分散式阻斷服務攻擊防禦系統研究與實作
- 專題報告書

作者：

壹、在 Windows 上實踐分散式阻斷服務攻擊的防禦機制：

徐曼妮、魏廷芸、張智諺、曾思維

貳、系統優化-heartbeat 完整化與慢速攻擊偵測：

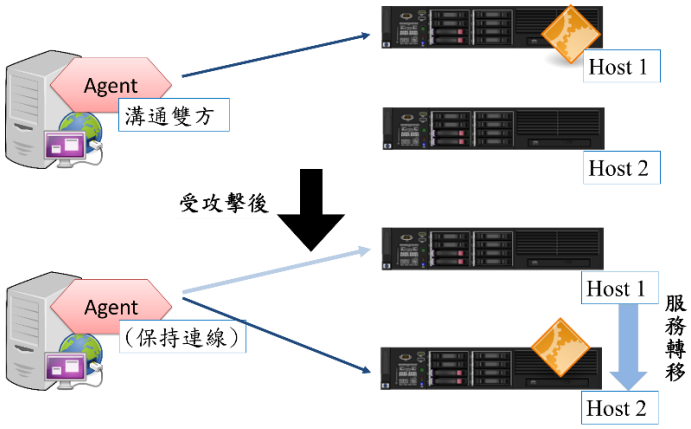
徐曼妮、魏廷芸

指導教授：許富皓

壹、在 Windows 上實踐分散式阻斷服務攻擊的防禦機制

一、摘要

阻斷服務攻擊主要是指耗盡特定網路服務的資源，而使得原本正常用戶無法使用此服務，分散式阻斷服務攻擊則是指有一個以上被侵占的電腦作為客戶端(一般稱為「殭屍」)對同一個特定服務進行阻斷服務攻擊。目前本實驗室已有在 Linux 上實現偵測分散式阻斷服務攻擊的防禦機制，主要是將一個代理安裝在使用 Linux 作業系統的客戶端，透過代理處理使用者與主機伺服器的連線過程，當主機伺服器透過偵測攻擊的程式，偵測到分散式阻斷服務攻擊時，服務會轉移至其他主機，同時通知代理，透過代理在不斷線的情況下改變連線過程與轉送封包的位址至新的主機伺服器，以保持服務的正常。本次研究希望可以透過設計，在 Windows 作業系統的實現該機制。



系統示意圖

二、研究動機與研究問題

當前科技越來越進步，人類對於科技的依賴性也越來越高，伴隨而來的是資訊安全的重要性。目前對於分散式阻斷服務攻擊，大多企業是採取增加硬體設備來解決此問題，而對於小型企業來說，唯一的處理方式就只能坐以待斃。並且一旦惡意人士使用高強度的分散式阻斷服務攻擊攻破硬體設備的限制，成功阻斷服務，則除了等待停止也別無他法，也可能因此讓被拒絕服務的使用者留下不好的印象。本實驗室藉由成功偵測分散式阻斷服務攻擊，並進行相對應的防禦機制，使電腦可以免於遭受分散式阻斷服務攻擊的危害，目前現有做法是將一個代理安裝在使用 Linux 作業系統的客戶端，利用代理進行防禦機制，但由於代理是專門設計安裝在使用 Linux 作業系統的客戶端，因此本項技術只能應用在 Linux 作業系統上，然而，實際上很多客戶端皆是使用 Windows 作業系統，因此希望能在目前現有的防禦機制基礎上，使其可以在 Windows 上運行。原本有關封包資訊的蒐集與處理方式只能應用在 Linux 作業系統而 Windows 作業系統不支援，因此

需要尋找並利用相關可支援 Windows 作業系統處理封包的工具來實行此項目標。

三、文獻回顧與探討

1. DDoS 防禦

DDoS 攻擊是世界上非常氾濫的一種惡意網路攻擊。發起攻擊者可以隱身幕後，使用殭屍網路(botnet)對目標進行攻擊，面對龐大數量由不同來源所發送的封包，至今，真正的防範阻斷服務攻擊仍非常困難。在進行此項研究計畫前，欲瞭解現行對於 DDoS 攻擊之防禦技術，因此透過文獻回顧與探討的方式，整理出了現今對於 DDoS 防禦的三種類型——攻擊偵測、攻擊追蹤、攻擊過濾。[1]

i. 攻擊偵測 [2]

偵測可分為基於特徵偵測(Signature-based)與異常偵測(Anomaly-based)兩大類型。特徵偵測主要為基於過去之攻擊行為，建立異常特徵資料庫。往後根據資料庫內的特徵對惡意行為進行偵測。基於異常偵測的入侵則是利用機器學習(Machine Learning)的方法，利用資料集進行訓練，建構出攻擊行為，當作辨識的依據。

ii. 攻擊追蹤 [3]

此種防禦類型大多需要透過路由器幫助追蹤，大致分為五種類型，分別是 ICMP 追蹤法法（ICMP traceback）、封包標記法（Packet Marking）、過濾法(Filtering)、封包記錄儲存法（Host-based Identification）、穿隧法（Tunneling）。

1. ICMP 追蹤法

指路由器在傳遞封包時，依照一定低機率取樣封包，對封包資訊做摘要處理，加上本身路由器資訊後，以 ICMP 訊息格式傳送到接收端，接收端可以收集帶有此訊息的封包資訊，知道此流量中的封包經過了哪些路由器，以重建出完整攻擊路徑。

