Android 上的殭屍網路攻擊偵測

學生：劉恩榜　　指導教授：曾文貴 博士

國立交通大學資訊科學與工程研究所碩士班

摘　　要

殭屍網路是現今網路上一大嚴重威脅，感染到殭屍病毒的電腦會不自覺地成為被控制的傀儡，不僅造成資料外洩、系統損壞、甚至成為重大網路攻擊的跳板。隨著智慧型手機的高度發展下，手機所提供的功能不只是傳統通電話或是傳簡訊，還包含了上網以及基本資料處理的功能，許多個人的資料、密碼還有相關私密的圖片、影片都會存放在手機裡，手機儼然成為一個小型 PC，因此近年來許多駭客不斷發展手機上的病毒、木馬、殭屍網路等惡意軟體，去竊取手機隱私資料、發送廣告簡訊和垃圾郵件等等。因此本論文提供一個針對 Android 手機上的殭屍網路偵測系統，在手機的流量中，根據殭屍網路的 group activity 特質和異常連線，於手機前端使用 Snort 這款強大的 IDS 做即時偵測，並安裝收集殭屍網路異常封包的 filter，將過濾好的封包上傳到後端的偵測中心，偵測中心從眾多手機的資料中，使用相似度演算法去判斷哪些手機是感染到殭屍病毒且正遭受惡意控制。

Mobile Botnet Detection on Android

student：En-Bang Liu　　Advisors：Dr. Wen-Guey Tzeng

Department of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

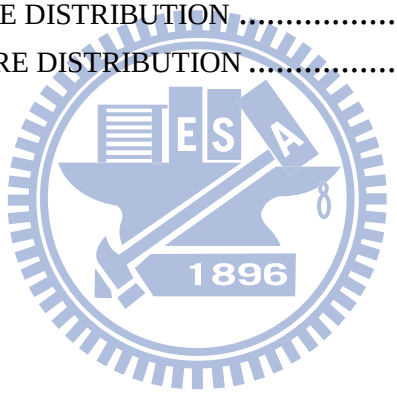
Botnets are now a serious threat to the internet . The infected computers will become a puppet (zombie computer), and controlled by attacker unconsciously . This impact not only resulted in leakage of information, system damage , but also make the computers become a springboard for a major network attacks .With the high development of smart phones , the phone is not just for calling or sending messages like before , also contains the ability of surfing the internet and basic processing data ; hence many personal data , passwords , private pictures/videos are stored in the phone. The smart phone has become a mini-PC. So in recent years , many hackers continue to develop viruses , Trojan Horses , bot virus and other malicious software on mobile phones to steal private information , send advertising messages and spam e-mails. Therefore in this paper , we provide a mobile Botnet detection system on Android. Based on the group activities model and abnormal connections metric , installing the Snort IDS to detect real time traffic and the Botnet packet filter to collect abnormal traffic in the frontend. Then upload the abnormal traffic to the detection center . After collecting traffic data from many mobile phones , the center uses similarity algorithms to determine which phone is infected with the bot virus and controlled by attacker.

致　　謝

首先感謝我的指導教授曾文貴老師，在我碩士班兩年間的學習過程中，帶領我深入密碼學的領域，老師積極認真的教學態度，嚴謹準確的做事準則，使我收益良多。另外，我要感謝我的口試委員，清大孫宏民教授，交大謝續平教授以及交大黃育綸教授，在論文上給我許多指導與建議，讓我的論文更加地完善。除此之外，我還要感謝實驗室學姐林孝盈學姐的許多幫助，在我論文面臨瓶頸感到無比焦慮時給予適時的指引和所需相關資訊，還有陳毅睿學長在論文修改過程的指導、沈宣佐學長在程式方面的技術支援以及其他碩士班同學不厭其煩的與我互相討論，給予彼此加油打氣的力量。最後，我要感謝我的家人，不論在精神上以及物質上都給我極大的支持，讓我在無後顧之憂的情況下可以順利完成學業。在此，謹以此文獻給所有我想要感謝的人，謝謝菩薩，謝謝老天爺。

目　錄

圖目錄



表目錄

一、 研究背景介紹

在 2010 年 12 月 29 號，由防毒軟體公司 Lookout Mobile Security 在中國發現一款有史以來宣稱最為複雜的 Android 木馬病毒，名稱為「Geinimi」，不但會竊取手機中的資料，也是第一個具有類似殭屍網路攻擊的 Android 惡意程式。當手機下載包含此病毒的應用程式後，木馬程式會以系統的背景模式執行，除了暗中與遠端伺服器建立連線來接收指令外，並定期將手機中的位置座標、裝置識別碼(IMEI)、SIM 卡的用戶識別碼(IMSI)或其他資料上傳到遠端伺服器，造成個人隱私的外洩，而且此病毒都會偽裝在一般正常的應用軟體中讓人防不勝防，由於 Anrdoid 手機不斷蓬勃發展下，為了避免更多人遭受到如此嚴重的危害，勢必要提供一個有效的偵測方法來防範手機上的殭屍攻擊。

1.1 Android

Android是以Linux為核心基礎所開發的開放原始碼手機作業系統，除了作業系統外，還包含手機應用軟體、中介軟體(Middleware)負責硬體與應用程式之間的溝通和Android SDK應用軟體開發套件，因此Android基本上就是一種嵌入式Linux系統加上重要的手機應用開發軟體。Android 系統架構主要有四個層級，如圖1.1所示，最底層採用2.6 Linux Kernel，負責硬體的驅動程式、網路、電源、系統安全以及記憶體管理功能；第二層是函式庫，由一些開放原始碼的函式庫所組成，有C函式庫Libc、OPENSSL、SQLite以及Webkit負責Android網頁流覽器的運作，另外還有OPENGL和SGL圖形與多媒體Library負責支援各種影音和圖形檔的播放。緊接著是和第二層Library並行的Android Runtime，提供Android特有的JAVA核心Library以及可轉換JAVA bytecode成 .dex檔案格式的Dalvik虛擬機器。第三層則是應用軟體架構，為所Android核心應用程式Framework API的總集合，可讓程式開發者方便取用這些常用的應用程式設計架構，以便增加應用軟體發展速度。在最上層的即是JAVA應用程式，包含通訊錄、瀏覽器、檔案管理、相機程式、MP3撥放器等等應用程式。

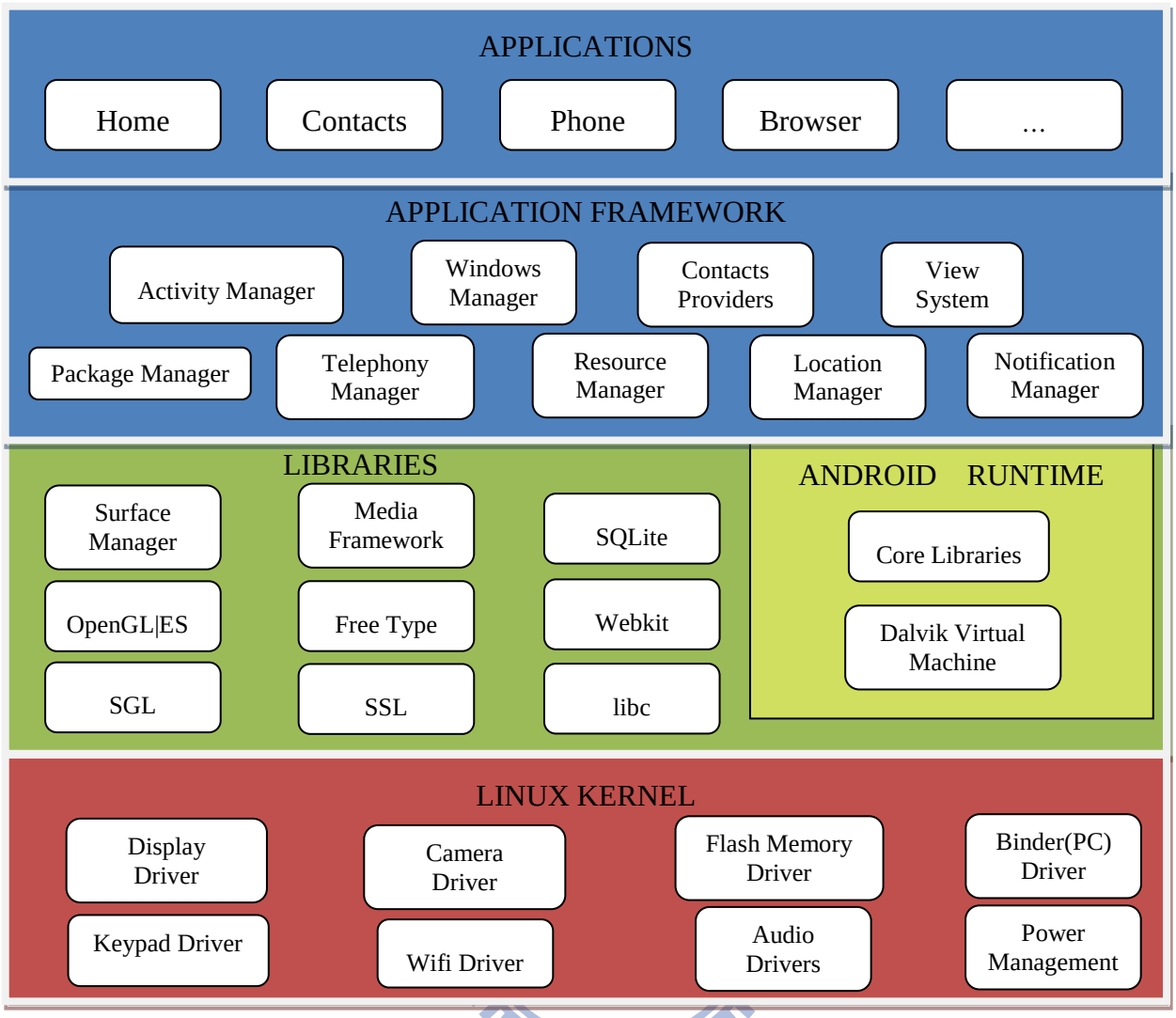


圖 1. 1

目前限定所有應用程式都必須以 JAVA 語言來撰寫，再以 Dalvik Virtual Machine 來轉換成 DX bytecode，因此程式開發和 JAVA ME 類似，應用程式的介面部分則是由 XML 程式來設計。簡單來說，即是系統由 LINUX 來執行，應用程式則透過 JAVA 語言來開發以及執行。這裡我們使用之前學長提出移植 Snort 的方法，以 Cross-Compiler 將 Snort 編譯成可在 Android 執行的版本，將其放入 Android file system 再利用前端程式去呼叫執行，本篇論文也利用此方法移植 Argus (Audit Record Generation Utilization System)網路知名的封包過濾軟體來收集手機上的 Botnet 封包進行之後的分析。

1.2 殭屍網路

Botnet 稱為殭屍網路(Zombie Network)或機器人網路(Robot Network)，主要是一群感染惡意軟體(bot code)後遭受攻擊者(botmaster)所控制的電腦主機形成的網路，攻

擊者除了竊取電腦上的私密資料外，並透過傳送指令使得受害者成為發動網路攻擊的跳板，其中的bot程式結合各種惡意程式的特性，如木馬(Trojan Horse)、病毒(Virus)、蠕蟲(Worm)及間諜軟體(Spyware)等等惡意攻擊特性，經由攻擊者不斷修改Bot程式，快速發展各種新穎殭屍網路攻擊技術以致難以完全及時防範與抑制，成為近年最重大的威脅。

為了使殭屍網路所控制的受害電腦能夠達到最大量，殭屍網路攻擊者常透過作業系統、瀏覽器和部分軟體的一些漏洞來進行攻擊，同時利用使用者資訊安全上的不足的來讓使用者誤入殭屍網路的陷阱，如利用系統更新軟體，使用者會認為這是系統正常的程序，因此殊不知背地裡執行發送spam mail，這類的電子郵件通常都會加上吸引人的標題或話題來騙取使用者點擊，當使用者被惡意導引到其他含有惡意程式的網址時，常常會被殖入感染惡意軟體，殭屍網路會藉由以上方式散播，擴充現有的殭屍網路規模。

典型的殭屍網路是中央集權架構，常見的通訊模式有IRC Based [13]、P2P Based [15]、HTTP Based [14]等三種方式，當中最具代表性的為IRC botnet，攻擊者(Botmaster)使用IRC協定，透過控制與命令伺服器(Control and Command, C&C server) 來控制受感染的電腦主機。Botmaster預先建置IRC Server作為 C&C Server使用，當 Bot成功連線至 IRC Server時，會加入 Botmaster預先設置好的聊天頻道，並等待 Botmaster下達命令，等到 Bot執行完命令將結果回傳到 IRC Server的聊天頻道，優點為連線速度快，可與攻擊者直接溝通，缺點為一但C&C伺服器被發現，殭屍網路即會瓦解。但也因為架構簡單，早期的殭屍網路都以此架構為主。

整體的Botnet life cycle 如圖1.1所示。在❶❷階段說明一般電腦會經由瀏覽惡意或釣魚網站感染到bot code，❸❹❺階段則是受感染的電腦成為受害者主機(victim或bot主機)後，會建立一條後門連線至C&C伺服器，此時攻擊者透過中介者C&C伺服器來下達控制指令以獲取電腦的內部資料，並進行維護、定期更新病毒版本來獲取更高權限。當擁有一定數量的殭屍網路，最後階段❻即可發動大規模的網路攻擊行為，例如同時對單一伺服器發動阻斷式服務(Distributed Denial-of-Service ，DDoS) [16]攻擊，大量寄發廣告郵件或帶有惡意連結的垃圾郵件。

