 來模擬一個被追蹤的核心系統，這樣會比較方便。

在下一節，我們將簡單敘述封包在核心的處理流程，及點出兩套軟體在流程中的運作位置。而 3,4 節，則分別深入探討這兩個軟體之功能，並從程式碼介紹其內部運作的流程及重要的資料結構。最後是我們的結論。

2.封包處理流程

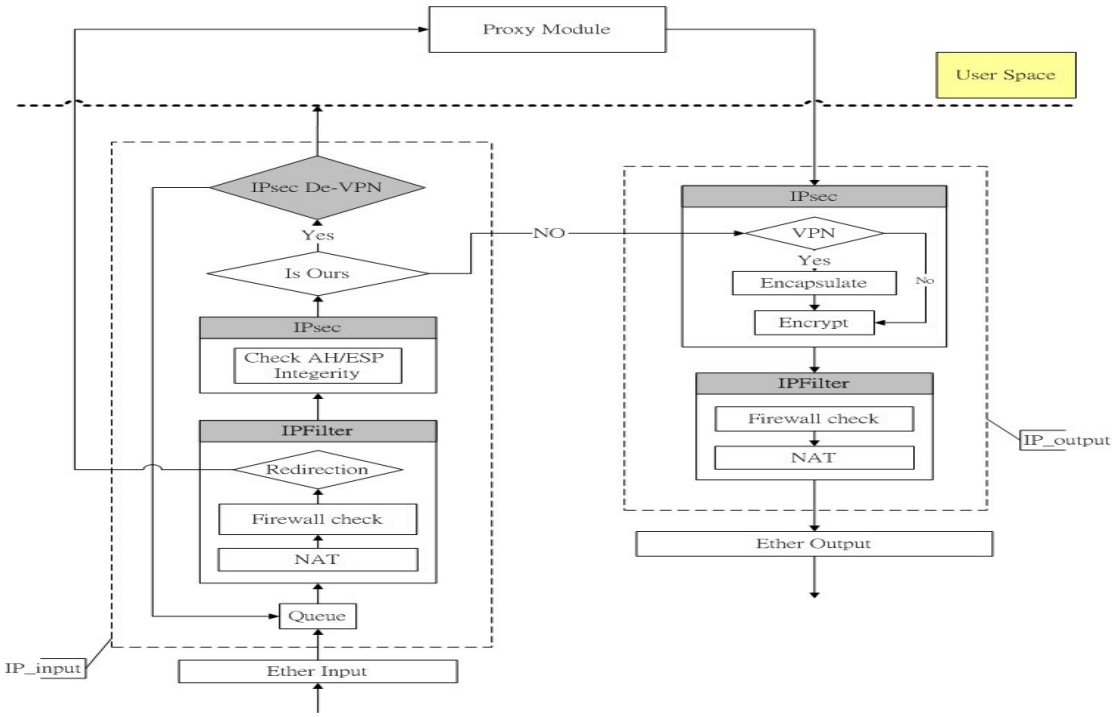

圖 1：封包處理流程

如圖 1 所示，這是封包流過核心的處理程序。圖左半部是流入的處理程序，右半部則是流出的程序。圖中灰色的部分標示出 IP Filter 及 IPsec 這兩套核心模組所處的位置。對一個封包來說，在流入後，首先會被 IPFilter 做 NAT 轉換動作與防火牆規則比對。根據防火牆規則，可能也會被轉給所指定的代理伺服器。接著交給 IPsec 模組檢查封包內容是否被中途攔截修改過。並確定是要轉送的還是我們自己的封包,對於因 VPN 封裝起來的封包,在反封裝後,將會重放入 IP queue 中,再處理一次。至於要送出封包時，先看要不要為了提供 VPN，去作封裝及加密的動作。接著丟給 IPFilter 比對防火牆規則，然後看是不是要做 NAT 處理。最後丟進 Ether_ouput 的 queue 中，等待送出。

3. IPFilter

IPFilter 是套能提供 NAT 與防火牆功能的軟體，相較於其他防火牆，他內建 NAT 算是比較特別的。目前已內建於 NetBSD1.6.1 核心中。只需在/etc/rc.conf 中設定，並在/etc/ipf.conf 建立規則後，便可啟動。他主要能夠依據來源/目的地位址，網路介面，IP 通訊協訂，19 種 IP options 或 8 種 IP 安全等級的服務種類(Type of Service, TOS), 是否為分段封包等多個項目*來拒絕或允許封包的流入流出*。除此之外，IPFilter 還能：

- 為 TCP,UDP,ICMP 封包保留封包狀態資訊
- 為 IP 封包保存分段資訊，用同樣的規則套用到所有的分段封包上
- 提供重新導向(redirection)
- 提供封包表頭給使用者指定的程式做認證用

由於功能眾多，在本次的追蹤中，我們僅將焦點放在與網路安全有關的防火牆功能上，其他相關部分請參考 IPFilter 網頁[5].

3.1 規則與資料結構

```
block in quick on xl0 all head 1
//規則 1　群組 1 的頭，規則是擋掉所有從 x10 流進的封包
block in quick on xl0 from 192.168.0.0/16 to any group 1
//規則 2　屬於群組 1，檔掉所有從 x10 流入，來源是 192.168.0.0 的封包
pass in on xl0 all group 1
//規則 3　屬於群組 1，允許所有從 x10 流入的封包通過
block out quick on xl1 all head 2
//規則 4　群組 2 的頭，檔掉所有從 x11 流出的封包
```

```
pass out on xl1 proto tcp from any to 20.20.20.64 port = 80 keep state group 2
//規則 5　屬於群組 2，允許所有從 x11 流出，目的地是 20.20.20.64，port 爲 80 的 TCP 封包，並保存狀態
pass out quick on xl1 proto udp from any to any port = 53 keep state group 2
//規則 6 屬於群組 2，允許所有從 x11 流出，port 爲 53 的 udp 封包，並保存狀態
rdr xl1 0.0.0.0/0 port 21 -> 127.0.0.1 port 21 group2
//規則 7 屬於群組 2，將所有從 x11 流入的 FTP 封包重新導向給 FTP 代理伺服器
block out on xl2 all
//規則 8 檔掉所有從 x12 流出的封包
pass in quick on xl2 proto tcp/udp from 20.20.20.128/25 to any keep state
//規則 9 允許從 x12 流入，來源爲 20.20.20.128/25，協定爲 tcp/udp 的封包，並保存狀態
```