2. 封包標記法

在封包經過路由器時，會將路由器資訊標記到封包欄位裡，待受害端收集夠多的封包，可依照資訊重建出攻擊路徑。又可依照標記機率分為機率封包標記（Probabilistic Packet Marking, PPM）和固定封包標記（Deterministic Packet Marking, DPM）。

3. 過濾法

在網路中佈下過濾器，比對封包的攻擊特徵。能加以判斷封包是否為攻擊者封包並加以丟棄。

4. 封包記錄儲存法

在路由器中儲存傳送過的封包摘要及相關資訊於記憶體中，受害端可請求防禦系統協助，系統下指令查詢封包的紀錄，找出

有哪些路由器記錄傳送過此攻擊封包，可找出攻擊封包經過的路徑，並由此重建出攻擊路徑。

5. 穿隧法

來自穿隧通訊協定（Tunneling Protocol）構想。它能將封包轉送、或提供一種安全的傳輸模式傳送資訊，抑或是在一個不相容的網路上傳送封包。將封包再封裝一層新的標頭，再對封包內容加密，而新的標頭內含有新的傳送端及目的端位址。目的端再藉由有接收解讀含穿隧新標頭的能力之目的路由器或目的端本身解讀封包內容。

iii. 攻擊過濾 [1]

攻擊過濾防禦方式有早期的 Packet Score 過濾方法、ALPi、Hop Count 過濾方法三種，及近期所提出之 CBF 方法。

1. Packet Score 過濾方法

以封包中的 TCP/IP header 屬性，利用貝氏定理來計算封包的分數，藉此判斷該封包是否來自正常管道。但此方法在計算過程中會消耗過多成本，使處理效率低落，因此不適合用來過濾大量的封包資料流量。

2. ALPi

Packet Score 過濾方法的延伸，它對於計算處理較 Packet Score 簡化，因此有助於提升執行的速度。

3. Hop Count 過濾方法

以封包的來源 IP 位址及 Time to Live(TTL)兩者之間的關係來執行過濾的動作。雖然 HCF 方法被證實有效且容易部屬，但此方法僅能針對偽造 IP 的攻擊事件且無法對付分散式攻擊。

4. 近期提出之 CBF 方法

透過雲端運算的方式，利用封包的 TCP/IP header 屬性作為關聯依據計算出信賴值/CBF 值，並以此來判斷封包是否異常。

2. DDoS 防禦方式與本研究之結合

上述三種 DDoS 防禦機制皆可達到一定功效，而本實驗室則是希望輔以上述幾種防禦方式，再配合目前正在研究之轉移代理而不斷線機制，進一步提升對 DDoS 的防禦能力。

在本研究室正開發的 linux 客戶端程式中需要更動 iptable 及下 shell 指令，這兩者若移至 windows 上執行，會有作業系統原因之異同，因此需另覓方法來加以處理。

3. 基於 pydivert 之客戶端程式改寫 [4][5][6]

在查找後發現 WinDivert[4]是可以應用在 Windows 上的封包處理工具。它可以攔截封包並轉送，也能對於封包做相對應的處理，而 PyDivert[5] [6] 將 WinDivert 整合至套件中，提供一個更簡單使用

WinDivert 的方式，因此可以嘗試利用 PyDivert 達成目標。

故本研究將會根據上述文獻中之防禦方式及適用於 windows 之封包處理工具加以整合、應用於研究中。

四、研究方法及步驟

本系統假設目標 server 做轉移時會通知使用者，且使用者使用 Windows 作業系統。

研究分成兩個步驟：

步驟一、一般情況，使用者正常與 server 建立連線

1. 將使用者連線的 IP 固定，再利用 PyDivert 導向動態更新或是已經設定好的 server IP
2. 運行後，程式將會與 server 建立連線

步驟二、server 受到攻擊，而使用者仍與 server 保持連線

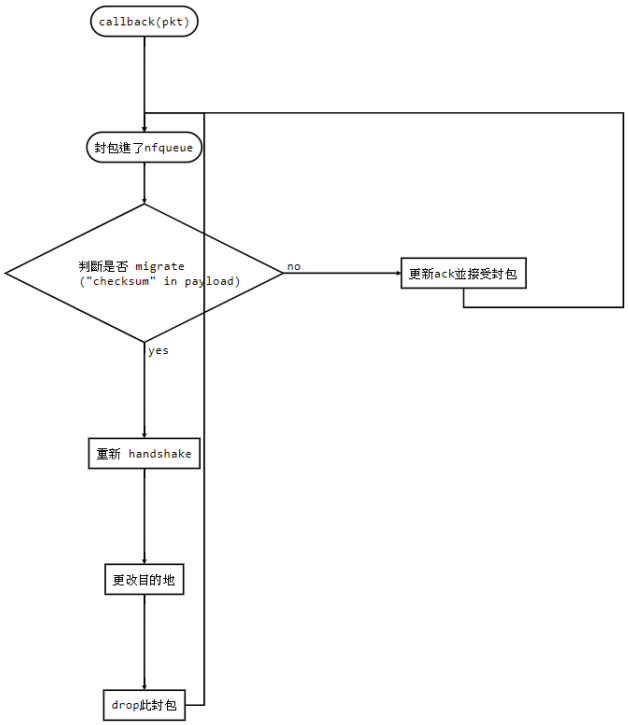
1. server 受到攻擊而進行轉移，並在轉移結束，server 會通知使用者其即將進行轉移
2. 使用者收到轉移通知，暫時取消部分封包的接收與發送(如含有 RST Flag 的 TCP 封包)，並更改 PyDivert 的設定，使固定的連線 IP 導向至新的 server IP
3. 與轉移後的 server 進行 handshake 建立連線
4. 成功建立連線後，移除限制封包接收與發送的設定，繼續運行

