逢　甲　大　學

資訊工程學系專題報告

TCP syn flood 的防禦

學生: 張志豪 (四丁)
林子棠 (四丁)

指導教授: 劉振緒

中華民國九十三年五月

摘要

在現今網路環境中, 用的甚為廣泛的就是 www 的應用了, 舉凡各大知名入口網站, 或是網路交易, 都是藉由網頁且配合其他技術來達成溝通交易的目的, 所以 Web Server 就扮演著服務溝通的重要角色。可是網路的便利, 讓有些人得以在其中利用網頁伺服器的設計缺失來惡意破壞, 使得 Web Server 無法正常服務其他合法使用者。而 TCP SYN flood 攻擊則為其中一種。

在我們的實作中, 將針對 DoS 中的 SYN flood 手法做一些實驗, 將使得遭受此種阻斷式攻擊的 Web Server 可以在防火牆的遮蔽下還能正常地服務線上的客户。並以 Apache Web Server 為實驗對象, 來實作如何利用開放原始碼的防火牆 IPtables 來防範。

但是, 這個防火牆是屬於網路過濾型的, 其中有些地方的功能不太能夠符合我們這次研究的內容, 像是阻擋突然暴增主動連線, 在 IPtables 上的做法是以一種從最大量限制的方式, 而超過此量的連線都會被擋掉, 所以有可能在這個數值沒有考慮好的情況下被設定了, 這會造成正常客户的連線有可能也被擋掉。

所以我們使用 IPtables 作者在官方套件上所附上的一套可延伸的 API: libIPq。它可以提供一套介面可以讓使用者可以接觸到每個經過 Netfilter 核心模組的封包, 而且還可以讓實作者能額外擴充另外的對於網路封包的處理。我們將利用它來完成我們的實作。

目錄

圖目錄

第 1 章
前言

1.1 研究動機

阻斷式攻擊難以防禦的原因是由於在防範的一方必須要能夠分辨那到底一道連線才是他們所服務的顧客的, 因為在 TCP/IP 的規範下, 一般的使用者必須要通過三道一定的程序 (Three-Way Handshaking) 才能建立雙方的溝通管道。

然而, 惡意的攻擊者利用這樣的規範反而造成了網頁伺服器管理者的困擾, 因為攻擊者一直發出三道程序中的第一道 (傳送 syn 訊息) 給網頁伺服器, 讓伺服器一直在處理像這樣的連線請求 (回應 syn/ack 訊息), 然後會有一堆處於 SYN_RCVD 的的連線狀態出現在網頁伺服器的連線表格上, 這個 SYN_RCVD 是指伺服器正在等待使用者的第三道 Three-Way Handshaking 的 ack 訊息, 使得使用者對伺服器的連線狀態成為 ESTABLISHED, 然後可以開始正常的網頁瀏覽服務。但是, 如果網頁伺服器的某個連線 entry 一直處於 SYN_RCVD 的狀態的話, 那這個 entry 將佔住一個伺服器的資源 (也就是說, 其他使用者少了一個連線的機會), 在伺服器等到第三道從使用者那兒來的 ack 訊息前將沒辦法做任何處理。若是上述的 entries 被攻擊者使用 syn flooding 佔住極大部份, 那這個網頁伺服器等於是當機了, 就達成攻擊者的目的了。

網頁伺服器特別容易遭受到這種攻擊, 因為一開始的連線一定要通過 Three-Way handshaking 的三個步驟, 所以這個缺點就被惡意的人給利用來做一些惡意的阻斷服務攻擊, 讓某個他想要攻擊的網站停擺, 而且持續停擺的時間和攻擊的規模還能由他來決定。

由於這個情況讓網頁伺服器管理者苦惱於到底要如何防止這類型的惡意侵擾, 讓我們想要試試看是否可以利用一些自由開放軟體的程式來做一些實驗, 為了要找出如何做才能加強網頁伺服器在遭受攻擊時還能正常的服務它原本要服務的客户的程度。

1.2 軟硬體工具

在軟體元件方面, 我們的實驗在 GNU/Linux 核心版本 2.4.18 的作業系統上進行, 使用了一支附有原始碼的 SYN generator 做主動連線來產生含有 syn 旗標的網路封包, 還有 GNU/Linux 的套件 IPtables 來做為我們的過濾型防火牆。這個套件分成兩個部份, 一個在核心中, 一個讓使用者寫入規則來控制防火牆的過濾方式。

在核心之中的是 Netfilter 模組, 專門在處理流通於機器之間或之內封包的導向, 有的要濾掉, 有的要放行等等的機制。在這個部份網路封包是 userspace 所碰觸不到的, 它只屬於核心對它的操作, 基於此點 IPTables 的作者也有考慮到, 所以他寫了一個可以將 kernelspace 的封包內容導入 user-space, 讓實作者可以另外寫出一套應用程式來對導出來的封包做其他的處理 (在原本的 IPtables 所沒有的處理), 這個介面 libIPq: 是一組函式, 在封包被導出, 然後被實作者的應用程式處理後, 會再將此封包在導回 ker-nelspace。再來, 在配上應用程式的決定 verdict (放行 NF_ACCEPT, 或丟掉 NF_DROP), 核心會依照這個應用程式的決定來處理又被導回來的網路封包。

在硬體方面, 使用了個人電腦四台, 都安裝 GNU/Linux 作業系統, 執行我們的實作的防火牆的 PC 上要用版本 2.4.18 的核心, 因為這個版本中的有著我們要使用的 Netfilter 模組。

而其他三台的 PC 上的核心版本就不太需要注意了, 其中一台執行網頁伺服器, 一台執行 syn generator 程式, 一台執行一般的網頁瀏覽動作, 像是: 使用 lynx 或 mozilla 等等的程式瀏覽網頁伺服器上的網頁。

而連接的方式為:

在執行防火牆的 PC 上安裝兩張網路卡 (一張對網頁伺服器的

PC, 一張接上 HUB); 在執行攻擊的 PC 上的網路卡與執行瀏覽動作的 PC 上的網路卡都連接到 HUB 上。

然後在執行防火牆的 PC 上開啟 Network Packets Forwarding 的功能, 讓有兩張網路卡的防火牆 PC 可以讓兩邊的封包可以流通這一條唯一通到網頁伺服器的網路連線 (不用 Network Address Translation 的功能)。

