



國立中山大學資訊管理研究所

碩士論文

Department of Information Management

Nation Sun Yat-sen University

Master Thesis

基於多層漸進式分群之惡意程式分析

Multi-layer Incremental Clustering for Malware Analysis

研究生：王奕凱

I-Kai Wang

指導教授：陳嘉玫 博士

Dr. Chia-Mei Chen

中華民國 104 年 9 月

September 2015

國立中山大學研究生學位論文審定書

本校資訊管理學系碩士班

研究生王奕凱（學號：M014020030）所提論文

基於多層漸進式分群之惡意程式分析

Multi-layer Incremental Clustering for Malware Analysis

於中華民國 104 年 8 月 4 日經本委員會審查並舉行口試，符合碩士學位論文標準。

學位考試委員簽章：

召集人 賴谷鑫 賴谷鑫

委員 陳嘉玫 陳嘉玫

委員 林耕霖 林耕霖

委員 楊竹星 楊竹星

委員 _____

委員 _____

指導教授(陳嘉玫) 陳嘉玫 (簽名)

誌謝

我要感謝我的指導教授陳嘉玫老師，老師她不論是在學業上或是作人處事上都是我最重要的老師。就像每個碩士生一樣，我在研究上遇到許許多多的難題，像是該怎麼挑選題目、該怎麼作出自己的研究貢獻、該怎麼撐過漫長難熬的研究過程等等，老師一直盡全力在幫助我。當我的研究內容走偏時，老師會不厭其煩的提醒我。當我在研究室待得太晚而影響生活起居時，老師會像媽媽一樣關心我。當我屢遭瓶頸灰心喪志時，老師會和我聊聊天放鬆心情。老師是我生命中的貴人，她不只以嚴謹的教學態度來指導我，還用和藹的關心來撫慰學生焦躁不安的心情。謝謝妳，老師，真的謝謝妳。

再來我要感謝土撥學長，學長總是可以一眼看出我的研究出了甚麼問題，然後立刻指出我該前進的方向。在研究過程中總是有許許多多困難，每當我絞盡腦汁也無法解決問題，土撥學長就是永遠的救星。沒有學長解決不了的問題。謝謝學長總是忍受我煩人的問題，沒有學長的話我絕對無法完成這研究。

當然還要感謝雅惠學姐、則堯學長、饅頭學長的各種指導。感謝季苹、文翎、嵩健的一同並肩作戰。最後要感謝我的父母在這段期間的支持，讓我可以完成我的研究。

摘要

惡意程式的威脅是現今資安的頭痛議題，隨著惡意程式的快速成長與變種，資安的防範手段也要跟上這個成長的速度才行。但資安人員總是只能在被動的立場去解決惡意程式的攻擊，當新的惡意程式展開攻擊時，資安人員才能想辦法去偵測，然後再去恢復受到攻擊的損害，最後再建構出有效的防禦機制。這是一場與時間的賽跑，每個環節都要以最快的效率來完成。

現今針對惡意程式的偵測有許多方式，坊間也有許多廠商所設計的防禦軟體。但大多都只把重點放在找出惡意程式上，很少有軟體會去找出惡意程式之間的關連，而這也是加速偵測惡意程式的關鍵所在。就算我們遇到了之前未曾碰過的惡意程式，如果可以找出它與之前碰過的惡意程式之間的關聯性，就可以更快的訂定出應對策略，減少損失。

本研究提出一個以偵測惡意程式之間關聯性為主的多層分群系統。主要是擷取惡意程式中的代表性特徵，包括各種程式碼檔中的指令結構、二進位檔的向量特徵、惡意程式中的結構特徵等。再利用兩種不同的分群演算法，改良漸進式分群和延伸式 1-NN，來將未知的惡意程式加以歸類，並且以多層分群的方式，分析惡意程式之間的家族關係。最後再與最具公正力的線上掃描網站 VirusTotal、知名防毒大廠 Avira 作比較，證實本研究可以作到更好的偵測效果。

關鍵詞：漸進式、惡意程式、靜態分析、多層分群、惡意程式家族

Abstract

The threat of malware is definitely the most important topic of internet security. As the growth of malware is faster ever and ever, the defense method of security must evolve. Unfortunately the IT expert only can start to deal with attack problem after the new malware have already invaded our system. The usual steps for malware attack issue is to collect the evidence first. Then the IT expert can analyze these evidence to find out the solution. At last, we need to improve our system in case that there will be another malware attack.

In this paper, we propose a malware analysis system to accurately cluster new malware. We extract the significant feature from malware sample. For source code file, we extract the syntax string as the feature. For binary file, we transform the binary file to image file, and extract the matrix vector from the image as the feature. Then we adopt two different clustering algorithm, advanced incremental clustering and extended 1-NN, to cluster our malware sample. Finally, our system can offer a detailed report about the malware family relationship. In our research, there are four experiments to verify our system. We compare the performance and accuracy about the two different clustering algorithm, and verify the system's maturity with random sample analysis order. We also compare our system with Virustotal.com and Avira software, and the result confirms that our system can do better efficient clustering.

Keywords : incremental clustering, malware detection, static analysis, malware family,

目次

第一章 緒論.....	1
1.1 研究背景.....	1
1.2 研究動機與目的.....	2
第二章 文獻探討.....	4
1.1 惡意程式分析.....	4
1.2 惡意程式家族的偵測方式.....	7
第三章 研究方法.....	9
3.1 系統架構與流程.....	10
3.2 相似度公式.....	23
3.3 分群演算法.....	27
第四章 系統實驗.....	35
4.1 樣本收集與觀察.....	35
4.2 實驗一 改良漸進式分群與延伸式 1-NN.....	37
4.3 實驗二 調換樣本順序以驗證本系統的分群效率.....	44
4.4 實驗三 本系統與 VirusTotal 之比較.....	51
4.5 實驗四 本系統與 Avira 防毒軟體之比較.....	56
第五章 結論.....	59
參考文獻.....	60

第一章 緒論

1.1 研究背景

資訊安全絕對是現今網路世界的一大議題，在我們享受網際網路帶來便利的同時，也要想辦法應付網路駭客在暗中的蠢蠢欲動，而駭客所使用的惡意程式更是讓使用者防不勝防。惡意程式主要包括病毒(Virus)、蠕蟲(Worm)、木馬程式(Trojan Horse)。以前的惡意程式主要都是著重在破壞，但現在人們愈來愈依賴電腦網路，所以使用者的電腦中會存放愈來愈多的重要資訊，而對於駭客來說，竊取這些資訊是有利可圖的，這也導致現今的惡意程式大多都是以控制和竊取資訊為主要目的。

惡意程式所代表的不單單只是駭客用來攻擊的工具，它是一個足以牽扯到數億美元的產業市場。在 2013 年底，就有大約 4 千萬左右的信用卡資料在網路黑市中流通[1]，這些資料都是網路犯罪份子藉由各種惡意程式所竊取而來的，這些帳號資料的外洩所造成的財物損失是無法想像的巨大。而惡意程式的產值還不止於此，在網路黑市中可以買到各式各樣的攻擊工具，從簡單的蠕蟲到昂貴的零時差漏洞攻擊工具都有。這也讓駭客的目的不再只是癱瘓資訊系統，而是想盡辦法從目標身上竊取有價值的資料。

目前幾乎所有的伺服器都會再加裝防火牆、防毒軟體等防禦的機制，而網路犯罪分子為了要入侵主機，一定會去使用後門程式或是混淆技術來讓入侵行為成功，而混淆技術對惡意程式所造成的變種，讓惡意程式種類增加的速度愈來愈快，種類也愈來愈多。就因為攻擊手段變化速度非常快，防禦的難度也愈來愈高。使用者面對這麼多種攻擊手段，被入侵的機率也大幅提升。

不單只是一般個人電腦中的使用者資料，企業組織的資訊系統中所存放的各種資訊更是眾多駭客的首要目標。根據賽門鐵克的統計[2]，2013~2014 之間

在美國就有 12 起重大資料外洩的資安事件，每個資安事件都牽扯到超過 1 千萬筆的資料外洩。而另外規模較小的資料外洩事件，2013 年就有 253 件，2014 年就有 312 件，兩年外洩的使用者資料加總起來更是超過了 3 億筆。這些使用者資料中除了使用者在網路社群上會用到的各種帳密、信用卡資料，還有使用者在真實世界中的居家資料、社會安全碼等。不難想像這些資料一但落入有心人士手中會造成多麼大的損害。

就因為惡意程式的利益價值如此可觀，使得駭客們更汲汲營營於設計出更新、更難被偵測的惡意程式。惡意程式的成長速度非常快[3]，在 2013 年就有 2 億五千萬個新品種惡意程式被偵測出來，在 2014 年就有 3 億 1 千萬個新品種惡意程式被偵測出來。這些新的惡意程式有些是全新的品種，有些是從舊的惡意程式中加以變化的。面對這麼多惡意程式的威脅，資安人員必須快速地分析各種攻擊事件，判斷出這是藉由哪種惡意程式的發起的攻擊。這個惡意程式可能以前偵測過或是可能先被誘捕系統收及過，這樣的話就能大幅降低分析時間，但如果是新發現的惡意程式，就必須用較多的時間去分析並訂定接下來的預防措施。

1.2 研究動機與目的

惡意程式的定義就是網路犯罪份子為了達到某些非法目的而設計出來的程式工具，其內含的程式碼都是用來破壞、竊取或是非法獲利的。這些年來，網路犯罪份子利用惡意程式所造成傷害是愈來愈高，其中一個原因就是 3C 工具的普及和網路使用人口的激增。如同金融詐騙事件的頻傳是來自於大眾對於金融體系的不熟悉，網路犯罪的成長速度愈來愈高也是由於大眾只看到網路表面上的便利，而沒有去詳細理解身在網路世界該了解的規範和風險。為了應付層出不窮的網路攻擊，資安人員必須和駭客們賽跑，必須在防禦機制上提高效率 and 準確率。

分析惡意程式有非常多的方式，大致上可以區分為動態分析和靜態分析。動態分析主要是主動去讓惡意程式執行，執行的期間觀察惡意程式對內的程式碼的執行動作，或是惡意程式對外做的行為，再做進一步的分析。靜態分析不會去主動執行惡意程式，而是去分析惡意程式裡的檔案組成、程式結構。

本研究是以程式原始碼和二進位檔的分析為主，提出一個對惡意程式進行多層分群的方法，藉此改善對惡意程式分群上的準確度。本研究的方法主要分為以下四個步驟。第一，對於誘捕系統所收到的惡意程式，本研究會先檢查是否有被壓縮過，如果有壓縮的情況，會自動化解壓縮再抽取本研究需要使用的原始碼。第二，從惡意程式內的程式原始碼取出所需要的特徵值來進行比對。第三，未歸類的惡意程式會使用上述步驟所採集的特徵值與系統目前已知惡意程式樣本的特徵值進行比較，以決定未知的惡意程式與已知的惡意程式群集之間的相似度。這邊所採用的分群演算法分別有兩種：改良漸進式分群和延伸式1-NN，根據分別使用這兩種分群演算法所得到的分群結果來比較其優劣。第四，在分群過程中會經過多層次的分群演算，多層分群除了可使準確率提高，也可推斷出群集與群集之間的相似程度。最後再根據惡意程式群集之間的關係，整理出惡意程式的家族關係。

本研究主要貢獻在於提出一套以多層漸進式分群來分析惡意程式的系統，並且能夠依據理論實作出系統。藉由先對未知的惡意程式進行分群，整理出惡意程式的家族關係，便可以加速資訊安全相關人員對所蒐集到的未知惡意程式初步的了解，可以讓他們知道此惡意程式是已知類型的變種或是新類型，來去決定是否需要進一步針對此惡意程式進行分析，因此可以避免需要對未知惡意程式進行分析的必要性。

第二章 文獻探討

2.1 惡意程式分析

分析惡意程式一直都是分成兩個方向來著手，靜態分析和動態分析。靜態分析著重在惡意程式的內容，針對其中的二進位檔和程式碼檔的內容作分析。動態分析則是會藉由觀察惡意程式在虛擬機器或砂盒中的行為，記錄它對主機中的系統檔案或記憶體的影響。兩者各有優缺點，也有很多研究是根據這兩個方向去分析惡意程式。

動態分析由於只要根據惡意程式的行為來作分析，所耗費的時間較短，也不用擔心惡意程式經過模糊化或加殼的處理後會難以辨認其內容，只要出現可疑動作即可進行分析。H. Bai[4]等人以相當典型的動態分析來偵測惡意程式，特別的是他們建立了 MBO(malicious behaviours and outcomes)資料庫來記錄惡意程式的動態特徵，並且根據這個資料庫去計算每個樣本的特徵矩陣，最後再以 SVM 來分類不同的惡意程式。MBO 結構可以清楚的針對惡意程式在沙盒中的行為而確定其為哪種惡意程式類別，這種行為與結果的連結是相當具有鑑別性的。

S. Wen[5]等人針對作用在電子郵件中的惡意程式作動態分析，他們相當創新地將這類惡意程式中的特徵定義成再感染和自我感染這兩類，並且設立虛擬的使用者來模擬惡意程式的行為，這讓分析人員可以從電子郵件的多次傳遞中找出惡意程式的繁殖細節。

L. Cen[6]等人分析 Android 行動裝置上惡意程式的執行權限，並且反組譯其中的程式碼，找出 API 特徵，從這兩方面去分析。他們額外採用了 F1 標準來取代較多研究所使用的 AUC 標準來評分惡意程式的分類程度。

靜態分析主要是針對惡意程式的內容，若有二進位檔則會透過反組譯來將

難懂的機器語言轉換成組合語言，再作指令字串的分析。

n-gram 就是很常被應用的一種字串相關的演算法，n-gram 則為一種語言中立的斷詞方法，依照所取的 n 值，在二進位檔案中從任意位置開始取出連續的 n 個位元組，利用統計方式從這些眾多的斷詞中取出最有關聯性的樣式來代表惡意軟體的特徵值。D. Yuret[7]基於 n-gram 的概念而設計出一個新的演算法，FASTSUBS，這個演算法不但會計算文章之間 K 個最常出現的子字串，還會將每個單字的長短考慮進去，有了更多的參考一句可以讓計算相似度的過程大幅簡化，讓他們再處理擁有百萬個以上單字的文章時可以比以往快上 30 倍以上。N. Beebe[8]等人只採用 n-gram 中的 unigram 和 bigram 兩種斷句方式，搭配 SVM 的處理來找出關聯性，他們一共實驗了 30 種檔案形態和 8 種文字形態，比較兩種不同核心的 SVM，讓誤判率降到最低。N. Jain[9]等人為了能夠清楚辨認兩群檔案叢集，先根據檔案群中控制流程圖的先後順序訂定清楚，再配合 n-gram 的字串擷取，這樣才能在正確的順序中判斷相似與否。

在二進位檔案分類的研究上其中一種方式是使用 Windows 作業系統上的惡意軟體二進位檔(Portable Executable binary code)抽取出 API call 當作特徵值，惡意軟體會因為其行為的不同，而去呼叫不同的 Windows API，所以可以成為區分惡意軟體類型的一種指標，像是 Ye 等人利用程式碼所出現的 API 序列關聯性，運用資料探勘的方式分類出良性與惡意軟體。Shankarapani[10] 等人雖然也同樣採用 API call，但是他們不考慮 API call 的序列，而是採用 TF-IDF(term frequency - inverse document frequency)方式計算特定 API call 出現於程式碼內部的頻率來當作特徵向量，而再使用支持向量機的方式來去分類。