五、系統架構

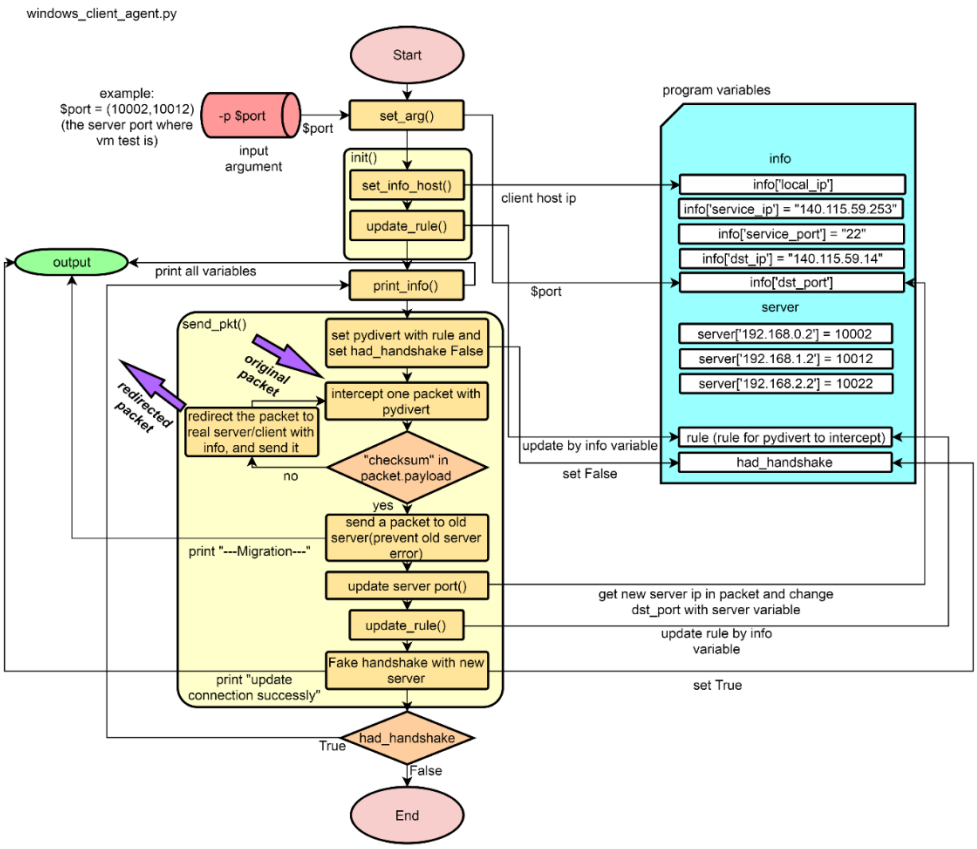
使用者連上已知 ip 後，透過 clientAgent 將 ip 修改為實際的伺服器 ip，達成伺服器遭 DDoS 攻擊轉移至另一主機後，使用者能在不需重新連線的情況下直接轉移到新主機上繼續使用服務。

