VPN、NAT、Firewall、Routing Table Lookup
在 Linux 核心上的效能檢測

李佩璇 張順理 林盈達
國立交通大學資訊工程所
October 21，2009

摘要

在電腦安全領域上，如何保護個人或企業上網的安全性日漸重要，因此安全閘道器也成為現今不可或缺的要角。故本文就常見的安全閘道器功能分析其效能，其中又以在 Linux 開放原始碼的安全閘道器為目標。本篇內文將分別量測單一封包經過防火牆(Firewall)、虛擬私人網路(VPN)、網路位址轉譯器(NAT)以及路由查表在 kernel space 執行時所需花費的時間，並客觀分析可能造成效能低落的原因和行為。 防火牆和網路位址轉譯器現今 Linux 已整合在 iptables 上操作，而 VPN 使用 OpenSwan 搭配 IPsec-tools。實驗中可知花費時間最長為 VPN 的加密，約 8.2ns，其次是 md5，約 6.1ns；網路位址轉譯大約 2.6ns，防火牆約 2.38ns；最後墊底的是路由查表，約 0.7ns。

關鍵字：Linux 安全閘道器，防火牆，虛擬私人網路，網路位址轉譯器，路由查表，效能測試

一. 簡介

要保護內部網路(intranet)，企業普遍建構安全閘道來阻斷網路上的攻擊和竊聽行為，本文介紹三種方式，分別是配置防火牆於內部和外部網路之間的過濾機制，以虛擬私人網路(VPN，公用網路上使用密道及加密方法)建置資訊安全傳輸通道，或透過位置轉換器(NAT,常用於處理 IP 不足的問題),利用在轉換 IP 的過程中間接隱藏原 IP 資訊。本實驗選擇在 Linux 系統上安裝安全閘道平台，主要是看中 Linux 穩定且支援開放原始碼的特色，並利用 iptables[1]、OpenSwan[2]/IPsec[3]，來扮演閘道上的不同角色。下面我們將先逐一介紹上述套件的基本操作和原理。

套件名稱	功能	核心空間執行	採用版本
iptables	Firewall/NAT	Linux 內建 NetFilter 封包過濾機制	V1.3.5
OpenSwan IPsec tool	VPN	核心內建 Netkey	U2.6.18/K2.6.18.8

表一 套件介紹

防火牆[4][5]

作為內部網路和外部網路的區隔通口，防火牆依據使用者預先設定的規則，提供檢查和過濾所有進出內部網路的封包的功能。以下分成三大類簡介：

(1)Packet filtering：根據使用者設定的規則，透過檢查進出防火牆的封包標頭(Header) 、通訊連接埠、通訊協定(TCP/UDP)等條件，允許或拒絕存取網路。

(2)Stateful-inspection： Packet filter 的強化版本，可檢測多重封包流。另一個差異點，是前者對於放行的通訊協定採永久開通模式，但後者僅在必要的時候開通（動態調整），因而大大提升安全性。

(3)Proxy 代理伺服器：藉由 proxy 伺服器中介處理 client 和外部網路之間的傳輸。

網路位置轉換器[4][5]

NAT 大多應用於 IP 分享器的功能;允許多個私有 IP 透過一個公用 IP 連上網路，在 NAT 後的所有主機外表看起來就像是使用同一組 IP 上網。又可以依修改出去的封包或是修改回來的封包區分為 SNAT(修改 IP 包中的源位址)和 DNAT(修改 IP 包中的目的位址)。由於 NAT 可屏蔽內部網路，故也具有類似提供安全性的功用。

虛擬私人網路[4][6]

VPN 是利用公用網路來做私人網路傳輸，常用於利用公用網路來存取公司資源，方法是透過具有保護資料的私密通訊。由於網路應用幾乎都採用網際網路協定(IP)，因此，IETF 訂出一套使用於 IP 網路的安全性機制－網路安全協定 IPsec (IP Security)。IPsec 提供兩項機制：IP 認證標頭(Authentication Header)，以及 IP 封裝安全裝載(Encapsulating Security Payload)。前者確保資料傳輸的過程中沒有被動過手腳，後者則是讓傳送出去的檔案經過加密的手續。