Cesare[11] 等人是採用 X86 的組合語言指令建立控制流程圖，再將流程圖轉為特徵值字詞，再以編輯距離的方式計算兩個二進位檔案的相似度。

Kang[12]等人是使用 IDA Pro 軟體產生控制流程圖，再將圖內每個節點所包含

的指令抽取出當作特徵字詞，最後再將這些字詞以 hash 方式轉化為特徵向量進行比對。Agrawal[13]等人則是專注於處理多態(polymorphic)的控制流程圖間的比對。

最後一種方式為將二進位檔轉化成圖像檔，然後再從圖像檔裡面抽取出特徵值進行比對。Conti[14]等人首先提出將二進位檔內的二進位碼轉換成灰階圖像格式，藉此可看出每一個不同類型的二進位檔在影像圖形呈現上也不相似，而同一個類型的圖形結構上極為類似。而 Nataraj[15]等人將此應用於惡意軟體二進位檔案的分類上，將轉化後的二進位檔灰階圖形再以 GIST 影像特徵方式擷取特徵向量，再以歐氏距離去計算兩個二進位檔間的距離來分類。

Louk[16]等人希望可以結合動態和靜態的分析方式，他們以靜態的方式從 PE 類檔案中擷取出 DOS 標頭和 PE 標頭等特徵，為了避免因模糊化技術而影響特徵擷取，又另外針對 PE 檔的行為模式作分析，以這兩方面下去作分類演算和訓練。Choi[17]等人同樣也是結合動態和靜態，不過他們設計出新的分析架構，GATTACA，他們將惡意程式的靜態特徵和動態特徵稱為 Mal-DNA，靜態特徵的擷取較單純，他們採用 ssdeep 和 ClamAV 等工具作擷取，動態特徵則使用 Procmon 和 WinDbg debugger，目的就是希望能融合兩方面的優點。

雖然現今有許多研究對惡意軟體二進位檔案的靜態分析取得不錯的成效，但是仍然會受到惡意軟體二進位檔案使用模糊化的技術而限制。模糊化的技術包含為二進位檔加殼(Packed)、插入無用的代碼(Dead code insertion)、控制流程模糊化(Control flow obfuscation)、重新指派暫存器模糊化(Register reassignment Obfuscation)等方式[18]，造成 API call、CFG 或是 n-gram 等方法在特徵擷取上會產生出無用的特徵值而導致判斷出現誤差，藉此躲避惡意軟體二進位檔案的靜態分析偵測。雖然有研究開始將目標致力於反模糊化，也

就是忽略或是去除這些模糊化所產生的程式碼，Cesare[11]等人的研究就在進行特徵擷取之前先做反加殼的前處理行為，但是隨著模糊化的技術不斷地進步而出現新型變種時，舊有的反模糊化方式可能會因此不適用而失敗，造成分類的正確率降低。至於二進位檔轉影像檔的分析方式，因為不需要對二進位檔進行反組譯等動作來還原成組合語言等程式碼，因此不必要進行反模糊化的動作，這會比 API call、CFG 和 n-gram 方式還要快速、花費成本少且有延展性，因此本研究採用此方式來進行惡意軟體內二進位檔的分析。

2.2 惡意程式家族的偵測方式

每個不同的惡意程式之中，常常會有擁有類似的動作，像是掃描網路區段中的現存 IP、清除 log 記錄檔、掃描主機裡有開啟的連接埠等等，甚至可以模組化這些功能，讓其他惡意程式也可以引入相同的功能。而這些類似的程式碼常常會被防毒軟體用來當作辨識的特徵，為了躲避防毒軟體的偵測，駭客會將惡意程式複雜化，複雜化的方式有多型(polymorphic)和變形(metamorphic)。多型是利用加密技術來偽裝惡意程式，變形是將惡意程式中的空白區塊任意調換。

雖然惡意程式種類的成長速度飛快，但其中仍有很多是藉由舊的惡意程式變種而來。當資安人員偵測到未知的惡意程式，如果可以用低成本的分析技術來確認這是和某個已知惡意程式隸屬相同家族之中，將可以省下很多時間成本。Hesham Mekky 等人是利用 Independent Component Analysis (ICA)來解析惡意程式對外部的複雜流量記錄，從多重交叉的信號中擷取出有效的特徵，再根據特徵來將樣本分類到正確的家族分類中[19]。

Jeff Gennari 和 David French 針對惡意程式的程式碼來作特徵值擷取，事先建立一個特徵資料庫並且設定好標準 bootstrap criteria，接下來的惡意程式家族成員的鑑定都根據上述標準[20]。

Yang Zhong 和 Hirofumi Yamaki 等人利用 LCSs 來擷取惡意程式中的特徵值，再利用 ARIGUMA 分群演算法來進行樹狀分群，每個群集的特徵值也是以樹狀結構來表示，群集的代表特徵有函式呼叫、迴圈複雜度等等樹狀資料結構的森林。最後的以 ARIGUMA 把樣本群分成一棵一顆的樹狀結構[21]。

為了對付越來越多的惡意程式模糊化技術，Blanc[22]等人設計出一套反模糊化的演算法，為了對付肆虐於 Web2.0 上的 XSS 蠕蟲和僵屍網路，他們將所要偵測的目標的網頁程式語言改寫，像是 ActionScript、VBScript 或 JavaScript，這是少數用靜態分析的方式去對付靜態分析最頭痛的模糊化技術。

第三章 研究方法

本研究的最主要的目的就是建立一個以靜態分析為架構的惡意程式多層分析系統，將惡意程式的內容仔細分析後，找出惡意程式的特徵，整理所有特徵並比對特徵之間的相似程度後，計算惡意程式之間的相似程度，再藉由分群演算法將相似的惡意程式分為同一群集，並藉由多層分群來界定群集之間的變種關係，找出惡意程式中的家族體系。

每個惡意程式都有其獨特的特徵，這些特徵可以在惡意程式中的二進位檔或程式碼檔裡整理出來，如果可以在兩個惡意程式之間找到類似的特徵，就可以定義這兩個樣本是屬於同一種類。但是有很多惡意程式在設計時，通常都會在舊的樣本中額外加上一些不同的功能，所以新設計出來的惡意程式比起舊的惡意程式會有一些相同的特徵，也多出一些不同的特徵，這種情況就是所謂的”變種”。經過變種的惡意程式有著各自獨特的特徵，彼此之間卻又有部份相似的特徵，這些惡意程式之間的關係就可定義為家族的關係，這也是本研究最大的特色之一，就是找出每個惡意程式群集之間的家族關係。

本系統針對每個惡意樣本進行詳細的靜態分析，從惡意程式中的二進位檔、程式碼檔和檔案結構中擷取出能夠代表此惡意程式的特徵值。二進位檔可以藉由轉化為灰階圖片後再找出其特徵矩陣來作為計算依據，程式碼檔則可擷取出程式碼中的重要指令，組合成一串特徵字串來作為計算依據，檔案結構則是去整理惡意程式中每種檔案類型的數量，來作為計算依據。擷取出每個惡意程式的特徵後，即可藉由相似度公式來找出兩者之間的相似程度，相似度公式包括計算字串特徵值和向量特徵值。接下來再透過改良漸進式分群和延伸式 1-NN 這兩種分群演算法將惡意程式進行分群，在分群過程中會透過多層分群來找

出群集之間的相似關係。最後整理出惡意程式群集之間家族關係，以提供給使用者參考。以下就針對本系統的每個處理流程一一作解釋。

3.1 系統架構與流程

本研究從架設在學術網路的 honeypots 上收集惡意程式，將惡意程式送進本系統作分析。首先對惡意程式作前處理，針對惡意程式的封裝格式作相對應的解壓縮，且先將其中的二進位檔轉化為灰階圖片，以加速下一階段的特徵擷取。再來針對惡意程式的內容擷取出特徵值，包含樣本中程式碼檔的語法結構、二進位檔的特徵向量、第二層目錄的檔案結構，將每項特徵值記錄下來。接下來開始計算該惡意程式和系統中每個群集中心點的相似程度，將先前記錄的特徵值帶入相似度公式作計算，一旦相似度達到閾值，即可將該惡意程式歸類到該群集中。最後再整理出群集之間的家族或變種關係。圖 3-1 為本系統架構圖。

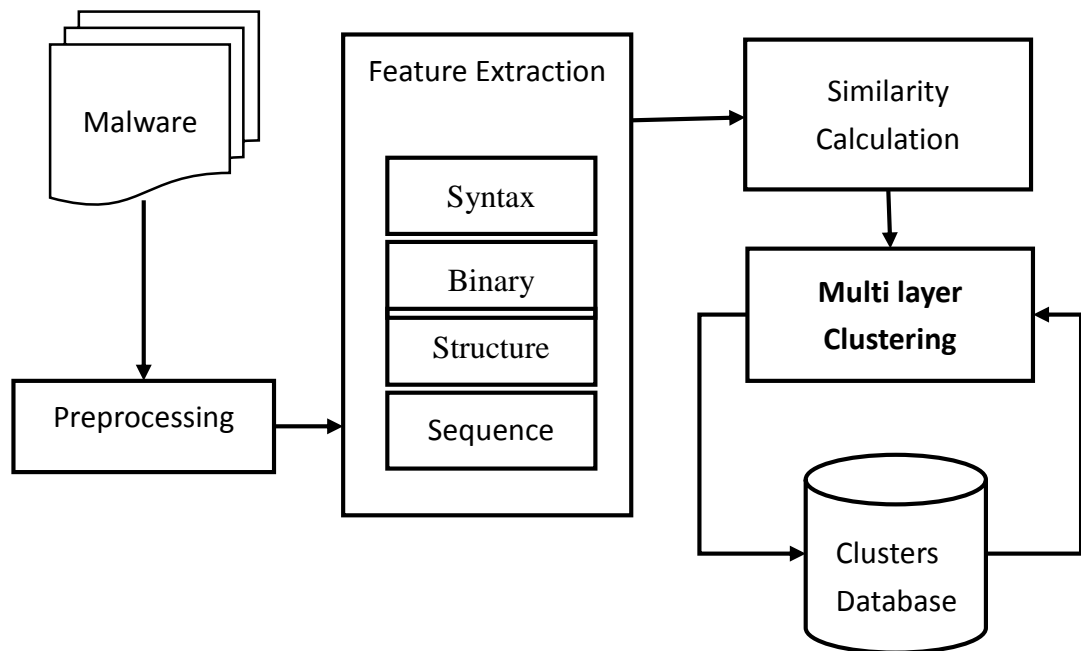


圖 3-1 系統架構圖

在本研究的系統中有五個主要階段：Preprocessing、Feature extraction、Similarity Calculation 和 Clustering。接下來會就各個階段做詳細說明。

<1> Malware：

本研究所分析的惡意程式來源主要是學術網路中所架設的 Honeypot，透過 Honeypot 就可以捕獲到網路犯罪分子所使用的惡意程式，或是藉由 Honeypot 中的日誌所記錄的犯罪分子使用的 URL，並自行透過此 URL 來收集樣本。由於本研究所使用的 Honeypot 都是架設在 Linux 作業系統中，所以本研究的惡意程式樣本也都是針對 Unix 系統架構所設計的惡意程式。

收集到的樣本主要會有兩種形式存在，單一檔案和惡意組件包

1. 單一檔案：

這類型檔案內容就只有單一檔案的存在，主要是為 perl 程式碼檔或是二進位檔，用來執行 DDos 或感染為 Bot 的單一動作。

2. 惡意組件包：

惡意組件包(malware package)指的是內含數個不同惡意程式的封裝檔，而本研究所收集到的樣本有九成以上都是屬於這類的惡意組件包。惡意組件包的內容包括 shell script 檔、c 程式碼檔、perl 程式碼檔、二進位檔和各種文字檔，將這些檔案封裝成壓縮檔的格式以方便傳輸，圖 3-2 是其中一個惡意組件包的內容。shell script 檔主要作為執行內部指令和執行惡意程式內的二進位檔。c 程式碼檔和 perl 程式碼檔為惡意程式內二進位檔的原始碼，或是用來編譯執行以產生惡意行為。二進位檔主要進行惡意程式的主要行為，如網域掃描、暴力破解、殭屍網路主程式等作用。其餘的文字檔包括惡意程式的說明文件、設定檔、或是字典檔等用途。

名稱	類型	大小
randfiles	檔案資料夾	
bash	檔案	577 KB
cyc.acc	ACC 檔案	1 KB
cyc.help	HELP 檔案	22 KB
cyc.levels	LEVELS 檔案	2 KB
cyc.pid	PID 檔案	1 KB
cyc.set	SET 檔案	2 KB
go	檔案	1 KB
pico	檔案	165 KB
pico.tgz	TGZ 檔案	83 KB
stealth	檔案	14 KB
udp.pl	PL 檔案	2 KB

圖 3-2 解壓縮後的惡意組件包範例

<2> Preprocessing :

在惡意組件包的前處理這部分中，主要有兩部分：解壓縮和二進位檔的圖像轉換。解壓縮部分是為了針對各種不同的封裝壓縮格式作處理，而二進位檔的圖像轉換則是下一階段的二進位檔特徵擷取的前置作業，這樣的前處理主要是為了讓接下來的特徵擷取更為容易，也讓本系統在功能層面上可以模組化。

1. 解壓縮：本研究所收集到的惡意組件包大多是壓縮檔的格式，有些雖然是壓縮檔但副檔名有修改過，有些則是只有單一檔案。系統在分析前會針對可以解壓縮的惡意組件包解壓縮。解壓縮後系統會對壓縮檔所還原之目錄內的程式碼檔先進行前處理的動作。對目錄內的檔案檢測是否為 C、Perl、Shell Script 所寫成之程式原始碼，若為上述之三種程式原始碼之一，則會進一步對該原始碼進行註解去除動作，並針對不同語言的程式碼檔在下一階段會有不同的特徵選取方式。

2. 二進位檔的圖像轉換：惡意組件包中的二進位檔幾乎都是由 C 語言程式碼所封裝而成的執行檔，除非是經過反組譯程序，否則系統通常無法直接檢視其中內容，而反組譯的過程相當耗時，其所得到的結果也相當不穩定，如果駭客在撰寫惡意程式時故意做了模糊化，那反組譯的結果將會大大誤導接下來的分析，所以本研究將二進位檔轉化為另外一種可以互相比較的格式。本研究參考 Nataraj[15]的方法，將二進位檔中的二進位碼量化為數字，而每個數字則代表灰階圖形的色彩，如此就可以形成一張灰階圖案。將這張圖案經過傅立葉轉換成方塊矩陣。這個矩陣即可當作量化比較的依據。

<3> Feature extraction :

將惡意組件包做完前處理後，本系統會針對惡意組件包中不同的檔案類型作不同的特徵擷取。組件包中所含的檔案類型有程式碼檔、二進位檔、純文字檔和子資料夾。這其中本系統指會針對程式碼檔和二進位檔作詳細的分析，純文字檔通常都是字典檔或是說明文件，並不具有特徵意義，所以本系統不會特別去分析純文字檔。

而程式碼檔中又包含了 C 語言檔、PERL 檔和 Shell Script 檔，系統會擷取出程式碼中的特定指令作為特徵字串，藉以代表這個程式碼檔的特徵。不過這三種程式語言都有各自獨特的指令和撰寫方式，本系統會針對不同種類的程式碼檔作合適的特徵擷取。二進位檔的特徵擷取比較特殊，由於二進位檔的內容無法直接以文字的方式顯示出來，所以本系統採用 Nataraj[15]的方法，將每個二進位檔轉化為灰階圖片檔，再將灰階圖片透過傅立葉轉換成矩陣向量，再去比較向量之間的差異。