1.3 研究目標

藉由分析 DoS 之中的 SYN flood 攻擊模式, 來了解攻擊者發動 SYN flood 攻擊之中會造成的問題, 知道它如何地影響網頁伺服器的運作與正常使用者能成功連線的機會, 然後嘗試在其中找出可能的解決方法, 例如如何設定或擴充防火牆功能來防禦這類攻擊, 並建立出一個縮小的環境, 來實驗我們所做的防火牆擴充功能是否能減輕 SYN flood 攻擊對於網頁伺服器運作的影響的程度。

我們期望能做出既能夠阻擋這樣的惡劣攻擊, 又能使合法使用者在網頁伺服器正遭受 syn flooding 的攻擊時還能毫無網路延遲的感覺的上網, 瀏覽網頁伺服器上的網頁內容。

也就是說即使防火牆被攻擊到 "當機" 了, 也不會影響到網頁伺服器的安全。

第 2 章
原理簡介暨問題發生

2.1 何謂 DoS ?

DoS 為 Denial-of-Service 的縮寫, 是一個阻斷式攻擊方法的統稱。

通常攻擊者常會大量地對網頁伺服器發出主動連線請求, 而網頁伺服器所能做的事情就是一一接收所聆聽到的所有的連線請求, 並且一一為這些請求設成 SYN_RCVD 狀態, 然後期待這些要連線的 Clients 一一回應確認的訊息 (ack number) 回來, 讓它可以將這些已經是 SYN_RCVD 狀態的 entries 變成 ESTABLISHED 狀態, 然開始處理 Clients 的額外的需求。

但是實際上, 網頁伺服器不太可能收的到這些被攻擊者大量假造的 source IP address 所回應的 ack 訊息, 所導致的結果是:

> 網頁伺服器中可以處理的 entries 的數量是有限的, 然而, 攻擊發出既是量大又是假造的連線請求, 將大部分甚至至全部的可以理正常使用者連線的資源被消耗待盡, 只要是攻擊者還沒停止攻擊 (發送大量的假造連線請求), 很有可能這個網頁伺服器就一直停擺, 無法再處理其他任何的連線服務。

另外還有許多方法可以達成 DoS 的目的[4], 例如常見的 “Ping of Death” 產生超過所允許最大封包, 造成當機。以及我們將要討論的 “SYN flood”, 也是利用大量封包來迫使伺服器資源不足, 不能應付大量請求。在詳論 “SYN flood” 之前, 先在下一節說明 TCP/IP 的連線原理。

2.2 TCP/IP 連線原理

現有的一種已經成為標準的網路模型是 “開放式系統互連” (Open System Interconnection, OSI) 參考模型, 它是由國際標準組織 (ISO) 所產生的, 主要有七個的不同作用的層次。

TCP/IP 為 Transmission Control Protocol/Internet Protocol 的縮寫, 雖然它只被分成四個層次, 但是這四層的作用與 OSI 的七層的作用是相同的。它只有四層, 有一個顯著的優點是網路封包傳輸時的 overhead 相對的就減少一些 (因為多了一層的話, 就得多出一次的 header 紀錄的讀寫)。這四層由上到下分別為:

- 應用層
- 傳輸層
- 網路層
- 鏈結層

傳輸控制協定, 簡寫為 TCP 在網路分層的概念中屬於 “傳輸層”(Transport layer) 實作, 它負責很多功能, 如在網路環境中的行程(*Process*) 通訊, 流量控制, 以及封包傳送後回應的錯誤處理。

為了要能在遠端的兩台主機做溝通, 所以 TCP 也負責了如何建立連線的機制, 我們稱 TCP 是 Connection-Oreinted 的協定, 是由於:

1. 以 TCP 為基礎的傳輸層在送出資料以前, 會先送出訊息到目的地的傳輸層, 告訴它將要送出資料並且應該使用那一種應用層軟體接受此資料。然後, 它等待訊息的回應之後才開始傳送資料。在這個方式裏, 送出資料之前, TCP 傳輸層會先建立連接。

2. TCP 傳輸層是藉由回應與在傳送的合作, 來證實所有片段的訊息已經成功地傳送到目的地, 所以它又被稱為 “可靠的協定”。

而 TCP 建立連線的最初步驟, 我們稱之為 Three-way handshaking, 如下圖所示, 其中 Sender 是欲連線的一方, 有時稱為 Client; Receiver 是接受連