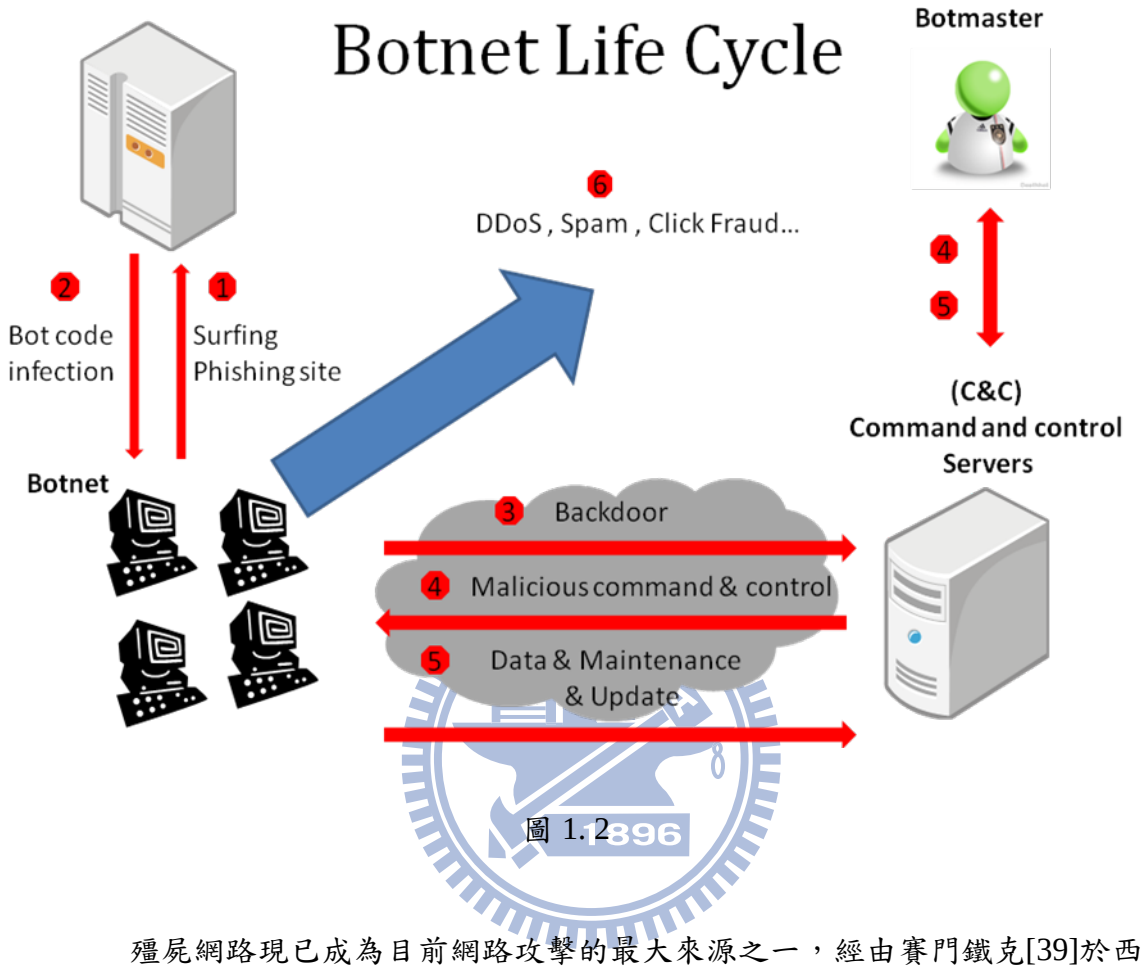


圖 1. 2

殭屍網路現已成為目前網路攻擊的最大來源之一，經由賽門鐵克[39]於西元 2011 年的統計數據，台灣在全球殭屍網路的感染排名第三，並藉由教育學術網路-系統安全與惡意程式偵測技術研發建置計畫提供從 2010/1/1~2010/12/31 全球感染殭屍網路的分布圖(圖 1.3)所示，歐亞洲地區感染的情況最為嚴重。

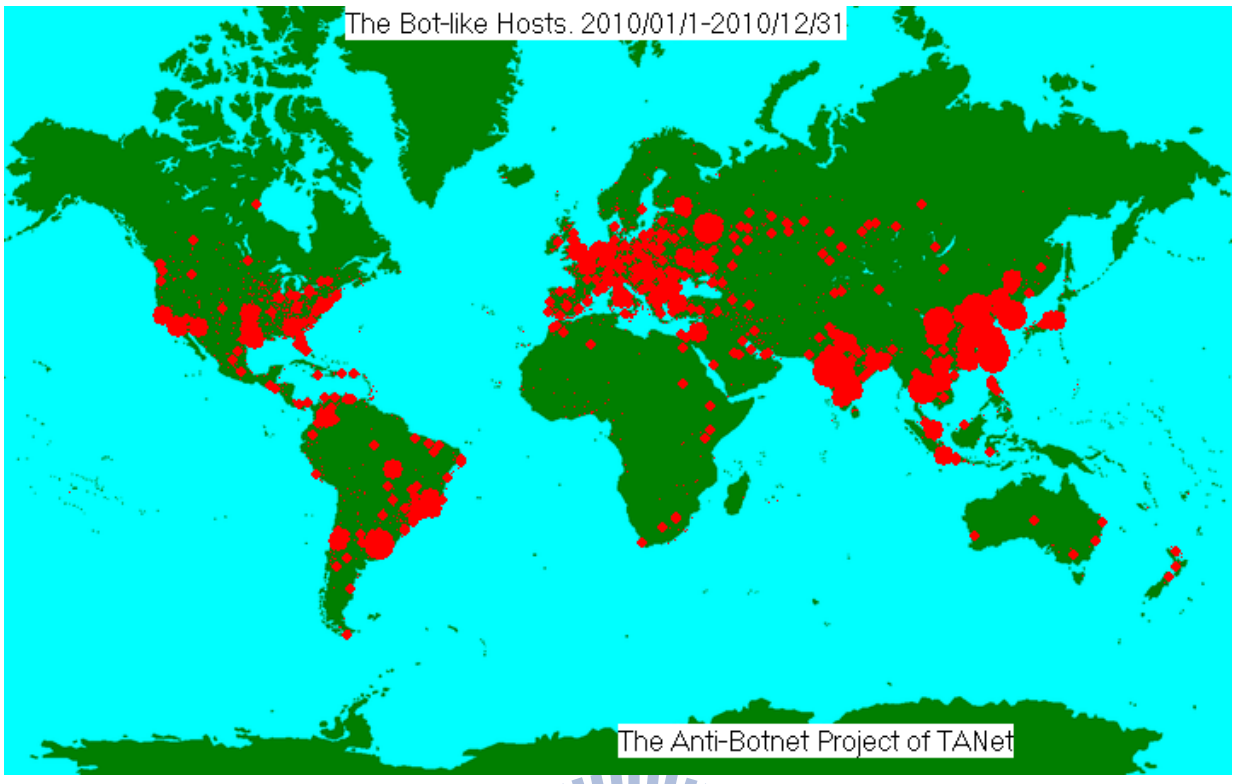


圖 1. 3

1.2.1 架構與分類

目前殭屍網路架構大致可以區分為四種模式: 中央集權模式、多重伺服器模式、階層式模式、隨機連線模式。

(1) 中央集權模式：

具有一台攻擊者拿來與受害主機相互溝通的C&C伺服器，具代表性的是IRC botnet，IRC Botnet使用 IRC協定來進行溝通，替Botmaster預先建置IRC Server作為 C&C Server使用，當 Bot成功連線至 IRC Server時，會加入 Botmaster預先設置好的聊天頻道，並等待 Botmaster下達命令，等到 Bot執行完命令將結果回傳到 IRC Server的聊天頻道，優點為連線速度快，可與攻擊者直接溝通，缺點為一但C&C伺服器被發現，殭屍網路即會瓦解。但也因為架構簡單，早期的殭屍網路都以此架構為主。

(2) 多重伺服器模式：

相對於IRC botnet，多重伺服器模式同時具有多台C&C伺服器，優勢在於不會因為單一節點伺服器被破壞而整個殭屍網路被關掉，還是能透過其

他C&C伺服器維持整個殭屍網路的運作，但缺點為布置成本較高，需先規劃好整個殭屍網路架構，要有多台C&C伺服器主機位置備用。

(3) 階層式模式：

階層式架構的殭屍網路擁有多層次的控制主機架構，攻擊者不直接使用單台或多台C&C伺服器對受害主機下達指令，而是將其中一部分受害主機當成下指令的跳板C&C伺服器，而真正的駭客躲藏於更上一層，多了一層的保護，使階層式架構的殭屍網路就算當跳板的受害主機被發現，也無法真正取得C&C伺服器主機位置。

(4) 隨機連線模式：

隨機連線模式的殭屍網路架構不具有C&C伺服器，每一個受害主機分別連線至一個小型的殭屍網路，具有高度的回覆特性，因為每個受害主機同時有很多個溝通管道與其他受害主機聯絡，而缺少C&C也使訊息傳遞的延遲率較高，單一節點不一定能夠正確傳達訊息給其他受害主機，單一節點被發現時，也不會影響到整體殭屍網路，是最難瓦解的殭屍網路架構。

1.3 入侵偵測系統

對於網路上層出不窮的攻擊，防範的方法也有很多種，其中一種最有效也最多人使用的方式，就是使用 Intrusion Detection System(IDS)，也就是所謂的入侵偵測系統，利用每種攻擊的特徵，對系統以及網路中的各項資料去進行比對，IDS 從偵測方式上可分為兩種，分別為 Anomaly-based detection 以及 Signature-based detection，前者是利用監控作業系統中的各項資源，去找出資源異常使用的地方，若有某項資源的使用量超出一般情況很多，則判定為異常，可能是受到攻擊或是使用者有異常行為，便會發出警告給系統管理員；後者則是利用字串比對的方式，去對要檢查的內容進行比對，以判斷是否含有會對系統造成危害之字串，從定義上防毒軟體可以視為被包含在這個分類裡面。從偵測目標上則可分為 Host-based IDS(HIDS)以及 Network-based IDS(NIDS)，顧名思義，前者是對主機去進行監控，而後者則是對網路封包去進行監控。其中 NIDS 大多都使用 Signature-based 的偵測方式。

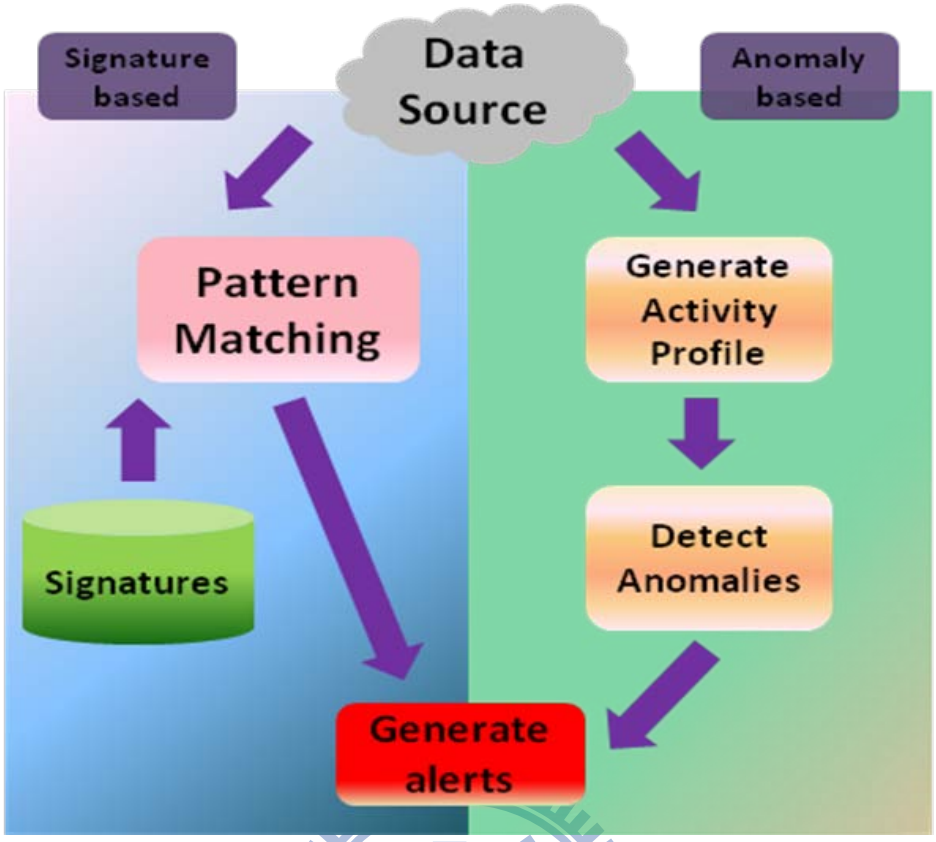


圖 1. 4
IDS 的偵測架構圖

這裡我們引用 Porting Snort on Android[17]這篇論文提出的移植技術來建置 Android 上的入侵偵測系統，Snort 是網路上很受歡迎的一款 IDS，不但功能強大而且還是共享軟體，其良好的軟體設計理念，使得 Snort[11]到現在還是非常多人使用，並且仍然有在進行功能更新。

本論文研究的目的在於針對手機感染殭屍網路病毒後會與 C&C 伺服器進行溝通的異常連線封包，除了設計 Botnet Snort rule 去進行特徵比對，並移植網路上知名的 packet sniffer, Argus (Audit Record Generation Utilization System)到手機上分析網路流量，將這些異常連線的紀錄擷取出來，且從中擷取出相關特徵屬性值，上傳到遠端偵測中心伺服器，透過殭屍網路特有的 group similarity 特性，對於手機間的相似度比對並分析統計判斷手機是否遭受到殭屍網路攻擊。本研究希望能達到以下的目標：

1. 利用簡單的封包擷取方式，取出 TCP、UDP 的流量，以時間視窗做數量上的統計並從特徵屬性中找出不同流量的差異。

2. 針對兩種不同的流量種類，normal、bot，制定能夠分類出不同種類的特徵屬性。

3. 透過群組相似性比對方式，將各個手機上的的流量做有效的分類與辨識出是否有遭受殭屍網路感染。

4. 製作手機上的殭屍網路攻擊來測試。

5. 對實驗架構和結果做評估，是否能有效達成偵測殭屍網路的目的。

本論文架構如下：第二節討論相關研究，介紹殭屍網路相關的研究偵測方法；第三節說明建置偵測手機殭屍網路的環境；第四節敘述本論文提出的偵測系統的架構以及運作流程；第五節進行殭屍網路偵測系統的實作與實驗結果分析；第六節是本論文主要的貢獻；第七節是結論。

二、 相關研究

目前已有相關資訊安全專家投入殭屍網路偵測與分析的相關研究，來防止更多的電腦落入殭屍網路中。Honeynets[18]和 IDS (Intrusion Detection System 入侵偵測系統)是兩個常被使用的偵測技術。Honeynets 部署在分佈式虛擬電腦中，提供收集殭屍網路攻擊的訊息，Honeypots 在檢測安全威脅、收集惡意軟體及了解犯罪者的行為和動機的能力很強，可用來檢測大範圍的網路。IDS 如 BotHunter[26]，Snort 依據現有的殭屍網路的行為，針對可疑的活動進行分析以及偵測，無論是來自於內部主機的存取動作或透過遠端存取，若在這些合法的存取動作間夾帶惡意攻擊，能提供相對應的規則描述來偵測發現潛在的攻擊。

2.1 現有研究情形

PC 上殭屍網路常見的的偵測分析方法有以下幾種:

建構 Honeynet 來暗中加入殭屍網路[32,33]，使用 host-based scanning software 來偵測 rootkits, trojans, and malwares；分析 traffic，收集主機內外通訊的流量，並使用 Tāmd 工具來辨識新感染的 bot 主機[34]；使用過濾的方法逐步將 traffic 做檢查，並自行架設驗證殭屍網路所開發的管線化系統是否能有效將殭屍網路的流量找出。

殭屍網路為了要讓受害主機可以成功的與下達指令的 C&C 主機連結，所以將 IP 位置改成用 DDNS 的方式來連線，因此[21]使用”RIPPER”演算法針對 DNS 封包資料進行分析，進而將殭屍網路瓦解，使其無法吸收僵屍成員。

Hyunsang Choi 利用 DNS 的封包來作分析[27]，將 DNS query 分為正常的要求和惡意的要求，電腦受感染後會試圖與 C&C 伺服器連線，C&C 伺服器大多會使用動態網域名的方式，所以會有 DNS query，若無法連上則會試圖再與其他的 C&C 伺服器做連線，產生大量 DNS query。作者設計針對 DNS query 的演算法，依照 DNS query 的時間和 IP 來做分群，判斷 DNS 的封包是否合法。

Frederic Giroire 觀察連線到目的端的持續性並測量連線時間上的規律性來找出異常的連線位置[35]，並利用白名單機制，將長時間連線的目標加入名單中，略過正常的連線目標，統計異常的連線規律找出 C&C 伺服器。

觀察聊天型 bot，傳送訊息在時間間隔上、以及傳送訊息的長度大小，與正常人的行為有差異性。所以提出一種可偵測和分類已知和未知 bot 的分類系統[36]，系統技術上分成兩種方法，第一依據聊天內容上的差異性計算其 entropy 來偵測未知的聊天

型 bot，第二利用機器學習的方式來偵測已知的聊天型 bot，再針對 21 個不同聊天室的記錄檔上，發現 14 種不同類型的聊天型 bot，顯示結合這兩種技術的偵測可準確區分聊天型 bot 與人類行為的差異。

Peter Wurzinger 提出建立接收命令與回復命令做出相對應活動之間的對應偵測模型，現有觀察整體行為的偵測方式，需要有至少兩台以上的大量主機來觀察相同的行為，這裡從 Honeynet 技術系統 Nepenthes[37]的系統捕捉各種 bot，利用這些 bot 觀察單一主機的中毒行為，透過網路行為上的回覆行動以及從封包內容找出接收命令的相對應字串來建立偵測模型，藉由這技術能產生 IRC、HTTP、P2P 和 Strom 類型的 bot 偵測模型。

BotHunter [26]，偵測惡意軟體感染過程中的某個特定階段，包含內網掃描、感染入侵、惡意程式下載，與外網的 bot 協調對話，對外網的攻擊散佈將極可能是被由成功感染的本地主機所引發的內部入侵警報的對話追蹤和外部溝通模式結合在一起，當有一系列照順序符合 BotHunter 感染對話模型的證據，會產生一份合併報告，報告被捕捉到且在感染過程中起作用的相關事件和事件來源。也提出另一基於網路行為異常的偵測系統。

BotSniffer [23]，不需事先知道各種偵測特徵，觀察網路流量與 C&C 伺服器之間的時間與時空間的相關性和關聯性利用自行設計的統計演算法，找出殭屍網路。

手機上殭屍網路常見的的偵測分析方法有以下幾種:

在 2010 年 4 月，維吉尼亞理工大學發表一篇偵測手機殭屍網路的論文[31]，使用 host-based framework，根據手機上 keystroke and mouse click 記錄來判定是否為正常 key in 或是遭受到遠端攻擊者下達指令。

A sms-based mobile Botnet using flooding algorithm[38] ，除了自行模擬 SMS-based Botnet，偵測方法上在硬體裡加入 non-software-controlled signals 來判斷是否有使用者不知情下背地裡發送簡訊。

Mobile Botnet detection using network forensics[39]，觀察手機上的簡訊發送異常行為來判定是否攻擊者利用簡訊來傳送命令控制手機。

2.2 現有研究之不足

目前針對 Botnet 主要以 honeypot 和 IDS 的偵測技術，部屬在電腦、operator 或路由器上來分析網路上的惡意封包，但有鑑於手機上的計算能力以及電量的高度消耗，加上手機上網主要透過 wifi 或 3G 的方式，容易到處移動更換基地台，使手機所擁有的 ip 不斷的更動，因此無法直接套用現有的方法運作於上面，所以本論文特地設計輕量化的偵測方法，於手機端安裝已簡化過的 Snort 版本做即時分析，並搭配後端伺服器的離線分析，雙方面判斷來達到一定的準確度。

三、 建構手機 Botnet 偵測環境

整體可分成系統移植和後端偵測系統兩大部分來探討

3.1 系統移植

在這裡主要根據[9]這篇論文移植 Snort 的方法進行修改達成移植 Argus 到 Android

系統移植的步驟上，大略可分為三個步驟

- 取得Android上的Root權限
- 將需移植的系統或程式編譯成可在Android上執行的執行檔
- 實作前端控制界面並連結到該執行檔

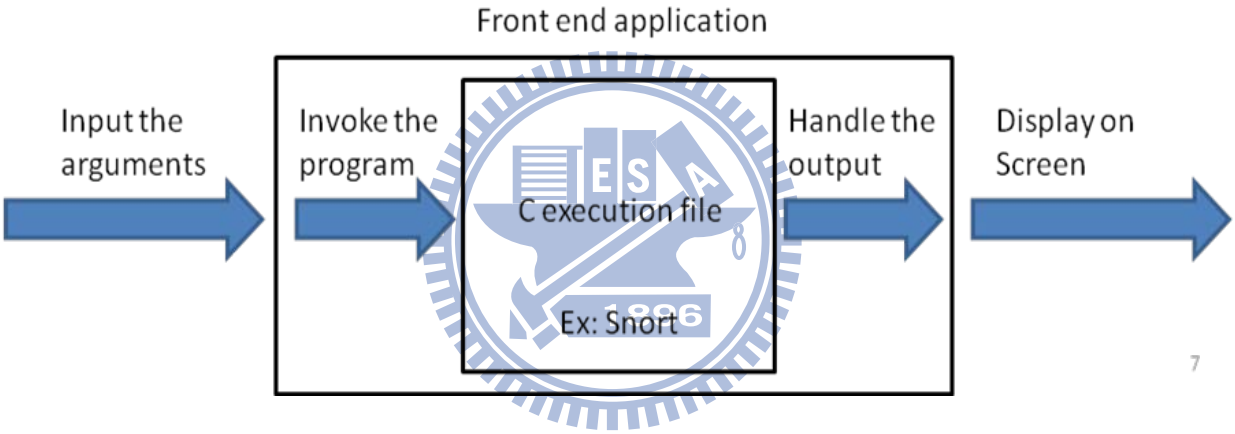


圖 3. 1

JAVA應用程式端呼叫C應用程式示意圖

3.1.1 Audit Record Generation Utilization System (Argus)

Argus 為一款 Linux 上即時網路流量監測工具，類似 Tcpdump，使用 Packet capture library (libpcap) 來擷取網路上的封包，並提供更多程式功來處理所需的封包格式。Argus 每個版本主要包含伺服器端與客戶端，伺服器端主要是做 sniffer，客戶端讀取分析 sniffer 後的 traffic data，另外也提供遠端 access control 機制跟 server 端直接取回資料。

伺服器端程式	用途
argus()	針對TCP/UDP來收集手機上的header封包

表 3. 1

客戶端程式	用途
ra()	讀取argus資料並擷取所需header資訊
racount()	讀取argus資料並統計各header屬性的資料量總和
rasort()	對argus資料進行排序
rafilteraddr()	過濾白名單(IP address)
rasqlinsert()	將argus資料上傳到SQL server
ragraph()	描繪出網路流量的圖型

表 3. 2

本篇是將兩端系統都移植到手機上，達成即時收取並立刻分析過濾所需封包內容

3.1.2　交叉編譯Argus到Android上

作業系統	Ubuntu 8.10
手機硬體	HTC Magic 32A
手機作業系統	CyanogenMod (Android 2.1-update1)
Cross-Compiler	Sourcey G++ Lite
流量偵測工具	伺服器端 ：argus-3.0.2
	客戶端 ：argus-client-3.0.2
函式庫	Libpcap 1.0.0
	Bison-2.4
	Flex-2.5.35

表 3. 3

軟硬體版本表

因為不同版本所需的函式庫版本也會有所差異，那以上是我們提供可行的建議版本。首先將上述的 libraries compile 成 static library，即.a 之結尾檔案，之後可利

用 linux 內建指令 objdump 查看是否 file format 為 elf32-little 得知是否有編譯成功。以下以/Target 表示編譯完之目標資料夾，可以改成任意一個放置編譯結果的資料夾。

檔案下載完後解壓縮，利用以下參數去修改 Makefile 的設定

CC： 指定.c 檔案的 compiler。

Host：指定編譯出來的執行檔要執行的平台。

Prefix：指定編譯出來的執行檔要放在哪個資料夾。

Disable-shared：不生成動態函式庫

LDFLAGS/CFLAGS : linking 的選項，主要用來指定是 statically linking 還是 dynamically linking。

CXX：指定.cpp 檔案的編譯器。

接下來要說明編譯 argus 伺服器端的程式，首先編譯 Libpcap，一開始沒有 Makefile 存在，需執行./configure 生成 Makefile。

CC=arm-none-linux-gnueabi-gcc ./configure --host=arm-linux --prefix=/Target LDFLAGS=”-static”
make
make install

再來用類似方法編譯 Bison

CC=arm-none-linux-gnueabi-gcc ./configure --host=arm-linux --prefix=/Target LDFLAGS=”-static”
Make
make install

最後則是編譯 argus-3.0.2

CC=arm-none-linux-gnueabi-gcc AR=arm-none-linux-gnueabi-ar RANLIB-arm-none-linux-gnueabi-ranlib

./configure --host=arm-linux --disable-shared --prefix=/Target LDFLAGS=”-static” CXX=arm-none-linux-gnueabi-g++ --with-libpcap-libraries=/Target/lib --with-liby-libraries=/Target/lib
Make
make install

除了給定參數外，還指定好程式去 linking 時所需 library 資料夾的位置，也就是目錄/Target，再利用 Android SDK 中提供的 adb tool 將編譯好的程式放入手機系統資料夾中。

adb push argus /data/filter

adb shell

cd /data/filter

./argus

圖 3. 2

成功編譯後的執行結果

針對 argus 客戶端的交叉編譯一樣先針對所需 library 去編譯，首先 Bison 使用上述一樣的指令去執行，再來是編譯 Flex

ac_cv_func_malloc_0_nonull=yes ac_cv_func_realloc_0_nonnull=yes CC=arm-none-linux-gnueabi-gcc ./configure --host=arm-linux --prefix=/Target LDFLAGS=”-static”
Make

make install

最後則是 argus-client-3.0.2

CC=arm-none-linux-gnueabi-gcc AR=arm-none-linux-gnueabi-ar RANLIB-arm-none-linux-gnueabi-ranlib ./configure --host=arm-linux --disable-shared --prefix=/Target CFLAGS=”-static” CXX=arm-none-linux-gnueabi-g++ --with-liby-libraries=/Target/lib --with-libfl-libraries=/Target/lib --with-libfl_pic-libraries=/Target/lib
Make
make install

adb push ra /data/filter/

adb push rafilteraddr /data/filter/

這裡我們是取 ra 和 rafilteraddr 這兩功能來做手機前端的封包處理

```
# ./ra -r out
   22:41:57.732138  e *       tcp    140.113.207.137.1234     <?>      192.168.0.199.50127       166
      26390   CON
   22:41:57.732138  e *       tcp      192.168.0.199.50127    <?>    140.113.207.137.1234        166
      26390   CON
   22:42:21.999517  e         tcp      192.168.0.199.53767     ->    140.113.207.137.21          31
```

圖 3. 3

擷取所需 header 欄位

```
# ./rafilteraddr -f whiteadd.txt -r out
   22:41:57.732138  e *       tcp    140.113.207.137.1234     <?>      192.168.0.199.50127       166
      26390   CON
   22:42:21.999517  e         tcp    140.113.207.137.21        ->      192.168.0.199.53767        31
      2424   FIN
   22:42:22.774517  e         tcp    140.113.207.137.29895     ->      192.168.0.199.43167         8
```

圖 3. 4

過濾白名單

3.2 後端偵測系統

這裡利用 AppServ 整合 Apache、PHP、MySQL 及 phpmyadmin，以下是相關環境設定:

- 作業系統 : Ubuntu 8.10
- Web Server : Apache2
- Web Language : PHP5
- Database : MySQL3.05
- 資料庫管理 : phpMyAdmin

四、 研究方法與系統設計

在這裡我們將手機上的流量當成所要研究的原始資料，分成兩部分的偵測階段，第一部分藉由 Snort 這套 IDS 系統，搭配我們自行設計出來針對 Android 手機殭屍網路攻擊的 Snort rule 來判定是否手機有執行開後門等一些惡意程式，做為第一道防護，再來透過我們所移植的 sniffer 系統，從受感染的手機與一般未受感染的手機中，過濾出異常連線的流量，從中找出差異性的特徵屬性，進一步將這些資料做分群處理，再透過相似性演算法去分析是否給定的流量有所異常。

4.1 問題分析

現今智慧型手機上大多數的殭屍病毒都會偽裝在一般正常的應用軟體上，如果使用者一旦不小心下載到手機上，此軟體就會背地裡利用手機上的漏洞透過緩衝區溢位攻擊(buffer overflow)設法執行 bot code，目的是要跟遠端伺服器建立連線後等待攻擊者下達指令，甚至設法取得手機的 root 權限來達到完整控制。因為手機不同於一般 PC 會當作 server 來使用，大部分的網路連線都是以手機向外連出為主，但為了接收攻擊者所下來的指令必定會出現 inbound 的流量，因此接下來我們便針對此特性來做探討。

4.2 異常連線特徵(現象)

主要有兩大特徵值去做判斷：

1. Inbound traffic:

   如果使用 Android 手機的用戶本身有 gmail 帳號，那登入後 google server 會跟手機自動建立好一條連線，不定期的連回手機來提供 mail 通知或其他軟體更新的資訊。那除了此情況外手機幾乎沒有流入的連線，因此一但有表示極有可能是攻擊者正在下達指令的情況。

2. TCP header 的 State(Flag) 欄位

   TCP 協定在傳送封包之前，會先以三方交握(Three-Way-Handshake)和目的端進行連接。簡單地說，首先送出 SYN 封包給目的端，表示要求建立 TCP 連線，假如目的端接受連線時會傳回 SYN/ACK 封包，當來源端收到目的端傳回的 SYN/ACK 封包時，即可開始傳送資料，一旦資料傳輸結束便會送 FIN 封包給目的端結束連線狀態，但我們知道攻擊者與手機會持續保持著連線的