除了程式碼檔和二進位檔的特徵之外，本系統會針對惡意組件包內的檔案結構作分析，將各種不同類型的檔案數量記錄下來作為特徵向量。不過此處只分析惡意組件包內的第二層檔案結構。由於惡意組件包的第一層

檔案結構都非常類似，各種檔案類型的數量差距很小，所以不加以分析。而第三層以上的檔案結構則幾乎不存在，大部分的惡意組件包都只有兩層檔案結構，所以第三層以上也不考慮。

總而言之，本系統所著重的惡意程式特徵為二進位檔的內容結構、程式碼檔的語法結構和惡意組件包的組成結構。表 3-1 就是本系統針對惡意程式所截取的六種特徵。接下來詳細解釋每種特徵的擷取方式。

特徵名稱	特徵來源
1. 總體二進位檔相似度	二進位檔
2. 呼叫過二進位檔相似度	二進位檔
3. C 程式碼檔相似度	C 語法結構
4. Perl 程式碼檔相似度	Perl 語法結構
5. Shell script 檔相似度	Shell 語法結構
6. 第二層的檔案結構相似度	檔案結構

表 3-1 惡意程式的特徵分類

- 語法結構：**惡意組件包中的程式原始碼包含了 C、Perl 和 Shell script 三種程式碼檔，本系統會將程式碼檔中的語法結構和特殊指令擷取出來，擷取出來的每個單一指令之間再用分號連結起來，就可以串連成一系列特徵字串。至於擷取的指令範圍不是 100%全部取出，事前會先經過多次的人工觀察，將每個惡意組件包的程式碼內容詳細分析過後，再決定哪些指令是有擷取出來的價值，畢竟有些指令所造成的影響會造成本機系統某種程度的改變，而這種行為就值得懷疑它有惡意的用途。另外有些指令在每個程式碼中出現太過頻繁，如果這也要擷取出來的話則會讓特徵字串的獨特性降低，所以本系統只針對部份特殊的指令和語法結構作擷取。接下來針對三種程式語言來解釋不同

的擷取方式。

● C 語言：

C 語言在語法和撰寫方式上有著非常制式化的特色，這讓 C 語言的程式碼整體看起來很整齊，也很容易去比對。就算是不同的人用不同的邏輯去設計同一目的的 C 語言程式碼，在語法結構上還是會顯得非常類似，如果換成是 JAVA，則會因為語法太過自由而有很多種不同的寫法，因此在原始碼相似度計算上可以利用此特性來進行比對，語法結構可分為：

- 循序結構：主要為程式碼內部出現的指派(assign)語法來當作特徵值，如 “=”。
- 選擇結構：取 if、if-else 等條件判斷語法來當作特徵值。
- 重複結構：取出 for 和 while 兩個迴圈語法來作為特徵值。
- 比較運算子：取出 “==”、“!= ”、“>”、“<”等比較運算子來作為特徵值。
- 函式實作名稱與函式呼叫名稱：一個函式在實作時會因所呼叫的其他函式種類、順序而會與其他函式產生不同的組合結構，所以取出函式所實作之名稱，以及函式內部實作所呼叫之其他函式名稱來當作特徵值。
- 程式區塊：程式內部在函式實作或是判斷式的區塊範圍，通常都會以左大括弧 “{” 來表示程式區塊的起始點，而以右大括弧來表示程式區塊的結束點。因此取程式內部的對稱大括弧來當作特徵值。

表 3-2 為從 C 語言程式碼中擷取特徵值的範例。

code	Feature
<pre>int function_name_1(x, y) { int i = 0; int j =1; /* test */ if(i < j) { function_name_2(i); i = i + 1; } }</pre>	<pre>function_name_1(;); {; =; =; if(;<;);function_name_2(;);=;+; }; }</pre>

表 3-2 從程式碼所擷取的特徵值

● Perl

Perl 語言在語法結構上比起 C 語言自由許多，不過在段落的結構上則是和 C 語言相差無幾，一些重要的指令也與 C 語言類似，因此針對 Perl 語言所選取的特徵值與上述 C 語言的情形大致是一樣的，但是因 Perl 自身所具有的特殊語法仍需新增一些特徵值如下：

- 選擇結構：if-elsif、unless
- 重複結構：foreach
- 比較運算子：~=

● Shell Script

Shell Script 在某些語法結構上和 C 語言類似，像是迴圈語法，但是 Shell Script 和 C 語言最大的差異在於 Shell Script 沒有段落的形式。Shell Script 在撰寫上通常是將許多指令彙整在一起，在執行上只要從頭組譯到尾就可以了。如果 Shell Script 需要作到類似呼叫函式之類的功能，則會另

外寫一個 Shell Script 程式碼檔去做呼叫用。使用者只需執行 Shell Script 的原始碼便能處理系統內複雜的動作，以及執行該惡意程式所含有的二進位檔案，如圖 3-3 就是一隻執行 pscan2 二進位檔進行網段掃描，和執行 sshf 二進位檔進行暴力登入，最後以 mail 方式寄送暴力登入結果的 Shell Script 程式碼，圖 3-3 紅框中的指令就是本系統從 Shell Script 中擷取的指令，擷取出來的指令字串就如圖 3-4 所示。因此 Shell Script 的特徵值選取除了跟上述 C 語言一樣之外，還需選取 Linux 的指令，以及執行二進位檔案的程式碼片段來作為比對。

```
#!/bin/bash
if [ $# != 1 ]; then
    echo " usage: $0 <b class>"
    exit;
fi

./pscan2 $1 22

sleep 10
cat $1.pscan.22 | sort | uniq > mfu.txt
oopsnr2=`grep -c . mfu.txt`
echo "# SA VEDEM CE PULA MEA FACEM"
echo "#      _\ _____) \_____ "
echo "#      (_) [ _bY_ ] {} <MaLa> "
echo "#      /      ) _/      "
echo "#.....si DE root ..... "
echo " "
echo -e "Checking\033[1;34m user file\033[0m pass 1"
cp 1 pass_file
./ssh-scan 100
sleep 3
echo -e "Checking\033[1;31m root file\033[0m pass 2"
cp 2 pass_file
./ssh-scan 100
```

圖 3-3 shell script 程式碼內容

```
if ; ./ ; cat ; sort ; uniq ; cp ; ./ ; cp ; ./ ;
```

圖 3-4 圖 3-3 之程式碼所擷取的特徵值

但是本系統並沒有把所有的 Shell Script 指令全部擷取出來，像是圖 3-

3 中的 sleep、echo、cp 等指令就沒有被擷取出來。這是因為有些 Shell Script 指令出現頻率過高，如果擷取出來的話會讓特徵字串的鑑別性下降。並且本研究在取捨該擷取哪些指令時，會根據過去觀察其他 Shell Script 惡意程式的經驗，來判斷哪些指令才是真正有惡意攻擊嫌疑，至於其他和惡意攻擊無關的指令就會捨棄不用。大部份有惡意攻擊嫌疑的指令都是會改變本機系統檔案的指令。

2. 二進位檔之檔案特徵：

壓縮檔內的二進位檔特徵值在本系統中主要採用 Nataraj[15]的方法來去擷取，主要流程如下圖 3.10 所示：

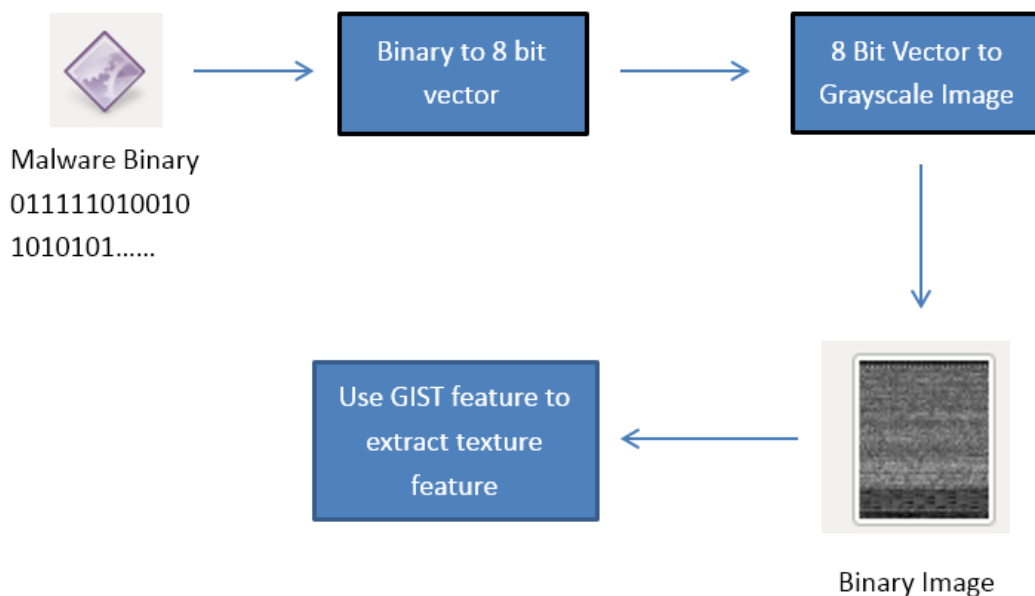


圖 3-5 二進位檔案特徵擷取流程圖

- Binary to 8 bit vector :

主要將一個二進位檔內的二進位碼轉換為 8bit 為一個元素的向量。像是一個二進位檔為 0101111101010101.....，就會以 8 個 bit 來進行切割，就變為 01011111, 01010101,，最後會再轉為十進位的方式為 95, 85,

的型態儲存至一個一維陣列。

- 8 Bit Vector to Grayscale Image :

上述一維陣列所儲存的數字代表著灰階圖形的色彩，範圍是 0~255，0 為黑色、255 為白色。一維矩陣在此步驟會轉換成二維矩陣，以形成二進位檔的灰階影像檔。其中影像的寬度會因為二進位檔本身的檔案大小而有所不同，以下表 3.2 是各二進位檔檔案大小與寬度的對照表：

二進位檔檔案大小範圍	影像寬度
<10kB	32
10kB-30kB	64
30kB-60kB	128
60kB-100kB	256
100kB-200kB	384
200kB-500kB	512
500kB-1000kB	768
>1000kB	1024

表 3-4 二進位檔檔案大小與影像寬度對照表

- Use GIST feature to extract texture feature :

經由上述步驟所轉化的灰階影像檔會在這個步驟去抽取出代表性的特徵值。本系統所使用的是 GIST 圖形特徵擷取方式，GIST 主要講求將一張圖片經由轉化成低維度的特徵向量去進行相似度比對。首先會將一張圖片先進行切割成 k 個小區塊，然後每個小區塊進行傅立葉轉換，最後會再由

multiscale oriented filter 將這些區塊轉化為 k 個邊長為 n 的方塊矩陣，其中的 n 是在 filter 中每個小區塊的邊長。

本系統會將產生的影像檔先統一重新調整為 64x64 的大小，然後再進行特徵向量轉換的流程，過程中是以 k=20、n=4 去計算，所以圖形最後會產生 320 個維度的特徵向量。

- Total binary & Used binary :

二進位檔通常是由 shell script 程式檔去呼叫所執行的，但並不是其中所有的二進位檔都會被呼叫。本研究會從 shell script 程式碼中去截取出呼叫二進位檔的指令，並記錄下哪些二進位是有被呼叫的、哪些是沒有被呼叫的。而最後在比對二進位檔案特徵時，就會分成 total binary 和 used binary。

3. 惡意組件包之檔案結構

同類型的惡意組件包在檔案結構上是類似的，檔案結構指的是惡意組件包中所含有的各種檔案類型數量，本研究選取惡意組件包中第二層目錄內的檔案類型數量，所計算的檔案類型分別有資料夾、c 程式碼檔、perl 程式碼檔、shell script 程式碼、二進位檔和其他文字檔，依序形成一個特徵向量表示，假設某個惡意組件包解壓縮後第二層目錄內擁有 2 個資料夾，而這兩個資料夾中共有 2 個 shell script 程式碼檔、1 個二進位檔和 10 個其他文字檔，則此壓縮檔第二層的檔案結構特徵會表示成(2, 0, 0, 2, 1, 10)。

至於為甚麼只擷取第二層的檔案結構而不擷取第一層呢？是因為各個不同群集的第一層檔案結構特徵相似程度太高。因為第一層目錄中的檔案數量一般來說偏少，每種檔案類型都只有個位數的檔案數量，這會讓特徵值在作相似度比較時無法區分群集之間的差異。至於第三層以上的檔案結構，由於很少有惡意組件包會有第三層的檔案結構，所以一樣不採計。而第二層目錄的檔案結構就大大不同了，通常惡意組件包中的第二層目錄都

是存放該組件包在執行時所需的參考指令庫，而不同的惡意組件包所用的指令庫都有很大的鑑別性，藉由觀察第二層的指令庫就可以猜測此組件包的種類。雖然有些較小的並沒有第二層目錄，並不影響第二層目錄檔案結構的鑑別性。所以在本系統中，在第一層目錄中針對每個程式碼檔和二進位檔的內容作分析，在第二層目錄中針對整體的檔案結構作分析，這樣所搭配的特徵值才能最有效代表這個惡意組件包的特色。

<4> Clustering :

經過特徵擷取之後，每個惡意組件包都會擁有六個特徵值，這六個特徵值分別是(1)所有的二進位檔相似度 (2)呼叫過的二進位檔相似度 (3)C 程式碼檔相似度 (4)Perl 程式碼檔相似度 (5)Shell script 檔相似度 (6)第二層的檔案結構相似度，這六個特徵值會經過相似度公式的計算，而得到每個組件包與其他群集中心的相似度距離。接下來再利用分群演算法將擁有相同特徵的惡意組件包分到同一群集中。本研究使用的分群演算法有兩種：1. 改良漸進式分群、2. 延伸式 1-NN。透過這兩種演算法而得到兩種不同的分群結果，再比較其優劣。

這兩種分群方式各有特色與缺點。在改良漸進式分群中，一旦需要分析的惡意組件包愈來愈多時，就不用耗時地與系統中每個成員做比較，只要和每個群集的中心比較即可。而且在漸進式分群中，可以再以 hierarchical 的方式去增加分群的規則，讓分群準確率更加提高。但群集中心的產生方式需要精細的計算，如果沒有事先經過細微的觀察，很難找出適合的中心點表示方式，畢竟這個中心必須擁有足以代表這個群集的特徵。

在延伸式 1-NN 中，每個群集中心的決定方式較為簡單，中心點就是一個實際的惡意組件包。但是在找出中心點的計算過程中會稍為耗時一些，不過耗時的程度影響不大。在延伸式 1-NN 中，比較沒辦法去細微區分出兩

種差異不大的惡意組件包的種類。

本系統的分群演算法除了區分不同群集之外，也能判別出群集與群集之間是否特殊的關聯性，也就是惡意程式家族的概念。有很多惡意組件包樣本在基本架構上是一樣的，但卻各自有一點不一樣的額外功能，那就可以合理的懷疑這其中有變種的可能性。例如 A 惡意組件包和 B 惡意組件包都擁有同樣的二進位檔和程式碼檔，但是 B 惡意組件包卻多了一個 PERL 程式檔來作為額外的 portscan 功能，大部分的防毒軟體都會將 A、B 兩個惡意組件包分為不同的種類，但是本系統期望的是能夠找出兩個惡意組件包之間的關聯性，並提供給使用者作參考，因此使用多層的分群演算。