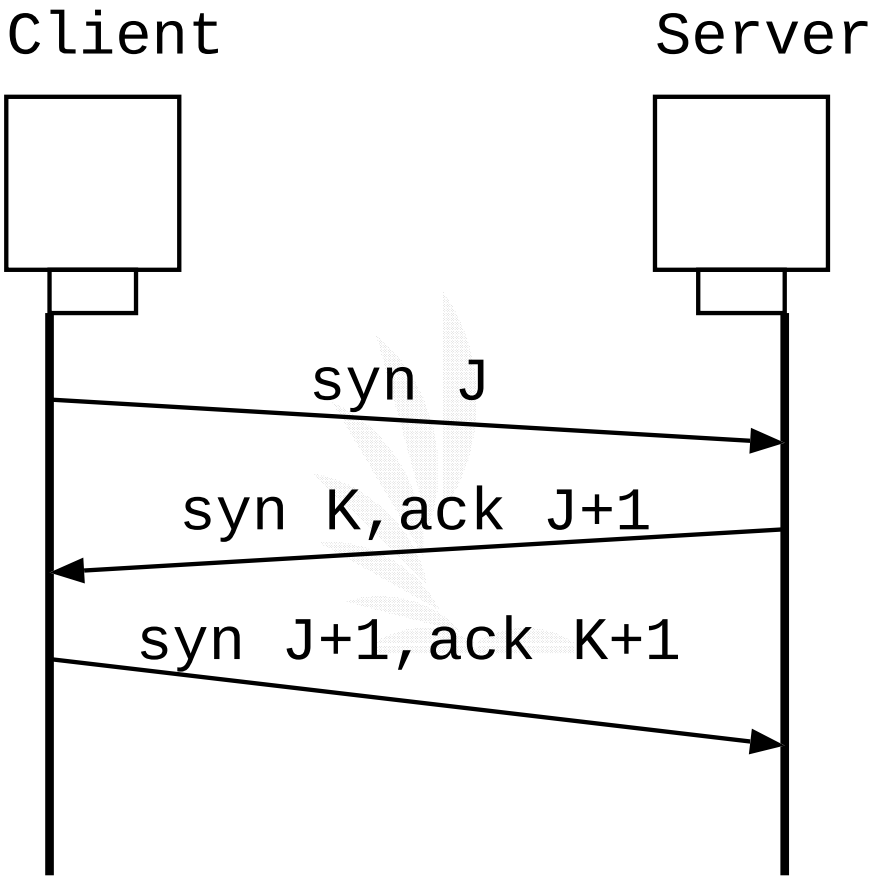


圖 2.1: Three-way handshaking

線的一方, 有時稱為 Server。

箭頭線段代表送出一 TCP 封包及送出方向, 括弧內為 TCP 封包中 Sequence number 和 Acknowledgment number 欄位的內容, 其內容是收發封包用的流水號。

正當 Sender 欲對 Rreceiver 發出連線請求, Sender 會往 Rreceiver 送出 TCP-flag 為 SYN 的封包, 其中的 Sequence number 為 J。當 Receiver 收到這 Sequence number 為 J 的 SYN 封包, 它就知道有使用者要和它做連線請求。

接著 Receiver 就送出 Sequence number 為 K 且 Acknowledgment number 為 $J+1$ 的 SYN+ACK 封包做回應, 其用意在於確認 Receiver 收到 Sender J 的封包。

正常情形下, 當 Sender 得到這個帶有 TCP-flag 為 SYN/ACK 的封包, 它即確定 Receiver 已準備好接受它的連線了。接著, Sender 送出一個 ACK number 為 $K+1$ 的封包, 通知 Receiver 可以開始建立與 Sender 的連線了, 形成 ESTABLISHED 的狀態。

接下來, 我們來查看封包的格式, 來說明若要完成以上動作, 封包內的資料會有什麼變化。

2.3 封包格式

以下是一個標準的 TCP 封包的標頭 (TCP Header) 格式。我們現只對上節提到的動作相關的欄位作說明: 我們之前知道 TCP 有行程間通訊的功能, 因此其中的 Source port address 欄位和 Destination port address 欄位就記錄這項資訊。

在 Sender 要送出第一個 TCP-flag 為 SYN 的封包之前, Sender 的作業系統就會決定連線的合法 Source port address, 而 Receiver 所提供接受連線的服務的 port address 是固定而且大家公認的, 因此第一個 SYN 封包的 Destination port address 欄位填的是 Receiver 接受連線的服務的 port address。以 Web Server 來說, 通常都會固定 port address 為 80。Sender 的作業系統也會在 Sequence number 欄位填入一值以決定此封包的流水號,

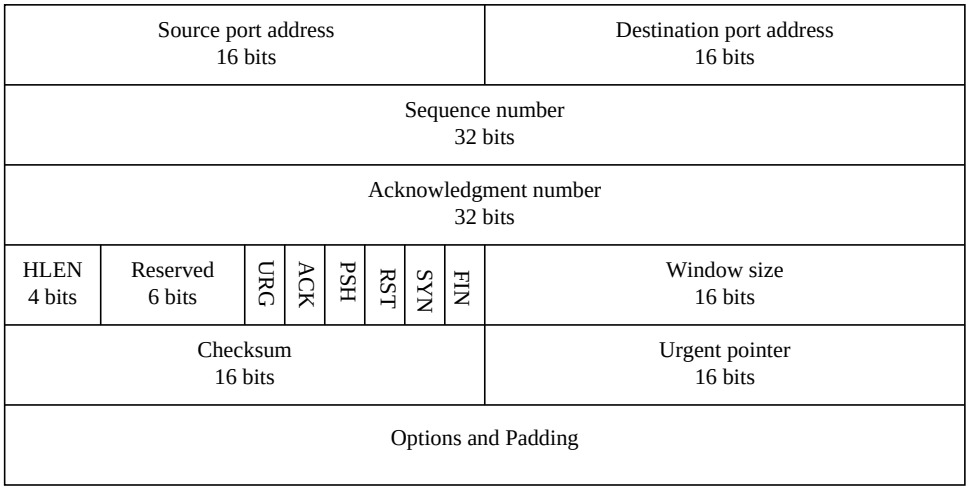


圖 2.2: TCP Header 的格式

然後其中有佔 1 bit 的 SYN 欄位會被設為 1。

對 Receiver 要回給 Sender 的 TCP-flag 為 SYN+ACK 封包, 當然 Source port address 欄位填入固定大家公認的 port address, Destination port address 欄位填入剛才 Sender 使用的 source port address。Receiver 的作業系統會在 Sequence number 欄位填入一值外, 還有在 Acknowledgment number 欄位填入剛得到 SYN 封包的 Sequence number + 1 。並且 SYN 和 ACK 欄位皆被設為 1。

同理, Sender 要給 Receiver 的 ACK 封包, 兩個 port address 欄位如剛才方法填入, Acknowledgment number 欄位填入剛收到的 SYN+ACK 封包的 Sequence number + 1 。ACK 欄位被設為 1。

2.4 SYN flood 攻擊原理

承上對 TCP 連線的簡單介紹, 我們可以來分析 SYN flood 的攻擊原理。基本上來說它利用上述的 Three-way handshaking 行為的設計未考慮到的地方, 來產生攻擊攻擊, 大量假造正常使用者的請求連線。

假設此種方式能保證連線成功, 但是 Server 在接受到 Client 的連線需求後, 還不知這筆連線會不會建立, 於是除了送出確認的 SYN+ACK 封

包外, 也必須要保留之前的 SYN 封包以茲核對用。在經過一段時間之後 ACK 封包沒被 Server 收到, 則 Server 才丟棄這些 SYN 封包。然而, 這是一種確認的過渡時期, 因此系統暫時保留這些 SYN 封包。當 Client 回傳 ACK 封包且 Server 收到後, 雙方連線建立, Server 只要記錄那一 Client 的那一 port 和它連線建立即可, 保留的封包就可丟棄。