表 1:IPfilter 的規則範例

基本上，IP Filter 根據規則(rule)來描述允入或允出封包的條件。因此要使用 IPFilter，便要懂得它規則的寫法，表 1 列了一些規則及簡單解釋，更進一步的介紹，請直接參考 IPFilter HOWTO 網頁[6]，大概有 35 頁吧。表 1 中的規則會被建立成圖 2 的資料結構，在看規則解釋時請對照著看。IPFilter 在做規則比對的時候，是由最上面一條比對下來，但卻採用*最後一個比對成功的規則*。換句話說，他不會在第一個符合時，便結束比對工作。如果確切希望一比對成功便馬上採用的話，則必須在那條規則上加 quick 這個關鍵字，例如表 1 的規則 1。另外，關於規則 5 的 keep state 關鍵字，這是幹什麼的呢？由於 TCP 是連線導向的協定，所以通訊兩端會先建立連線才傳送資料。因此我們在規則上加上 keep state，告知 IPFilter 只需比對一開始建立連線的封包，判斷是否准予通過。如果成功建立連線後，則對於往後該連線的封包，便不再需要比對，而可以快速通過。規則也可以使用群組的概念，群組的功用就是用來加速判斷，例如你有兩張網路卡，你可以將網路卡 x10 的規則全部列為群組 1，網路卡 x11 的規則全部列於群組 2，當一個封包流入時發現不是屬於網路卡 A 所處理的，那所有網路卡 x10 的規則都不需要比對了，可以直接跳去群組 2，從處理網路卡 x11 的部分開始比對。規則 1.2.3、4.5.6.7 分別是兩個群組，我們等會會再詳細介紹。

了解規則後，我們來看 IP Filter 中三個重要資料結構：frentry、frgroup 與 fr_info，這些都定義在 netinet/ip_fil.h 裡面。其中 frentry 與 frgroup 是用來描述規則的，表 2 及 3 是這兩個結構的重要欄位。

欄名	欄位用途
fr_ref	用來紀錄所屬群組有多少條規則，只有當自己是群組的最前頭時

	這個欄位才會不是 0，也只有當此值為 0 時才能被刪除
fr_next	指到下一條規則
fr_ip	一個資料結構為 fr_ip 的欄位，用來紀錄 rule 主要的資訊，例如來源 IP，目的地 IP，TOS…等
fr_grp	如果自己是屬於某個群組的頭，這個會指到群組規則串列前端的那條規則

表 2： frentry 結構中的重要欄位

欄名	欄位用途
fg_num	紀錄這個群組的編號
fg_next	指到下一個群組
fg_head	指到這個群組的第一條規則

表 3: frgroup 結構中的重要欄位

如圖 2 所示，當我們在讀入表 1 的規則時，每條規則會被建成一個 frentry 的資料結構，規則之間是有階層關係的，所有有關鍵字 head 的代表是一個群組的頭，然後所有的規則會依序被串成一個單向鍊結串鍊，如果某些規則屬於同一個群組，規則中需要指定關鍵字 group，這些規則會以類似下圖中規則 1,2,3 方式與群組 1 串接起來。以表 1 的例子看來，該設定檔有 9 條規則，分作 2 個群組，1.2.3 條規則屬於第一個群組，4.5.6.7 條規則屬於第二個群組，其中 1,4 條規則分別是這兩個規則的頭。8.9 兩條規則則不屬於任何群組。

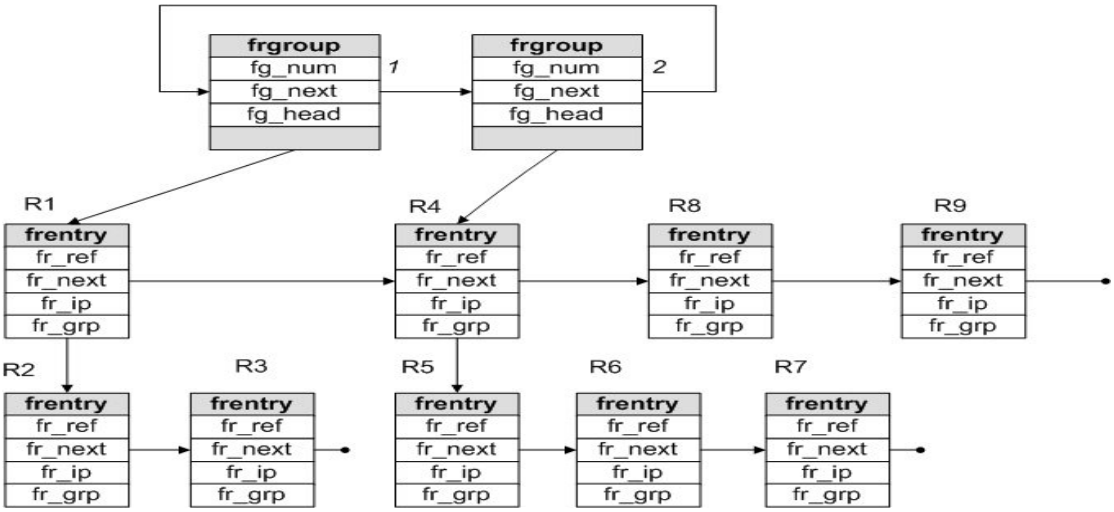

圖 2:frentry 與 frgroup 串接示意圖

接著我們看 fr_info。fr_info 是用在處理封包時，將封包中有用的資訊存在這個資料結構中，然後拿這個資料結構來比對。所有 IPFilter 中的處理，都是以這結構來運作。表 4 是列出這個結構的重要欄位。

欄名	欄位用途
fin_fi	為一個資料結構為 fr_ip 的欄位，存放要處理的封包的一些資訊，包括來源/目的位址..等
fin_out	紀錄這個封包是 IN 還是 Out
fin_tcpf	TCP 表頭旗標
fin_icode	ICMP 錯誤傳回
fin_group	所屬群組編號，-1 代表沒有屬於某個群組
fin_fr	指向這個封包最後比對成功的規則
fin_id	這個 IP 封包的 ID 欄位

表 4: fr_info 中的重要欄位

3.2 流程簡介：

圖 3 是整個 IPFilter 處理封包流入/流出的流程圖，左邊是封包流入處理圖，右邊是封包流出處理圖，不管流入還是流出的處理，IPFilter 都是呼叫 fr_check()這個函式來處理封包，由於流入處理比流出處理複雜，因此我們簡介流入的處理流程，流出的部分讀者可比照流入處理：