二. 程式追蹤

要追蹤核心模式下的系統呼叫函式，我們採用 Kernel Function Trace [7]這個套件來輔助收集執行過程中所呼叫到的所有函式，再以人工的方式來畫出 call graph。

KFT 是一套可以偵測和觀察當前核心處理程式狀態的套件。安裝及使用的步驟很簡單；首先將 KFT patch 到目標的核心上(本實驗環境是 kernel 2.6.18.8)，編譯完核心並且重開機後，就可以在/proc/目錄下看見 kft 的指令。使用上我們利用設定 trigger 函式的做法。舉例來說，我們想要收集從 IP 層接收到封包後核心接下來所呼叫到的核心函式。已知第一個被呼叫到的程式是 ip_rcv，我們就去/proc/kallsym 找到它對應的位置，接著照 KFT 說明文件中的格式寫進一個檔案後，如以下所式：

```
new
begin
trigger start entry 0xc02797c0    (ip_rcv 在 kallsym 查詢到的位址)
logentriesend    20000
(設定可儲存的容量大小,滿了之後就會停止收集 system call 資訊)
end
```

我們可以收集一定數量內的所有在核心執行呼叫的函式名稱。透過這些函式名單，我們再進一步利用人工的方法去對核心程式碼進行 source code 追蹤；最常使用的資源是 LXR 這個 Linux 核心程式碼的網站資源[8]。

iptables 及 NetFilter [5][9][10][11]

這一節承接上一章的基礎，但更深入去分析執行過程中的環境架構。我們將防火牆和網路位址轉換合併起來討論，因為它們實際使用上具有一些共通性，也就是皆可透過 iptables 設定。iptables 是 NetFilter 位於 User Space 的工具，其組成分為兩部分，其一為 NetFilter 的 HOOK，其二則是儲存 HOOK 函數的運作規則。

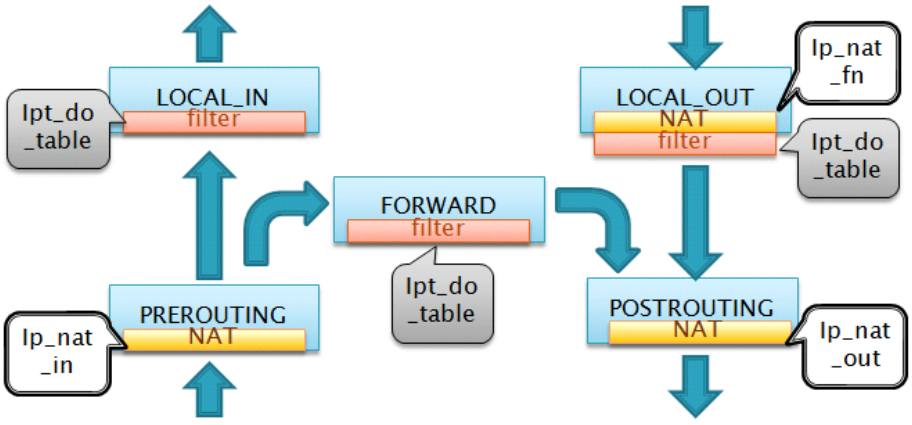
```
iptables   [-t table]     ACTION [PATTERN]      [-j TARGET]
      table 項目              規則                  處理方式
```

NetFilter 是 kernel 的子系統，可以進行封包過濾實現防火牆，以及提供連線追蹤(ip conntrack)實現 NAT。NetFilter 的框架如以下分析：

(1) 五個 HOOK 註冊(NF_IP_PRE_ROUTING、NF_IP_LOCAL_IN、NF_IP_FORWARD、NF_IP_LOCAL_OUT、以及 NF_IP_POSTROUTING)，也就是五種 Chain。

(2) 我們可以將模組掛載到 HOOK 上(利用函式 nf_hook_ops)，在核心註冊該 HOOK 後，當往後執行到 HOOK 時，就可對封包依訂立的規則做處理。