六、程式架構

1. Linux version：

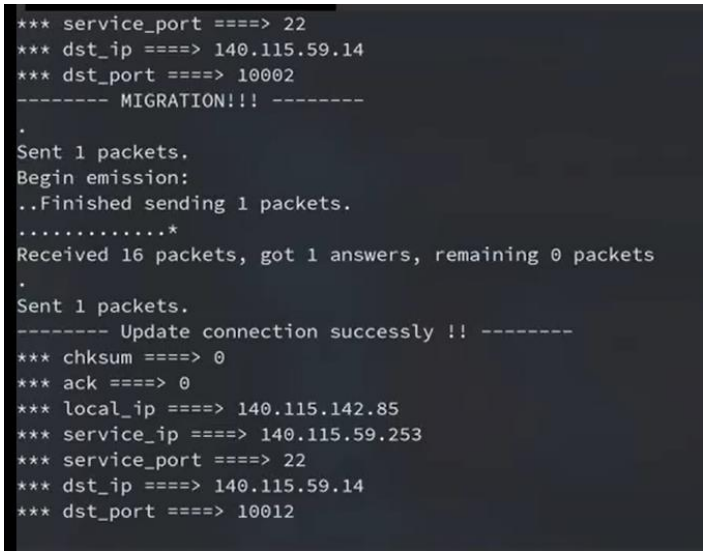


2. Windows version：

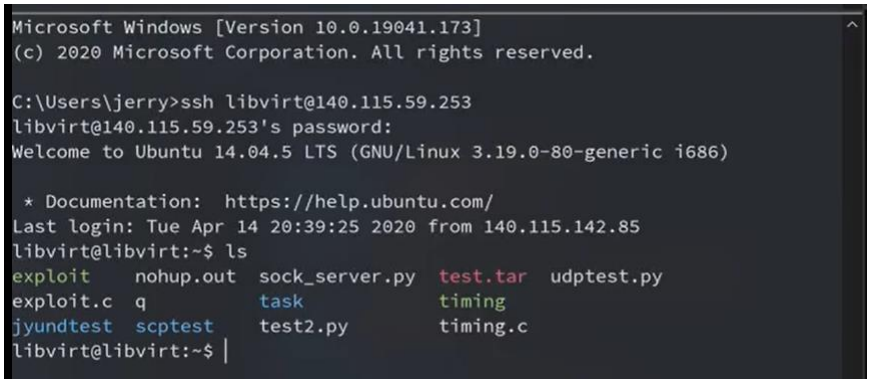


七、測試結果

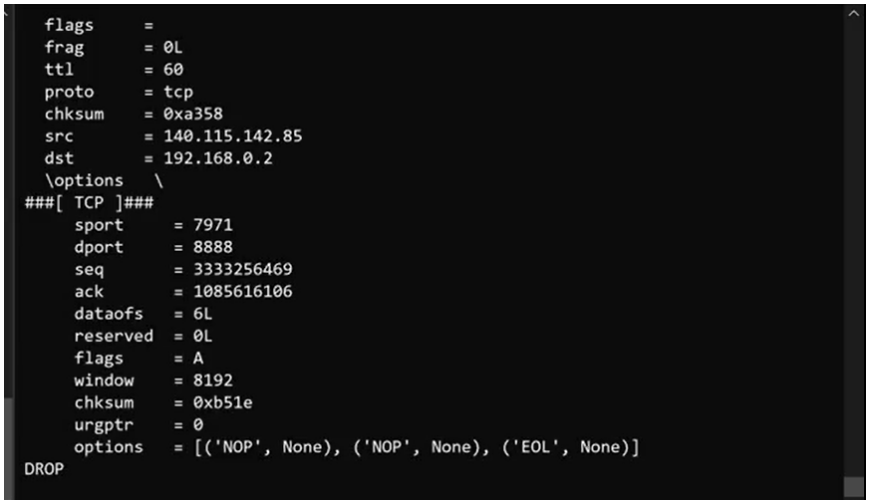
Windows client agent 顯示 migration 完成：



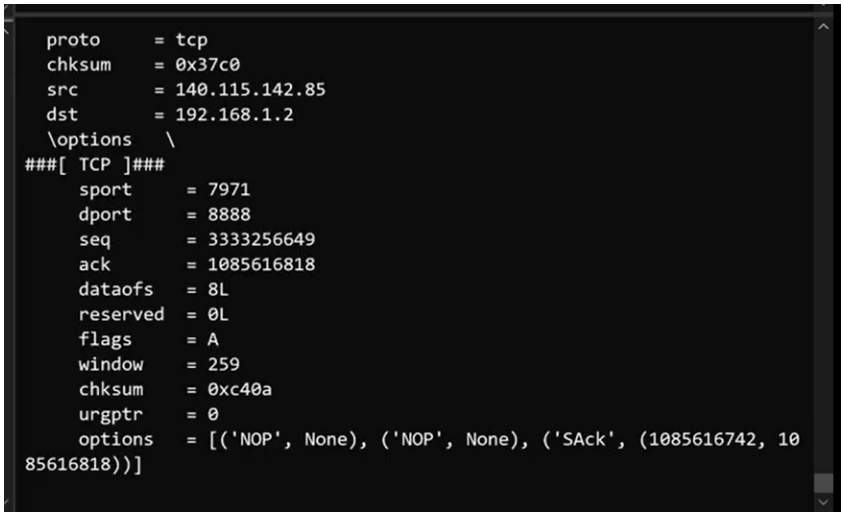
此時，在 libvirt 內下 ls 指令，成功獲得回應：



原本的 host 停止接收到封包：



新的 host 開始接收封包：



根據圖片結果顯示，在經過 live migration 後，windows client 仍與 server 保持連線。

八、參考文獻

1. 李毓展/2016/雲端環境下改良式 DDoS 防禦機制之研究
2. 謝昌融/2017/使用類神經網路和 Apache Spark 平台之 DDoS 偵測系統
3. 夏怡華/2010/利用異質性追蹤器防禦 DDoS 攻擊
4. WinDivert GitHub/ https://github.com/basil00/Divert
5. PyDivert Github/ https://github.com/ffalcinelli/pydivert
6. PyDivert 官方 document/ https://ffalcinelli.github.io/pydivert/
7. Basil(2020)。WinDivert 2.2: Windows Packet Divert。檢自：https://www.reqrypt.org/windivert-doc.html

貳、系統優化-heartbeat 完整化與慢速攻擊偵測

一、heartbeat 完整化

1. 研究目的

heartbeat 機制的目的主要是保持 server 與 client 間的連線，一般 HTTP server 機制下，client 連線在特定時間後會斷開，造成下次傳送封包需要重新建立連線，為了避免上述情形，heartbeat 程式會在小於該特定時間的時間間隔內預先傳送一個 heartbeat 封包(一般多為沒有 payload 而只有 header 的空封包)，來維持恆久連線。

在本實驗室現有的程式中，正需要 heartbeat 機制來維持 client 與 server 間的通訊，因此，本專題的目的正是將 heartbeat 機制完整化，並實裝至 server 程式 RPCAutoRedirect.py.HB 中。

2. 研究方向

heartbeat 機制是定時發送一個自定義的 structure（heartbeat packet），讓對方知道自己還活著，以確保連接的有效的機制。[1]

在 TCP 的機制中，本身就含有類似 heartbeat 的機制：SO_KEEPALIVE。

根據參考文獻[2]中，2.4 節中提到：” Because of the physical limits of these machines, they can only keep a finite number of connections in their memory. The most common and logical policy is to keep newest connections and to discard old and inactive connections first.”