本系統的多層分群指的是對樣本群使用多層次分群演算，每一層的分群所採用的特徵值和權重值都不一樣。如圖 3-6 所示，第一層分群中先採用總體二進位檔相似度和 Shell script 檔的語法結構，這層所分出來的群集比較綜觀，因為二進位檔和 Shell script 檔是所有惡意組件包都有的。接下來檢查每個群集之間是否達到合併的標準。第二層分群中把所有特徵值都計算進去，就式為了把群集內每個成員的細微差異找出來。

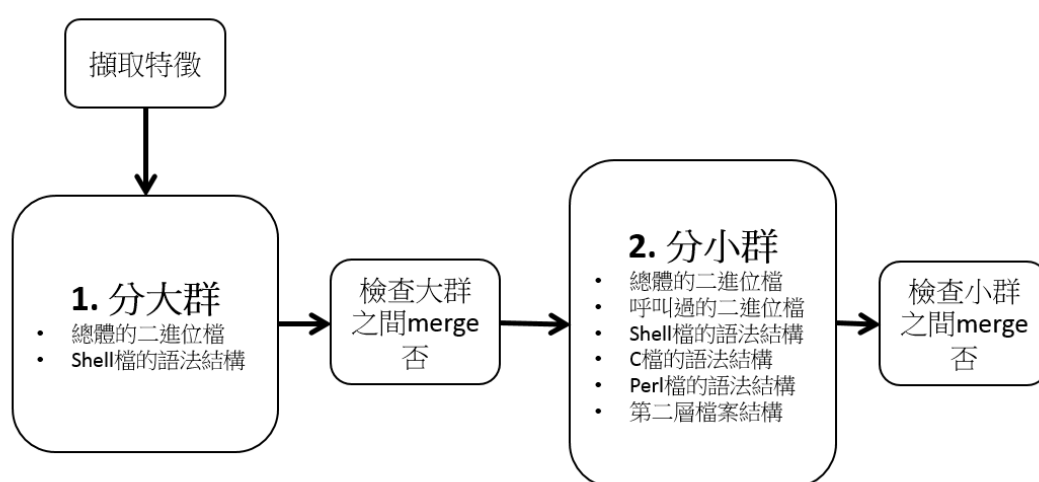


圖 3-6 多層分群演算法

這樣的分群過程將可把樣本群分成如圖 3-7 所示，在一個比較大的群集之中又可以分成許多小的群集。這些小群集各自有著自己的特色，但又

同時隸屬於同一個大群集之中，都同時有著這個大群集的特色。這就是惡意程式家族的概念，每個家族成員都同相同的家族遺傳特色，但每個成員各自都有自己的特色。一般的分析系統只會將這些惡意組件包通通分在同一群集中，這是不夠詳細的。本系統的目的就是希望能將這些有著細微差異的惡意組件包區分出來。

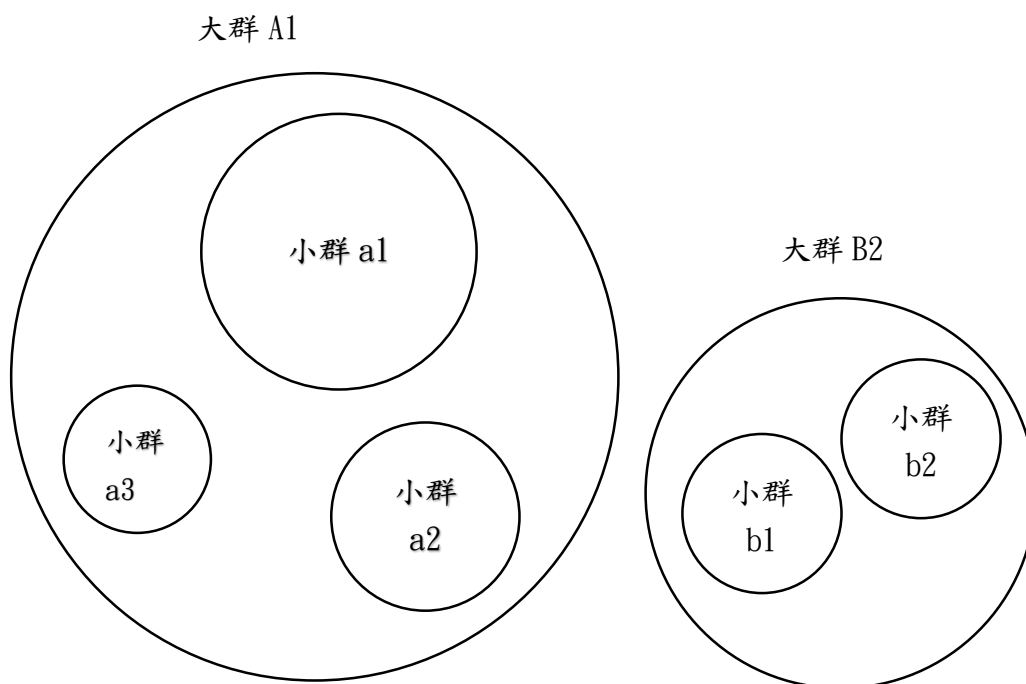


圖 3-7 多層分群範例

3.2 相似度公式

每個惡意組件包的內容大致上是由二進位檔、程式碼檔、文字檔和子資料夾所組成，而程式碼檔則包含 C 語言檔、Perl 檔、Shell script 檔。有些惡意組件包內容只有單一程式碼檔而已，這種類型的檔案系統會特別獨立出來作分析。

每個新惡意組件包進入到系統中，經過特徵擷取之後都會擁有自己的特徵值，而接下來就是做惡意組件包之間的相似度計算比較，計算出兩個惡意組件包之間的相似程度到底有沒有達到分為同一類別的標準。不過每個惡意程式的特徵值型態不太一樣，有的是字串、有的是向量，當然也要用不同的公式去計

算。每個特徵的重要程度也不同，給予每個特徵的權重比例當然也就不一樣，這些都是需要經過詳細的實驗比較的。針對兩個群集是否合併，需要將兩個群集中心做相似度的計算比較。不管是新進惡意組件包或是群集中心，都有以下的 6 個特徵值：

- (1) 所有的二進位檔相似度
- (2) 呼叫過的二進位檔相似度
- (3) C 程式碼檔相似度
- (4) Perl 程式碼檔相似度
- (5) Shell script 檔相似度
- (6) 第二層的檔案結構相似度

在這六個特徵值中，二進位檔和檔案結構的特徵值是以向量的形式來表示，程式碼檔的特徵值則是以字串的形式來表示，而接下會將依照各種不同的特徵值分別說明相似度計算方式：

● 二進位檔相似度計算

不論是所有的二進位檔相似度或是呼叫過的二進位檔相似度，計算方式都是一樣的。惡意組件包中的二進位檔都會經過 Nataraj [15] 的方式轉化為 320 個維度的向量。而兩個二進位檔(f, b)之間的相似度計算就是以兩個 320 維度的向量依歐式距離公式來計算，再用 1 剪去所得到的距離，即可獲得相似度。

$$BinaryPattenSim(f, b) = 1 - EuDist(BinaryPatten_f, BinaryPatten_c)$$

不過惡意組件包中常會存在 2 個以上的二進位檔。假設惡意組件包 p 為比較方，群集中心 c 為被比較方。惡意組件包 p 中的每一個二進位檔都會去和 c 的所有二進位檔做計算，然後選相似度最高的做代表，將惡意組

件包 p 的所有 binary 相似度做加總，再除以群集中心 c 內的二進位檔個數，即為惡意組件包 p 和中心 c 之間的 binary 相似度，其值會介於 0~1 之間。若是兩群集中心做比較，方式亦同。

$$BinarySim(p, c) = \frac{\sum_{i=pf_1}^{pf_n} Max(BinaryPattenSim(i, cf))}{BinaryNum_c}$$

● 程式碼檔相似度計算

不管是 C 程式碼檔相似度、Perl 程式碼檔相似度或 Shell script 檔相似度，相似度計算方式都相同。一個惡意程式中的程式碼特徵值，都是一串由程式碼指令所組成的字串，而這裡就是要比較兩字串的相似度。

兩串程式碼字串之間的相似度計算，要先算出兩字串之間的 Edit Distance，然後再除以兩字串之間的最大長度，再以 1 剪去，即可獲得結果。

$$CodePattenSim(f, c) = 1 - \frac{EditDist(Patten_f, Patten_c)}{Max(PattenLen_f, PattenLen_c)}$$

當兩個惡意組件包之間要計算程式碼相似度時，而兩個惡意組件包又各擁有 2 個以上的程式碼檔，此時系統會先依照程式碼種類分別計算，也就是 C 程式碼檔只與 C 程式碼檔比較、Perl 檔只與 Perl 檔比較、Shell script 只與 Shell script 比較。

假設惡意組件包 p 為比較方，群集中心 c 為被比較方。惡意組件包 p 的每個 C 程式碼檔會去和中心 c 的所有 C 程式碼檔做計算比較。然後選相似度最高的做代表，將惡意組件包 p 的所有 c 程式碼檔的相似度做加總，再除以群集中心 c 內的 c 程式碼檔個數，即為惡意組件包 p 和中心 c 之間的 c 程式碼檔相似度，其值會介於 0~1 之間。再用相同方式計算出 Perl 檔和 shell script 檔的相似度。若是兩群集中心做比較，方式亦同。

$$CodeSim(p, c) = \frac{\sum_{i=pf_1}^{pf_n} Max(CodePattenSim(i, cf))}{SourceCodeNum_p}$$

● 檔案結構相似度計算

在本系統中只針對惡意組件包的第二層檔案結構做計算，第一層檔案結構因為差異太小而捨棄。惡意組件包的第二層檔案結構特徵值，會以一個 5 維度的向量值所呈現，這 5 個維度分別代表這個惡意組件包第二層的檔案結構中的子資料夾個數、shell script 檔個數、perl 檔個數、c 程式碼檔個數和二進位檔個數。當兩個惡意組件包之間要做結構相似度計算時，只要將兩個向量中每個維度的差異數加總，再除以被比較方的檔案數量，最後再用 1 去剪，即可獲得相似度。在這裡本研究不考慮使用其他文字檔的檔案數量差異數是因為相同類型的惡意程式可能會因製作者在說明文件、字典檔等放入的數量不同，而讓文字檔數量上差異過大，會使得在檔案結構的相似度計算上造成相似度下降。

$$diff(p, c) = |N_{dir1} - N_{dir2}| + |N_{sh1} - N_{sh2}| + |N_{c1} - N_{c2}| + |N_{pl1} - N_{pl2}| + |N_{B1} - N_{B2}|$$

$$StructSim(p, c) = \left(1 - \frac{diff(p_{second}, c_{second})}{\max(second\ dir\ file\ number)}\right)$$

最後兩個惡意組件包的總相似度則用以下公式所計算：

$$Similarity(p, c) = \omega_1 * CSim(p, c) + \omega_2 * ShellSim(p, c) + \omega_3 * PerlSim(p, c) + \omega_4 * StructSim(p, c) + \omega_5 * BinarySim(p, c)$$

其中 ω_1 、 ω_2 、 ω_3 、 ω_4 和 ω_5 分別為以上五種相似度的權重值。

3.3 分群演算法

本系統所用的兩種分群演算法基本上都屬於漸進式分群，其流程圖如圖 3-8。漸進式分群的特色就是以群集中心點為主軸，來改善分群效率的方式。當新進惡意組件包進入到系統中，系統不需要將整個群集重新計算比較，取而代之的是將新惡意組件包與現存的所有群集中心比較即可，這樣既可以維持一定的準確率，又可以大幅降低系統分析所需時間。

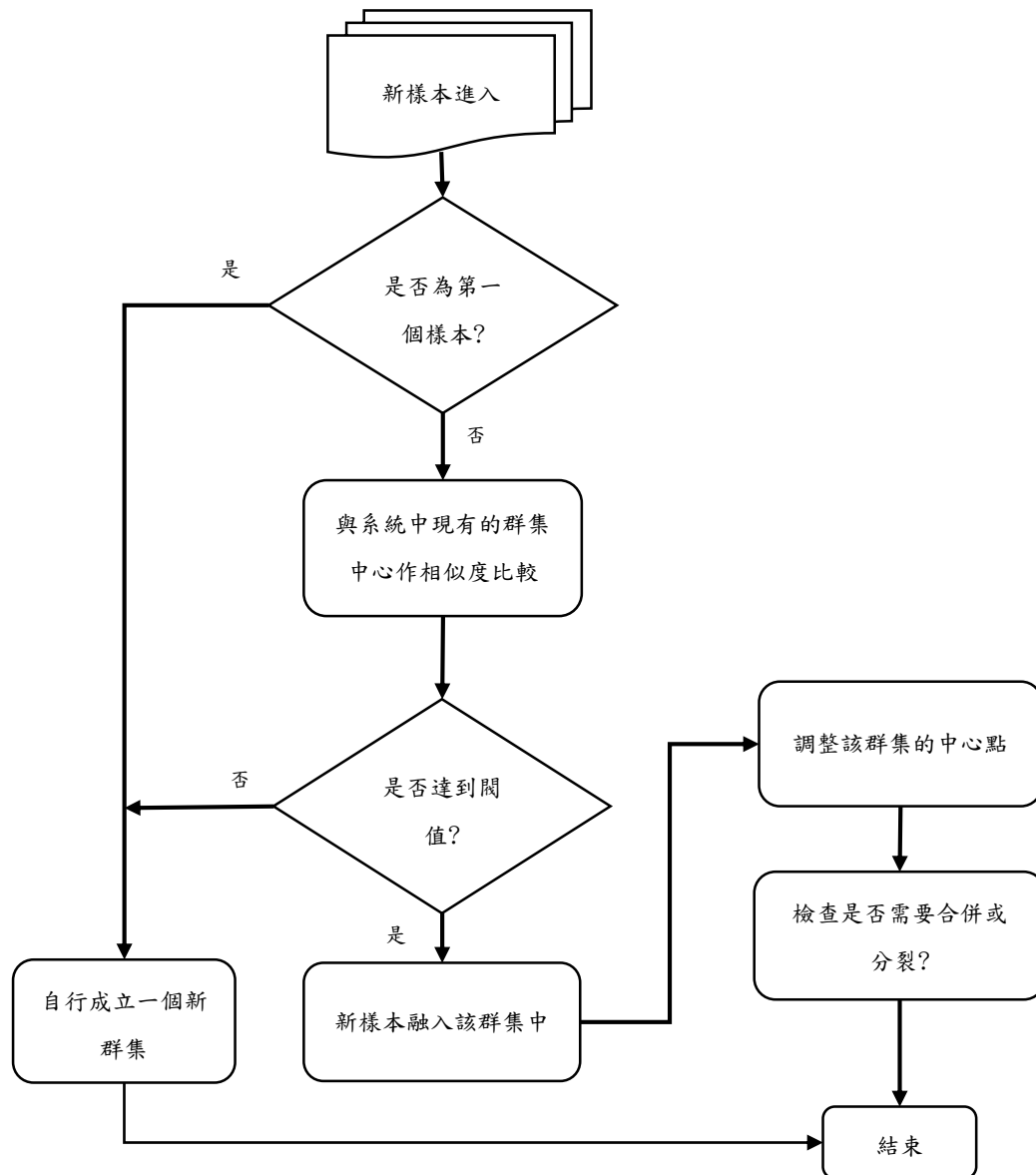


圖 3-8 漸進式分群流程圖

當第一個惡意組件包進入系統時，變自行成為第一個群集，並建立群集中心。接下來每個新惡意組件包進入系統，都會與當時系統中每個群集的群集中心做相似度比較，只要相似度達預定的閾值，就可把該惡意組件包納入該群集中，並根據自定的演算法來重新調整此群集的群集中心。萬一沒有任何群集適合新惡意組件包，則自行成為一個新群集，並建立群集中心。