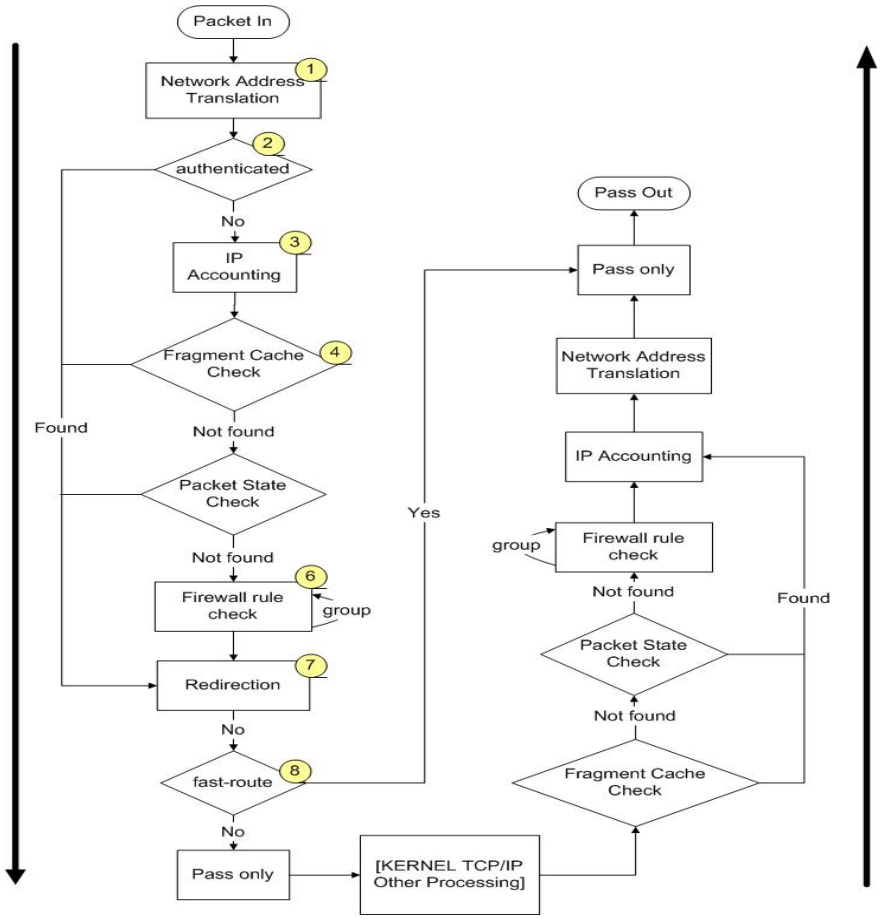

圖 3:IPFilter 的流程圖(fr_check)

IP Filter 流入的主要處理步驟為：

1. NAT 轉換：這邊會呼叫 ip_natin()，看是不是需要做 IP 位址轉換，如果需要就做 IP 位址轉換
2. 使用者認證(提供使用者用自己的程式判斷封包可否流過)
3. 統計封包資訊，例如流過多少位元組等
4. 分段檢查：如果是分段封包，檢查分段快取，看是否之前有處理過屬於同一分段的封包，有就直接採用快取中找到之處理規則。
5. 狀態檢查：(netinet/ip_state.c)

   狀態檢查就是之前介紹規則時所提到的保留狀態(Keep State)的動作。如果有在狀態快取中找到處理規則，就直接採用找到之規則。
6. 規則比對：(netinet/fil.c)

   當之前使用者認證、分段檢查與狀態檢查都失敗的時候，才會做規則比對的動作。這邊是一條條規則一路比對下去。
7. 重新導向：根據處理規則看是否需要將封包導給代理伺服器。

8. 快速路由：如果規則中有指定快速路由，我們不會將封包丟給核心的 IP 堆疊處理，而是直接查詢路由表，決定要從哪個介面送出，並不會做將封包的 TTL 值減少的動作。快速路由主要是希望在他人在做 traceroute 類的動作時，不會發現我們這台機器的存在。

我們詳細介紹其中三個比較有趣的步驟：

(i)分段檢查(netinet/ip_frag.c)：

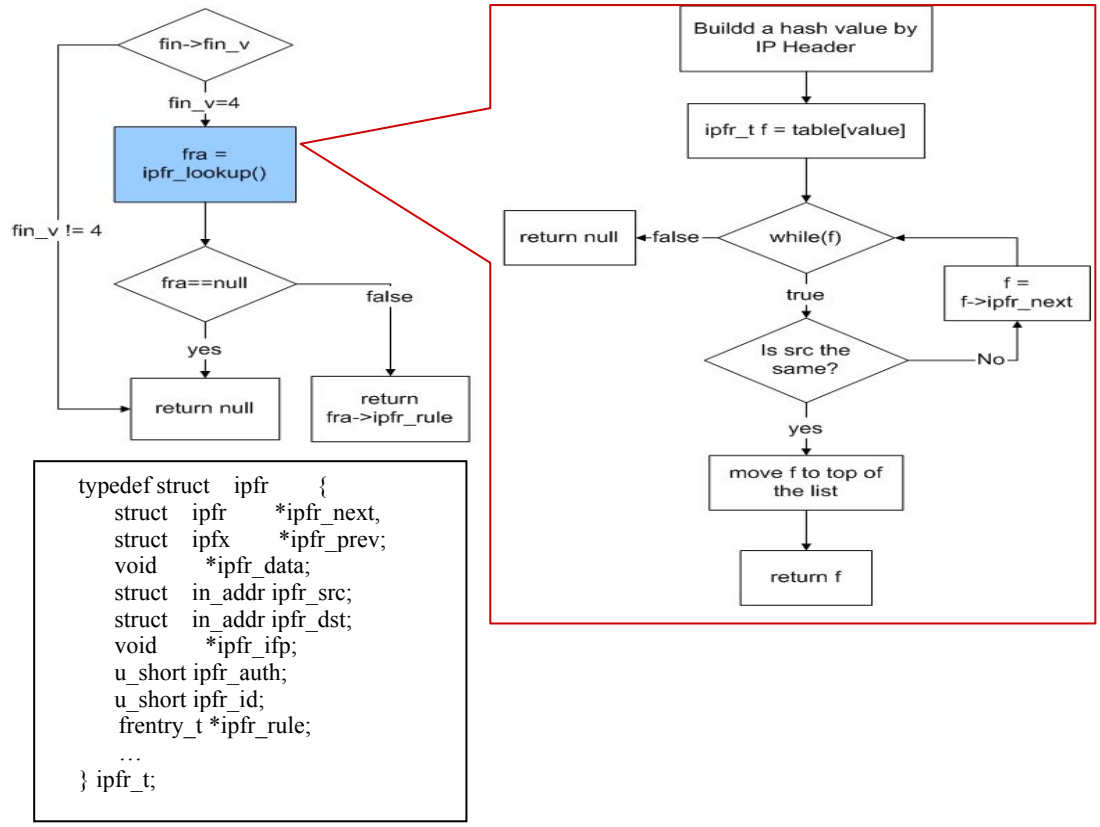

圖 4: ipfr_knownfrag()流程圖

這部分主要由 ipfr_knownfrag()所負責。圖 4 是這個函式的流程圖。一開始會傳入資料結構為 fr_info 的 fin 變數。首先看這個 IP 封包版本是不是 4，如果不是，那就直接跳回，這是因為 IPv6 的分段是要由使用者自行負責而不是路由器負責的關係。如果是 4，那就呼叫 ipfr_lookup 函式，這個函式會利用 IP 封包表頭資訊計算出一個雜湊值 idx，然後從分段快取雜湊表 ipfr_heads[]的第 idx 位置開始搜尋。ipfr_heads[]為資料結構為 ipfr 的陣列，如果有分段封包被處理過，那處理完的資訊會被建成 ipfr 的資料結構存在 ipfr_heads 裡面。ipfr 紀錄的資訊括來源(src)、目的位址(dst)的 in_addr 資料結構，還有處理此分段封包用的規則。

這邊搜尋分段快取雜湊表的方法為線性探測。搜尋是依據 ipfr->src 欄位是否一致，如果有找到一致的，先將找到 ipfr 移到 ipfr_heads[]的 idx 位置，這樣可以加快之後的搜尋(因為分段封包幾乎都是一次進來一群)，然後再將找到的資訊傳回，我們便可利用找到的規則(ipfr->ipfr_rule)處理分段封包了