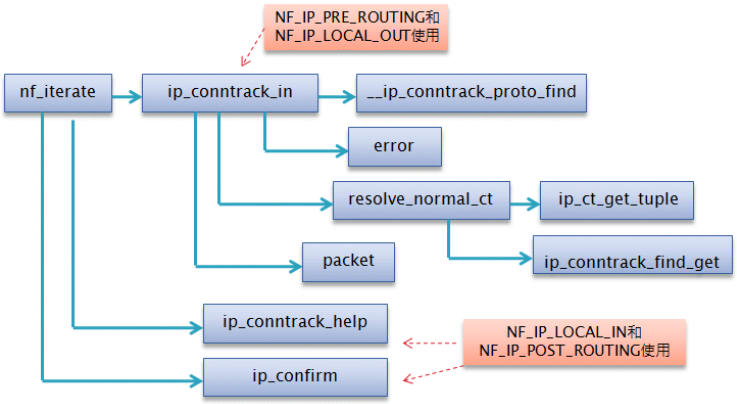
(3) NetFilter 的流程 flow 請看下圖：



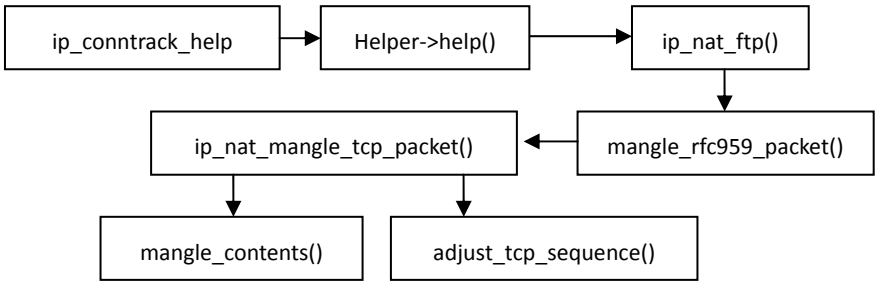
圖一 封包通過 NetFilter 流程[5]

除了五種 Chain 外，iptables 在核心還有三種類別的 table：filter，nat，和 mangle。Filter 用來過濾封包(Accept 或 Drop)；nat 可修改封包的來源和目的，以及 IP 偽裝(Masquerade)，也和連線追蹤有關。 Mangle 較不常用，可修改封包特定欄位(TOS，skb 或 TTL 欄位)。

Fierwall 主要是對應 filter 處理封包的功能，這部分較無爭議。重點在於 nat，要將封包在兩張網卡間 forward 出去，需要一連串追蹤流程的前置處理和後置作業。封包首先進入 ip_conntrack_in()，呼叫 ip_conntrack_proto_find()尋找該封包使用的連線追蹤，接著呼叫 error()檢查封包的正確性，下一步若 resolve_normal_ct()判斷連線不存在，則會建立一個新的連線追蹤。ip_conntrack_help()判斷是否進來的封包有輔助資訊要求(例如 FTP 主動模式,如圖三所示)；最後，ip_confirm()會再次確認連線追蹤，並在此時真正寫入 state table，如圖二所示。