群集與群集之間若要判斷是否合併，則將兩群集的群集中心做相似度比較，若達預定的閾值，則將兩群集合併。某群集之中若要判斷是否分裂，則將該群集內所有成員重新分析，若成員之間的相似度低於預定的閾值，則將該群集分裂。

漸進式分群的演算方式可以視為中心點基礎分群法與密度基礎分群法的合併。希望可以極大化的節省時間，同時又能兼顧準確率。但漸進式分群最大的挑戰就是群集中心點的演算方法。在漸進式分群中，群集中心點要足以代表整個群集，一但中心點過於偏頗，一定會大大影響準確率。每當有新惡意組件包加入群集後，群集中心點的調整方式一定要非常慎重。以下為本系統所採用的兩種分群演算法，這兩種演算法在群集中心點的演算法上各有優劣。

● 改良漸進式分群

本演算法的特色就是將群集中所有惡意程式的特徵值都納入中心點的計算中，最後算出的中心點是一個虛擬中心點，它並不是實際存在的惡意程式節點，但它卻能最大化的代表這個群集中的每一個惡意組件包。

本系統中每個惡意組件包都擁有六個特徵值，所以群集中心點會含有六個表格，每個表格代表其中一個特徵值。每個特徵表中會記錄下該群集所有惡意組件包的不重複特徵，並且還有該特徵的權重。每個權重代表的是群集中擁有

該特徵的惡意程式個數比例。例如：C1 群集中有 5 個成員，表 3-5 就是其 5 個成員的 shell script 特徵，冒號為區隔符號。

	Shell script 特徵值
成員 1	pwd:./:crontab:crontab:grep:if:if:./:./:chmod:
成員 2	pwd:./:crontab:crontab:grep:if:if:./:./:chmod:
成員 3	grep:for:grep:if:
成員 4	if:if:./:./:
成員 5	pwd:./:crontab:crontab:grep:if:if:./:./:chmod:

表 3-5 群集成員的 Shell script 特徵

由表 3-5 可以以看出，5 個成員的 shell script 特徵中，有三個是一樣的，另外兩個則各自不相同。表 3-6 就是該群集中心的 shell script 特徵。

	Shell script 特徵值	權重
特徵 1	pwd:./:crontab:crontab:grep:if:if:./:./:chmod:	3
特徵 2	grep:for:grep:if:	1
特徵 3	if:if:./:./:	1

表 3-6 表 3-5 成員們的群集中心表

表 3-6 就是該群集的群集中心點的 shell script 特徵值。每當有新惡意組件包要與此群集中心做相似度比較時，新惡意組件包的 shell script 特徵值必須和上表的所有特徵值做計算，還要再乘上表中的權重值，並取三個數值中最大的當做最後的 Shell script 特徵相似度。

一旦新惡意組件包要納入本群集中，就要進行中心點調整。將新惡意組件包的 shell script 特徵值與中心點的特徵值比較相似度，若相似度達一定的閾值，就將兩特徵字串做最大共同字串計算，得到結果則做為新的中心點特徵

值，且將權重加 1。若是低於閾值，就將新惡意組件包的特徵字串直接納入為中心點的新特徵值，權重設為 1。這個閾值的設定必須以極高的標準去考慮，否則特徵字串會隨著計算次數的增加，漸漸縮短。

當新惡意組件包是系統中的第一個成員時，該惡意組件包的所有特徵值就是該群集中心的所有特徵值。接下來當新惡意組件包確定要納入某群集時，會將新惡意組件包的特徵與該群集中心的特徵做比較。當新的群集中心計算完畢後，必須整理群集中心的各個特徵表，將權重太低的特徵值刪除，以避免特徵表愈來愈長。

● 延伸式 1-NN

本演算法的特色是從群集的每個成員中挑出一個做為群集中心，這個中心與其他的群集成員的相似度必須是最高的，藉此來將這個中心點代表整個群集。由於中心點是實際存在的成員節點，所以不必花心思去計算並建立一個虛擬的中心點，畢竟計算愈多就會有愈多的不確定性。

圖 3-9 就是延伸式 1-NN 的群集中心點比較方式，每當有新惡意組件包要與所有群集中心做比較時，會有三種情況。(a)如果該群集的成員只有一個，那該成員就是群集中心，就與該成員做相似度比較。(b)如果該群集的成員有兩個，那這兩個成員同時都是群集中心，和這兩個成員所做相似度計算成果取平均值即為最終的相似度。(c)如果該群集的成員有三個以上，那會有其中一個成員為該群集的中心點，新惡意組件包只要直接與該中心點做比較即可。

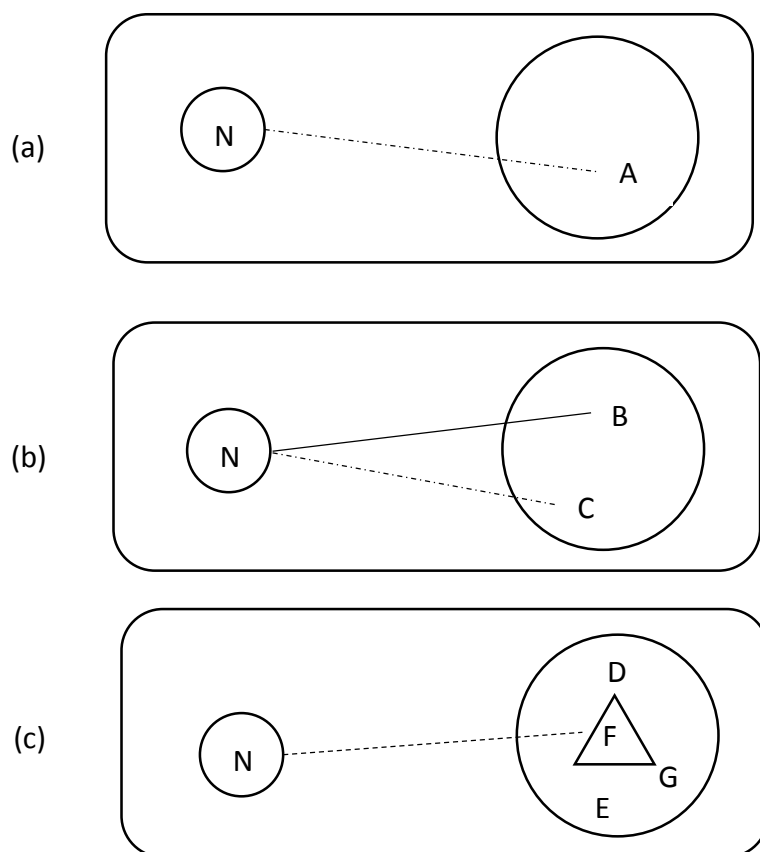


圖 3-9 延伸式 1-NN 的中心點選取

相似度比較後，若新惡意組件包與所有群集中心的相似度都未達閾值，則新惡意組件包自行成為一個新群集。若新惡意組件包要納入某群集中，該群集如果只有一個成員，那直接加入即可。若該群集成員有兩個以上，則新惡意組件包進入後必須與所有成員節點做相似度比較，再將計算結果加總。從所有成員中找出相似度加總結果最高的成員，即為群集中心點。

延伸式 1-NN 演算法雖然以簡單明瞭的方式來確立群集中心點，但是在找出群集中心點的過程中，必須讓群集裡所有成員兩兩比較，才能找出加總相似度最高的成員。雖然時間複雜度是 $O(N^2)$ ，但每當有新惡意組件包進入群集時，新惡意組件包只需與現有成員個別計算一次即可，畢竟群集成員並不會太多，計算過程所耗時間也沒有想像中那麼長。

● 多層分群

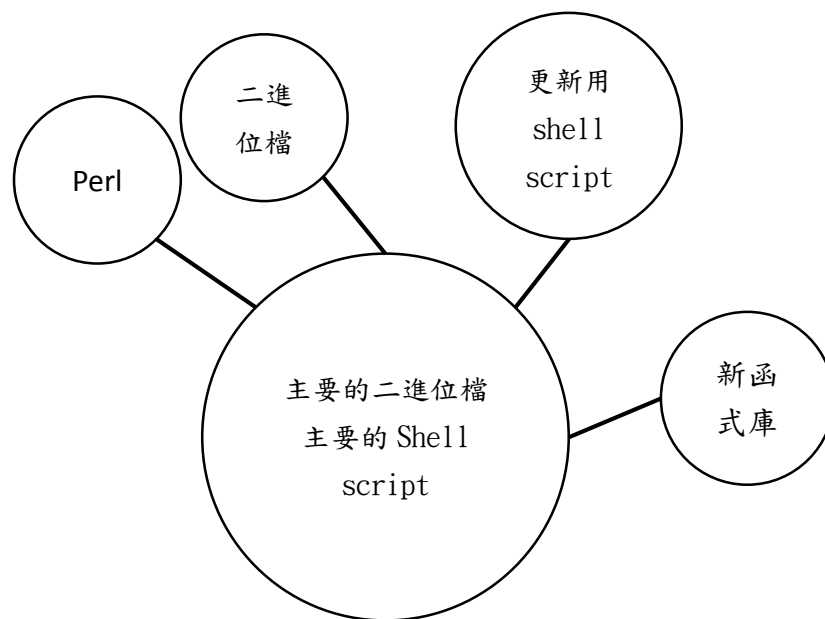
一般分群演算法目的皆是在於將樣本群中找出一群群的群集，而每個群集之中的所屬成員彼此之間的相似度極高，進而證明同屬一個群集中的成員擁有相同的特徵。但分群過程中並不是永遠都這麼單純，同一群集之中仍有可能在加以分成兩個不同群集，這在以密度為基礎分群演算法中相當常見。在密度基礎的分群演算過程中，一開始可能因進入系統的樣本數不夠多，某些較疏遠的樣本也會先分在同一群集之中。但隨著樣本數愈來愈多，就會發現這個群集之中的有些區塊的密度比較稀疏，當這個群集中的密度低到某種程度後就可以分裂成兩個群集。這兩個新分裂出來的群集彼此之間的相似程度比起其他群集要高得多，所以這兩個群集之間是有共同相似的特徵，卻又因為相似程度沒有達到閾值而被分成兩個不同群集，若是照傳統的分群方式來定義這兩個群集並不够詳細，應該要找出這兩個群集之間相似的地方所在。本研究希望可以在惡意程式的分析過程中，找到類似這種關係的惡意程式群集，進而證明這兩個群集之間在哪些部分是關係緊密、在哪些部分是關係疏遠的。

幾乎所有的惡意組件包內容中都有 Shell script 檔和二進位檔，且二進位檔的執行動作都是透過 Shell script 檔中的指令呼叫。一個惡意組件包的攻擊行為必定是以 Shell script 檔和二進位檔的搭配為主體，Shell script 檔和二進位檔也就是惡意組件包的主要必備功能，另外可能再搭配其他的額外惡意程式，像是 perl 檔或其他字典黨。如果有兩個惡意程式擁有相同的 Shell script 檔和二進位檔組合，但卻又各自有額外不同的 perl 檔或二進位檔。這兩個惡意程式在分群過程中會被分到不同的兩個群集類別，但它們兩個是有相似的特徵存在。

本研究在設計多層分群的先後順序時，參考了以上的觀察結果，所以在第一層的分群中先採用 Shell script 檔和二進位檔的特徵來作相似度計算，因為幾乎所有惡意程式都有這兩種特徵。以這兩個特徵所計算出的分群結果會是一

個較綜觀的群集關係，此時的同一群集中的成員都擁有相同的 Shell script 檔和二進位檔組合，但每個成員之間的相似程度不會非常高，它們可能各自額外擁有不同的惡意檔案。不過每個不同群集之間的差異度極大。

接下來在第二層的分群中，將六個特徵值都納入計算中。本研究希望能在這層分群中找出同樣隸屬於一個大群集中的成員之間，彼此間有哪些細微的差異，而這些差異就是讓這些成員可以再分出多個群集的關鍵。有些成員會有額外不同的二進位檔作其他的攻擊行為，有些成員會有額外不同的 Shell Script 檔作字典檔的更新動作，有些成員會有額外的 perl 檔。這些細微差異導致這個大群集之中可以再作分群，分成更多的小群集。這些小群集都共同擁有一些相同的攻擊行為，這就是這個大群集的特色，而每個小群集卻又會有各自的不同特色。



本系統就是透過這樣的多層分群結構去找出惡意程式之間的家族關係，每個大家族都有其家族特有的攻擊行為，其下每個家族成員也都遺傳了這樣的家族特性。但是家族內的每個惡意程式成員在被駭客設計出來的過程中，都會被加上一些額外的功能或是修改一些舊的缺點，使的每個家族成員之間都有不同

之處。

第四章 系統實驗

接下來本研究將會透過四項實驗來驗證本系統的效率。第一個實驗會先將本研究中的兩種分群演算法：改良漸進式分群和延伸式 1-NN 所計算出了分群結果作比較，確認兩種演算法的特色和缺點。第二個實驗，為了驗證本系統的執行效率，透過任意調換樣本群進入本系統分析的順序，驗證分群準確率和執行時間是否前後變動過大。第三個實驗會將本系統和網路上最具公正力的惡意程式分析網站 VirusTotal 來作比較，將本研究的樣本群進行分析，建立一套分群準確率的標準。第四個實驗將本系統和現今個人電腦中免費防毒軟體使用率最高的 Avira 作比較，驗證本系統和一般單機防毒軟體的差異程度。透過這四項實驗，藉以驗證本系統的準確率和實用程度。

4.1 樣本收集與觀察

本研究使用的惡意樣本，主要都是透過在幾個學術研究機構所架設的 Honeypot 所收集到的。由於這些 Honeypot 都在架設在 Linux 的作業系統上，所以收集到的惡意樣本也都是針對 Unix 系統架構所設計的。收集時間是從 2011 年 3 月開始到 2013 年 9 月為止，共 2 年又 6 個月，樣本數為 545 個。

所收集到的樣本可以粗分為兩類，單一檔案和壓縮封裝檔。單一檔案中包含 PERL 程式檔、二進位檔和 xml 檔。其中 xml 檔經人工檢查後，其中所包含的網址連結並沒有惡意成分，xml 檔本身也沒有攻擊行為，故在本研究不予參考。壓縮封裝檔都是以 Linux 作業系統中常見的壓縮格式所封裝，其內容物可能包含了 C 語言程式檔、PERL 程式檔、Shell Script 檔、二進位檔和純文字檔，沒有其他類型的程式碼檔了。在 Windows 平台中常見的 Java 檔在此也完全沒有發現。

接下來本研究先以人工觀察的方式來將 545 個樣本分類，人工觀察的分類重點規則如下：

- 檔名相同與否不重要
- 帳密字典檔、說明文件和無意義的文字檔一律不重要
- 檢查 shell script 中的重要指令執行順序
- 檢查 shell script 中所呼叫的二進位檔的順序和次數
- 檢查 perl 檔中的重要指令執行順序
- 檢查二進位檔的內容相同與否
- 檢查樣本中的各類檔案結構

根據以上規則，先自行手動將 545 個樣本分成 38 個類別，表 4-1 即是其分裙結果，每個類別的命名方式是將該類別的所有樣本上傳到 virustotal 網站上，取重複率最多的名字即可，關於 virustotal 的相關實驗在下一小節會有詳細說明。