說明了因為因為硬體設備的限制，無法保持所有連線。因此在必要的時候，會選擇踢掉一些不活躍的連線。

也因此文獻[2]說明若定期通過網路發送 packet，可以降低連線被剔除的風險：” Because the normal implementation puts the connection at the top of the list when one of its packets arrives and selects the last connection in the queue when it needs to eliminate an entry, periodically sending packets over the network is a good way to always be in a polar position with a minor risk of deletion.”

同理，本專題希望能根據 TCP 保持連線的機制，推廣且應用於保持實驗室環境中 server 端與 client 端 HTTP 連線上，通過在固定時間內由 server 端發送封包給 client 端的方式來保持兩者間的連線。

原先 RPCAutoRedirect.py.HB 程式中已有一份測試 heartbeat 機制的 function 程式碼，但該 function 並沒有真正達到 heartbeat 的作用。

原程式中有 set_heartbeat()、do_heartbeat()、heartbeat()三個函數，透過在 main function 呼叫 thread：set_heartbeat()以開始 heartbeat。

然而，在 python 中，多個 thread 會相互等待執行，並不會平行處理，所以這段程式碼只有在 main thread 接收到封包後，才會進行一次 heartbeat 封包的傳送，並不會每隔一段時間就發送封包給 client。[3][4]

在期中的修改後，我們已將 thread 改為 mutiprocessing，而下半學期新增的功能如下：

(1)client list 實時更新與刪除

(2)server 端跟 client 端間互相收發 heartbeat 封包

3. 程式架構

關於新增的兩點功能，程式的基本設計架構如下：

(1)client 的更新與刪除：

將原先的 function 全部移入 class 中，並透過在主程式 callback 中呼叫 set_heartbeat(self, cn)函式實時更新 client list，以解決更新 client 的問題。

(2)server 端跟 client 端的收發訊息問題：

我們讓 server 端每隔 3 秒對所有連線的 client 發送一個 heartbeat 封包，並在封包 header 裡的 option 欄位中加入 'heartbeat'選項以辨識其為 heartbeat 封包。當 client 收到封包，會根據 option 是否帶有'heartbeat'欄位而判斷是否為 heartbeat 封包，若有則對 server 端做出回應。

4. 程式碼設計說明

(1) server

a. class：HBtoClient

此 class 為 heartbeat 機制的主要 function。

內有 init、set_heartbeat、run 三個 funtion。

I. Init 初始化 heartbeat function 內的 client list。

II. set_heartbeat 用於實時更新 heartbeat function 內的 client list。

III. run 每隔 3 秒對所有 list 內的 client 發送封包進行 heartbeat

b. main：在 main 中建立 multiprocess hb_thread，並開始執行

(2) client

client 內透過 option 內有無'heartbeat'欄位，來判斷是否收到 heartbeat 封包。

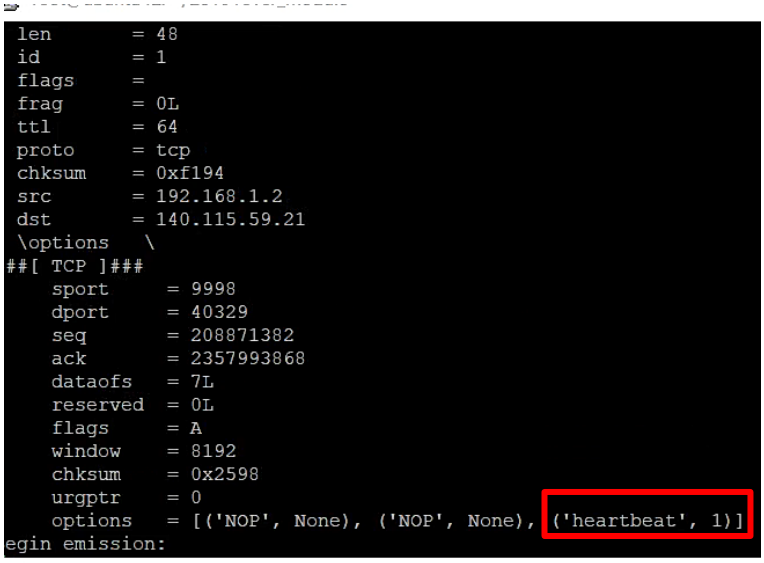
(3) /usr/share/pyshared/scapy/layers/inet.py

在 option 中新增 heartbeat 選項。

5. 測試結果

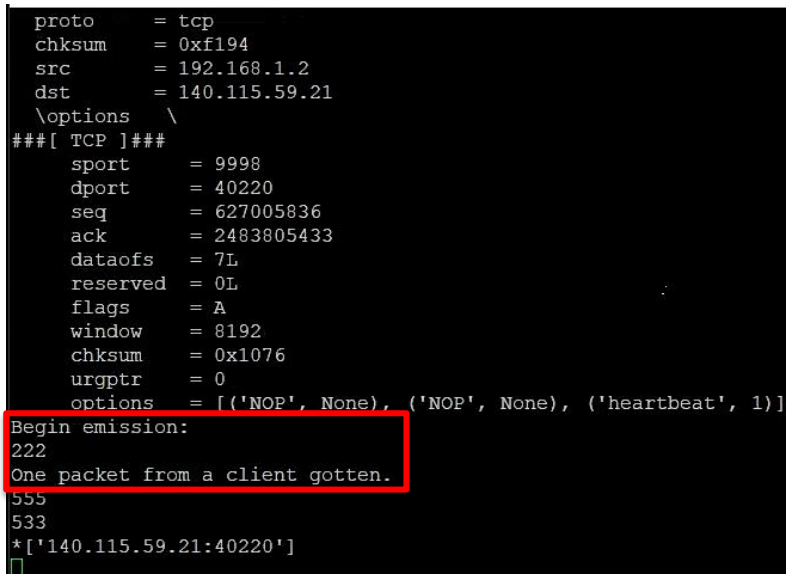
如下圖，經我們測試後 server 已可成功送出 heartbeat 封包，client 也有確實收到封包並回覆。