但若有人存心送大量的 SYN 封包, 而且其中的來源位置資訊可以被假造, 當通通送到 Server 時, Server 不知來源被假造 (因為和平常正常的 SYN 封包沒什麼兩樣, 意味著大部分根本沒有那樣多的連線需求。), 所以 Server 保留了這些大量的 SYN 封包, 而且送出 SYN+ACK 去詢問要求的連線, 我們知道那是等不到回應的, 因為來源位置可能不存在或是位置存在但此 port 並沒有如此的連線或位置存在但從沒送出過 Sequence number 為 J 的封包。遭受攻擊的的 Server 為應付接踵而來的 SYN 封包, 耗費大量的資源保存它們, Server 也得等到 Timeout 的時間已到時才丟棄它們。因為保留而消耗的資源的速度永遠比丟棄的速度快, 而且 Web Server 的程式也無法管理 Three-way handshaking, 因為那是 TCP 的機制, 所以 Server 系統資源漸漸地耗光, 則網頁伺服器可能就發生當機。

以上就是 SYN flood 攻擊。我們將在下一章提出可能解決此種困境的辦法。

第 3 章
解決方案與實作

由以上所示的成因, 我們可以知道防禦 SYN flood 是有一些困難的:

- SYN 封包和一般無不同, 只是內部來源位置被假造, 而網頁伺服器還是要發送 SYN+ACK 確認。
- SYN 封包相當小, 沒有什麼資料, 只有 Header 示意連線需求而已, 所以一次可發出大量的 SYN 封包。
- 丟棄 SYN 封包的 Timeout 時間太長了。

但是我們期望能發現一些方法。

3.1 如何防禦

經由觀察, 我們不難發現因為 Server 的資源被此種攻擊耗光因而不足以應付大量連線請求, 而導致網頁伺服器當機。

所以增加軟硬體資源, 是解決空間資源不足的方法, 如擴增連線表, 或是增加更多的 Server。但是花費大筆金錢, 效果提升有待考慮, 因為 SYN flood 若四面八方而來, 再強力的硬體資源還是有限。若以時間問題考量, 縮短 Timeout 時間也是一個方法, 但是 SYN 封包大量而來, 一來即丟棄, 結果連正常連線都不能做了。或追蹤主事源頭 (但是來源位置被篡改), 追蹤不易。有人也提過利用 IP 封包的 TTL 欄位來推測和主事者的距離, 但是只能下一次的預測, 以及事情已發生才能用。

現在最流行的就是利用防火牆的技術來防禦此種攻擊, 大體上, 還分成兩種不一樣的模式: 其中一種是用欺騙 Web Server 的手法, 也是我們所採用方式。如圖 3.1 所示, 其中圖兩端線條意指 Client 和 Server 的時間進行, Server 是以 Web Server 為例, 中間長方塊是防火牆內部的運作; 兩端全為大寫字母是送出或收到所轉變而成的狀態; 有弧角的小方塊意指當時傳送封包的內容, 為了簡化起見和說明方便, 由左上至右下欄位分別是 Source IP Address, Destination IP Address, Source port Address, Destination port Address, 以及最後一欄為連線的封包屬性。

當 Web Server 面對要求的 SYN 封包時, 必須先透過防火牆, 防火牆也無法知道立即知道此 SYN 封包有無被造假, 因此

- 先行讓其通過。
- 或者保留一些封包內容。

接著 Web Server 回應 SYN+ACK 封包, 一定會先通過防火牆, 防火牆除了讓 SYN+ACK 封包依然通過外, 最重要的是利用此 SYN+ACK 封包, 來製作兩個 ACK 封包並直接發回一個給 Server。

因為防火牆和 Web Server 的距離很近, Web Server 可以很快地進入 ESTABLISHED 狀態。Web Server 也不用為了等待 ACK 封包而保留 SYN 封包。然後防火牆在將另一份由確認用的 ACK 封包保存起來, 以利事後比對用。

這就有如 Client 想要和 Web Server 溝通要經過防火牆, 當 Client 反向詢問是否要溝通, 而防火牆立即對 Web Server 肯定回答, 但是事實上 Client 可能還未有回應。若此 Client 是正常使用者請求連線的, 那麼防火牆將收到一個送往 Web Server 的 ACK 封包, 防火牆比對剛才保留的 ACK 中相關的封包, [1] 若部份指定欄位內容符合, 因連線已被建立隨即被丟棄 (NF_DROP)。

若此 Client 沒回應出 SYN+ACK 的封包, 就等於說 SYN 封包被造假, ACK 必然無法被收到, 如何解決? 防火牆會事先在收到 Web Server 傳回

[1]不大可能完全一樣, 因為系統對每個不同封包有不同的 padding(TCP header), 其中 TCP header 的一些欄位內容也不見得相同, 但圖中所示的欄位 (有含 IP header), Sequence number 和 Acknowledgment number 至少要相同。