效果，不會馬上結束，因此 header 的 state 中必定會出現 CON 這個狀態值。

State	SYN (Synchronize sequence numbers)	FIN (Finish)	CON (Connected)	INT (Initial)	RST (Reset the connect)
說明	代表要求雙方進行同步處理，也就是要求建立連線	用來釋放連接，表明雙方已經沒有資料傳送了	代表雙方的連線已經建立好並持續保持連接的狀態	一開始請求連線的狀態	因某種原因引起初線的錯誤連接，也用來拒絕非法資料和請求

表 4. 1

常見的 TCP state 欄位屬性

	proto	saddr	sport	daddr	dport	dir
異常	tcp	140.113.207.137	1234	192.168.0.199	57038	->
正常	tcp	192.168.0.199	39118	66.220.147.47	80	->

spkts	sbytes	Dpkts	Dbytes	state
8	463	8	574	**CON**
13	2136	12	10035	FIN

註: 192.168.0.199: 手機 IP

表 4. 2

異常與正常封包內容示意圖

4.3 系統架構與流程

本論文的系統架構，手機前端安裝 Snort 和針對手機殭屍網路的封包過濾器，後端收集所有上傳的手機封包去做分析。Snort 即時為手機做封包異常偵測，判定手機是否有連上釣魚網站、執行惡意 shell code、backdoor 連線和使用系統內部指令來控制手機，同時利用移值到手機的 filter 過濾所需的封包 header 資訊，利用 FTP 上傳到後端 server 後，根據 Botnet group activity 的特性去做相似度的比對，下圖為偵測架構圖和系統偵測流程圖。

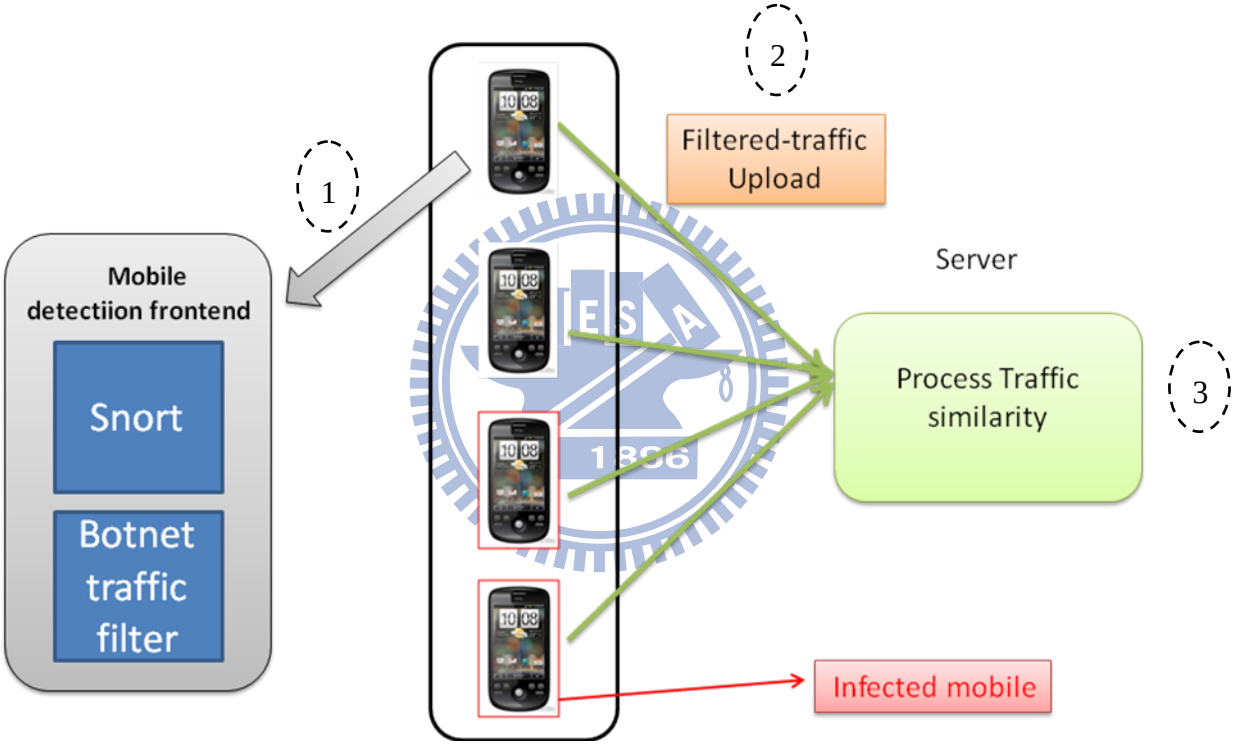


圖 4. 1

偵測系統架構示意圖

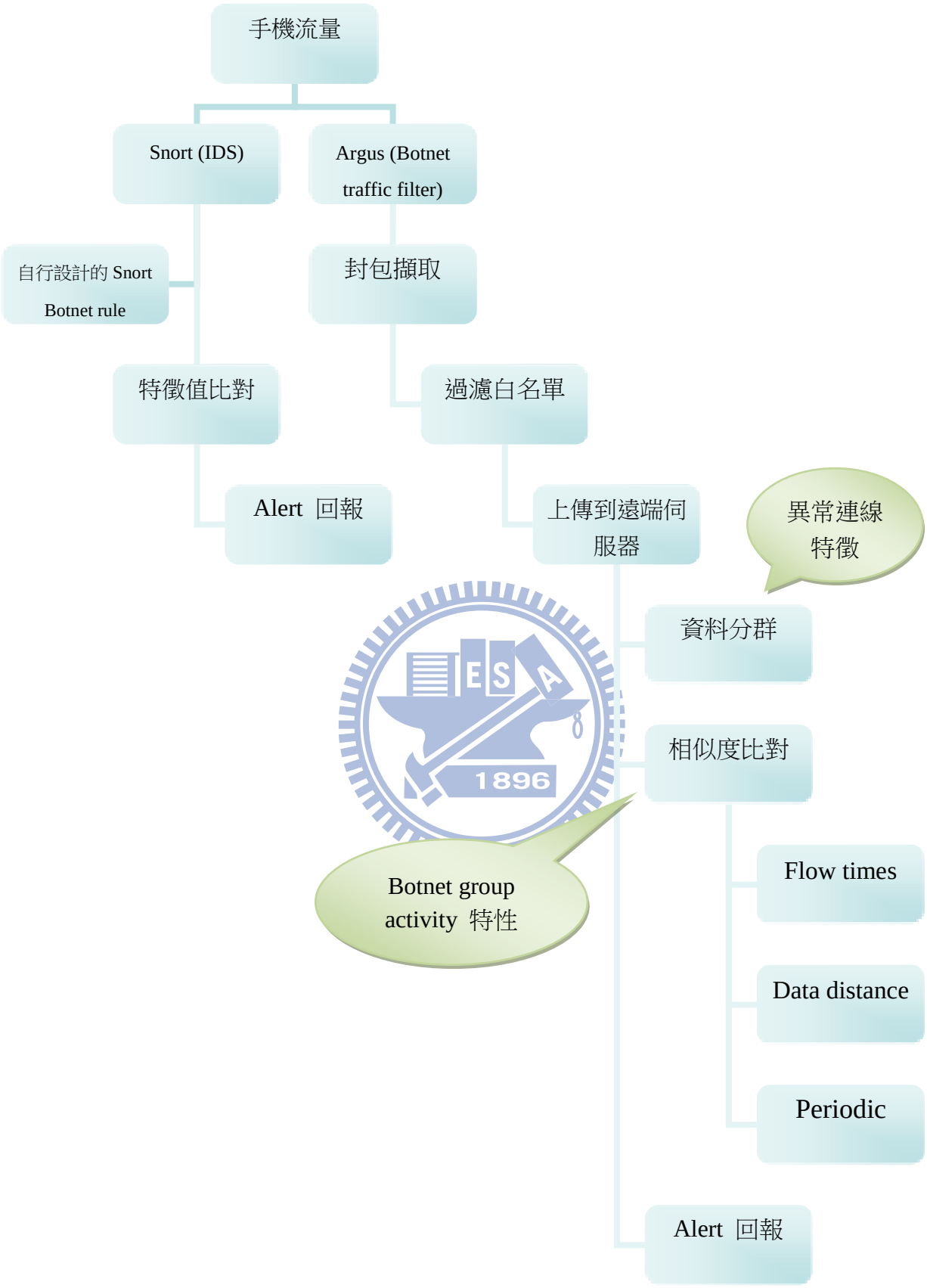

手機流量
Snort (IDS)
Argus (Botnet traffic filter)
自行設計的 Snort Botnet rule
封包擷取
特徵值比對
過濾白名單
Alert 回報
上傳到遠端伺服器
異常連線特徵
資料分群
相似度比對
Botnet group activity 特性
Flow times
Data distance
Periodic
Alert 回報

圖 4. 2

系統偵測流程圖

以下小節就分別來說明每個流程的詳細內容。

4.4 提供 Android 上的 Snort Botnet signature

根據手機上觀察到的殭屍網路攻擊，設計出 Snort rule 來偵測，觀察到的攻擊主要有以下幾種:

1. 後門程式：

   由一位名叫 MJ.Keith 的作者提供[40]，主要針對 Android(2.0/2.1)裡 Webkit 瀏覽器的漏洞，使用 use-after-free 技巧去植入/system/bin/sh 這個指令到記憶體中並執行，主要目的就是促使手機連回一條 shell 到指定的遠端 server，達成後門攻擊。

2. 執行系統指令：

   Android 的指令集都放於/system/bin 這個資料夾底下，一旦攻擊者使用裡面的指令，便可以查看手機內部資料、端看系統 port 資訊。

3. 遠端安裝應用軟體：

   Android SDK 有提供一個 Adb tool，此 tool 是平常手機藉由傳輸線跟電腦連線後，提供相關指令讓使用者利用電腦觀看手機內容並安裝相關手機應用程式，但也有提供遠端連線功能，只要電腦跟手機是同網域的情況下，在得知手機 IP 並且開啟 5555 port 後，便可以無線方式跟手機進行連線，這方式也是目前駭客使用的手法之一。下圖為 signature 的內容與說明。

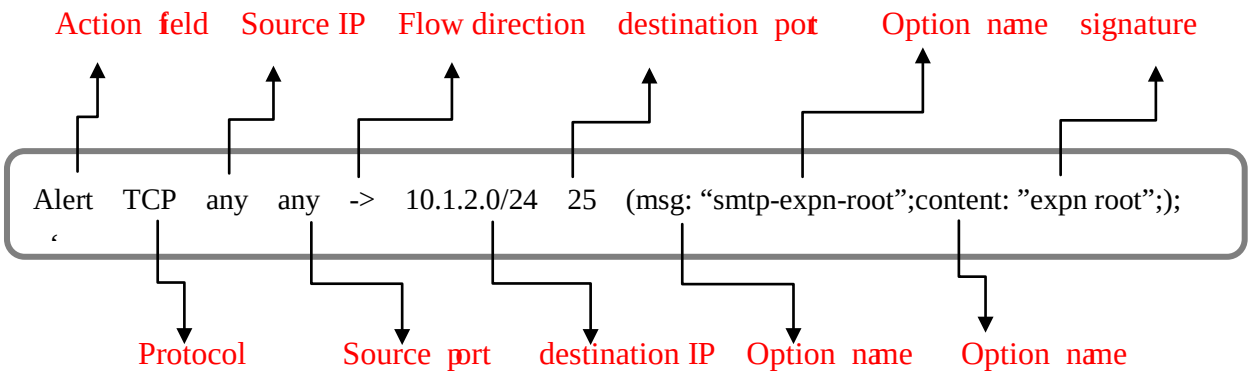


圖 4. 3

Snort 偵測規則(rule structure)

Signatures	說明
alert tcp any any -> any any (msg: "shell Backdoor access1!"; content:"|75 37 33 37 39 75 36 35 37 34 75 32 66 36 64 75 32 66 36 64 75 36 39 36 32 75 32 66 36 65 75 36 38 37 33|";nocase;)	Content 內容為 /system/bin/sh
alert tcp any any -> any any (msg: "shell Backdoor access2!"; content:"NaN(ffffe00572c60)";nocase;)	利用 webkit 浮點數 function 無法正確驗證 NaN 的漏洞
alert tcp any any -> $HOME_NET 5555 (msg:"adb connect";)	遠端跟手機建立 adb 連線
alert tcp any any -> any any (msg:"Port scan";content:"/system/bin/netstat";nocase;)	遠端查看系統 port 資訊
alert tcp any any -> any any (msg:"System stole";content:"/system/bin/";nocase;)	遠端操控手機內建指令
alert tcp any any -> any any (msg:"Http Get";content:"GET /shell.html HTTP/1.1";nocase;)	連上惡意網頁(包含後門程式))

註: $HOME_NET = 192.168.0.199(手機 IP)

表 4. 3

4.5 手機過濾封包

本論文主要擷取的異常連線共有 TCP 和 UDP 等兩種類型，根據目前 PC 上現有的 bot code 的分析，大部分以 TCP 來確保連線的穩定性，因此我們也以 TCP protocol 為主，在這裡我們將手機進來的封包以 flow 為單位，根據所預設的時間間隔內，記錄此 flow 的時間以及 ten header records : protocol、source IP、source port、destination IP、destination port、direction、duration、source packets、source bytes、state，再經過 http://www.alexa.com/topsites 提供的白名單過濾 IP 後上傳到遠端偵測伺服器進行資料分群。圖 4.4 為手機封包過濾流程圖。