(ii)狀態檢查(netinet/ip_state.c)：

IPfilter 不僅能替 TCP/IP 做保存狀態的動作，也能處理 UDP 與 ICMP，這也是 IPFilter 比另一套防火牆 IPFW 強的部分。在此我們僅詳細解釋處理 TCP/IP 的流程，其他兩個僅在圖中顯示，有興趣的讀者可從 IPFilter HOWTO 網頁[6]瞭解處理 UPD 與 ICMP 的狀態資訊的方法。

檢查狀態這部分主要是 fr_checkstate()這個函式負責，圖 5 是這個函式的完整流程圖，這個函式的步驟簡介如下：

1. 一開始也是根據 IP 封包表頭算出雜湊值 idx，然後根據通訊協定做不同處理。例如對於 TCP/IP 是 SYN/FIN 封包，而且標示 RST(重新設定連線的)。我們就不做狀態處理。
2. 與分段封包處理一樣，這裡也有一個資料結構為 ipstat 狀態快取雜湊表 ips_table[]，紀錄需要做保存狀態的封包資訊，當規則有指定保存狀態時，第一個要求建立連線的 TCP 封包(標設為 SYN)資訊會建立一個 ipstat 的資料結構，裡面包含比對成功的規則與其他有用資訊，然後會存在 ips_table[]中，之後進來的封包會從 idx 位置開始與 ips_table[]中每一個元素比對，如果有比對成功，就代表是屬於某個 TCP 連線，就能馬上知道用哪一條規則處理，這邊比對的部分是呼叫 fr_matchsrcdst()處理。
3. 最後會看這個 TCP 封包是不是結束連線的封包(旗標設為 FIN)，如果是，就將 ips_table[]中找到的 ipstat 紀錄刪除，然後回傳找到的規則，如果不是，就單純回傳找到的規

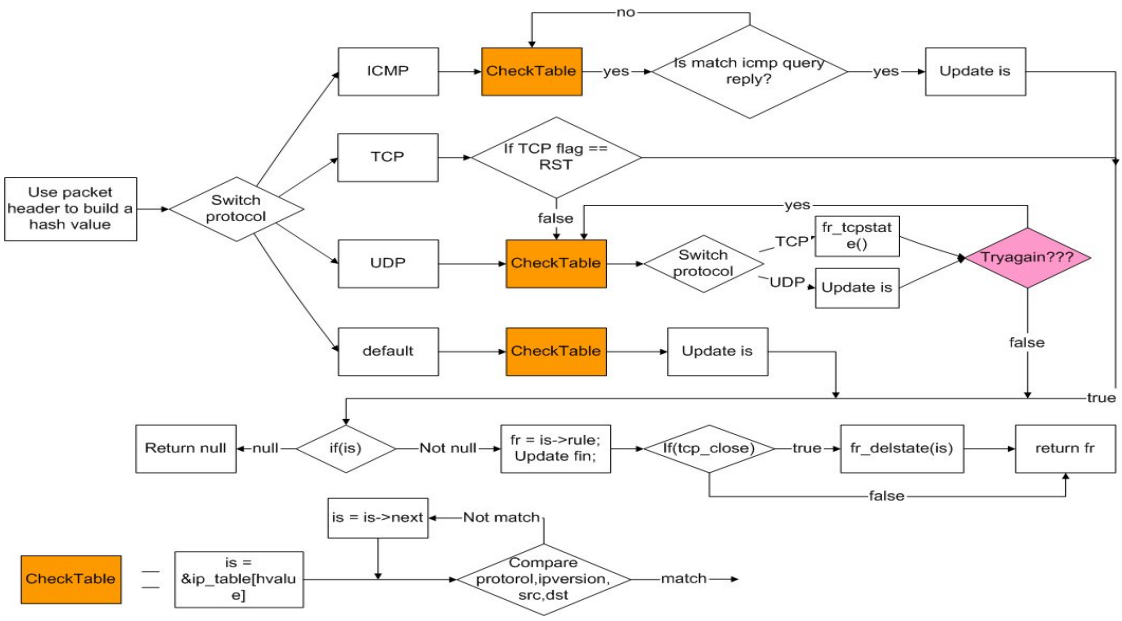

圖 5: fr_checkstate()流程圖

(iii)規則比對(netinet/fil.c)：

如果之前的步驟都不能讓封包快速通過的話，便會進入這個步驟。這步驟呼叫 fr_scanlist()一路比對整條規則的鏈結串鍊，圖 6 是這個函式的流程圖，其詳細比對步驟如下：

1. 記得我們說過，規則是有階層關係的，因此一開始會從最上層的從第一個規則開始，依序以網路介面，封包協定種類這兩個項目來初步判定封包是否符合。如果是 TCP 或 UDP 封包，就呼叫 fr_tcpudpchk 進一步比對來源、目的地還有 TCP 旗標。如果是 ICMP 封包則必須該規則也是用來處理 ICMP 時，才會進一步比對。如果都不符合，則換下一條規則比對。
2. 紀錄比對的結果或符合某規則的封包總數，比對成功次數等
3. 接著會看這條規則的 fr_grp 欄位是不是 null，如果不是就代表這條規則屬於某個群組的最前頭，並且這個群組還有其他規則存在，接著會遞回呼叫 fr_scanlist，從 fr_grp 所指的群組串列第一個規則開始處理。這部分請參閱圖 2 來看，會比較清楚。
4. 最後比對會看這條規則是不是有設關鍵字 quick，如果有，那比對到這條規則便直接回傳能不能通過。如果沒有就會一直比對下去。