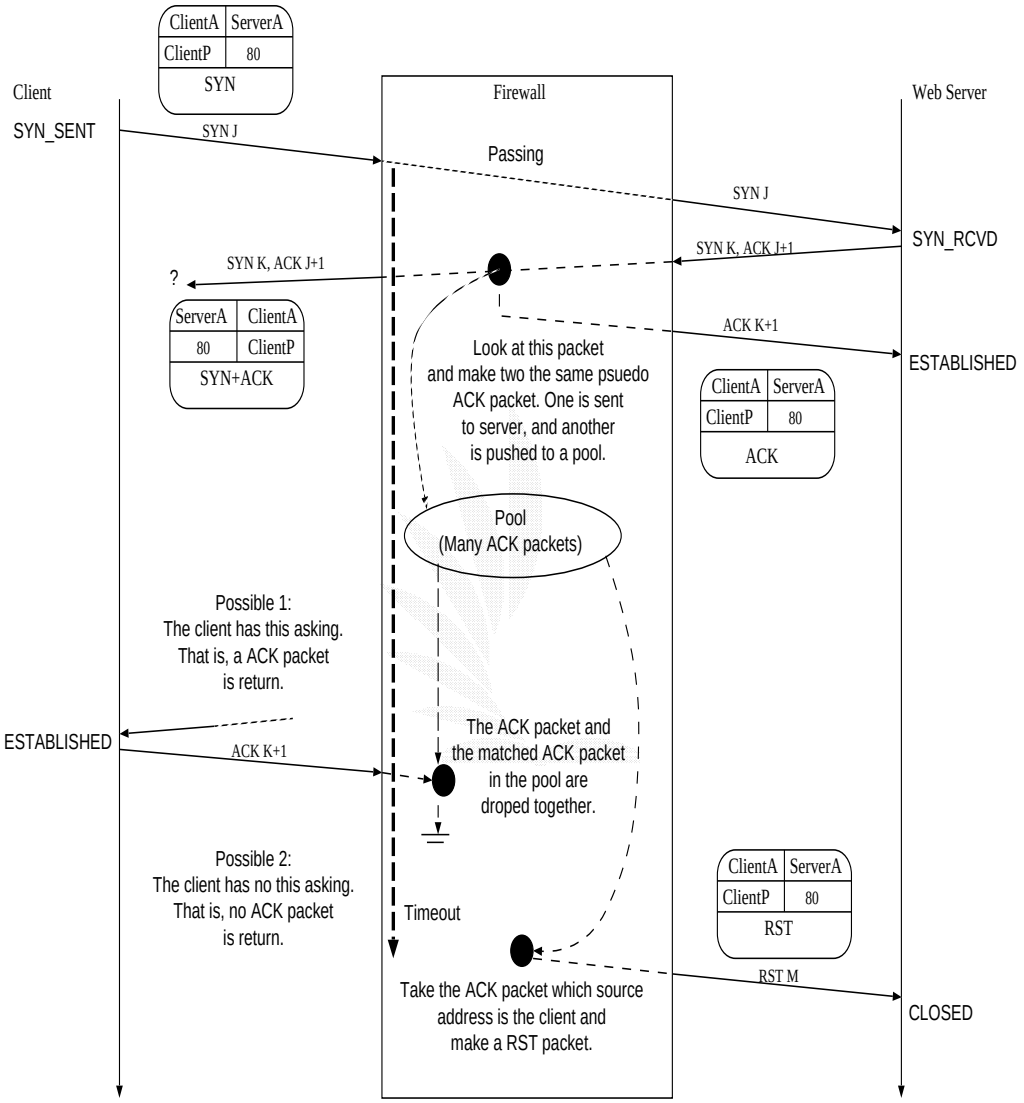

ClientA
ServerA
ClientP
80
SYN
Client
Firewall
Web Server
SYN_SENT
SYN J
Passing
SYN J
SYN_RCVD
SYN K, ACK J+1
SYN K, ACK J+1
?
ServerA
ClientA
80
ClientP
SYN+ACK
ACK K+1
ESTABLISHED
Look at this packet
and make two the same psuedo
ACK packet. One is sent
to server, and another
is pushed to a pool.
ClientA
ServerA
ClientP
80
ACK
Pool
(Many ACK packets)
Possible 1:
The client has this asking.
That is, a ACK packet
is return.
ESTABLISHED
ACK K+1
The ACK packet and
the matched ACK packet
in the pool are
droped together.
Possible 2:
The client has no this asking.
That is, no ACK packet
is return.
ClientA
ServerA
ClientP
80
RST
Timeout
RST M
Take the ACK packet which source
address is the client and
make a RST packet.
CLOSED

圖 3.1: 防火牆的防禦法

來的 SYN+ACK 封包後, 開始計時, 當隔一段時間防火牆一直沒收到來自 Client 的 ACK 封包, 就將保存下來的 ACK 封包清除, 表示時間等太久。由此可見, 控制這個時間間距的控制將是我們重要的關鍵。

另一使用防火牆的方法, 即是防火牆完全地代替 Server 做 Three-way handshaking, 當證實此 Client 存在且可信任的, 再次替 Server 進行 Three-way handshaking 的動作, 促使 Server 轉為 ESTABLISHED 的狀態。無論如何, 使用防火牆的方法, 都要有一個耐攻擊且速度快的機器做為防火牆, 才能面對排山倒海而來的 SYN 封包。

3.2 實作過程

在說明我們實作的程式演算法前, 有些前提要先完成。

先說明 libIPq API 的運作方式:

1. Initialization:
   呼叫函式 IPq_create_handle 來綁定 (bind) 一個要被 IP_queue 所使用的 Netlink socket 然後傳回一個 opaque context handle 讓接著下面的函式能夠呼叫它。

2. Setting the Queue Mode:
   呼叫函式 IPq_set_mode 來指定要 API 從核心的 Netfilter 模組處理的網路封包, 要導出什麼樣的資料, 這有兩種:

   - packet metadata
   - payload and metadata

   我們用的是連封包的整個內容都拿到的 payload and metadata 這個方式。
   在指定完成之後, 它還通知核心模組 IP_queue, 讓這個模組知道在 user-space 的應用程式已經準備好了, 要把封包內容給導引下來做處理。

3. Receiving Packets from the Queue:
   呼叫函式 IPq_read 讀進在核心模組 Netfilter 的 queuing message (一次次 queue 到 Netfilter 的網路封包) 並且將這個封包的內容裝到我們程式中的暫存位置。
   不過, 在它做讀取動作之前得先做確認, 到底讀出來的東西是封包的內容, 還是這個讀取的動作其實沒有成功。所以還得先由呼叫函式 IPq_message_type 來做檢驗, 而它傳回的結果如下:

   - Packet message:
     這個傳回值表示傳回的是封包內容, 然後交由呼叫函式 IPq_get_packet 來處理這個內容。
   - Error message:
     這個值表示這個讀取的動作失敗了, 呼叫函式 IPq_get_msgerr 會接手處理。

4. Issuing Verdicts on Packets:
   呼叫函式 IPq_set_verdict 發出對於現在正在處理的封包做出判定, 有兩種:

   - NF_ACCEPT
   - NF_DROP

   已經下了判定或者是傳回被修改過封包內容, 會使得核心模組 Netfilter 對於再導回來的封包做不一樣的處置。讓它繼續通過防火牆 (NF_ACCEPT) 還是把它丟棄 (NF_DROP)。

5. Error Handling:
   呼叫函式 IPq_perror 會印出在函式 IPq_errstr 所定義出的多種錯誤訊息 (在比對 IPq_errno 過後) 之一在 stderr 上。

6. Cleaning Up:
   呼叫函式 IPq_destroy_handle 來還原這個應用程式所佔用的記憶體空間。

以上從第一到第六項有著一定的次序, 一一完成不同的部份。

現在我們可以開始說明我們使用的演算法:

先假設 Clients 中包含著正常連線使用的, 與攻擊者大量假造的主動連線請求。

1. 在 Client 發出一道主動連線請求後, 必須經過防火牆內部 (我們所做的應用程式), 在其內部我們會選擇保留一些必要的資料下來到我們自訂的資料結構上。然後給這個封包的判定為 NF_ACCEPT, 讓它回去繼續通過防火牆內部的後半段, 直到網頁伺服器上, 請求伺服器給予連線。

2. 在網頁伺服器上, 它聆聽到有連線請求時, 會將這個請求連線的狀態紀錄變成 SYN_RCVD, 然後回應出第二道訊息到 Client 上, 發出的封包在途中也會經過防火牆內部。
   這時, 我們將封包導引到我們的應用程式上, 複製一些必要的資料在我們的自訂結構上。
   最重要的一個步驟是:

   > 在我們的應用程式取到這個 Three-Way handshake 中的第二道訊息的封包內容後, 得馬上使用 raw socket 將必要的資料填入, 然後模擬 Client 應該會回應的 (不論這個 Client 是否被假造) 帶有 TCP-flag ACK 的封包, 傳回網頁伺服器使得之前的連線狀態紀錄先成為 ESTABLISHED 再說。

   再來的步驟:

   > 直接讓這個封包的判定為 NF_ACCEPT, 讓它再回到核心模組 Netfilter 後繼續通行到 Client 上。然後啟動計時模組。

(時間持續計時中...)

3. 再回到 Client 上, 現在可能的情況有兩種:

   - Client 存在:
     它會回應從網頁伺服器來的 TCP-flag SYN/ACK 封包, 發出 TCP-flag ACK 的 Three-Way handshake 的第三手的確認封包。
   - Client 是假造的:
     假造的 Client 將不會回應任何的訊息。

   (時間繼續計時中...)

4. 現在看回防火牆上, 可能出現的結果也有兩種 (對應上面的兩種情況):

   - 等到了回應的訊息 (ACK 封包)。
     我們得將此訊息的封包判定為 NF_DROP (因為我們之前先替它做了確認的動作了)。
   - 沒有等到回應。
     這時也有一個重要的步驟要執行:

     > 假設這個等待回應的時間已經超過了之前的計時限制 (時間計時到達...), 那麼我們本應該要再次使用 raw socket 填入之前所蒐集的資料, 模擬 Client 發出帶有 TCP-flag FIN 的連線終止的封包給網頁伺服器, 之後進行四道的步驟, 使得連線終止。但是我們發現若改發出 TCP-flag RST 的重新連線的封包給網頁伺服器, 要求網頁伺服器中斷並等待新的連線要求, 只要一道步驟使網頁伺服器隨即中斷 ESTABLISHED 的狀態, 是較快速的作法。發完 TCP-flag RST 的封包後, 即將佔用的記憶體空間還給系統。至於真正的 Client 已在 Web Server 建立的連線[2] 會不會被中斷, 我們將在下一章介紹。

[2]當然也是防火牆代為連線的, 只是 Client 真的有回 ACK 封包而且防火牆也收到了。

5. 在防火牆裡面也要有資源的控制:
   我們常常使用到動態記憶體配置來做封包資料的轉移, 複製, 和產生。大部分的資源用來接受 Clients 傳來的連線請求, 若是連線的量突然暴增, 那這個防火牆的機器會有 "當機" 的可能性。

   雖然防火牆被及垮了, 但這也表示再也沒有連線請求會到達網頁伺服器的機器上面了, 然而, 網頁伺服器依然正常運作。

這樣完成了大致上我們實作的演算法。

另外, 有一些重要觀念 (關於 IPTables 的) 也必須介紹:

現看看圖 3.2 在 IPTables 採用則堆疊的方式過濾封包。

現在假設有一個封包到達, 可能的流程如下:

1. 當一個封包進入網路卡驅動程式, 它會現檢查 PREROUTING, 然後檢查它的目的 IP, 判斷是否要將此封包轉送出去:
   封包進入 $\longrightarrow$ PREROUTING $\longrightarrow$ FORWARD $\longrightarrow$ POSTROUTING $\longrightarrow$ 出去

2. 如果目的 IP 為本機, 接著跳到 INPUT 進行過濾; 如果目的 IP 需要轉送到其他機器, 跳到 FORWARD 進行過濾。

3. 如果此封包需要轉送到別台機器去, 則再檢查 POSTROUTING 。

4. 如果此封包是來自本機的, 則檢查 OUTPUT 以及 POSTROUTING 。

3.3 防火牆的資源控制

在上一節中, 我們知道防火牆需要保存大量的欲連線的 Client 的資訊, 當時間一到, 我們如何清除這些資訊, 不讓防火牆為了一直保存而很快地垮了。

在此, 我們考慮到當真正來自 Client 的回應訊息 (ACK) 必須要和我們保存的 Client 的資訊相符合, 而且尋找速度能快一些, 所以我們採用 binary search tree 來存放 Client 的資訊, 因為攻擊者為了使 Web Server 短時間好