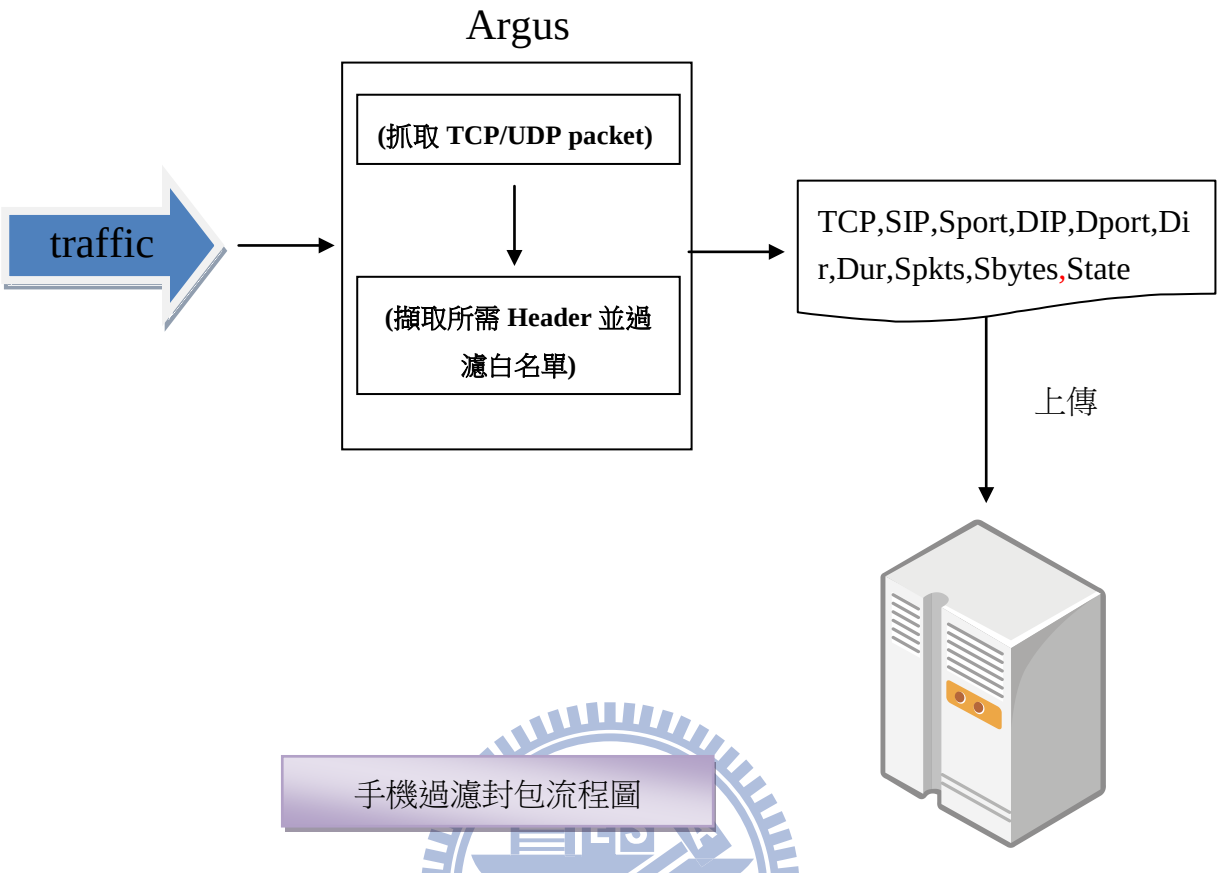


圖 4. 4
手機封包過濾流程圖

4.6 手機資料分群

在這裡偵測伺服器中心收到許多手機上傳的封包資訊後，根據 Lookout 手機防毒軟體公司的報導，感染 Geimini 殭屍病毒的手機大約每五分鐘會跟遠端 C&C server 做溝通，表示攻擊者會定期的下指令來控制 bot 行為，再根據之前的異常連線特性，我們接下來就依據四個特徵值來做分群 ：

1. Time interval : Wi
2. Inbound traffic
3. Bi-direction flow
4. Same source IP

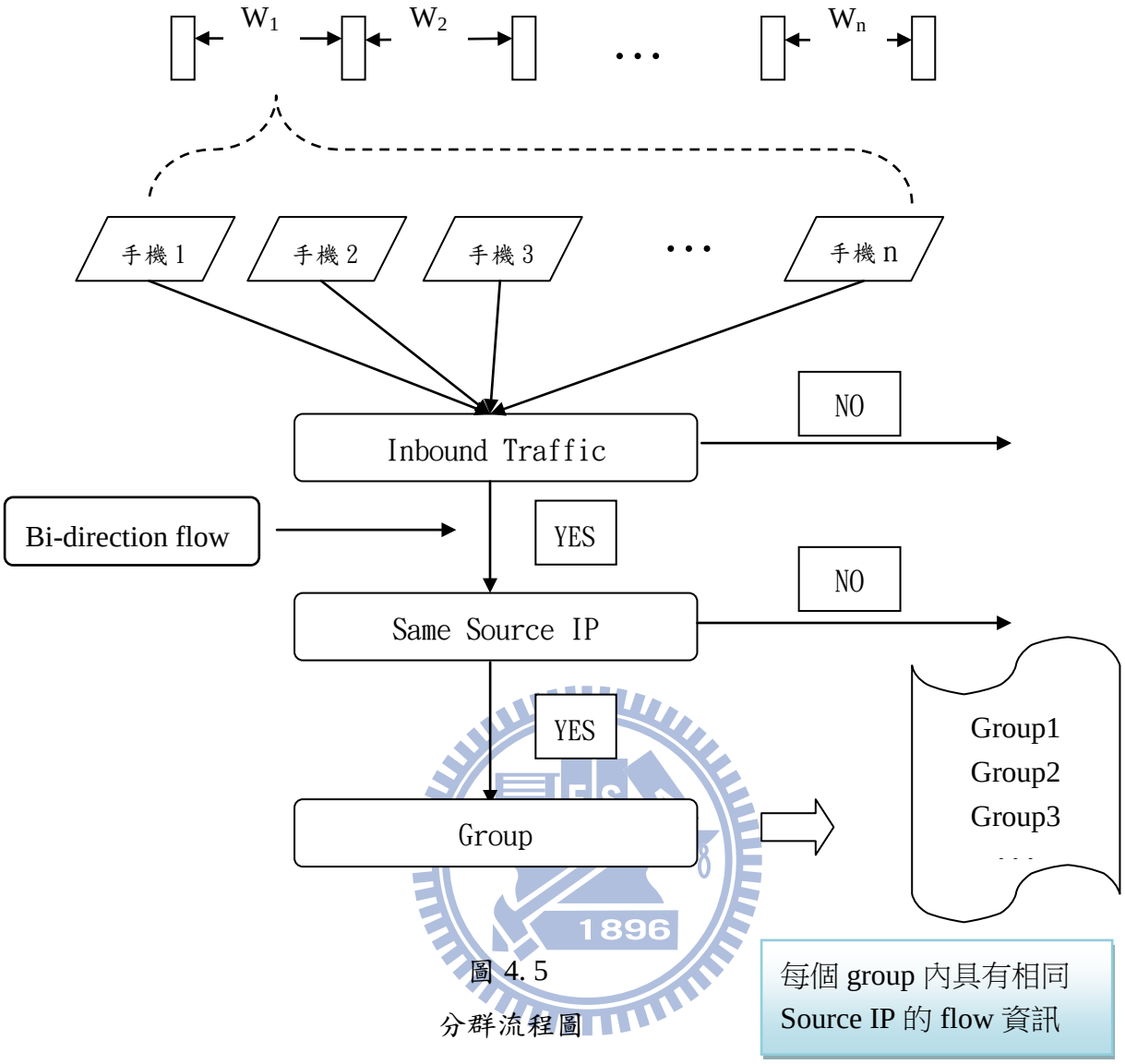


圖 4. 5
分群流程圖

在W_1時間內的每個group 內的DIP都具有相同的source IP，以下是Group 1和Group 2的例子說明:

Group	SIP	DIP	Flow 次數
1	140.113.207.137	手機1	3
		手機2	3
2	74.125.153.188	手機1	7
		手機2	8
		…	…

表 4. 4

之後針對一個 group 中，將每個 DIP 的 flow 次數表示成向量，向量的表示方法為利用每個 flow 的 state 值，如果 **state 值為"CON"**，代表此 flow 的 SIP 與手機是保有異常連線，則 Tuple 值標為 1，否則標為 0。

If (flow_state = "CON") ⟶ Tuple = 1

Else ⟶ Tuple = 0

之後每個手機的 Tuple 值轉換成 f_i 值的方式去表示，利用每個手機的 f_i 當作相似度演算法的輸入，算出兩兩手機間的 flow 次數相似度，最後平均做為整個 group 的 flow times 相似度值。最主要使用 flow times 這個特徵是因為 C&C server 會有**同時**下達指令給所有 bots 的行為，所以 bots 間彼此跟 server 的連線次數一定會非常相近，表示成向量則是為了增加時間上的關連性。最後再新增 average bytes per packet、average bytes per second、average packet per second 三個欄位值來做 data distance 的分析。表 4.5 是以 Group 1 來說明 ：

Group	SIP	DIP	Flow 次數	state	Tuple	f_i
1	140.113.207.137	手機1	3	CON	1	$f_1 = (1,1,1)$
		手機1		CON	1	
		手機1		CON	1	
		手機2	3	CON	1	$f_2 = (1,0,1)$
		手機2		FIN	0	
		手機2		CON	1	

Average Bytes Per packet	Average Bytes Per second	Average Packet Per second
61	944	15
58	797	13

表 4. 5

4.7 group 相似度分析

在這裡我們比對的設計概念主要根據 Botnet group activity 的特性，這是 Botnet 獨有的一個特性，主要包括下列 4 種群體行為:

1. 攻擊者下達指令給 bots，bots 接收指令並傳送資料
2. Bots 下載更新 bot code
3. 遷移 bots 的連線到其他 C&C server
4. Bots 執行 Spam、DDoS 或 Click Fraud 等大量網路攻擊

每次針對一個 group 分成三部分的相似度分析，第一部分計算整體 flow 次數的相似度，使用 flow 次數 這個特徵是因為 C&C server 會有**同時**下達指令給所有 bots 的行為，所以 bots 間彼此跟 server 的連線次數一定會非常相近。第二部分計算整體資料傳輸量的相似度值，根據手機上收到平均每個 packet 的 byte 數、平均每秒多少個 bytes、平均每秒多少個 packet 數這三個特徵值來鑑定，之後結合第一、二部分的值來推出此 group 內的 source IP 是否為可疑的 C&C server。第三部分則是利用 server 會定期跟 bot 溝通的特性，評斷不同時間間隔的 group 中，手機 IP 的重複率判定是否 group 內的手機群已成為 Botnet 的成員。

4.7.1 Flow 次數 similarity

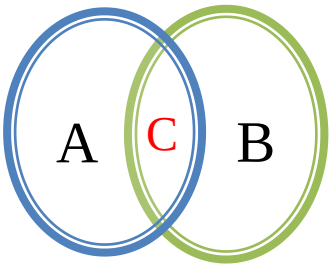
這裡我們使用 Jaccard similarity coefficient 來計算兩手機跟 C&C server 間的連線次數的相似度。

4.7.1.1 Jaccard similarity coefficient

又可稱為 Tanimoto coefficient，為 Jaccard coefficient 的延伸，去計算以 birnary 形式表達的兩個集合間的重覆率，相似值介於 0~1 之間，越靠近 1 代表兩集合的交集越大。

Jaccard coefficient

$$S_{AB} = \frac{C}{A + B - C}, \quad S \in [0,1]$$



根據 group 內每個手機的 f_i 值當做比對的向量集合，套用 Jaccard coefficient 的公式:

$f_i f_j^T$：對應項相乘並相加

n: *number of different DIPs in a group*

$$S_{Jacc(f_i, f_j)} = \frac{f_i f_j^T}{f_i f_i^T + f_j f_j^T - f_i f_j^T}, \quad i \neq j, \; i, j \in \{1, \ldots, n\}$$

每兩個手機算出 flow 次數相似度後，最終算出平均值，做為整體 group 的 flow 次數的相似度值。

$$\overline{S} = \frac{1}{\binom{n}{2}} \sum_{i \neq j} S_{Jacc}(f_i, f_j)$$

4.7.2 Data distance

4.7.2.1 Euclidean formula

這裡我們使用歐基里得演算法[32]主要是我們將三個特徵值 bytes per packet、bytes per second、packet per second 轉換為三維空間上的點，再以歐基里得距離來衡量每兩個手機與 C&C server 的資料傳輸量差異性，相似值我們取介於 0~1 之間，越靠近 0 代表兩手機的資料傳輸量越靠近。

$$D_i = (BPP_i, BPS_i, PPS_i) \ , \ D_j = (BPP_j, BPS_j, PPS_j)$$

$$norm\_diff_1(i,j) = (\frac{BPP_i - BPP_j}{BPP_i + BPP_j})$$

$$norm\_diff_2(i,j) = (\frac{BPS_i - BPS_j}{BPS_i + BPS_j})$$

$$norm\_diff_3(i,j) = (\frac{PPS_i - PPS_j}{PPS_i + PPS_j})$$

n : *number of different DIPs in a group*

$$\delta_{i,j} = \sqrt{\sum_{k=1}^{3} (norm\_diff_k(i,j))^2} \ , \ i \neq j \ , \ i,j \in \{1,...,n\}$$

每兩個手機算資料距離後，最終算出平均值，做為整體 group 的 data distance。

$$\delta_{distance} = \frac{1}{\binom{n}{2}} \sum_{i \neq j} \delta_{i,j}$$

Flow 次數的相似度跟 data distance 都算完後，相乘做為整個 group 的 score 值，再根據 threshold 值 T，若此 group 的 score 值超過 T > 0，就列為一個的 Botnet 的 group。

$T : threshold$

$$Score_{groupi} = \overline{S} \times (1 - \delta_{distance})$$

	$Group_i$
$Score_{groupi} \geq T$	異常
$Score_{groupi} < T$	正常

表 4. 6

以下是用 group 1 來舉例說明：

$$f_1 = (1,1,1)$$
$$\updownarrow \updownarrow \updownarrow$$
$$f_2 = (1,0,1)$$

$$S_{Jacc(\overline{f_1}, \overline{f_2})} = \frac{(1,1,1)\bullet\begin{pmatrix}1\\0\\1\end{pmatrix}}{(1,1,1)\bullet\begin{pmatrix}1\\1\\1\end{pmatrix} + (1,0,1)\bullet\begin{pmatrix}1\\0\\1\end{pmatrix} - (1,1,1)\bullet\begin{pmatrix}1\\0\\1\end{pmatrix}} = \frac{2}{3} \approx 0.66$$

$$\overline{S} = \frac{1}{\begin{pmatrix}2\\2\end{pmatrix}} \times 0.66 = 0.66$$

$$D_1 = (61,944,15) \ , \ D_2 = (58,797,13)$$

$$norm\_diff_1(i,j) = (\frac{61-58}{61+58}) = 0.025$$

$$norm\_diff_2(i,j) = (\frac{944-797}{944+797}) = 0.084$$

$$norm\_diff_3(i,j) = (\frac{15-13}{15+13}) = 0.071$$

$$\delta_{1,2} = \sqrt{(0.025)^2 + (0.084)^2 + (0.071)^2} = 0.112$$

$$\delta_{distance} = \frac{1}{\binom{2}{2}} \times 0.012 = 0.012$$

$$Score_{group1} = \overline{S} \times (1 - \delta_{distance}) = 0.66 \times (1 - 0.112) = 0.586$$