以上就是整個 IPFilter 運作流程，希望對讀著了解 IPFilter 有所幫助。

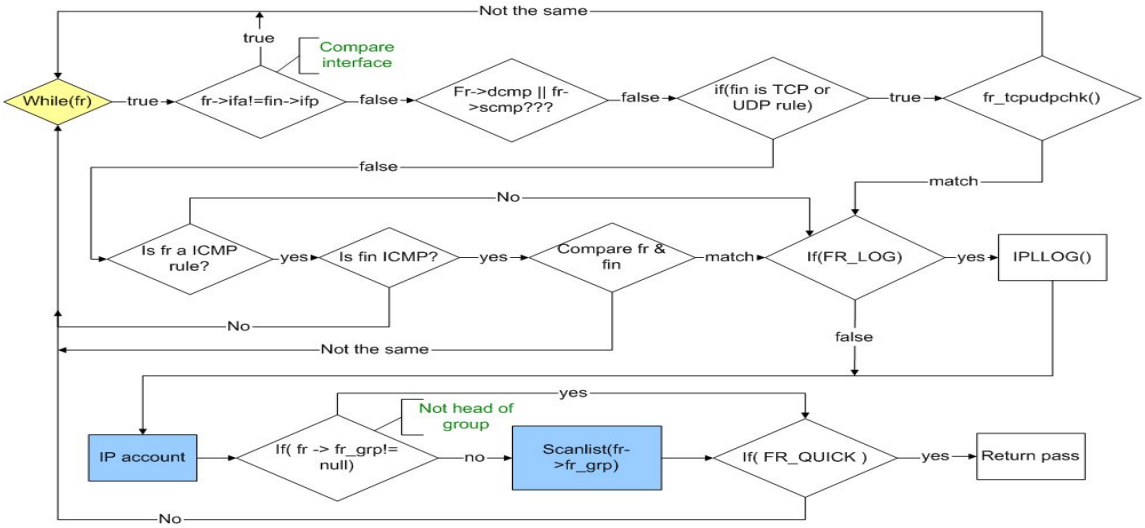

圖 6: fr_scanlist()流程圖

4. KAME 之 IPSEC

KAME[7]，是個很大的 Project，主要由日本許多公司參與研究，例如 Fujitsu, Hitachi, NEC, Toshiba…等。 其發展目標是在各式 BSD 平台上, 例如 NetBSD, FreeBSD, OpenBSD, 實作 IPv6, IPsec, 還有一些進階的網路應用例如 Packet queuing, ATM。而 NetBSD 從 1.5 版便已經將 KAME 所發展的 IPv6, IPsec 部分整合進核心中。KAME 整個 Project 包含了許多小 Project，Kernel 與 Userland 的部分皆有許多有趣的東西，而這次我們只挑選我們比較有興趣的 IPsec 部分做追蹤。

由於 IP 封包本身不具安全性，要偽造 IP 封包，修改 IP 封包或是偷看傳輸中的 IP 封包內容都是非常容易的事。因此 IETF 在設計 IPv6 時，提出相關的安全保密架構，後來該架構才獨立為「網際網路層安全協定」(IP Security Protocol)，簡稱 IPsec。IPSec 簡單來說就是 AH + ESP，也可以再搭配 IPcomp + IKE。其中 AH(Authentication Header)為認證標頭，提供封包來源的驗證、以及內容一致性的檢查，可以用來檢查 IP 封包在傳輸過程中沒有被修改。而 ESP(Encapsulating Security Payload)為封裝安全負載，可結合加密演算法對封包內容加密，不用擔心封包遭竊取。由於 AH 跟 ESP 的加密都是需要指定編碼金鑰，因此如果不想通訊兩端手動設定安全金鑰，便需使用網際網路金鑰交換(Internet Key Exchange, IKE)通訊協定來管理安全金鑰，這部分不在我們追蹤範圍內，有興趣的讀者可自行參考 RFC2409[9],RFC 3526 [10]。

提到 IPsec，就一定要提到安全政策(Security Policy, SP)與安全關聯(Security

Association, SA)。安全政策 SP 定義了兩地之間交通的原則，可以設定為丟棄，不處理或是交給 IPsec 去處理。而安全關聯 SA 則是 IPsec 的基礎，IPsec 使用的兩個協定(AH 與 ESP)都使用 SA。SA 其實就是通訊各方對下列這些事情的協議：對 IPsec 的通訊協定、通訊協定的運作模式(傳輸或是通道)、編碼演算法、保護交通用的編碼金鑰與金鑰的存活時間。如果 AH 與 ESP 兩者都要使用，那麼就需要兩組 SA。因此如果想要處理 IPsec，這兩個資料庫是必要的：

- 安全政策資料庫 (Security Policy Database)
- 安全關聯資料庫 (Security Association Database)

4.1 IPsec 主要資料結構

IPsec 用到的資料結構關係很複雜，依功能分為三類：

- 演算法：ah_algorithm, esp_algorithm (netinet6/ah.h esp.h)
- SPDB(安全政策資料庫)：secpolicy, secpolicyindex (netinet/ipsec.h)
- SADB(安全關聯資料庫)：secasindex, secashead, secasvar (net/keydb.h)

而 ipsecrequest 負責連結 SPDB 與 SADB。接下來，我們以一個例子來介紹這些資料結構，表 5 是一個假設的設定檔內容。

```
add 20.0.0.1 20.0.0.2 ah 7777 -A hmac-md5 "hogehogehogehoge";                    //sa1
add 20.0.0.1 20.0.0.2 esp 8888 -E blowfish-cbc "blowfishtest.001" ;              //sa2
add 20.0.0.2 20.0.0.1 esp 9999 -E blowfish-cbc 0xdeadbeefdeadbeefdeadbeefdeadbeef;  //sa3
```

```
spdadd 10.0.1.0/24 10.0.2.0/24 any -P out ipsec esp/tunnel/20.0.0.1-20.0.0.2/require
ah/tunnel/20.0.0.1-20.0.0.2/require;                                             //sp1
spdadd 10.0.5.0/24 10.0.6.0/24 any -P in discard;                                //sp2
spdadd 10.0.2.0/24 10.0.1.0/24 any -P in ipsec esp/tunnel/20.0.0.2-20.0.0.1/require ;  //sp3
```