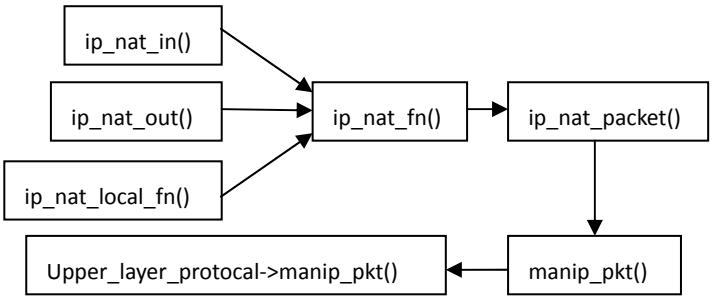


圖二 連線追蹤分析



圖三 ftp ALG

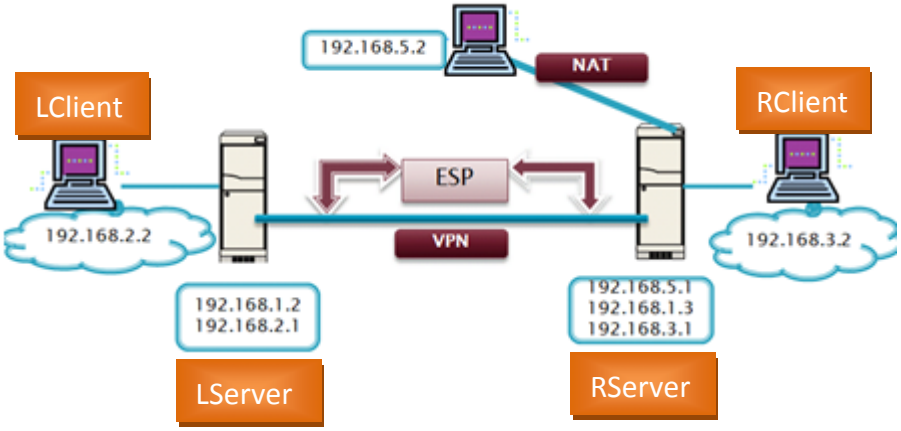
我們討論完 NAT 的資訊儲存，接著來看宏觀的 NAT 的處理部分。ip_nat_fn()是中心的處理函式，它會去呼叫 ip_nat_rule_find()，利用 ipt_do_table 去檢查 NAT 規則，最後呼叫 ip_nat_packet ()置換掉原 IP(使用 manip_pkt()去修改)。



圖四 NAT 流程圖

Open Swan 和 IPsec-tools [12][13][14][15]

Open Swan 是 IPsec 的軟體，pluto 是 key 的管理工具，而核心模組我們使用 linux 預設支援的 Netkey(所以需安裝 ipsec-tools)，而不是另外 patch Open Swan 發布的 KLIPS。如果成功安裝 Open Swan，我們輸入 ipsec –version 指令就可以看到 IPsec 相關的訊息，或是利用 ipsec verify 指令條列出當前環境的檢查結果。圖五是實驗環境的硬體架構，有兩台作為 gateway 的 server，作為兩台 client 間 VPN 溝通的橋梁。



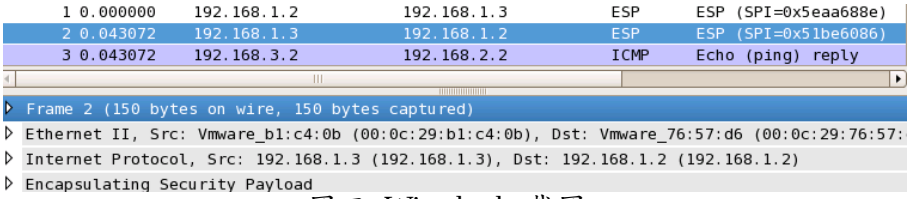
圖五 VPN 架構圖

接著我們要配置 IPsec 的相關設定。舉例來說，我們需要先取得作為安全閘道雙方的公鑰，然後在/etc/ipsec.conf 中填入公鑰資訊。實驗中我們採用以 RSA 認證，在 LServer 上執行 ipsec showhostkey –left 指令獲得 LServer 的公鑰，同理在 RServer 上執行 ipsec showhostkey –right 指令獲取 RServer 的公鑰。接著配置/etc/ipsec.conf，範例文件如下(只列出重要部分):