4.7.3 Periodic

C & C Server 會定時去跟 bot 做聯繫，所以我們取不同時間間隔中的兩個 group，若雙方具相同的 source IP 下，去比對裡面有相同 destination IP 的重複率，如果重複率高也代表為一種異常連線的情況。主要使用 Kulczynski similarity 這個相似度方法來評估。

|T1| : W_1 時間內 group i 的總手機數量

|T2| : W_2 時間內 group j 的總手機數量

|V| : group i 和 group j 具有相同手機 IP 的數量

Same Source IP			
W_1		W_2	
	DIP		DIP
group i	手機 1	group j	手機 1
	手機 2		手機 2
	手機 3		手機 4

表 4. 7

$$P = \frac{1}{2} \cdot (\frac{|V|}{|T1|} + \frac{|V|}{|T2|}), (|T1| \neq 0, |T2| \neq 0) \ , \ P \in [0,1]$$

P 的值若越接近 1，代表兩個 group 的 IP 重複率越高，也就表示這兩個 group 內的手機定期受到遠端 server 的連線溝通，可能已成為 Bonet 的成員。

五、 實驗結果與效能分析

5.1 實驗環境

5.1.1 模擬手機殭屍網路攻擊

因為目前無法取得手機上 bot code，所以我們參考防毒軟體公司的報告來實際模擬建置出手機的 Botnet 當作實驗的測試樣本，攻擊所具備的特性主要由 Geinimi 這隻病毒所擁有的行為做依據，以下是我們設計出來的 Bot 行為:

攻擊	目的
Weather application include: 1.後門程式 ([MJ.Keith]作者提供) 2.開啟 port 5555	讓手機連上惡意網頁執行裡面的後門程式，之後背地裡跟 C&C server 建立遠端連線，並開啟手機 port 5555。
Command and control server	Botmaster 每隔五分鐘透過 server 來傳遞指令給 bot，竊取系統和 SD card 上的資料，並上傳到 Botmaster 哪
Spam mail application	促使手機發送 spam mail

表 5. 1

5.1.2 角色與網路

角色	數目
Botmaster	1
C&C server	1
手機	2 (感染 bot code)
Android 模擬器	3 (未感染 bot code)
遠端偵測伺服器	1
手機版本	HTC Magic 32A
手機作業系統	CyanogenMod (Android 2.1-update1)
手機網路環境	Wifi(D-Link)
C&C and detection server	Ubuntu 8.10 Intel Core(TM)2 Duo 2.4GZ 3G memory

表 5. 2

5.2 實驗參數

5.2.1 特徵擷取時間間隔

我們設計的手機殭屍病毒大約每五分鐘會跟 server 進行溝通，每次溝通約五分鐘左右，因此我們每隔 10 分鐘去收集手機封包來分析。

5.2.2 Score 的 threshold

我們根據實驗的結果發現 normal group 跟 Botnet group 最大的差異在 flow Similarity。仔細觀察到流進手機的封包中，除了 C&C server 下達指令的連線外，只剩 google server 與手機有連線的動作，原因是當 Android 手機一開機後就會先跟 google server 建立好一條連線，server 就透過此連線發送相關通知訊息給手機，例如: gmail 的新郵件通知、系統更新資訊等等，但這樣連線的次數並不規律，不像攻擊者都是一起發送指令給每個手機，所以中了 Botnet 的手機間的 flow 次數大約差距 1 次或 2 次，因此 Bonet group 的 flow similarity 平均都會大於 0.7 以上，normal group 平均都 0.5 以下。目前實驗定 Threshold T = 0.5，之後可以根據更多病毒樣本後再去作一個調整。

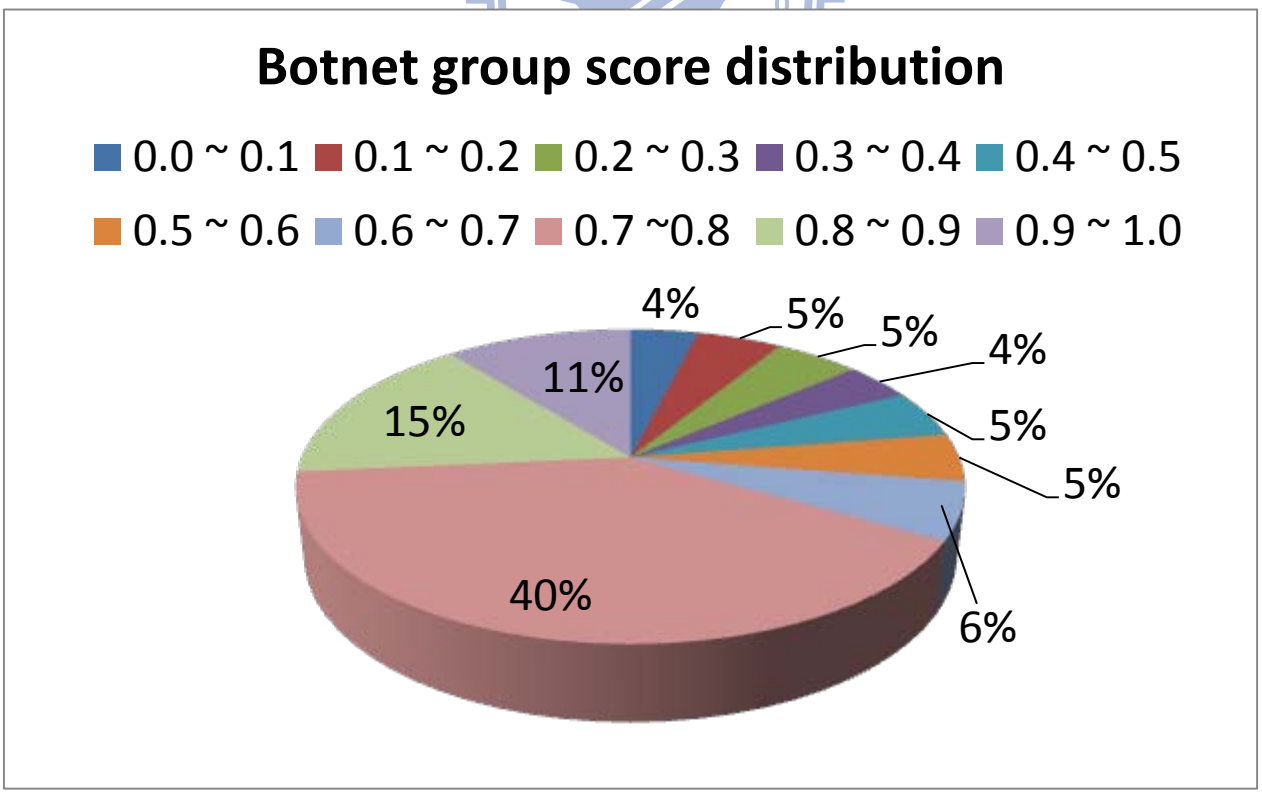


圖 5. 1

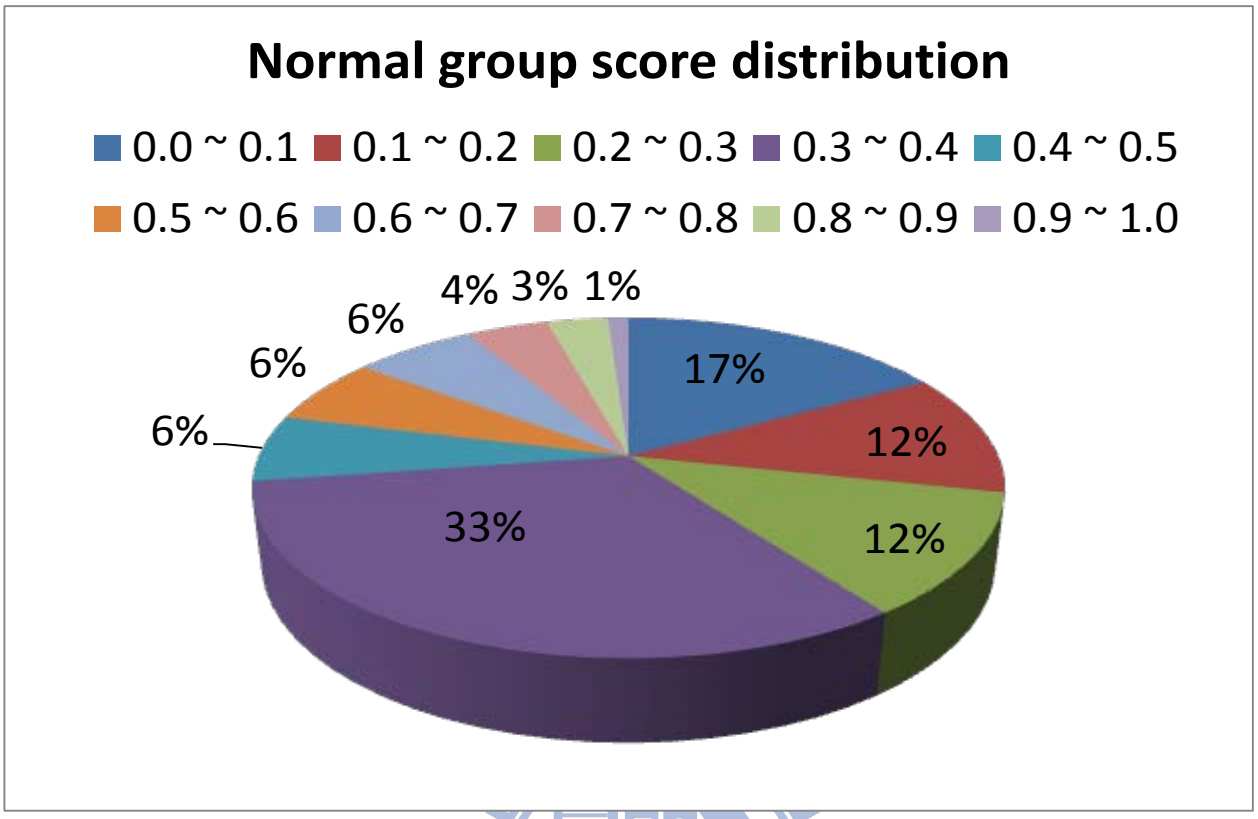


圖 5. 2

5.3 實驗結果

Normal group :google 與手機的連線

	$\overline{S}$	$1-\delta_{distance}$	Score	Threshold T = 0.5	Periodic
Normal group	0.42	0.84	0.35	< T	0
Botnet group	0.875	0.89	0.78	> T	1

表 5. 3

Group 分析的結果

	Bytes per packet	Bytes per second	Packets per second
Normal group	258.30	473842.47	1801.72
Botnet group	59.93	870.59	14.58

表 5. 4

Group 內資料量的平均值

5.4 效能分析

Programs	Java part	C part	Total
Snort	20720KB	4016KB	27748KB
Argus		3012KB	
Phone	25780KB		25780KB
Difference			1968KB

表 5. 5

手機偵測軟體執行所耗費的記憶體

如表 5.5 所示，在 Snort 跟 Argus 執行時所耗費的記憶體空間，大概比手機內建的 phone app 多 2MB 左右。

	Consumption per hour	Battery lasting time
Originally	7.2%	14 hours
Turn on Snort	9.6%	10 hours
Turn on Argus	8.1%	12hours
Turn on both	13.2%	8hours

表 5. 6

電量消耗(上 facebook)

如表 5.6 所示，在一般正常瀏覽網頁下(這裡我們是上 facebook 做測試)和開啟 Snort 跟 Argus 下手機所需額外耗費的電量。

CPU 消耗量

	Snort	Argus
Youtube	9~16%	5~10%
Facebook	5~8%	2~5%
Both	12~23%	7~16%

表 5. 7

如表 5.7 所示，在開啟 Snort 跟 Argus 下，如果有其他 APP 也在執行時，所需額外的 CPU Overhead。

本次實驗使使用 HTC magic(G2)，這款是 HTC 三年前出產的第二代手機，在 CPU 處理器跟 ROM 大小跟現金的手機相比都明顯低階不少，我們經由刷機過後將 ROM 提升到大約到 280MB，但若偵測軟體全開發現任會有吃力的效果，因此可建議使用 HTC Desire 或 Wildfire 等新款手機即可順暢許多。

六、 貢獻

1. 本論文提供第一個偵測 Android 手機上殭屍網路攻擊的系統
2. 偵測技巧結合即時特徵值比對和異常封包的離線偵測，提供雙方面的偵測效果
3. 仿效目前觀察到的手機殭屍網路攻擊情況，自我建構手機 Botnet 環境去做測試
4. 針對手機的耗電量和記憶體消耗做分析，除了在耗電量上會比較吃重外，記憶體並不會消耗太多造成手機的負擔。

七、 結論

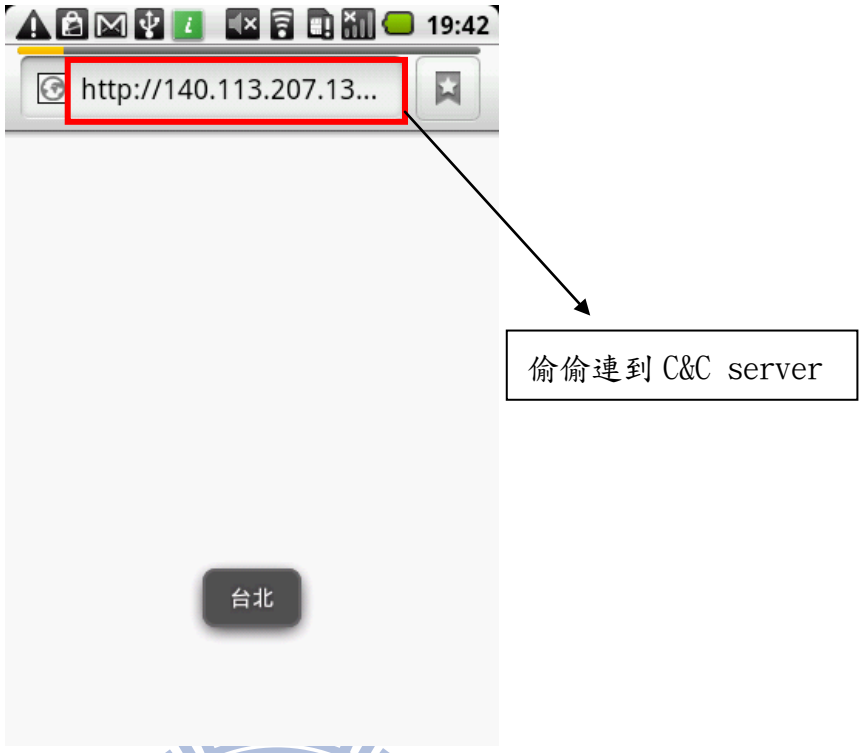
鑒於手機的功能越來越強大且計算能力不斷提升下，手機已漸漸跟筆記型電腦一樣具有多樣化的功能，因此過往只會在 PC 上出現的殭屍網路病毒也慢慢入侵到手機上，造成手機安全更大的威脅。