(1) server 送出 heartbeat 封包



(2)client 收到 heartbeat 封包，並送出 ACK

(3) server 收到回覆



7. 遭遇問題與未來規劃

(1) keepalive 時間到後 client 仍會斷線，無法起到 heartbeat 連線作用。

(2) RPC client list 有時會在網頁 hover 時出現大量同 ip 不同 port 的 client。

針對以上問題，我們提出以下嘗試：

(1) 使用 apache 測試

我們接下來規劃嘗試在 server 安裝 apache 並測試，觀察是否仍會受到 keepalive 的影響而斷線。

(2) 使用 payload 傳送

試著修復 payload 並以 payload 挾帶 heartbeat 傳送封包，測試當滑鼠 hover 的時候，是否仍會出現新的 port。

8. 參考資料

1. 心跳機制。檢自：https://baike.baidu.com/item/%E5%BF%83%E8%B7%B3%E6%9C%BA%E5%88%B6/2817701

2. TCP Keepalive HOWTO. http://www.tldp.org/HOWTO/html_single/TCP-Keepalive-HOWTO/

3. 一文看懂 Python 多程序與多執行緒程式設計。檢自：https://www.itread01.com/content/1549998912.html

4. ［筆記］python3 多執行緒與多核心平行計算。檢自：http://violin-tao.blogspot.com/2017/05/python3_26.html

二、慢速攻擊偵測

1. 研究目的

我們將慢速攻擊定義為 Flood 攻擊以外的所有攻擊。慢速攻擊不單純依靠大量封包來癱瘓伺服器，而是以其他方式來達成攻擊目的。相較於 DoS 而言，因為沒有短時間傳送大量封包的明顯特徵而不易被偵測。

本研究室現有的環境中已有程式碼：detectDoS.py 可以用來偵測 DoS/DDoS 攻擊。因此我們希望能新增偵測慢速攻擊的功能，使得系統能防禦的攻擊種類更加多元化。

2. 研究方向

「慢速攻擊是一種 DDoS 攻擊的變體版本。它通過向服務器發送正常請求，只不過請求的 header 或 body 內容特別長，發送速度特別慢，這樣每一個連線佔用的時間就會變得特別長，攻擊者會在短時間內持續不斷的對服務器進行請求，很快便會耗盡服務端的資源，從而令服務端拒絕服務。」[1]

常見的慢速攻擊類型有以下幾種[1][2]：

(1) Slow headers(slowris)：slowris 在傳送 header 時，故意不傳送結束符號"\r\n"，造成伺服器長時間等待 header 的傳送結束，使得伺服器資源被占用。

(2) Slow body：在 header 中標示 content length 後傳送 header，卡住 body 不傳送，在之後的時間以數秒 1byte 的速度緩慢傳送內容。

(3) Slow read：TCP 協議使用滑動窗口[2]來控制流量，client 會根據自己的狀況在 header 向 server 宣告窗口大小。攻擊者透過將窗口調的很小，使 server 誤以為 client 忙碌中，不得不持續向

client 送出接收資料的詢問請求，直到連線快逾時前才讀取 1byte 以保持與 server 的連線。

(4)其他：除以上三種，尚有 thc-ssl dos 等各類慢速攻擊存在。

上述幾種惡意連線達到一定數量後，皆會占用大量 server 資源，達成攻擊目的。

針對這些慢速攻擊方式，我們決定著眼於防禦第一種慢速攻擊：slowris 攻擊。

而面對慢速攻擊，主要有以下防禦策略[4]：

- Reject / drop connections with HTTP methods (verbs) not supported by the URL.
- Limit the header and message body to a minimal reasonable length.
- Set an absolute connection timeout.
- Define the minimum incoming data rate, and drop connections that are slower than that rate.

其中，我們決定參照文獻中設定 connection timeout 的防禦方式來進行防禦設計。

在我們的規劃下，設定了兩個 threshold，一個是對於 header 傳送時間的上限，另一個則是對於 header 傳輸速度偏慢的 client 數量的上限。

並使用兩種方式來偵測並防禦 slowris：

(1) 紀錄 header 傳送時間雖然偏慢，但尚未到達被判斷為超時的 client，當其達到一定數量後，判斷為遭受慢速攻擊。

(2) 切斷 header 傳送時間超時的連線

3. 攻擊偵測方式與想法

我們發現當傳送 Slow headers 攻擊時，並不會按照預期使收發封包有所延遲，更進一步觀察後發現，raw socket 會有一個類似 buffer 的機制，將沒有傳送完整的封包存放在 buffer 裡，等待封包傳送完成，也就是確實收到"\r\n\r\n"後，才處理此封包訊息。