表 5：IPSec 的一個設定檔範例

這個設定檔會建立起圖 7 的資料結構。我們將參考這個圖來，來仔細解釋三類型結構。

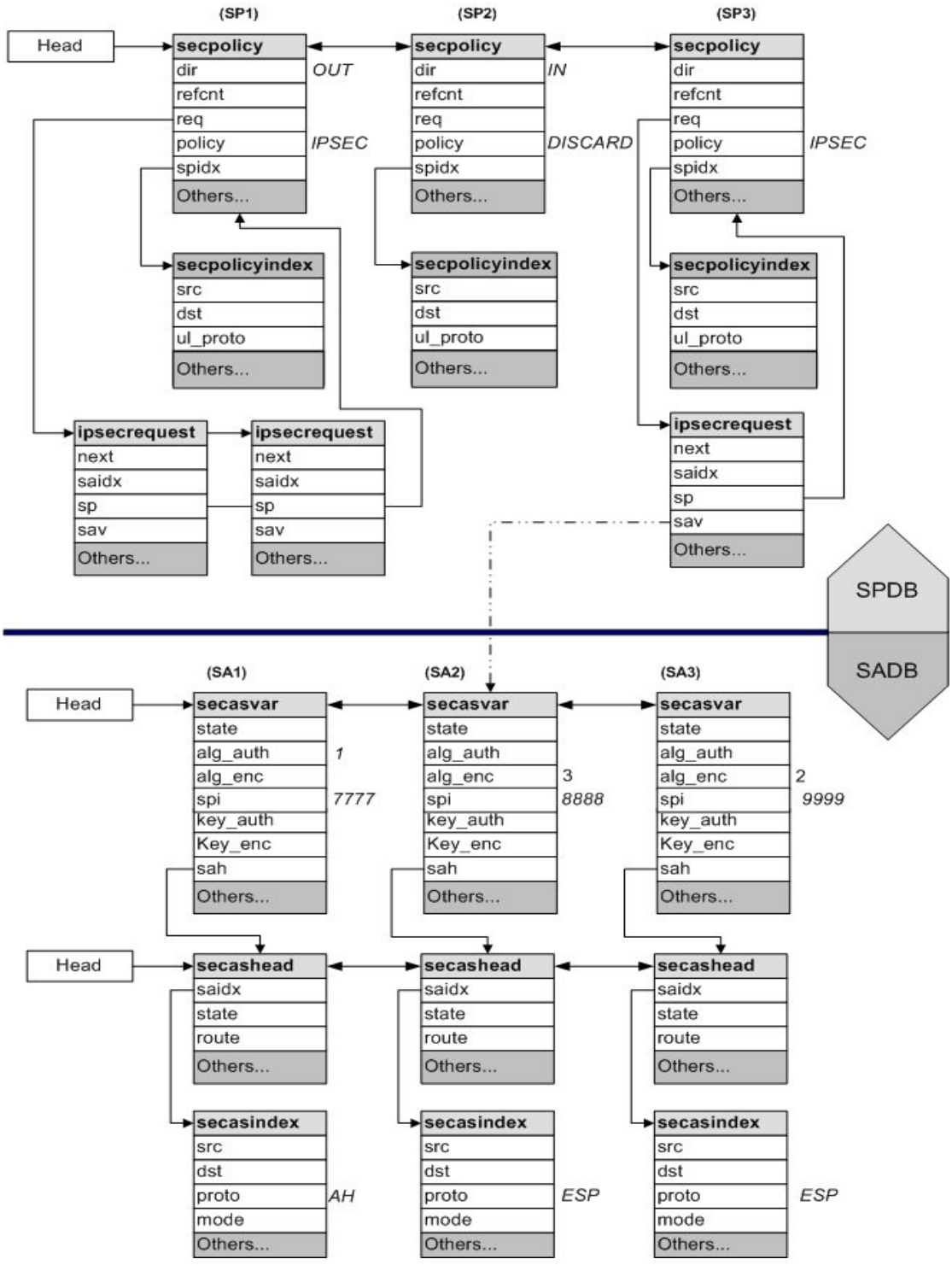

圖 7: IPsec 資料結構

(i) 演算法用的結構:

AH 所有能使用的演算法被存在一個資料結構為 ah_algorithm 的陣列中，每個 ah_algorithm 皆定義這個演算法使用的金鑰的最小與最大長度，以及五組函式指標：sumsiz, mature, init, update 與 result。可使用的 AH 演算法包括 MD5HMAC, SHA1HMAC, MD5 與 SHA。ESP 所有能使用的演算法被存在一個結構為

esp_algorithm 的 esp_algorithms[]陣列中，每個 esp_algorithm 如同 AH 演算法，也是定義了金鑰的的最小與最大長度，還有下面這些函式指標：encrypt, decrypt, mature, schedlen, ivlen。可使用的 ESP 演算法包括 DESCBC, 3DESCBC, BLOWFISHCBC, CAST128CBC, RIJNDAELCBC

(ii)建立 SADB 的結構：secasvar、secashead 與 secasindex。

圖八的設定檔前面三行會建立三個 SA，每個 AH 或是 ESP 皆須有一個獨立的 SA，而且 IN 與 OUT 的 SA 是獨立的。每個 SA 是由 secasvar、secashead 與 secasindex 組成的，在 secasvar 中會指定所用的演算法編號，還有 spi 與提供給演算法的金鑰，alg_auth 與 key_auth 是給 AH 使用的，alg_enc 與 key_enc 是給 ESP 使用的，sah 會指到一個 secashead 資料結構，secashead 包含了這一些資訊，包刮這個安全關聯目前狀態是死亡的還是可以使用的，還有一個 saidx 指到 secasindex 的資料結構，secasindex 這個資料結構是讓之後 SP 來搜尋 SADB 中符合的 SA 時用的，secasindex 記載來源位址(src)，目的地位址(dst)，與使用的通訊協定(proto)，看是 AH 或是 ESP。當 SP 搜尋 SADB 時就是以這三個欄位為搜尋依據來找出所需的 SA。所有的 SA 會被串成雙向鍊結串鍊，而且新加入的 SA 會被加到這條串鍊的頭端。

(iii)建立 SPDB 的結構：secpolicyindex 與 secpolicy。

表四中的設定檔，最後面的三行會建立三個 SP，每個 SP 是由 secpolicyindex 與 secplolicy 組成的，secpolicy 裡面的欄位比較重要的有 spidx，spidx 會指到一個 secpolicyindex 的資料結構，這個資料結構紀錄要處理的來源(src)，目的地(dst)與上層的協定(ul_proto)。當處理封包搜尋所需的 SP 時，便是以比對 spidx 來做搜尋。欄位 dir，紀錄這個 SP 處理的是 IN 還是 OUT，欄位 policy 指定這個 SP 是要怎麼處理，policy 可以設定為不處理(NONE)，或是丟棄(DISCARD)或是交由 IPsec 處理(IPSEC)，當指定為 IPSEC 時，req 欄位便會指向一串 ipsecrequest 資料結構構成的鏈結串列。ipsecrequest 這個資料結構裡有一個 saidx，saidx 資料結構為之前介紹的 secasindex，當 IPsec 要取得對應的 SA 時就是利用 ipsecrequest 的 saidx 欄位來搜尋 SADB。欄 sp 會指回我們的 SP，而欄位 sav 的資料結構是之前介紹的 secasvar，他不是馬上就指到 SADB 中的某個 SA，sav 一開始是設定為 null 的，也就是一開始處理設定檔時是與 SADB 沒有關聯的，在之後封包的處理過程中，當需要 IPSEC 處理時，ipsecrequest 才會利用 saidx 去搜尋 SADB，如果

找到對應的 SA，便將 sav 欄位指向找到的 SA。會這樣做是因為 SA 的金鑰可以有其時效性，因此舊的 SA 可能隨時會被刪除，為了避免衍生的問題才動態尋找。了解 ipsecrequest 後，要清楚一個 SP 可以有很多個 ipsecrequest，這是因為 AH 與 ESP 皆須有一個 SA 處理，如果一個 SP 定義同時要用 AH 與 ESP 處理，便須有兩個 ipsecrequest 指到對應的 SA，每個 SP 所用的 ipsecrequest 是用鏈結串列串起來的。

以上就是單一個 SP 的資料結構，所有的 SP 會利用包含頭節點的雙項鍊節串鍊串起來，當搜尋 SPDB 時，就是搜尋這個鏈結串列。在這邊附帶一提，IPsec 裡用的鏈節串列都是使用 sys/queue.h 定義的 LIST Macro 來建立的，LIST 是個有頭節點的雙向鍊結串列，有興趣的讀者可以 man queue，裡面定義了許多不同的 LIST 與 QUEUE 的巨集可使用。在解釋這些 IPsec 資料結構後，想必讀者以經開始頭昏眼花了，在此建議讀者看完解釋後再重新對照著之前的設定檔與圖 7 看一次，想必這樣應該能清楚許多。