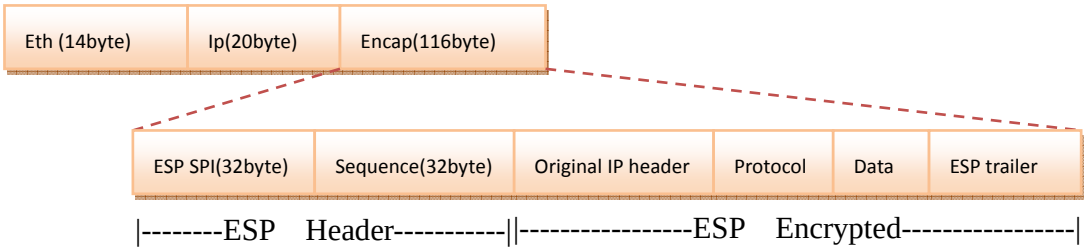
```
conn net-to-net
    left=192.168.1.2              # LServer 連接外部網路的 IP 地址
    leftsubnet=192.168.2.0/24     # LServer 連接內部 IP 子網域
    leftrsasigkey=qsdf…           # LServer 的公鑰
    right=192.168.1.3             # RServer 連接外部網路的 IP 地址
    rightsubnet=192.168.3.0/24    # RServer 連接內部 IP 子網域
    rightrsasigkey=ddsfs.. .      # Rserver 的公鑰
```

值得一提的是，在以往 VPN(Freeswan)採用 AH 和 ESP 共同為封包的完整性提供保護。但在新版的 Openswan 中，AH 已可由 ESP 取代掉了。ESP 現已提供認證的功能，故已不在同時使用 AH 和 ESP(也就是在 ESP 外頭再包一層 AH 結構)。

圖五和圖六列出實驗時擷取之 ESP 封包的結構(參考[16][17]):

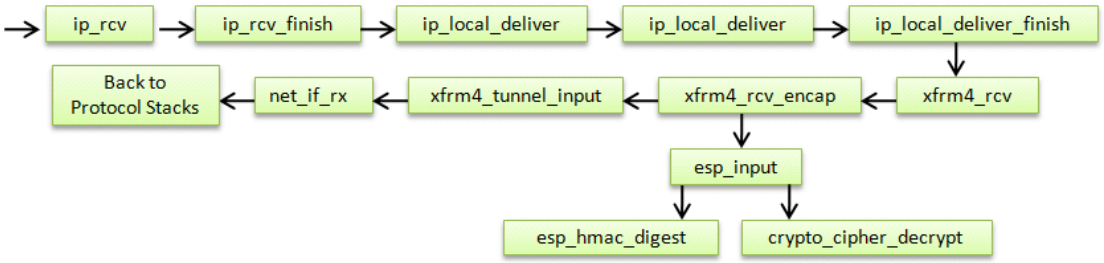


圖五 Wireshark 截圖

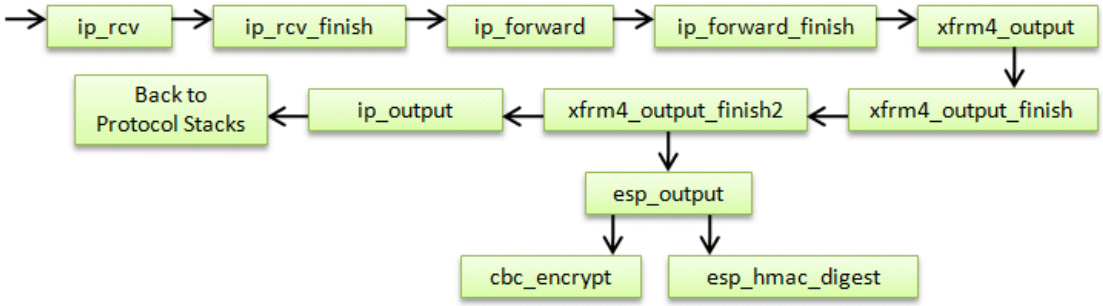


圖六 ESP 結構

另外，我們一般 VPN 傳輸的模式是採用通道模式(Tunneling Mode)而非傳輸模式(Transport Mode)。在官方出版的書上[14]有個很有趣的比喻；傳輸模式就好比一般將車由一個地點開往另一個地點情況，而通道模式則是在火車上開車－火車會將車載往目的地，但在要將車卸下的時候，你需要有一張有效的火車票。接下來，圖六和圖七就是我們實際上追蹤程式碼的結果，並參考資料[18][19]:



圖七 VPN 解密(安全閘道出口)



圖八 VPN 加密(安全閘道入口)