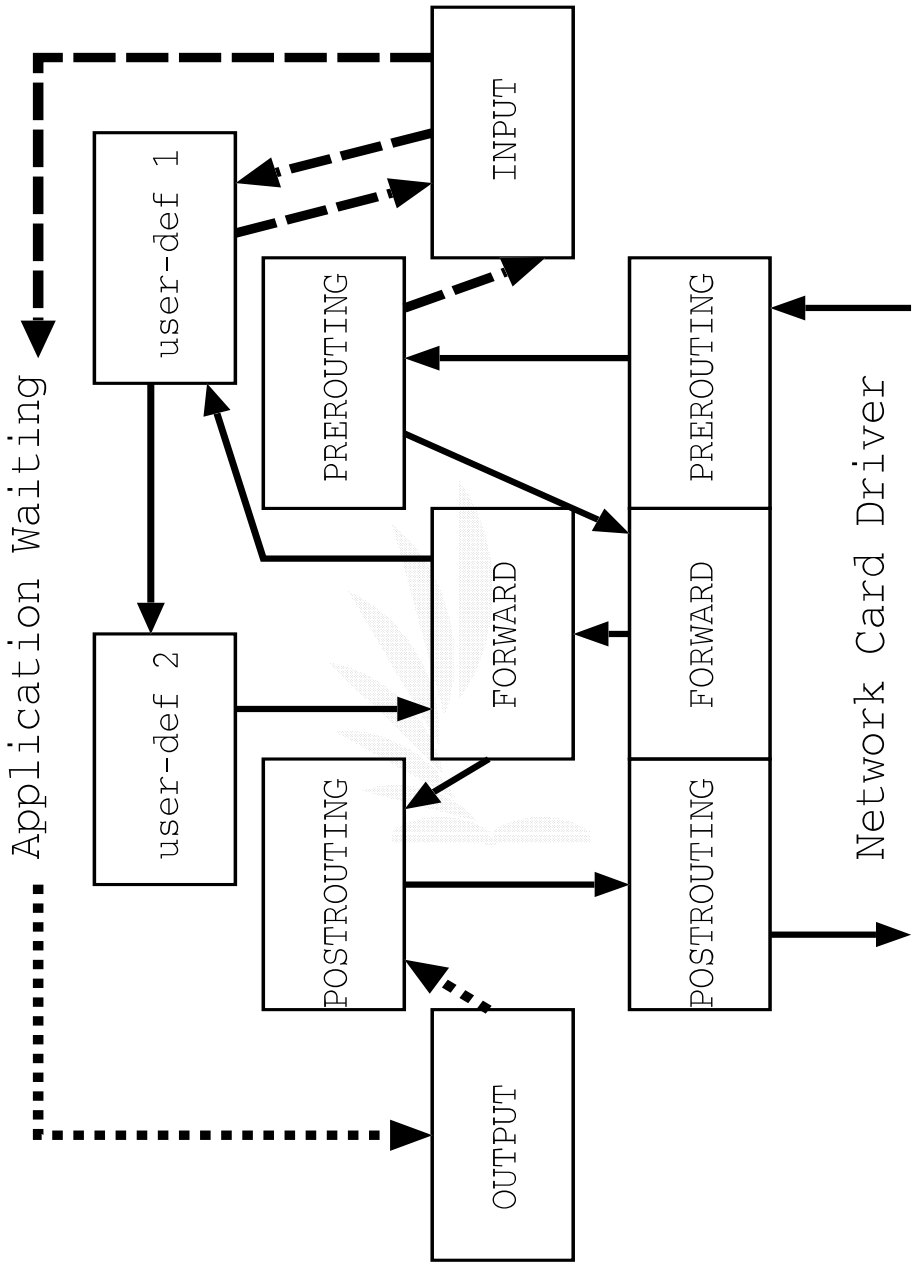


圖 3.2: IPTables

像有大量來自不同地方連線, 也為隱藏自己發封包的 IP 位置, 所以每一發出 SYN 封包的 Client IP 是隨機的, 甚至 Sequence number 和 Client port number 也是隨機的, 因而成為最差的 binary search tree (如同 linked list) 的可能比較小。那此 binary search tree 是以什麼資訊做為 key? 我們最後決定以 Client 的 IP 做 key, 若同一個 IP 有不同的 Client port number 就在此 tree node 開一個 linked list 存放, 理由如下:

1. 當我們尋找一個 Client 的連線資料時, 我們一般從 Client IP 開始尋找。
2. 同一 Client IP 會有不同的 port number 要求連線。
3. 當真正 Client 的 ACK 封包來時, 希望能快速找到相符合的資訊。因為攻擊者填入的 Client IP 是隨機的, 因此以 IP 做 key 可以造成不差的 binary search tree。而且 Client port number 也為隨機的, 所以造成 tree node (不同 Client IP) 的可能性比 linked list element (同 Client IP 但不同 Client port number) 還高。

所以我們資料結構如下所示:

```
struct acknowledgement
{
        char clear_it;
        unsigned short port;
        unsigned long seq_num;
        unsigned long ack_num;
        struct acknowledgement *next;
};
struct ackpool
{
        unsigned long IP;
        struct acknowledgement ack_ele;
        struct ackpool *left;
```

```
        struct ackpool *right;
};
struct ackpool *pool=NULL;  /* tree root */
```

如程式碼, struct ackpool 是每一 tree node 的內容, 以 IP 做 key, 指標 left 是較小的 IP 值的 tree node 位置, 指標 right 是較大的 IP 值的 tree node 位置, 並有 struct acknowledgement 來放某一封包的 Client port number, Sequence number, 和 Acknowledgement number, 如有相同 IP 則以 linked list 的結構存放。

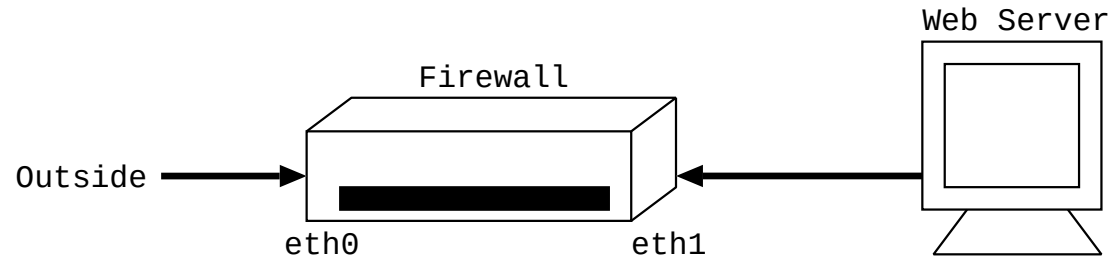
然而, struct acknowledgement 裡有一項 clear_it 欄位, 有何作用呢? 此 clear_it 欄位預設為 1, 表示時間一到要被清除, 並送出相對應的 RST 封包。若來自真正 Client 的 ACK 封包到來, 在我們保存的 Client 資訊中有相符合者, 就在 clear_it 欄位記為 0。當我們時間一到時, 本要清掉所有在防火牆的每一筆 Client 資訊。並伴隨送出 RST 封包。若 clear_it 欄位為 0 時, 雖然時間一到也要被清除, 但不送出 RST 封包。這樣一來, 除了不用為了只刪除我們保存的 Client 資訊的其中一筆而煩心[3], 而且真正的連線不會被中斷掉。