目前，我們已成功修改 DetectDoS.py，使之在解析出 TCP 層的資訊後，得以繼續解析出 HTTP 層的封包內容[5]。因此，透過判斷封包結尾是否含有"\r\n\r\n "，即可判斷 header 是否傳送完成。

我們預期可以透過建立一個 list 來記錄 header 尚未完成完整傳送的 client，來管理 list 中 client 傳送 header 的時間，並針對超出時間的 client 進行後續的防禦處理。

不過由於：

- 實驗室中的 http 服務使用 nginx，而 nginx 本身預設 keepalive 為 75 秒，當 75 秒還未發送第二個"\r\n"便會自動切斷超時的連線，已可基本防禦慢速攻擊。

- nginx 本身對於慢速攻擊尚有其他防禦機制，當我們在模擬攻擊時，偶爾會被伺服器阻擋而無法再次與其建立連線。

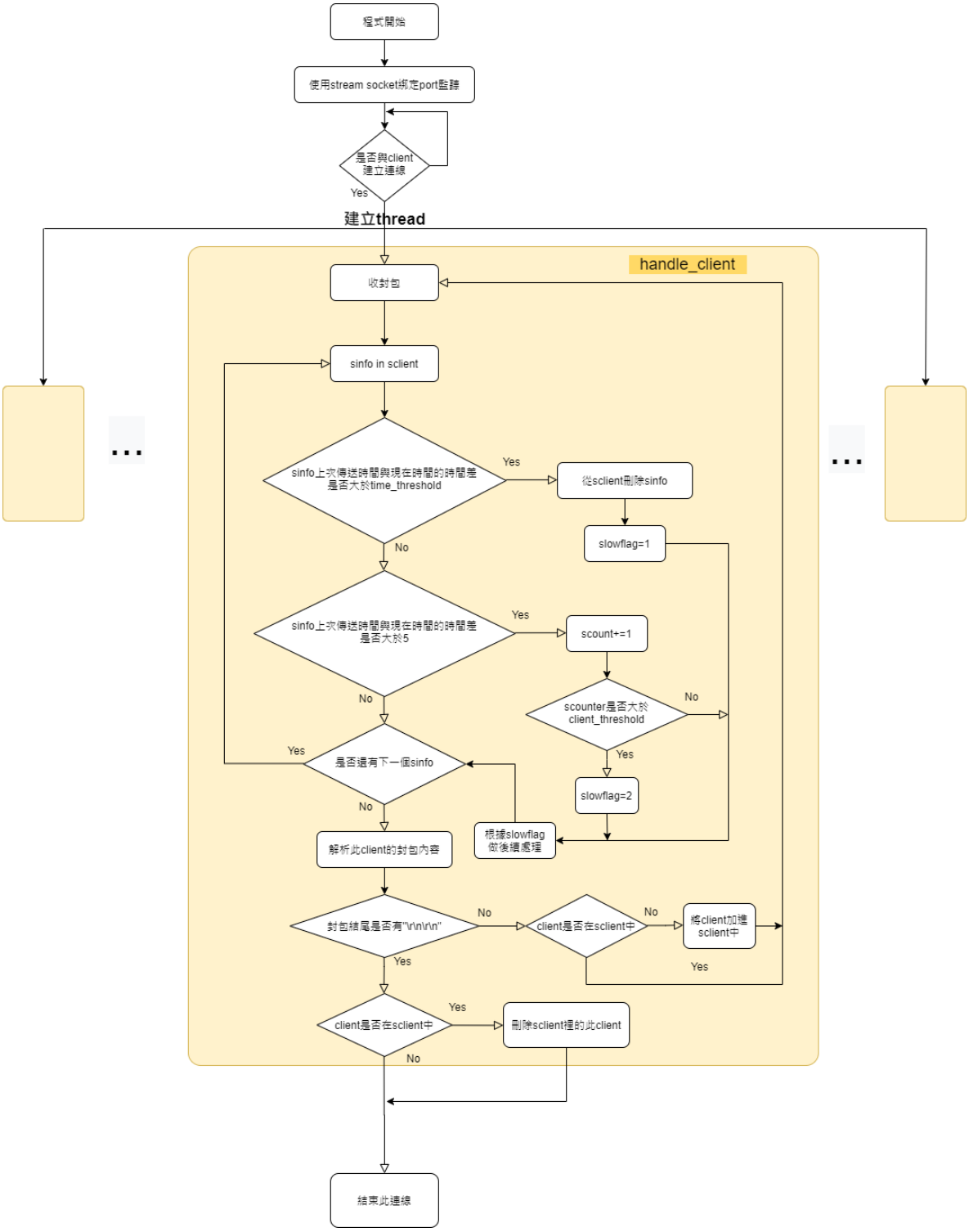
基於以上原因，為了方便在不受 niginx 防禦機制影響的情況下實作出防禦機制，我們決定設計一個 POC，利用 stream socket 取代 raw socket，使用類似的概念模擬 raw socket 收發封包時的狀況，讓 socket 在收到封包時，如果發現封包仍未傳送完畢，也就是結尾沒有兩個"\r\n"，便保持連線，持續接收封包等待其傳送完整。

4. POC 程式架構

我們利用期中時發現的解析封包方法[5]，建立一個字典檔 sclient 儲存尚未完整傳送"\r\n\r\n"的 client，記錄他們的 IP、PORT 以及傳送的時間，並在每次收到封包時計算 sclient key 裡的每個 client 上次傳送封包的時間與當下的時間差，根據以下兩個條件判斷是否為慢速攻擊：