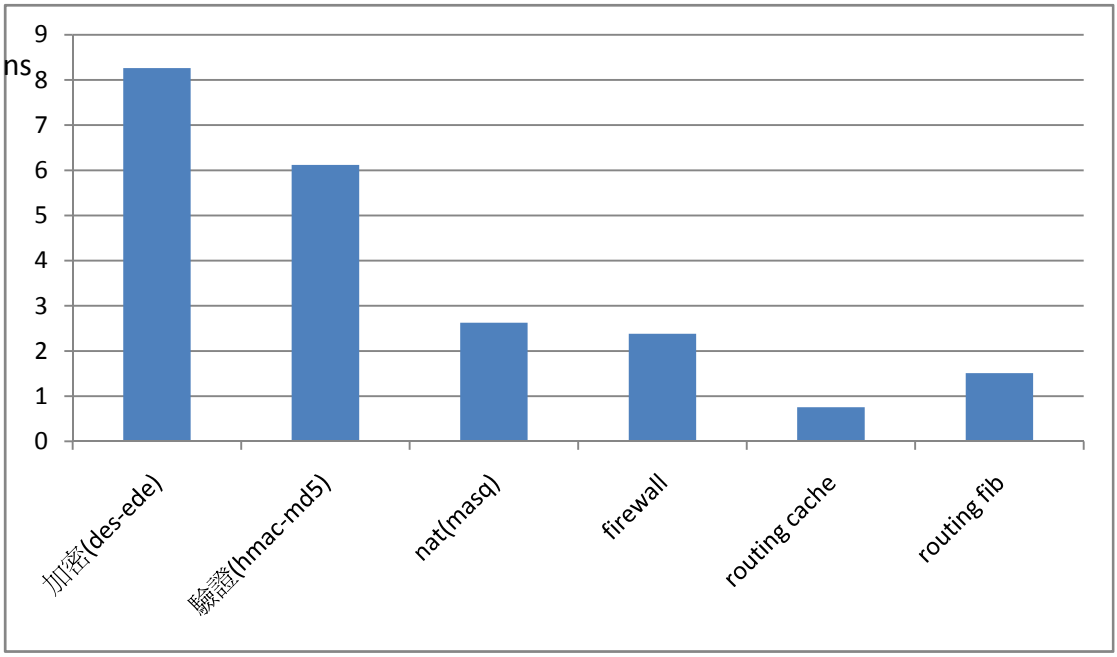
首先我們看到圖七，當加密過的封包經過安全閘道時，由於此時的 IP header 是仍是本機的 IP，會先進入本機端封包解密後，再重新跑一次接收流程，這時才真正將封包轉發給安全閘道後的 client 端。封包首先由 ip_rcv 接收後,會對其 header 做檢查和處理,結束時進入 ip_rcv_finish；下一步，呼叫 ip_local_deliver，目的是為了將 IP 數據包傳送到更高的協議層，同時進行 IP 碎片的重組，完成時同樣呼叫 ip_local_deliver_finish。之後由於是 ESP 協議的原故，故 ESP 會呼叫其 handler 函式 xfrm_rcv，此函式再呼叫 xfrm4_rcv_encap,設定封裝類型(設 0 表無封裝)；進入 esp_input 後，對資料進行解密。以上步驟完成後，重新進入 netif_rx，當成新封包來處理。

接著我們觀察圖八，當進行加密封包的過程時,會呼叫 ip_forward 函式去處理；在 ip_forward 函式內，會呼叫 xfrm4_route_forward 函式，為此封包建構一個安全路

由。以上結束後就進入 ip_forward_finish，在這裡會呼叫路由輸出函式 dst.output，此時的 dst.output 為 xfrm4_output(安全路由輸出函式)。進入到 xfrm4_output_finish2 時，會呼叫 ESP 協議類型的輸出(即為 esp_output)進行操作，例如加密、通道封裝等操作;最後進入 ip_finish_output 函式輸出封包。