既然不用為只刪除一筆資訊而煩心, 因此我們實作時也可以用陣列來模擬 tree 的結構, 只是要考慮容許存放 tree node 的量。所以 struct ackpool 改以下列方式:

```
struct ackpool
{
        unsigned long IP;
        struct acknowledgement ack_ele;
};
struct ackpool *pool[MAX_PAC];
```

只是我們要看 MAX_PAC 的值為何, 必須考慮我們一次短時間內來自不同 IP 的封包最大量若有 n, 那最差的 binary search tree 如同 linked list 一般

[3]刪除一個 binary-search-tree node 是相當麻煩的, 請見[7]



Get SYN+ACK

```
iptables -A FORWARD -p tcp -i eth1 -o eth0 -s WebserverIP
--source-port 80 --tcp-flags SYN,RST,ACK SYN,ACK -j QUEUE
```

Get real ACK

```
iptables -A FORWARD -p tcp -i eth0 -o eth1 -d WebserverIP
--destination-port 80 --tcp-flags SYN,RST,ACK ACK -j QUEUE
```

圖 3.3: 防火牆的設定

(IP 值皆遞增或遞減), 所以 MAX_PAC 值我們設為 2^n, 為了保險起見, 設為 2^{n+1} 或 2^{n+2} 也不為過。此為保存 Client 資訊的另一法。

3.4 實作結果

現在, 我們以上述的方法和架構做如圖 3.3 的設置, 在 Outside 處設置兩臺電腦, 一臺扮演攻擊者, 另一臺扮演欲連入 Web Server 的正常 Client 端, 但因機器有限, 我們將攻擊者和正常 Client 同設置一臺電腦。然後中間設置一機器做防火牆, 其上的 eth0 網卡連接攻擊者和正常 Client 端電腦連接, 形成一網域。Web Server 則和防火牆的 eth1 做連接, 形成另一個網域。

一開始, 我們先將防火牆預設兩端網卡通行所有封包, 只開啟 IP forward 的功能:

```
# echo 1 > /proc/sys/net/IPv4/IP_forward
```

此時這個防火牆如同平常的 router 一般, 沒有任何過濾的功能。我們要看看在沒開啟防火牆功能之前, Web Server 能承受多大攻擊, 使得正常的 Client

無法連入。接著我們先使攻擊者以每秒 1 個 SYN 封包發給 Web Server, 在正常的 Client 端以網頁瀏覽器嘗試連入 Web Server, 結果還能正常連線。再使攻擊者以每秒 20 個 SYN 封包發給 Web Server, 連線效率稍差, 但還能連線。再使攻擊者以每秒 50 個 SYN 封包發給 Web Server, 結果相當難以連上, 只有偶爾可以。這表示在每秒 50 個 SYN 封包以上已可以阻斷正常連線的目的了。

現在我們啟動防火牆:

```
# insmod IP_tables
# insmod IPtable_filter
# insmod IP_queue
# IPtables -F
# IPtables -X
# IPtables -Z
```

並設置圖 3.3 的規則, 我們說明過濾封包的規則: 我們已前述防火牆要收集來自 Web Server 的 SYN+ACK 封包, 還有真正要回應給 Web Server 的來自 Client 的 ACK 封包。所以我們的防火牆在前述的 FORWARD chain 中只使來自網卡 eth1 的 TCP 封包, 其來源發出 IP 為 Web Server IP, 來源發出 port 為 80, 旗標是 SYN 和 ACK 同時為 1, 以上條件符合就送入前述的 IP_queue 中。並且防火牆在前述的 FORWARD chain 中只使來自網卡 eth0 的 TCP 封包, 其目標要收的 IP 為 Web Server IP, 目標 port 為 80, 旗標是 ACK 為 1, 以上條件符合就送入前述的 IP_queue 中。

設置好防火牆外, 並跑起我們以 libIPq 並跟據上二節所述的處理流程, 名為 mewIPq 的程式, 其中程式設定每三秒清除所有已存的 Client 資訊。因為在未有防火牆時, 每秒 50 個 SYN 封包已足以阻斷正常連線, 所以我們的攻擊者以每秒 50 個 SYN 封包發動攻擊。而我們在 Web Server 中以此指令:

```
$ netstat -a
```

發現有很多連線進來已是 ESTABLISHED 的狀態, 多執行幾次, 發現 ESTABLISHED 的量有時會減少。看來我們成功地將假的連線要求造成 ES-

TABLISHED 的狀態, 並且防火牆時間中斷一到要清除時有送出 RST 封包。接下來試試網頁瀏覽器嘗試連入 Web Server, 結果還能連入, 雖然會等一些時候, 但是不會有難以連入的情形。若每秒 100 個 SYN 封包始的攻擊, 一開始情形如同每秒 50 個的攻擊, 但是很快地 Web Server 端也出現很多 SYN_RECV 的狀態, 表示防火牆來不及發送 ACK 封包給 Web Server。不過一旦攻擊停止, Web Server 過一會兒能夠很快回到沒被攻擊狀態。

第 4 章
結論

由上面的實作結果, 我們能防禦的 SYN flood 的攻擊量並非很大, 但防禦還是有效果, 這也證實此法是可行的。

我們所使用的防禦方法並非是我們提出的, 我們最主要的參考資料為[1], 並且兩種用防火牆來防禦的方法也都有商業產品, 只是還不多而已。我們最主要的成果在於利用 GNU/Linux kernel 裡的防火牆機制, 再添加上一些功能, 使其能防禦 SYN flood。但無法達成和使用同樣防禦方法的商業產品一樣的好, 這可能因為受限於我們處理 SYN+ACK 速度來不及所致。

參考文獻

[1] http://www.usenix.org/events/sec01/invitedtalks/oliver.pdf

[2] Comer, Douglas E. *Internetworking with TCP/IP*, vol. 1, 4th ed. Upper Saddle River, NJ: Prentice-Hall, 2000

[3] Forouzan, Behrouz A. *TCP/IP Protocal Suite*, 2nd ed. Burr Ridge, NJ: McGraw-Hill, 2003

[4] http://www.adcom.com.tw/product/sonicw/dos_att.htm

[5] http://www.netfilter.org/

[6] http://www.spps.tp.edu.tw/documents/memo/IPtables/IPtables.htm

[7] Horowitz, Ellis., Sahni, Sartaj., and Anderson-Freed, Susan. *Data Structures in C*, 9th printing. Computer Science Press, 2001