4.2 IPSEC 的封包流出處理

在瞭解 IPsec 的資料結構後，我們來看 IPsec 的封包流出流入兩個處理流程。在這一節我們介紹流出過程，至於流入過程則在下一節介紹。

NetBSD 中封包流出的處理部份是在 netinet/ip_output.c 中 ip_output 這個 function。在這個 function 中，與 IPsec 有關的流程圖如圖 8，在此由於篇幅有限，我們只介紹 ip_output 中 IPsec 的部分，原先 ip_output 處理的部分我們以虛線框起來：

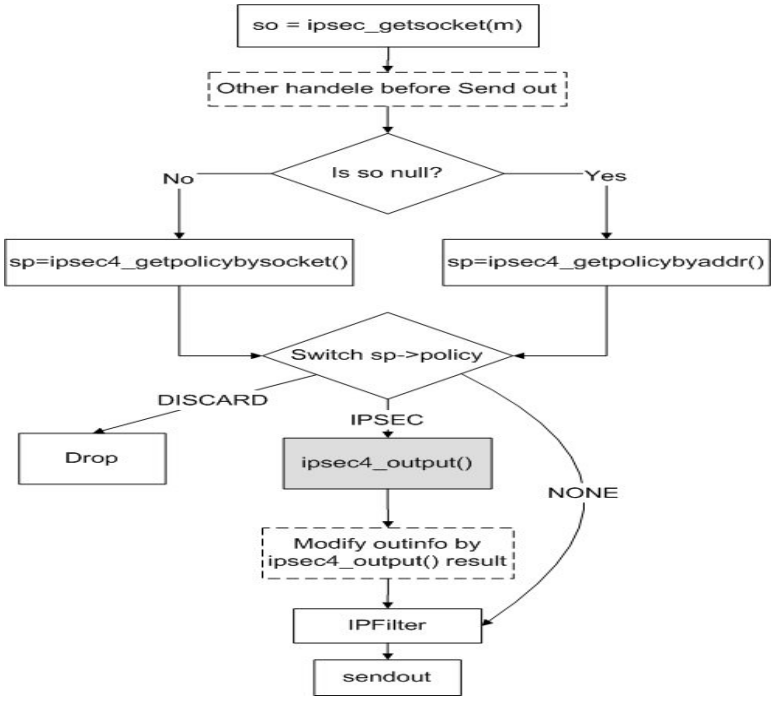

圖 8: IPsec 封包流出處理

1. 首先一開始，我們會根據傳入 ip_output 的 mbuf m 去呼叫 ipsec_getsocket()，根據 mbuf 裡面的資訊去看看是不是屬於某個 socket，如果是就會傳回所屬的 socket。
2. 接著我們要取得安全政策 sp，如果我們之前已經有取得 socket，我們會呼叫 ipsec4_getpolicybysocket() 來取得安全政策，否則我們就是呼叫 ipsec4_getpolicybyaddr()來取得安全政策。
3. 有了安全政策後，我們看 sp->policy 的這個欄位，如果這個欄位是設為 DISCARD，我們便直接丟棄這個封包，如果是設為 NONE，就會跳過 IPsec 的處理部份。如果是設為 IPSEC，那就代表要丟給 IPsec 去處理，處理的步驟就是呼叫 ipsec4_output()這個 fnction 去處理，這個 function 我們等等會再介紹。
4. 當 ipsec4_output()處理完後，由於目的地位址，路由資訊可能都會被修改，因此在這邊要利用 ipsec4_output()處理完傳回的資訊來更新 mbuf 資訊，更新完後，如果有指定使用 ipfilter，便會再把這個 mbuf 丟給 ipfilter 去做處理。處理完後如果沒有錯誤便會送出。

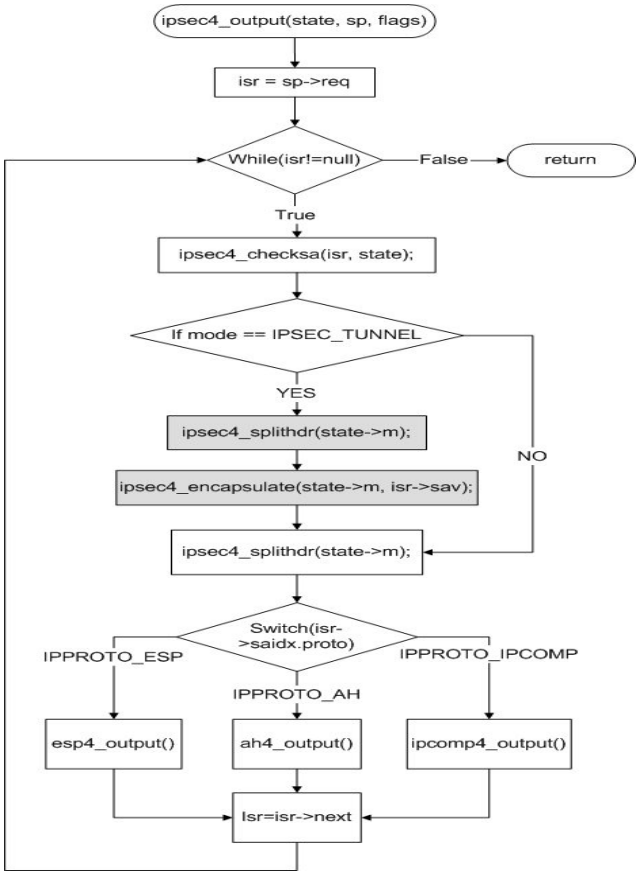

圖 9: ipsec4_output 流程圖

接著我們仔細來看看 ipsec4_output 這個 function 到底做了那些事，圖 9 是 ipsec4_output 的流程圖。ipsec4_ output 主要的處理步驟就是根據傳入的 sp，處理所有屬於這個 sp 的 ipsecrequest 鏈結串列，如同之前說的，如果這個安全政策定義需要 AH 與 ESP 處理，那就會有兩個 ipsecrequest。(請參考之前的資料結構圖)，而每個 ipsecrequest 處理的步驟如下

1. 呼叫 ipsec4_checksa()，從 SADB 中找尋屬於此 ipsecrequest 的的安全關聯 SA，找尋的方法是利用 ipsecrequest 中的 saidx 欄位來搜尋 SADB 中 secashead 的 saidx，比對成功如果找到就將 ipsecrequest 的 sav 欄位指向找到的 SA，如果找不就代表有錯誤，會傳回錯誤訊息。
2. 判斷找到的 SA 的模式是什麼，如果是 TUNNEL 模式，便要做 IP 封包封裝的動作。在 TUNNEL 模式下，會先將呼叫 ipsec4_splithdr()將 IP 表頭跟 option 從 payload 中分離出來，方便之後處理，接著會呼叫 ipsec4_encapsulate()，根據我們傳入的 SA 資訊產生一個新的 IP 封包表頭，並將原先的 mbuf 資料塞到新產生的 mbuf 的資料區內。如果不是 TUNNEL 模式，便是 TRANSPORT