三. 實驗結果

我們利用插入程式碼的方式來量測數據結果；即在量測的 function 的起始和結束點插入 rdtscll()。此函式可以記錄下當時的 clock 數量，故將起始和結束點採集到的數據相減之後，就得出目標 function 過程其間所花費的 clock 數。再乘上電腦當前 CPU MHz 值的倒數，所得的結果就是精準度以 ns 為單位的數據了；測試的環境是利用 ping 一個封包大小的流量，使用預設的 ICMP 封包大小。



表二 核心花費時間

四. 討論與結論

由結果可以看見，VPN 執行加密和驗證動作最耗費 cpu 時間，緊接著 NAT 和 firewall 的處理排名其次，最快的則是路由查表的部分。在此我們建議，若需提升整體效率，可將硬體加速技巧應用在 VPN 的加密與驗證動作。

此外，我們觀察在 Linux 核心中，加密和驗證所花費的時間比例（8.262ns 除以 6.118ns），約是 1.35 倍左右。為了探究加密與驗證工作的特性，我們亦在 Linux 用戶空間(userspace)使用 openssel speed 指令，計算不同封包大小的加密或是驗證所需要的時間。我們先查詢 md5 在 64bytes 封包大小下，每秒處理的封包數約是 69832k，而加密方面與我們實驗最接近的演算法為 des-ede，同在 64bytes 封包下每秒處理的封包數量是 51810.87k。兩者的倒數(也就是一個封包的處理時間)分

別是 14.32ns (md5)以及 19.3ns (des-ede)。將 19.3 除以 14.32，亦可得到近似 1.35 的比值。由此可顯示，a) 核心端的加密與驗證函式比用戶端更有效率，由用戶端時間除以核心時間可知約提升 2.34 倍；b) 不論是用戶端與 Linux 核心，des-ede 加密所需的時間約為 md5 驗證所需的時間的 1.35 倍。

參考文獻

[1] iptables ,http://www.netfilter.org/.

[2] Openswan, http://www.openswan.org/.

[3] IPsec-tools, http://ipsec-tools.sourceforge.net/.

[4] Yi-Wei Chnag and Chang-Bin Wang, “A method to improve the security of intranet network by combining firewall with virtual private network”, June 2007.

[5] 鳥哥的 Linux 私房菜 - Linux 防火牆與 NAT 主機 http://linux.vbird.org/linux_server/0250simple_firewall.php.

[6] Hui-Huang Yeh and Shi-Jinn Horng, “Implementing the IPSec VPN in a dynamic IP network”, July 2005.

[7] Kernel Function Trace(KFT), http://elinux.org/Kernel_Function_Trace.

[8] LXR / The Linux Cross Reference, http://lxr.linux.no/.

[9] 蘇建郡, “以 IPTables 設計高可用性叢集式防火牆”, 行政院國家科學委員會補助專題研究計畫, 2003-2004. http://ir.lib.stut.edu.tw/bitstream/987654321/2963/1/922218E218-14.pdf.

[10] Netfilter 連接跟蹤與狀態檢測的實現 http://www.chinaunix.net/jh/4/815129.html.

[11] Chien-Chih Wang and Cheng-Jen Tang, “The Way of Recording Network Packets or Blocking by Using The Open Source Software of BRCTL、TCPDUMP and IPTABLES”, May 2008.

[12] OpenSwan 安裝配置, http://blog.chinaunix.net/u2/73166/showart_1077960.html.

[13] OpenSwan 安裝配置指南, http://9ng.cn/1/viewspace_8112.html.

[14] Paul Wouters and Ken Bantoft, “Building and Integrating Virtual Private Networks with Openswan,” packtpublishing.

[15] Open Sourse Linux VPN－Openswan 安裝, http://9ng.cn/1/viewspace-8112.

[16] IPSec Architecture, http://docs.hp.com/en/J4255-90011/ch04s03.html.

[17] IPsec packet diagram, http://en.wikipedia.org/wiki/IPsec.

[18] Ralf Lehmann, Mirko Benz, Stephan Groß, and Maik Hampel, “IPsec Protocol Acceleration using Network Processors”, 2003 http://www.rn.inf.tu-dresden.de/uploads/Publikationen/LeBeGrHa2003.pdf.

[19] IPSec 在核心中的實現, http://blog.chinaunix.net.