A1-portscan(89 個)： (1) portscan_A (2) scrip_G (3) Backdoor.Linux.Zorg.B (4) Trojan.Hacktool.Linux.Pscan.A (5) Virtool.Linux.Shark.A	A2-mechbot(94 個)： (1) Backdoor.Mechbot.F (2) Backdoor.Mechbot.H (3) Spyware.Unix.Mech.A (4) Backdoor.Mechbot.G
A3-flooder(16 個)： (1) Trojan.Flooder.Linux.Small.N (2) Trojan.Flooder.Linux.Small.O (3) Trojan.Flooder.Linux.Small.P (4) Trojan.Flooder.Linux.Small.S (5) Trojan.Script.9765	A4-cyc(16 個)： (1) Trojan.Hacktool.Linux.Small.B (2) Backdoor.F

A5-sonmany(18 個) ; (1) Trojan. Linux. Infostealer. A (2) HackTool. Linux. CleanLog	A6-psybnc(43 個) :
A7-makefile(33 個) :	A8- LinuxExploit(3 個) :
A9- Exploit. CodeExec. 0(13 個) :	A10- Rootkit. Linux. Agent(12 個) :
A11- Win32. Worm. Linux. Adore. A(8 個) :	A12- Trojan. Horse. BU(7 個) :
A13-Hacktool. Prochider. A(2 個) :	A14- Virtool. Linux. Shark. A(15 個) :
A15- TrojScanA-Gen(12 個) :	A16- Backdoor. Perl. Shellbot. B(35 個) :
A17- Backdoor. Perl. Shellbot. F(19 個) :	A18-Linux. OSF. 8759(18 個) :
A19-Backdoor. Agent. AAQ(10 個) :	A20- Trojan. Script. 63933(6 個) :
A21- Linux. RST. B(21 個) :	A22-Linux. CornelGEN. 225(20 個) :

表 4-1 人工觀察所整理出的分群結果

表 4-1 的分類結果皆是以人工方式觀察、整理所得的，其過程不僅耗時且需要反覆比對和檢查，但相對的，分類結果可以百分之百的信任。希望能透過這成本耗費極鉅的過程，先得到一個可以信賴的分類結果，並透過接下來的實驗來證明，本系統可藉由較短的時間成本來得到一樣可信賴的分類結果。

4.2 實驗一 改良漸進式分群與延伸式 1-NN

本實驗為透過本系統去分析 Honeypot 所收集到的 545 個樣本，經過本系統的前處理和特徵擷取之後，接下來會有兩種分群演算法，1. 改良漸進式分群、

2. 延伸式 1-NN。透過這兩種演算法之後所得到的分群結果可以完全反映出這兩者的優缺點。而當作對照比對的第三方結果則是由人工的方式來做判斷，藉以決定兩種演算法的準確率。表 4-2 即是兩種分群演算法的分群結果比較。

名稱	群集總數	正確分群數量	錯誤分群數量	準確率	平均每樣本處理時間
改良漸進式分群	38	517	28	94.9%	8 分鐘
延伸式 1-NN	22	529	16	96.9%	10 分鐘

表 4-2 改良漸進式分群與延伸式 1-NN 的分群結果比較

漸進式分群的特色就是幫每個群集定義出一個中心點，如果要判斷新樣本是否要分至這個群集，只要將新樣本和此群集中心點作比較計算即可，不用和群集中每個成員比較。而本系統的改良漸進式分群則是改進了群集中心點的決定方式。每個群集的中心點都彙整了該群集中的所有特徵值，並記錄下每個特徵值的權重比例，畢竟每個群集成員的特徵字串不一定會 100%相同，而如果要使用 longest common subsequence 之類的演算法去整合出一個折衷的中心點，只要群集成員愈多，中心點的代表性就會愈來愈下降。而本系統的改良漸進式分群的缺點則是在演算法的設計上比較複雜，優點則是找出群集之間的家族關係，時間效率也不差。改良漸進式分群的群集總數會比較多，是為了從一個較大的群集中再畫分出不同的小群集，這些小群集彼此都有一定的親戚關係，卻又各自有一些決定性的特色，這就是惡意程式的變種。

表 4-3 就是以改良漸進式分群所得到的惡意程式家族的分析結果。每個方塊就是一個惡意程式家族，而方塊中的就是該家族成員。命名是事後將樣本上傳至 VirusTotal 而參考命名的。

A1-portscan(89 個)：	A2-mechbot(94 個)：
--------------------	-------------------

(1)portscan_A (2)scrip_G (3)Backdoor.Linux.Zorg.B (4)Trojan.Hacktool.Linux.Pscan.A (5)Virttool.Linux.Shark.A	(1) Backdoor.Mechbot.F (2) Backdoor.Mechbot.H (3) Spyware.Unix.Mech.A (4) Backdoor.Mechbot.G
A3-flooder(16 個) : (1)Trojan.Flooder.Linux.Small.N (2)Trojan.Flooder.Linux.Small.O (3)Trojan.Flooder.Linux.Small.P (4)Trojan.Flooder.Linux.Small.S (5)Trojan.Script.9765	A4-cyc(16 個) : (1)Trojan.Hacktool.Linux.Small.B (2)Backdoor.F
A5-sonmany(18 個) ; (1)Trojan.Linux.Infostealer.A (2)HackTool.Linux.CleanLog	A6-psybnc(43 個) : (1)with perl (2)without perl
A7-makefile(33 個) : (1) with perl (2) without perl	A8- LinuxExploit(3 個) :
A9- Exploit.CodeExec.0(13 個) :	A10- Rootkit.Linux.Agent(12 個) :
A11- Win32.Worm.Linux.Adore.A(8 個) :	A12- Trojan.Horse.BU(7 個) :
A13-Hacktool.Prochider.A(2 個) :	A14- Virttool.Linux.Shark.A(15 個) :
A15- TrojScanA-Gen(12 個) :	A16- Backdoor.Perl.Shellbot.B(35 個) :
A17- Backdoor.Perl.Shellbot.F(19	A18-Linux.OSF.8759(18 個) :

個)：	
A19-Backdoor. Agent. AAQ(10 個)：	A20-Trojan. Script. 63933(6 個)：
A21-Linux. RST. B(21 個)：	A22-Linux. CornelGEN. 225(20 個)：

表 4-3 改良漸進式分群所分析出的群集總數

表 4-4 是以改良漸進式分群加上多層分群所分析出的惡意程式家族間每個成員彼此的相似關係。

家族內關係
<p>A1-portscan(89 個)</p> <p>(1) portscan_A：最原始型</p> <p>(2) scrip_G：二進位檔和(1)完全不同，其餘相同</p> <p>(3) Backdoor. Linux. Zorg. B：二進位檔和(1)(2)完全不同，其餘相同</p> <p>(4) Trojan. Hacktool. Linux. Pscan. A：二進位檔比(1)多一個，在 shell 中執行的順序和次數也不同。</p> <p>(5) Virtool. Linux. Shark. A：二進位檔比(4)多 2 個，在 shell 中執行的順序和次數也不同。</p>
<p>A2-mechbot(94 個)</p> <p>(1) Backdoor. Mechbot. F：最原始型</p> <p>(2) Backdoor. Mechbot. H：比(1)多了 2 個二進位檔</p> <p>(3) Spyware. Unix. Mech. A：比(1)多了 1 個 shell script</p> <p>(4) Backdoor. Mechbot. G'：比(1)多了 1 個 PERL 檔</p>
<p>A3-flooder(16 個)：</p> <p>(1) Trojan. Flooder. Linux. Small. N：最原始型</p> <p>(2) Trojan. Flooder. Linux. Small. O：比(1)多了 1 個 shell script</p> <p>(3) Trojan. Flooder. Linux. Small. P：二進位檔和(1)有兩個不同</p>

(4) Trojan.Flooder.Linux.Small.S：比(1)多了 1 個 PERL (5) Trojan.Script.9765：二進位檔和第二層檔案結構和(1)不同
A4-cyc(16 個) (1) Trojan.Hacktool.Linux.Small.B：最原始型 (2) Backdoor.F：有 2 個二進位檔和(1)不同
A5-sonmany(18 個)； (1) Trojan.Linux.Infostealer.A：最原始型 (2) HackTool.Linux.CleanLog：第二層檔案結構完全不同
A6-psybnc(43 個)： (1) with perl：最原始型 (2) without perl：比(1)少個 1 個 PERL 檔
A7-makefile(33 個)： (1) with perl：最原始型 (2) without perl：比(1)少個 1 個 PERL 檔

表 4-4 改良漸進式分群所分析得出的惡意程式家族關係

在本系統的改良漸進式分群中，545 個樣本中有 28 個樣本是分群錯誤的。其中造成分群錯誤的主要原因是樣本中的檔案短缺和誤把純文字檔判為程式碼檔。由於本研究的樣本是來自於 Honeypot 的收集，在收集樣本的過程中的確是有可能會遺失部分檔案。但本研究很難去判定樣本中的檔案短缺究竟是攻擊者故意拿掉該檔案或是在收集過程中遺失的。而這些樣本在經過分群演算法的分析後，會因為特徵相似度不夠的關係而成為獨立的零星節點，不會將這種零星節點分到任一群集之中。

誤判純文字檔為程式碼檔則是由於，有些純文字檔是被撰寫為說明文件，而在其內容中節錄了幾段程式碼，當本系統在擷取特徵值時就會誤判該文字檔

為程式碼檔，正常的分析過程中會先判斷此文字檔是否為程式碼檔，再去針對程式碼種類來擷取特徵指令。但在惡意組件包中某些文字檔中只記錄下程式碼的指令，標頭卻沒有程式碼的正規格式，這種賺寫方式常見於 Shell Script，當此惡意程式要執行這些指令時才會去引入這些文字檔。這種情況並不多見，545 個樣本中只有 3 個樣本出現這種情況。

延伸式 1-NN 的特色則是在中心點的決定上相對簡單，只要直接將群集中最具代表性的節點抓出來當中心點即可，而如何找出哪個節點最具代表性則需要將群集中的每個成員兩兩計算比較，時間複雜度就會高達 $O(n^2)$ 。延伸式 1-NN 非常適合用在分析未知惡意樣本的狀況，系統只要告訴使用者新進入的未知惡意樣本可能隸屬於某個群集，使用者就可以採取該群集的應對方式來對付此未知樣本。但是延伸式 1-NN 無法分析出群集之間的家族關係，因為在調整群集中心點時考慮到新樣本和群集中心的距離，若新樣本融入某群集的閾值太高，雖然可以分析出較嚴謹的分群結果，但會造成零星結點過多，這樣反而失去的分析惡意程式的初衷。

表 4-5 就是延伸式 1-NN 的分析結果。

A1 - portscan(89 個)	A13 - Hacktool.Prochider.A(2 個)
A2 - mecbot(94 個)	A14 - Virtool.Linux.Shark.A(15 個)
A3 - flooder(16 個)	A15 - TrojScanA-Gen(12 個)
A4 - cyc(16 個)	A16 - Backdoor.Perl.Shellbot.B(35 個)
A5 - somany(18 個)	A17 - Backdoor.Perl.Shellbot.F(19 個)
A6 - psybnc(43 個)	A18 - Linux.OSF.8759(18 個)

A7 - makefile(33 個)	A19 - Backdoor.Agent.AAQ(10 個)
A8 - LinuxExploit(3 個)	A20 - Trojan.Script.63933(6 個)
A9 - Exploit.CodeExec.0(12 個)	A21 - Linux.RST.B(21 個)
A10 - Rootkit.Linux.Agent(12 個)	A22 - Linux.CornelGEN.225(20 個)
A11 - Win32.Worm.Linux.Adore.A(8 個)	
A12 - Trojan.Horse.BU(7 個)	

表 4-5 延伸式 1-NN 的分群結果

圖 4-1 與圖 4-2 是記錄改良漸進式分群和延伸式 1-NN 在分析樣本的過程中，每個樣本的處理時間變化。

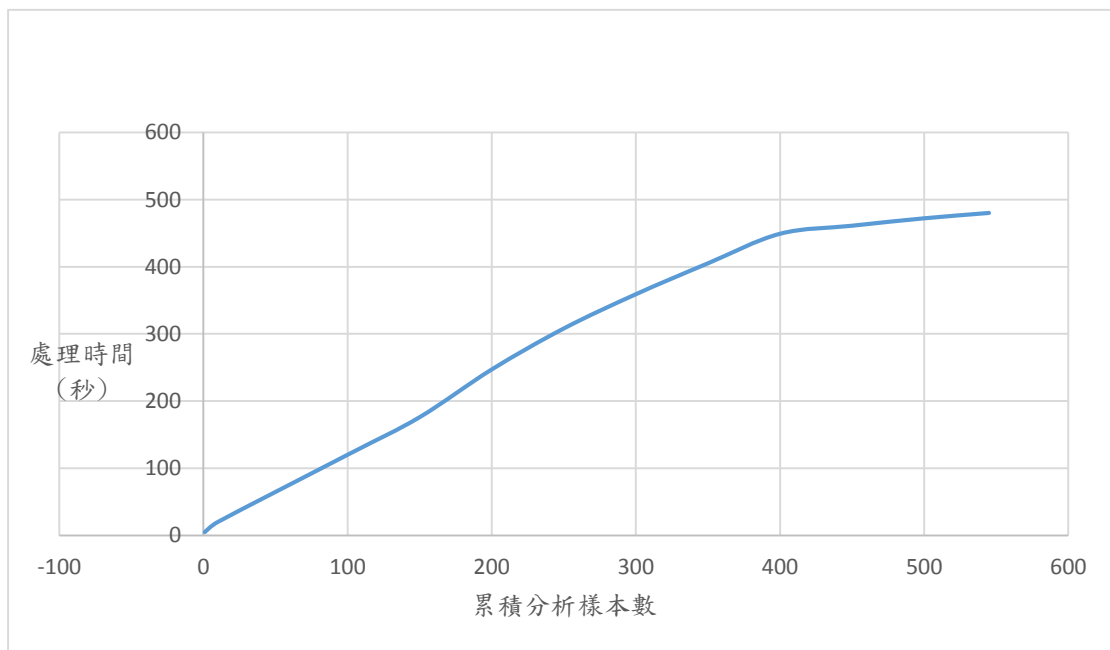


圖 4-1 改良漸進式分群中每個樣本分析所需時間

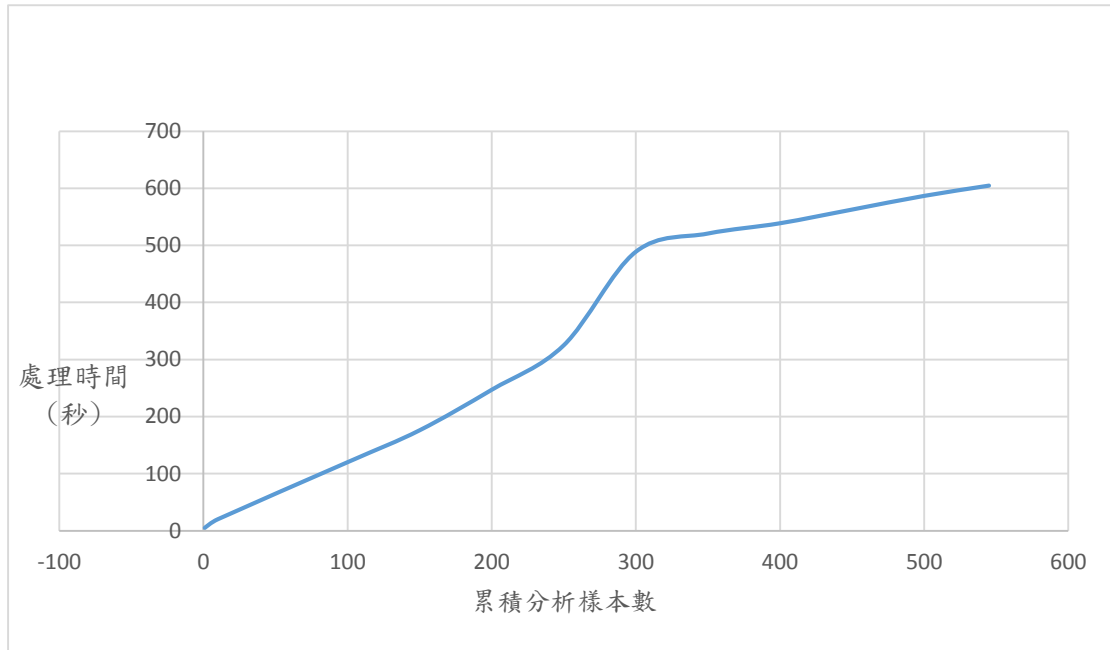


圖 4-2 延伸式 1-NN 中每個樣本分析所需時間

延伸式 1-NN 除了無法分析出惡意程式群集間的家族關係之外，延伸式 1-NN 最讓人擔心的還是時間效率的問題，從演算法公式中可以看的出來時間複雜度是 $O(n^2)$ ，而從實際分析數據上也可以看的出來延伸式 1-NN 的處理時間是比改良漸進式分群還高的。不過這是因為本研究的惡意樣本中，有幾個群集的成員數量特別龐大，所以在計算該群集時耗費太多時間。如果能夠將群集成員限制在 20 個以內，那耗費時間將會大幅下降，甚至比改良漸進式分群還低。