模式，上述的步驟便不需要了。

3. 在找到 SA 與處理完 TUNNEL 部分後，接著判斷我們的 ipsecrequest 所指的 sadix 裡面的 proto 欄位，看是要做 AH 還是 ESP 還是 IPCOMP 處理。如果是 AH，便呼叫 ah4_output()做處理，ESP 則呼叫 esp4_output()做處理。我們大略介紹一下 ah4_output()與 esp4_output()的流程。ah4_output 一開始會先用 SA 的 alg_auth 欄位來取得 ah_algorithms[]中對應的演算法 algo，接著呼叫 ah4_calccksum()，ah4_calccksum()會利用 algo 提供的 function 來產生檢查碼並更新 mbuf，將演算法產生的檢查碼加在 mbuf 中 IP Header 的後面。至於 esp4_output 的主要概念與 ah4_output 差不多，只是 AH 是修改封包表頭，ESP 會需要修改表頭與對封包內容加密，所以更複雜的多。一開始利用 SA 的 alg_enc 欄位來取得 esp_algorithms[]中對應的演算法 algo，然後處理 mbuf 資料的搬移修改，並呼叫 algo->encrypt 所指的 function 來對封包進行加密的動作。礙於篇幅，這兩個 function 無法詳細的說明，但是這邊算是加密的核心步驟，有興趣的讀者可以深入研究看看。
4. 最後便是看是否還有下一個 ipsecrequest 需要處理，如果有就重複 1 - 4 的步驟，沒有的話便結束。

以上便是整個封包流出時 IPsec 的處理步驟。

4.3 封包流入處理

NetBSD 中封包流入處理是由 netinet/ip_input.c 的 ip_input 函示處理，圖 10 是跟 ip_input 中 IPsec 有關的處理流程圖。這個流程圖的主要處理步驟也是一開始根據 mbuf 的內容建立 spidx 然後利用 spidx 去搜尋是不是有安全策略，如果有找到的話，會呼叫 ipsec_in_reject()來檢查 AH 或 ESP 的完整性，這邊主要是在檢查如果我們的 SP 有指定用 IPsec 處理，而且這個封包有使用 AH/ESP 協定，那是不是都能在 SADB 裡面找到對應的 SA，如果不能，就丟棄，如果找到，再看是不是要傳給我們的封包，不是的話就做 IP Forward 的動作，如果是要給我們的，那接下會看這個封包是不是使用通道模式，如果是，會做反封裝的動作，然後將解出來的封包再丟回 ip_input 的程序作處理。如果不是通道模式，那如果有使用 AH/ESP，就呼叫 ah4_input/esp4_input 處理，這邊 ah4_input 與 esp4_input 的處理動作與 ah4_output/esp4_outpu 類似，也都是利用 SA 找到對應的演算法然後利用演算法提供的函式去做解密的動作。

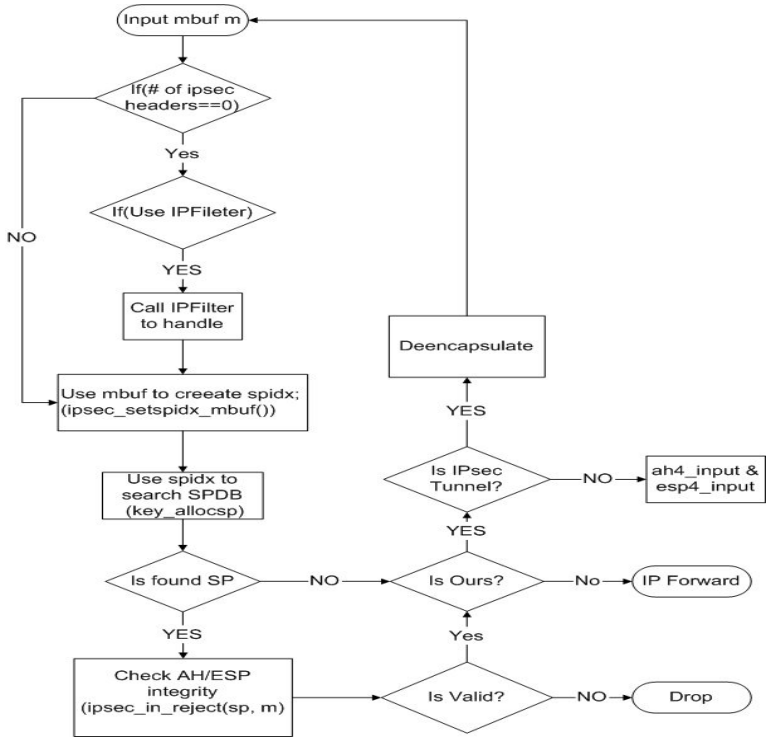

圖 10：IPsec 封包流入處理

以上就是封包流出與流入流程的介紹。IPsec 筆者只能大概介紹到如此，IPsec 的程式碼非常龐大，不只 IPsec 本身而已，還牽扯到許多 Key 的處理部份，因此很多細部無法介紹，有興趣的讀者可以花些時間研究看看，而在 NetBSD 的網頁上有介紹在 NetBSD 中如何使用 IPsec[8]。

5. 結論

為解決網路安全的問題，NetBSD 選定了 IPFilter 與 KAME 這兩個套件，來提供防火牆及封包加密的功能。我們可以發現在 IPFilter 中大量使用快取來處理各種情況，並且存取快取都是使用雜湊處理，因此速度可以有效提升，再加上 IPFilter 內建 NAT，因此 NAT 與防火牆之間的搭配處理能更快速，不像其他防火牆還需要與其他 NAT 軟體配合。而 KAME 的 IPsec 清楚的定義了各種資料結構來串起 SPDB 與 SADB，並對 IPsec 協定實做的非常完整，這也是就為何 NetBSD 會選他來作為內建 IPsec 堆疊的原因。

如果想更深入追蹤核心網路部分的讀者，在 W.Richard Stevens 所著之 TCP/IP Illustrated Vol.2[11]，有詳細 BSD 核心網路部分的實做。相信對想研究核心網路程式的人非常有幫助。此外，追蹤核心程式不是件很輕鬆的事，如能藉由

我們所提及之三個工具：GLOBAL, KGDB, Kernel Profiling，應能大大提升追蹤的效率。

6. 參考資料

[1] The NetBSD Project：http://www.netbsd.org/

[2] 蔡品再、林盈達；『追縱 Linux TCP/IP 的核心(上)』；*網路通訊*； 114 期，2001 年 1 月。

[3] Kernel Profiling HOWTO
http://www.netbsd.org/Documentation/kernel/profiling/

[4] Debugging the NetBSD kernel with GDB HOWTO
http://www.netbsd.org/Documentation/kernel/kgdb.html

[5] The IPFilter Project: http;//www.ipfilter.org/

[6] IPFilter HOWTO
http://www.obfuscation.org/ipf/ipf-howto.html

[7] The KAME Project：http://www.kame.net/

[8] NetBSD IPsec FAQ：
http://www.netbsd.org/Documentation/network/ipsec/

[9] RFC 2409 The Internet Key Exchange (IKE). D. Harkins, D. Carrel. November 1998.

[10]RFC 3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). T. Kivinen, M. Kojo. May 2003

[11]W.Richard Stevens; TCP/IP Illustrated Vol.2.