因此我們將功能強大的 Snort 配合殭屍異常封包的偵測技術，對傳入手機的封包進行檢測，確保手機隱私和上網的安全性。

最後目前手機上的殭屍病毒並不多，因此本論文在無法取得實際樣本下，透過一些手機病毒公司的描述與報導來實際模擬手機殭屍病毒，在未來一定會出現更多更強大甚至攻擊手法更多元的殭屍病毒，到時可以取得實際的樣本後，便可設計出更完善的偵測方法來抵擋手機的殭屍病毒攻擊。

攻 擊 畫 面

惡意天氣軟體，包含後門程式和開啟手機 port 5555



正在下載並執行 bot code

攻擊者使用系統指令觀看系統資訊

```
who
1        bot      127.0.0.1       <- botmaster
2        bot      140.113.207.170
3        bot      140.113.207.170
4        bot      140.113.207.137
/system/bin/id >a11
/system/bin/id
uid=10000(app_0) gid=10000(app_0) groups=1015(sdcard_rw),3003(inet)
id=10000(app_0) gid=10000(app_0) groups=1015(sdcard_rw),3003(inet)
id=10000(app_0) gid=10000(app_0) groups=1015(sdcard_rw),3003(inet)
uid=10031(app_31) gid=10031(app_31) groups=1015(sdcard_rw),3003(inet)
uid=10031(app_31) gid=10031(app_31) groups=1015(sdcard_rw),3003(inet)
```

/system/bin/id:目前攻擊者所擁有的權限

```
/system/bin/netstat >a11
/system/bin/netstat
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp        0      0 127.0.0.1:5037         0.0.0.0:*              LISTEN
tcp        0      0 0.0.0.0:5555           0.0.0.0:*              LISTEN
tcp        0      0 10.0.2.15:40996        140.113.207.137:1234   ESTABLISHED
cp         0      0 10.0.2.15:40996        140.113.207.137:1234   ESTABLISHED
cp         0      0 10.0.2.15:40996        140.113.207.137:1234   ESTABLISHED
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp        0      0 127.0.0.1:5037         0.0.0.0:*              LISTEN
tcp        0      0 0.0.0.0:5555           0.0.0.0:*              LISTEN
tcp        1      0 192.168.0.199:41785    140.113.207.137:1234   ESTABLISHED
tcp        1      0 192.168.0.199:41786    140.113.207.137:1234   ESTABLISHED
tcp        0      0 192.168.0.199:44544    74.125.153.188:5228    ESTABLISHED
tcp        0      0 192.168.0.199:35207    64.233.183.147:80      ESTABLISHED
udp        0      0 0.0.0.0:9000           0.0.0.0:*
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp        0      0 127.0.0.1:5037         0.0.0.0:*              LISTEN
tcp        0      0 0.0.0.0:5555           0.0.0.0:*              LISTEN
tcp        1      0 192.168.0.199:41785    140.113.207.137:1234   ESTABLISHED
tcp        1      0 192.168.0.199:41786    140.113.207.137:1234   ESTABLISHED
tcp        0      0 192.168.0.199:44544    74.125.153.188:5228    ESTABLISHED
tcp        0      0 192.168.0.199:35207    64.233.183.147:80      ESTABLISHED
udp        0      0 0.0.0.0:9000           0.0.0.0:*
```

/system/bin/netstat:觀看手機內部 port 資訊

```
/system/bin/ps
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     296    204   c009a694 0000c93c S /init
root      2     0     0      0     c004dea0 00000000 S kthreadd
root      3     2     0      0     c003f778 00000000 S ksoftirqd/0
root      4     2     0      0     c004aaf4 00000000 S events/0
root      5     2     0      0     c004aaf4 00000000 S khelper
root      6     2     0      0     c004aaf4 00000000 S suspend
root      7     2     0      0     c004aaf4 00000000 S kblockd/0
root      8     2     0      0     c004aaf4 00000000 S cqueue
root      9     2     0      0     c017bb3c 00000000 S kseriod
root      10    2     0      0     c004aaf4 00000000 S kmmcd
root      11    2     0      0     c006ecac 00000000 S pdflush
root      12    2     0      0     c006ecac 00000000 S pdflush
root      13    2     0      0     c007349c 00000000 S kswapd0
root      14    2     0      0     c004aaf4 00000000 S aio/0
root      21    2     0      0     c017933c 00000000 S mtdblockd
root      22    2     0      0     c004aaf4 00000000 S hid_compat
root      23    2     0      0     c004aaf4 00000000 S rpciod/0
root      24    2     0      0     c018e530 00000000 S mmcqd
root      25    1     728    176   c01537e8 afe0c7dc S /system/bin/sh
system    26    1     796    192   c019a854 afe0ca7c S /system/bin/servicemanager
root      27    1     832    260   c009a694 afe0cba4 S /system/bin/vold
root      28    1     656    164   c01a65e8 afe0d40c S /system/bin/debuggerd
radio     29    1     4396   380   ffffffff afe0d0ec S /system/bin/rild
root      30    1     82016  11368 c009a694 afe0cba4 S zygote
```

/system/bin/ps： 查看系統中正在執行的程式之程序資料

```
/system/bin/ls /sdcard >all
/system/bin/ls /sdcard
LOST.DIR
output
utput
utput
00001.vcf
00002.vcf
1
00001.vcf
Android
00002.vcf
Backgrounds
1
DCIM
Android
FxCamera
Backgrounds
HTCSync_1.0.1.exe
DCIM
HTC_ADP_1.6_DRC83_rooted_base.zip
FxCamera
HTC_Magic_A6161_????????????.pdf
HTCSync_1.0.1.exe
```

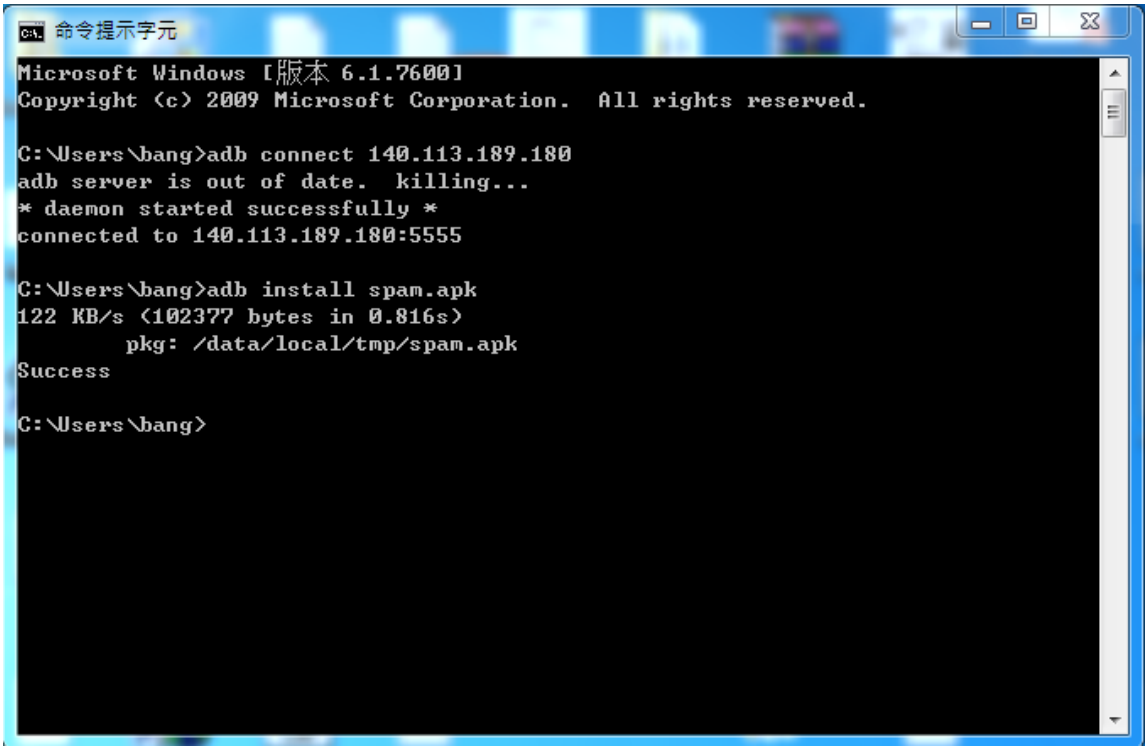
/system/bin/ls /sdcard： 顯示手機 SDcard 的內容

```
/system/bin/cat /proc/cpuinfo >all
/system/bin/cat /proc/cpuinfo
Processor       : ARM926EJ-S rev 5 (v51)
BogoMIPS        : 26.16
Features        : swp half thumb fastmult vfp edsp java
CPU implementer : 0x41
CPU architecture: 5TEJ
CPU variant     : 0x0
CPU part        : 0x926
CPU revision    : 5
PU revision     : 5
Hardware        : Goldfish
Revision        : 0000
Serial          : 0000000000000000
erial           : 0000000000000000
erial           : 0000000000000000
Processor       : ARMv6-compatible processor rev 2 (v61)
BogoMIPS        : 421.72
Features        : swp half thumb fastmult edsp java
CPU implementer : 0x41
CPU architecture: 6TEJ
CPU variant     : 0x1
CPU part        : 0xb36
CPU revision    : 2
PU revision     : 2
Hardware        : sapphire
```

/system/bin/cat /proc/cpuinfo :手機系統架構資訊

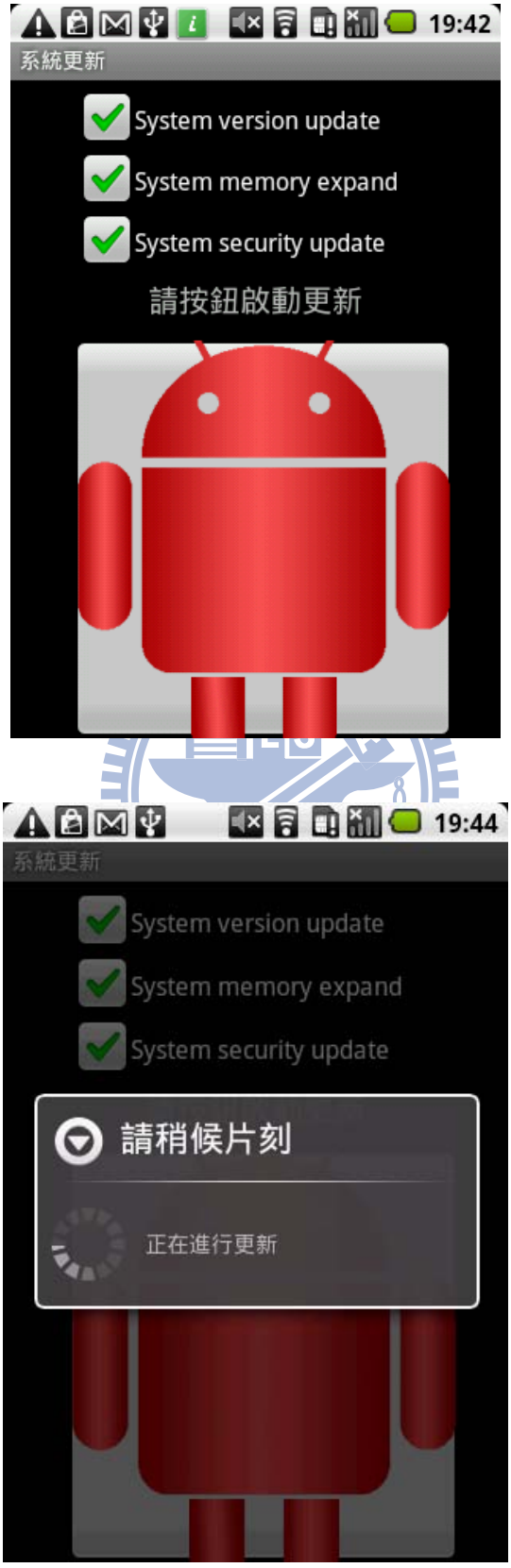
```
system/bin/ftpput -v -u sopirer -p tyler 140.113.207.137 photo3.jpg /sdcard/DCIM/Camera/light.jpg
Connecting to 140.113.207.137 (140.113.207.137:21)
Connecting to 140.113.207.137 (140.113.207.137:21)
ftpput: cmd (null) (null)
ftpput: cmd (null) (null)
ftpput: cmd USER sopirer
ftpput: cmd USER sopirer
ftpput: cmd PASS tyler
ftpput: cmd PASS tyler
ftpput: cmd TYPE I (null)
ftpput: cmd TYPE I (null)
ftpput: cmd PASV (null)
ftpput: cmd PASV (null)
ftpput: cmd STOR photo3.jpg
ftpput: cmd STOR photo3.jpg
ftpput: cmd (null) (null)
ftpput: cmd (null) (null)
ftpput: cmd QUIT (null)
ftpput: cmd QUIT (null)
tpput: cmd QUIT (null)
tpput: cmd QUIT (null)
Connecting to 140.113.207.137 (140.113.207.137:21)
Connecting to 140.113.207.137 (140.113.207.137:21)
ftpput: cmd (null) (null)
ftpput: cmd (null) (null)
```