4.3 實驗二 調換樣本順序以驗證本系統的分群效率

本實驗的目的是要藉由亂數調換樣本送進本系統分析的順序，驗證本系統的分群準確率和執行時間是否有過於明顯的變化。對於每個分群演算法的最大挑戰之一就是其中的參數調整，萬一參數設定的不夠恰當，就算在某次分群中的準確率很高，如果調換一下將樣本群送進系統分析的順序，準確率就很有可能降低許多。

本研究樣本的命名都是以該樣本被 honeypots 收集到的時間來作為開頭，

例如：若是在 2013 年 10 月 2 號下午 5 點 32 分時收集到該樣本，該樣本的命名開頭就是 201310021732。而樣本群的排序也是依檔名的字母排序，所以本研究實驗一和實驗二的樣本分析順序就是按照樣本被採集的時間順序，由舊到新。

接下來本實驗會用兩種方式來變換排列順序，第一種是完全亂數，第二種是分組亂數，將樣本群分成一組一組的，每組的樣本數量相同，每次從所有組別中取出第一號樣本送進系統作分析，等所有組別都取過一次後再重新從每個組中取出第二號，直到取完為止，換句話說就是樣本總數取 mod x ，就會變成每組 x 個，然後 x 值可以任意變動，圖 4-3 即是取 mod 5 的範例。分組亂數之目的主要是因為當初 honeypots 在收集惡意樣本時，常常會在某一時間區間中收到數個完全相同的樣本，內容完全一樣，只有在命名上有所不同。在原排序中分群時，就會有接連好幾個都是相同內容的樣本進入系統作分析，為了完全打破這種情形，才採取這種亂數分組的方式。

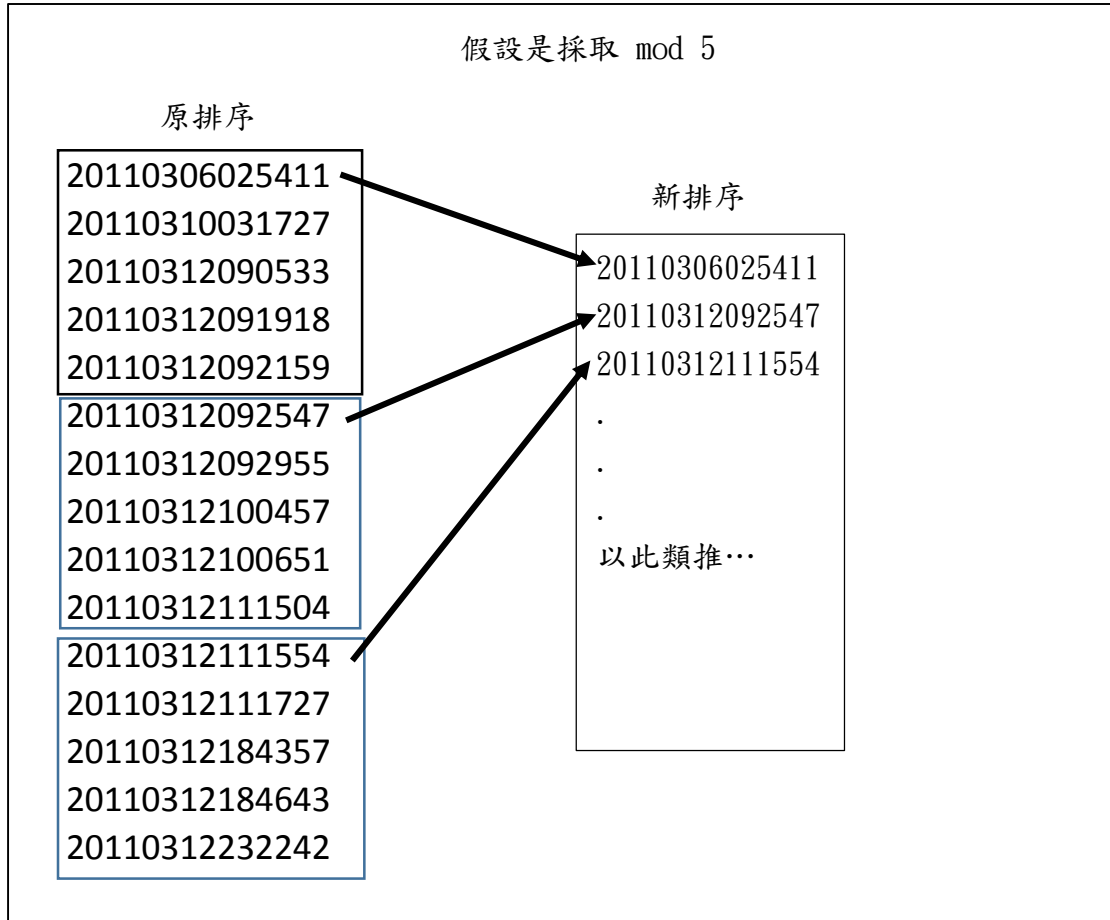


圖 4-3 分組亂數取的範例

接下來就開始比較原排序、完全亂數排序和分組亂數排序的分類結果，分別從<1>分類準確率、<2>群集合併次數、<3>群集分裂次數和<4>樣本分析時間來作討論。本實驗所使用的分群演算法都是改良漸進式分群，因為這演算法在分群上比較複雜，容易出現錯誤，因此需要藉此實驗來驗證。完全亂數排序的部分會作 20 次，每次都會重新打亂，再取平均值。分組亂數排序的部分，會從 mod 5 ~ mod 15，以便觀察其變異曲線。

● <1> 分類準確率

名稱	群集總數	正確分群數量	錯誤分群數	準確率
----	------	--------	-------	-----

			量	
原排序	38	517	28	94.9%
完全亂數排序 (20 次取平均)	38	512.2	32.8	93.98%
分組亂數排序 (16 次取平均)	38	513.3	31.7	93.77%

表 4-6 比較原排序與兩種亂數排序的分群準確率

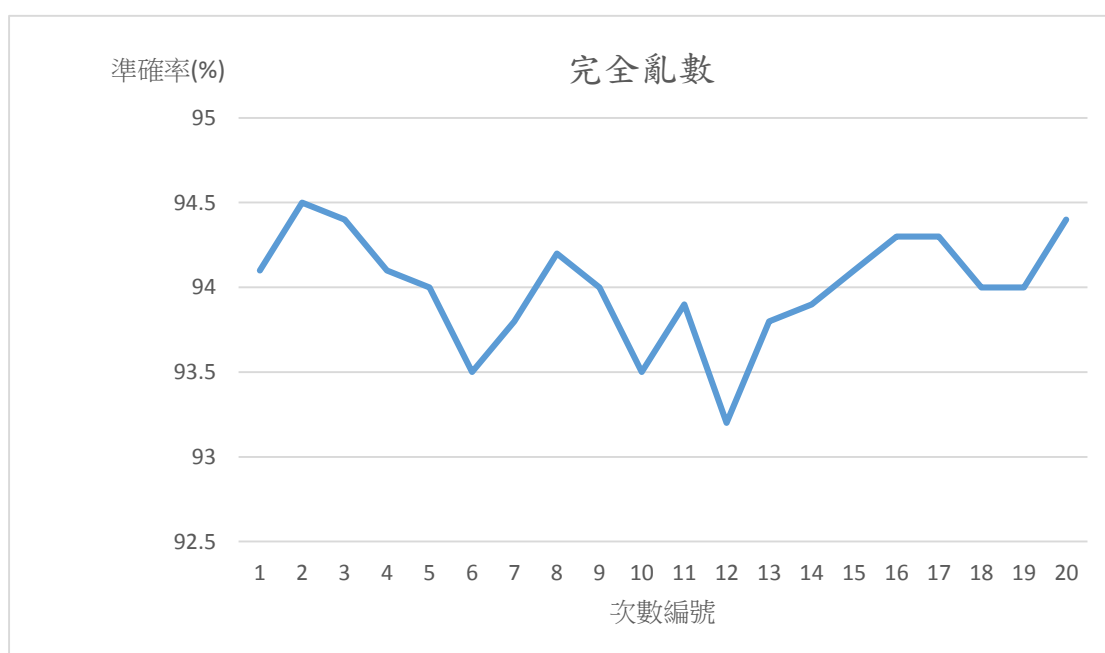


圖 4-4 20 次完全亂數的準確率變化趨勢

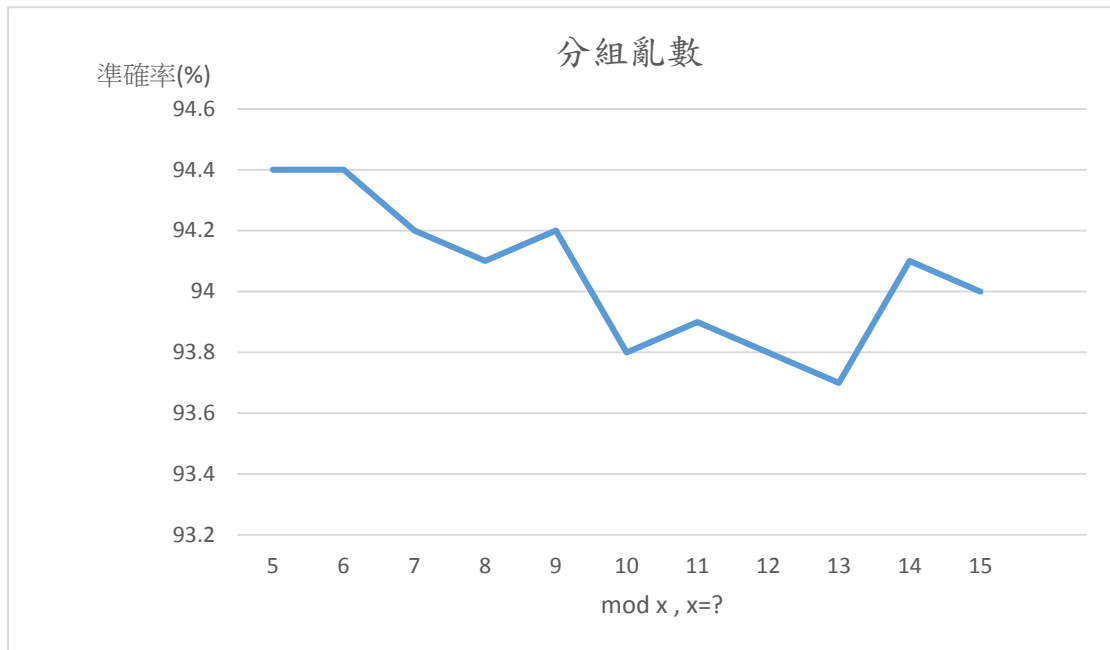


圖 4-5 16 次分組亂數的準確率變化趨勢

從表 4-6 可以看出，不管是完全亂數或是分組亂數的情況下，準確率都有些許的下降，但是從圖 4-4 和圖 4-5 中可以發現，每次亂數實驗的準確率都是在 94% 附近上下浮動，並沒有大幅度的變化。若是從分群過程中去仔細觀察，基本上大型群集都沒有變動，主要是有些小型樣本在分群上會有所變動，因為這些小型樣本的組成非常簡略，其內容只有 2、3 個二進位檔，沒有任何 shell script 檔，一般認為這可能是當初在檔案傳輸時有所遺失，或是 honeypots 在採集時沒有採集完全，並不是正常的惡意程式樣本，所以在分類時可能會在某個惡意程式家族下的小群中遊走。完全亂數形成的新排序和分組亂數形成的新排序所作出來的分群狀況是差不多的，所以在分群準確率上也都相差不大。

● <2> 群集合併次數

名稱	群集總數	群集合併次數
原排序	38	11

完全亂數排序 (20 次取平均)	38	14.3
分組亂數排序 (16 次取平均)	38	15.1

表 4-7 三種排序的分群過程中的群集合併次數比較

從表 4-7 中可以看出，在完全亂數和分組亂數下，平均群集合併次數都有上升的現象。但之所以會上升的原因，是因為第 10 次完全亂數排序的合併次數高達 20 次，而其他 19 次排序的合併次數都大約落在 10~13 次左右。在分組亂數排序中，在 mod 9 和 mod 14 時都接近 20 次的合併次數。如果仔細觀察這幾次合併特別多的排序，就可以發現這些排序在系統分析的初始階段，剛好每個不同種類的惡意樣本都進來 1、2 個，造成一開始的群集總數特別多但每個群集都只有少量成員。這在系統分析的中間階段，當進入系統分析過的樣本數愈來愈多時，就會開始整合一開始太多分散的群集。這幾次合併特別高的原因，可以說是遇到排序中的 worst case，就算是在分組亂數中，也不能果斷認定 mod x 的 x 愈高合併就愈高，畢竟在 mod 13 和 mod 15 的合併次數還是偏低的。

● <3> 群集分裂次數

名稱	群集總數	群集分裂次數
原排序	38	3
完全亂數排序 (20 次取平均)	38	3
分組亂數排序 (16 次取平均)	38	2.9

表 4-8 三種排序的分群過程中的群集分裂次數比較

從表 4-8 可以看到，以分裂次數而言三種排序的差異不大，幾乎都是三次。當初在設計改良漸進式分群的演算法時，造成群集分裂的條件比較嚴苛，必須在同一群集之中，有 2 群成員之間的相似度有所差異，且這 2 群各別之中的成員彼此之間相似度極高，這才構成群集分裂的條件。在本研究的樣本群集之中只有 Spyware.Unix.Mech.A 和 Spyware.Unix.Mech、Trojan.Flooder.Linux.Small.0 和 Trojan.Flooder.Linux.Small.N、Trojan.Linux.Infostealer.A 和 HackTool.Linux.CleanLog，這三個組合會出現群集分裂的情況。在調整演算法的時期就曾發現，群集分裂的條件若是不夠嚴苛，分群錯誤的機率極高，且分析時間也會提高很多。所以才會寧可在分群過程中的合併次數多一點，避免在系統分析的後段產生太多分裂。