(1) header 傳送時間偏慢但尚未超時的 client 達到一定數量後，判斷為遭受慢速攻擊

(2) header 傳送時間超時的連線，判斷為遭受慢速攻擊



5. POC 程式碼設計說明

(1) server

I. server 為 stream socket，會不停的接收 client 傳來的資訊直到收到完整的 header 為止。

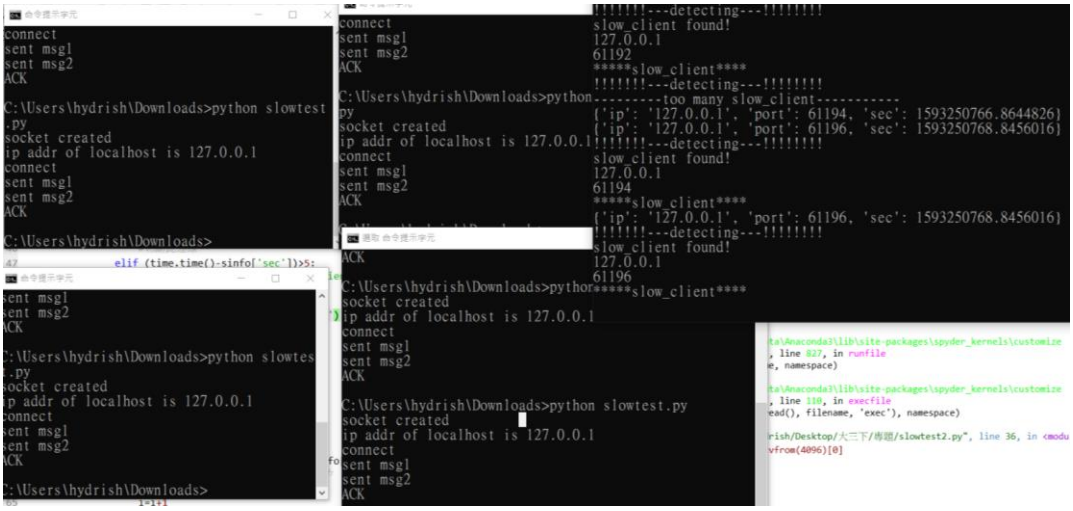
II. 透過時間以及 client 數的 threshold 來偵測慢速攻擊。

(2) attacker

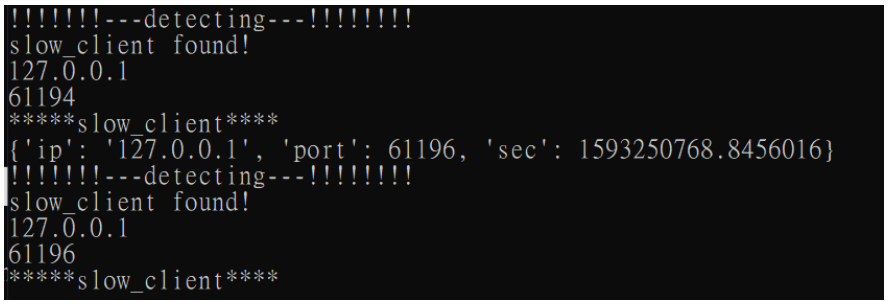
攻擊程式透過把 header 分為兩段傳送，兩者間隔超過 threshold 的時間限制(15 秒)來模擬 slowris 連線。

6. 測試結果

為方便測試，我們先將 threshold 調小，當 client 的 header 傳送間隔超過 time 的 threshold(10 秒)時，會被判斷為"slow client"，並切斷 client 的連線。



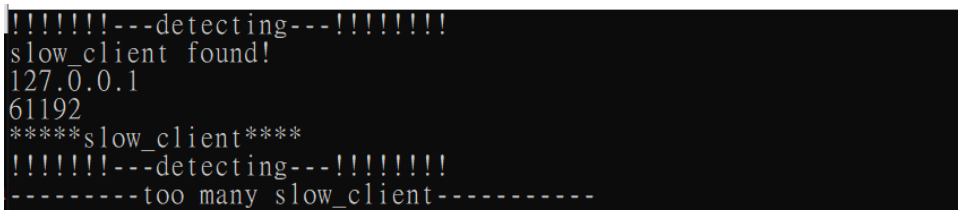
下圖為偵測到 slow client，並印出該 client 的 ip 及其他資訊：



而當傳送 header 時間間隔偏慢(5 秒)的 client 總數超過 client threshold(3 個)時，會判斷"too many slow client"，並觸發防禦 slowris 的 flag。

下圖為當慢速 client 超過 client 數 threshold 時，偵測 too many slow

client：



7. 未來規劃

- 將 stream socket POC 之架構移植至 raw socket 中：

我們目前將防禦 slowris 攻擊的偵測想法實作至 POC。之後希望可以把此想法推廣、整合至 detectDoS.py 內的 raw socket 中，使其有能力防禦 slowris 攻擊。

8. 參考資料

(1) 防 HTTP 慢速攻击的 nginx 安全配置。檢自：https://www.cnblogs.com/52py/p/10931089.html
(2) 信安之路。Slowhttptest 攻击原理。檢自：https://cloud.tencent.com/developer/article/1180216
(3) TCP-IP 詳解：滑動窗口（Sliding Window）。檢自：https://www.itread01.com/articles/1476615670.html