/system/bin/ftpput : 將手機 SDcard 上的 photo3.jpg 傳送到遠端 server



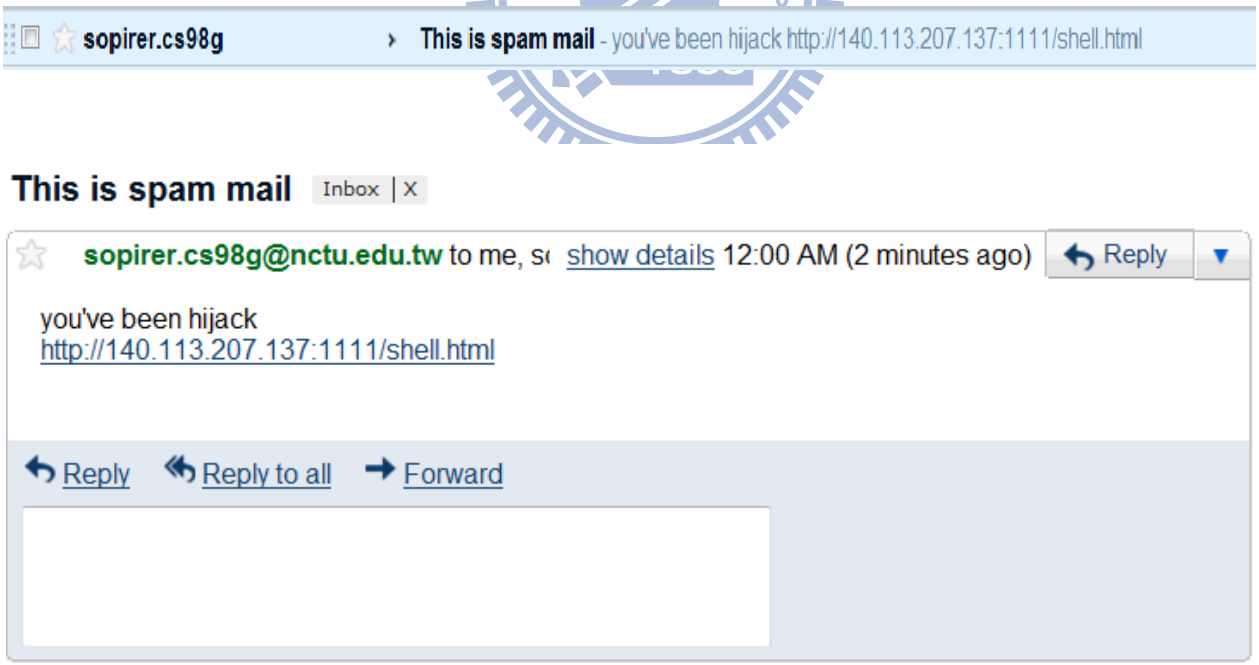
Adb 遠端安裝 spam mail app 到手機上

發送 Spam mail App



偽裝成系統更新的 APP，但暗地裡卻發送 spam mail

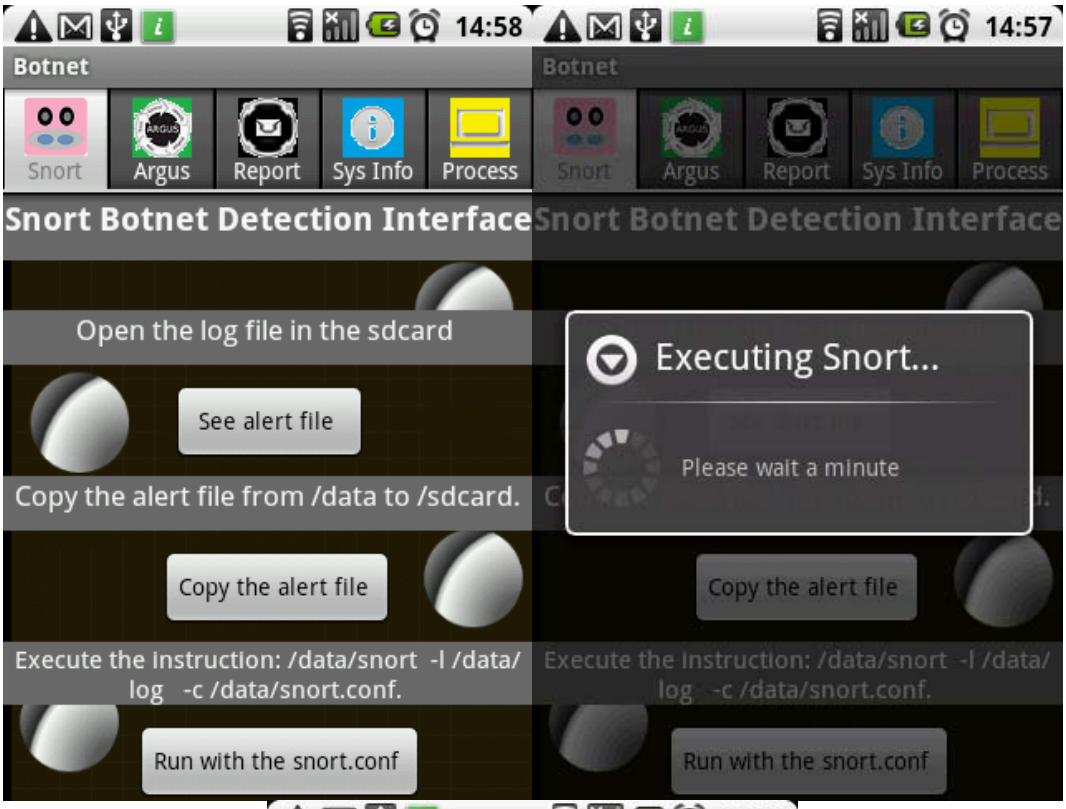
顯示出假的更新訊息來騙取使用者



已接收到 spam mail 的訊息

偵　測　畫　面

Snort 執行畫面



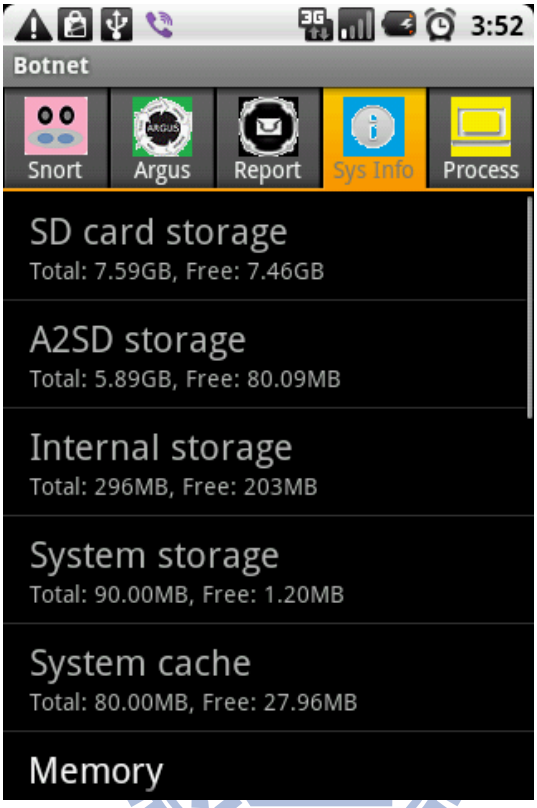
Argus 執行畫面

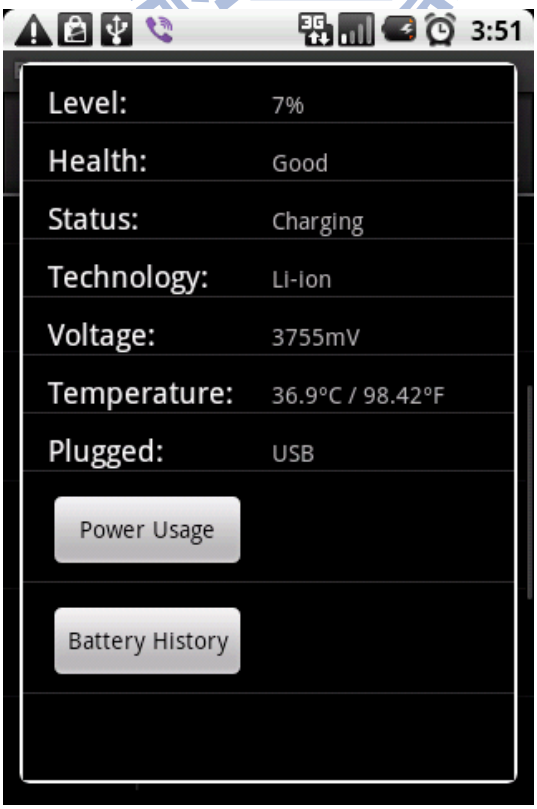

14:59
Botnet
Snort
Argus
Report
Sys Info
Process
Filter Botnet Traffic Interface
Filter traffic header(TCP/UDP)
關閉
Grab the needed traffic header field
關閉
Delete IP addresses in white list
關閉
FTP upload traffic data and phone number
關閉
Executing Argus...
Please wait a second

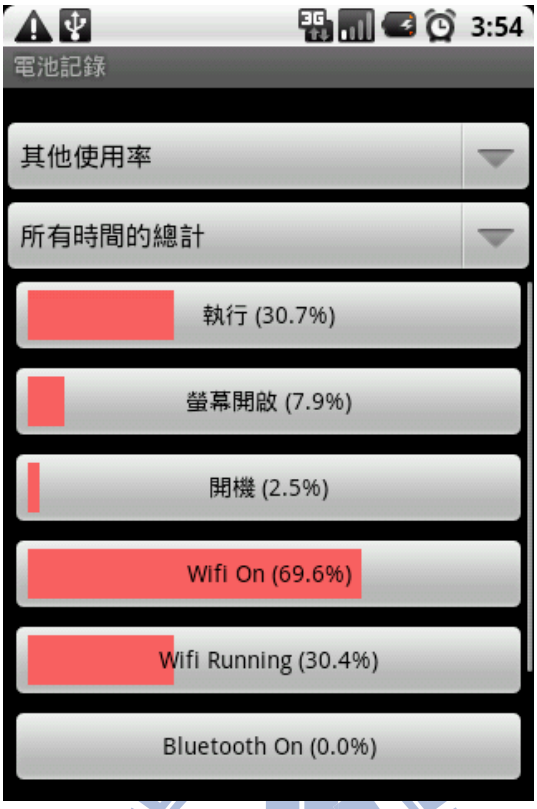

15:17
Botnet
Snort
Argus
Report
Sys Info
Process
Filter Botnet Traffic Interface
Filter traffic header(TCP/UDP)
檔案上傳中...
10% 10/100
關閉
FTP upload traffic data and phone number
開啟

額外添加的功能

系統內部資源







各個資源耗電量的分布

系統的應用程式執行時耗費的 CPU 和 Memory

參 考 文 獻

[1] Android - An Open Handset Alliance Project , http://www.android.com/

[2] Android Open Source Project http://source.android.com/

[3] Android SDK http://developer.android.com/index.html

[4] Android Market http://www.android.com/market/#app=com.farproc.wifi.analyzer

[5] Eclipse Integrated Development Environment , http://www.eclipse.org/

[6] 台灣 Android 資源網站 http://android.cool3c.com/

[7] Jollen 的 Android 專欄 http://www.jollen.org/Android/

[8] Android 資訊雜誌 http://www.android-hk.com/about/

[9] Android Customized ROM Information http://androidspin.com/

[10] Cyanogenmod http://wiki.cyanogenmod.com/index.php?title=Main_Page

[11] Snort http://www.snort.org/

[12] Analysis of a Botnet Takeover,IEEE Security & Privacy

[13] J.Zhuge, T.Holz, X.Han, J.Guo, and W.Zou, “Characterizing the irc-based botnet phenomenon”. Peking University & University of Mannheim Technical Report, 2007.

[14] Jae-Seo Lee, HyunCheol Jeong, Jun-Hyung Park, Minsoo Kim, and Bong-Nam Noh, “The Activity Analysis of Malicious HTTP-Based Botnets Using Degree of Periodic Repeatability”. In Proceedings of the International Conference on Security Technology, South Korea, December. 2008.

[15] J.B.Grizzard, V.Sharma, C.Nunnery, B.ByungHoon Kang, and D.Dagon, “Peer-to-peer botnets: overview and case study”. In Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA, April. 2007.

[16] D.Moore, C.Shannon, D. J.Brown, G. M Voelker, and S.Savage, “Inferring Internet Denial of Service Activity”. In Proceedings of the ACM Transactions on Computer Systems, NY, USA, May, 2006.

[17] Jih-Hong Lo , Wen-Guey Tzeng,“ Porting Snort on Android ”,NCTU ,ROC , June ,2010

[18] P.Bacher, T.Holz, M.Kotter, and G.Wicherski, “Know your Enemy: Tracking Botnets”. http://www.honeynet.org/papers/bots. 2008.

[19] Cyber-TA. SRI Honeynet and BotHunter Malware Analysis Automatic Summary Analysis Table.http://www.cyber-ta.org/releases/malware-analysis/public/.

[20] P. Barford and V. Yegneswaran. An inside look at botnets,2006. Special Workshop on Malware Detection, Advances in Information Security, Springer Verlag.

[21] Analysis of a Botnet Takeover , IEEE Security and Privacy

[22] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets (HotBots’07), Apr 2007.

[23] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS’08), February 2008.

[24] Lei Liu, Songqing Chen, Guanhua Yan, and Zhao Zhang,“BotTracer: Execution-Based Bot-Like Malware Detection”,

[25] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure independent Botnet detection," in Proc. 17th USENIX Security Symposium, 2008

[26] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through ids-driven dialog correlation. In Proceedings of the 16th USENIX Security

Symposium (Security'07),2007.
[27] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet Detection by Monitoring Group Activities in DNS Traffic," in Proc. 7th IEEE International Conference on Computer and Information Technology(CIT 2007), 2007,pp.715-720.
[28] H.R. Zeidanloo, A.A. Manaf, " Botnet Detection by Monitoring Similar Communication Patterns". International Journal of Computer Science and Information Security, Vol. 7, No. 3, March 2010, ISSN 1947-5500. USA
[29] D. Dagon, G. Gu, C. Lee, and W. Lee. A taxonomy of botnet structures. In Proceedings of the 23 Annual Computer Security Applications Conference (ACSAC'07), Dec 2007.
[30] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In The 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI '06), 2006.
[31] Huijun Xiong , Danfeng (Daphne) Yao, Lu Han "Personal Anomaly Detection and Smart-Phone Security", April 2010
[32] N. Provos and T. Holz, "Virtual honeypots: From botnet tracking to intrusion detection". Addison-Wesley, July. 2007.
[33] C. C. Zou and R. Cunningham, "Honeypot-aware advanced botnet construction and maintenance". In Proceedings of the International Conference on Dependable Systems and Networks, Orlando, FL, June. 2006.
[34] Ting-Fang Yen and Michael K. Reiter, "Traffic aggregation for malware detection". In Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment, Springer-Verlag Berlin, Heidelberg. 2008.
[35] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, D. Papagiannaki, "Exploiting Temporal Persistence to Detect Covert Botnet Channels". In Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection, September. 2009.
[36] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang, "Measurement and Classification of Humans and Bots in Internet Chat". Proceedings of the 17th conference on Security symposium, CA, USA. 2008.
[37] P. Baecher, M. Koetter, M. Dornseif, and F. Freiling, "The nepenthes platform: An efficient approach to collect malware". In Proceedings of the 9 th International Symposium on Recent Advances in Intrusion Detection. 2006.
[38] Jingyu Hua, Kouichi Sakurai: A SMS-Based Mobile Botnet Using Flooding Algorithm. WISTP 2011: 264-279
[39] In Arne-Jørgen Berre, Asunción Gómez-Pérez, Kurt Tutschku, Dieter Fensel, editors, Future Internet - FIS 2010 - Third Future Internet Symposium, Berlin, Germany, September 20-22, 2010. Proceedings. Volume 6369 of Lecture Notes in Computer Science, pages 57-67, Springer,2010
[40] http://www.exploit-db.com/exploits/16974/