● <4> 樣本分析時間

名稱	系統分析總時間	平均每個樣本的分析時間
原排序	46 小時 23 分	5.11 分
完全亂數排序 (20 次取平均)	47 小時 20 分	5.21 分
分組亂數排序 (16 次取平均)	47 小時 55 分	5.28 分

表 4-9 三種排序的樣本分析時間比較


從表 4-9 中可以看出，基本上這三種排序在分析時間上並不會有太多

差異，不過在群集合併次數的實驗中，完全亂數和分組亂數分別都出現幾次 worst case 的排序，群集合併次數愈高，樣本的分析時間也會有些許的提升，導致在平均的分析總時間會高一些。

4.4 實驗三 本系統與 VirusTotal 之比較

本實驗會先將本研究的 545 個樣本上傳到線上惡意程式分析網站 VirusTotal 作分析，希望能夠建立起一套值得參考的分群結果。接下來再去比較本系統和 VirusTotal 的分析結果，希望可以確認本系統的準確率。

VirusTotal.com 可以針對病毒、木馬或惡意連結作分析，而且是免費的。VirusTotal 收集了 40 個以上防毒軟體大廠的分析引擎，只要使用者上傳樣本，就會顯示出所有防毒引擎的分析結果，這樣就能將誤判機率降到最低。由於可能有些防毒引擎判定為良性的樣本，在其他引擎的分析中卻會判定為惡意樣本，所以 VirusTotal 提供了最完整的分析服務。圖 4-6 就是 VirusTotal 分析某惡意樣本後所得到的結果。

SHA256:	59d352b7070160383d767662ace3e8d292fd9afb66dfe7c3f2f363e9cd3413dc	
File name:	20111217230329_http___4u_moy_su_bnc.jpg	
Detection ratio:	36 / 55	
Analysis date:	2014-11-24 09:34:18 UTC (6 months, 2 weeks ago)	

Analysis	Additional information	Comments 0	Votes
----------	------------------------	------------	-------

Antivirus	Result	Update
AVG	PERL/ShellBot	20141124
AVware	Backdoor.Perl.IRCBot.a (v)	20141121
Ad-Aware	Backdoor.Perl.Shellbot.B	20141124
Agnitum	Perl.Shellbot.I	20141123
Avast	Perl:Shellbot-O [Trj]	20141124
Avira	Perl/Shellbot.B.4	20141124
BitDefender	Backdoor.Perl.Shellbot.B	20141124
CAT-QuickHeal	Perl.ShellBot.B	20141122
ClamAV	Linux.Script.IRC	20141124
Comodo	Unclassified/Malware	20141124
Cyren	Unix/ShellBot.AA	20141124
DrWeb	Linux.BackDoor.Shell.6	20141124
ESET-NOD32	Perl/Shellbot.NAK.Gen	20141124
Emsisoft	Backdoor.Perl.Shellbot.B (B)	20141124
F-Prot	Unix/ShellBot.AA	20141124
F-Secure	Backdoor.Perl.Shellbot.B	20141123
GData	Backdoor.Perl.Shellbot.B	20141124

圖 4-6 VirusTotal 分析惡意程式的結果範例

不過 VirusTotal 的缺點就是每個引擎的命名方式不同，每個防毒引擎都有自己的一套惡意程式命名辭典，這樣在判斷此樣本的類別時會造成很大的困難。就如上圖所示，同一個惡意程式經過分析，就有 10 幾種不同的命名，雖然可以觀察出這些命名有些共同點，但畢竟每個引擎都是知名大廠，不能任意捨棄其中一部分的命名。為了分群方便，本研究會從所有引擎分析出的名稱中，找重複率最高的那個當作此樣本的分類名稱。如上圖所示，該惡意程式的分群命名就是 Backdoor.Perl.Shellbot.B。

表 4-10 是將所有樣本上傳到 VirusTotal 的分析結果與人工觀察的分群結果比較。

分群方式	有效群集	零星群集(只含單一樣本)
VirusTotal	30 群	48 群
人工觀察	38 群	2 群

表 4-10 VirusTotal 與人工觀察的分群結果比較

從上述表格可以看到 VirusTotal 分析的結果就會發現，VirusTotal 分析出的有效群集少很多，這 30 群中，有些群集融合了很多相異的群集，顯然 VirusTotal 的命名並沒有定義清楚。有許多相異的樣本，其內容可能有 2、3 個關鍵二進位檔不同，但 VirusTotal 還是將這些惡意程式分在同一個群集之中。有些相異樣本的內容中只有一個關鍵程式碼檔是相同的，其餘內容完全不相同，但 VirusTotal 卻分為同一群集之中。由此本研究懷疑 VirusTotal 在惡意程式的分群上可能有些缺陷。

表 4-11 是把 545 個樣本上傳到 VirusTotal 後的分析結果和本系統中改良漸進式分群的分析結果作比較，而當作對照比對的第三方標準則是由人工觀察所得出的分群結果，藉以決定兩種分群方式的準確率，並找出 VirusTotal 在分群上有哪些缺點。

名稱	有效群集總數	正確分群數量	錯誤分群數量	準確率(與人工比較)	平均每樣本處理時間
改良漸進式分群	38	517	28	94.9%	8 分鐘
VirusTotal	30	392	153	71.9%	趨近於 0

表 4-11 VirusTotal 與本系統的改良漸進式分群比較分群結果

這裡的結果有點出乎意料，VirusTotal 身為一個具有相當公信力的分析網站，準確率有點太低。以下說明 VirusTotal 的三個缺點，<1>過於注重 PERL 程

式檔、<2>對於 Shell Script 的變化不太重視和<3>眾多防毒引擎的命名規則不盡相同。

<1> VirusTotal 過於注重 PERL 程式檔

VirusTotal 只要在樣本中有掃描到任何一個 PERL，不管樣本中還有其他多少個二進位檔或是多麼不一樣的 Shell script，都會無條件的將這個樣本分類到 Backdoor.Perl.Shellbot.F 或 Backdoor.Perl.Shellbot.B。而如果樣本中只有一個 PERL 檔，也是分類到 Backdoor.Perl.Shellbot.F 或 Backdoor.Perl.Shellbot.B。PERL 檔在惡意程式中僅僅只是擔任其中一個功能而已，或許是 port scan，或許是感染成 BOT。在惡意程式中還有很多二進位檔會被執行。而 VirusTotal 的大部分防毒引擎都太過重視 PERL 檔的存在，這會造成分類上很大的誤差。

表 4-12 即為經 VirusTotal 分析後有爭議的部分。

A2-mechbot Backdoor.Mechbot.G：共 12 個樣本，其中都含有一個 PERL 檔作 portscan 用，在 VirusTotal 中被 15 個引擎分類到 Backdoor.Perl.Shellbot.F
A3-flooder Trojan.Flooder.Linux.Small.S：共 2 個樣本，其中都含有一個 PERL 檔作 portscan 用，在 VirusTotal 中被 11 個引擎分類到 Backdoor.Perl.Shellbot.F。
A4-cyc： Backdoor.F：共 2 個樣本，其中都含有一個 PERL 檔，但這個 PERL 檔完全沒有被執行或編譯過，在 VirusTotal 中被 7 個引擎分類到 Backdoor.Perl.Shellbot.F。
A6-psybnc：

其中有 19 個樣本都含有一個 PERL 檔，在 VirusTotal 中被 10 個引擎分類到 Backdoor.Perl.Shellbot.B
A7-makefile： 其中有 3 個樣本都含有一個 PERL 檔，在 VirusTotal 中被 11 個引擎分類到 Backdoor.Perl.Shellbot.B

表 4-12 VirusTotal 對於含有 PERL 檔的樣本誤判

<2> VirusTotal 對於 Shell Script 的變化不太重視

Shell Script 在惡意程式中的角色有一部分是用來將針對作業系統中的系統檔案，可能是移動、刪除、重新命名或刪除，這類的 Shell Script 不會去呼叫執行二進位檔，通常是用來作為額外的更新檔，而 VirusTotal 對於這類的 Shell Script 非常不重視。若是有 a, b 兩個樣本，兩個樣本的內容完全一樣，但 b 樣本卻多了兩個上述的 Shell Script 檔，若是經過 VirusTotal 的分析，這兩個樣本會分在同一群集中。

下述表格即為經 VirusTotal 分析後有爭議的部分。

A2-mechbot Backdoor.Mechbot.H 中的 19 個樣本和 Backdoor.Mechbot.G 中的 18 個樣本被 VirusTotal 分在同一群集。但其中卻有 1 個 Shell Script 檔的差異。
A3-flooder Trojan.Flooder.Linux.Small.N 和 Trojan.Flooder.Linux.Small.O 被 VirusTotal 分在同一群集。但其中卻有 1 個 Shell Script 檔的差異。

表 4-13 VirusTotal 對於含有 Shell script 的樣本誤判

<3> 眾多防毒引擎的命名規則不盡相同

有些防毒引擎的命名過於簡略，並可能沒有去針對某些樣本作特定的命名。例如，Symantec 對很多樣本的命名都出現 Hacktool.Rootkit 和 Hack.Linux，這些命名太過籠統，無法表示這個樣本的獨特性。

很多防毒引擎都會在命名最後加上 A~Z，用來表示這些樣本是經過某個樣本突變而來，卻又有些不同之處。但實際上卻常常只用了 A 和 B 這兩個字母，C 之後的字母很少用到。而在 A、B 這兩個字母的運用上卻又看不出規則。例如，Linux.RST.A 和 Linux.RST.B 這兩個名字常被很多大廠用來命名，但每個被叫作 Linux.RST.A 並沒有其獨特的特徵，與 Linux.RST.B 也沒有太多相關。例如 a 樣本被 Panda 命名成 Linux.RST.A，又被 TotalDefense 命名為 Linux.RST.B。再來，b 樣本和 c 樣本都被 AntiVir 命名為 LINUX/Rst.B，但 b、c 樣本在內容上卻完全不同。

4.5 實驗四 本系統與 Avira 防毒軟體之比較

本實驗的目的是比較本系統和個人電腦中的防毒軟體在分群準確率上的差別。本實驗所採用的防毒軟體為 Avira，俗稱小紅傘。Avira 在使用者介面的中文文化相當完整，在台灣網路社群的知名度和市佔率都非常高。

而這裡需要先介紹一下 Avira 的掃描機制和結果呈現。Avira 會掃描樣本內容中的每一個檔案，然後再將其中會被認為是惡意程式的檔案標示出來，而這類的檔案可能會有一個以上。Avira 不會針對這個惡意程式的封裝檔去定義一個命名，而是去標示出封裝檔中含有幾個不同的惡意程式。圖 4-7 即是將一個名為

20121120133521_http___linuxtrade_webs_com_xxplex_linuxteam_tar_gz 的惡意程式經 Avira 掃描後所得到的結果。

```

I:\outside workshop\1_3_http___linuxtrade_webs_com_xxplex_linuxteam_tar_gz
[0] 封存類型: GZ
--> AV0000005d.AV$
[1] 封存類型: TAR (tape archiver)
--> scan/gen-pass.sh
[偵測] 包含 EXP/Spy.233 惡意探索程式碼的辨識模式
[警告] 無法修復封存中受感染的檔案!
--> scan/a
[偵測] 包含 LINUX/WrappSkript.A Linux 病毒的辨識模式
[警告] 無法修復封存中受感染的檔案!
--> scan/auto
[偵測] 包含 SPR/Tool.Sshscan.C 程式的辨識模式
[警告] 無法修復封存中受感染的檔案!

```

圖 4-7 Avira 分析惡意程式後的分析結果範例

可以看到這個惡意程式封裝檔中含有三個惡意程式檔：gen-pass.sh, a 和 auto。這三個檔案分別有個字的命名。在本實驗中只要兩個惡意程式中所含有的惡意檔案擁有相同的名字，就將兩個惡意程式分為同一類別。

本實驗中是以本系統的改良漸進式分群演算法來與 Avira 作比較，表 4-14 即為比較結果。所依據的標準仍是在第一節所列的人工觀察結果。

名稱	群集總 數	正確分 群數量	錯誤分群 數量	準確率
改良漸進式 分群	38	517	28	94.9%
Avira	42	497	48	91.2%

表 4-14 改良漸進式分群與 Avira 的分群準確率比較

由上方表格所示，Avira 的分群準確率有別於 virustotal，高達九成。若是仔細觀察 Avira 的掃描過程，可以發現 Avira 對於惡意程式的判斷相當公正，並沒有特意注重 PERL 程式碼檔或是二進位檔。不過在針對 Shell script 的判斷上卻會有些許的漏失，有些用來掃描網路區段中 IP 位址的 Shell script 和有些用來呼叫二進位檔作執行用的 Shell script 都沒有被 Avira 認

定為惡意檔案，所以仍有 48 個樣本被錯誤分群。表 4-15 是 Avira 所誤判樣本群集。

<p>A2-mechbot</p> <ol style="list-style-type: none">1. Spyware.Unix.Mech.A：這個類別中因為有 1 個 Shell script 和 1 個二進位檔沒有被 Avira 抓到，因此都被分到 Backdoor.Mechbot.F 中2. Backdoor.Mechbot.F 有三個樣本中的 1 個 Shell script 沒有被 Avira 抓到，而被自立為一個新群集。3. Backdoor.Mechbot.H 中有 2 個樣本中的二進位檔沒有 Avira 抓到，而被自立為一個新群集
<p>A3-flooder</p> <ol style="list-style-type: none">1. Trojan.Flooder.Linux.Small.0 有一個樣本中的 1 個 Shell script 沒有被 Avira 抓到，而被自立為一個新群集。
<p>A4-cyc：</p> <ol style="list-style-type: none">1. Backdoor.F 有 4 個樣本中有 2 個 Shell script 和 1 個二進位檔沒有被 Avira 抓到，而被自立為一個新群集。
<p>A1-portscan</p> <ol style="list-style-type: none">1. Virtool.Linux.Shark.A 有 3 個樣本中有 1 個 Shell script 額外被 Avira 抓到，而被自立為兩個新群集。
<p>A7-makefile：</p> <p>其中 3 個樣本有 1 個二進位檔額外被 Avira 抓到，而被自立為一個新群集。</p>

表 4-15 Avira 誤判樣本所屬群集

第五章 結論

學術網路和企業網路永遠是網路攻擊事件最頻繁的戰場，由於其封閉性和內部流量龐大，一旦惡意程式找到缺口而入侵到內部網路區段，造成的傷害是無法想像的。雖然在資安的角度上，防禦措施是設計的愈來愈嚴密，但資安中最大且最無法防禦的漏洞永遠是使用者，所以惡意程式的偵測是必要且緊急的。在惡意程式中，程式碼檔會負責去呼叫具攻擊行為的二進位檔，只要能阻止這兩者，就能阻止惡意程式的攻擊。

本研究提出結合兩種分群演算法的惡意程式分析系統，藉由分析惡意程式中的二進位檔、程式碼檔和檔案結構，擷取其中足以代表惡意程式的特徵值，透過改良漸進式分群和延伸式 1-NN 兩種分群演算法，期望能夠得到又快速又準確的分群結果，及整理出惡意程式群集之間的關係，以幫助加速資安人員對於惡意程式的分析與應對。根據第四章四個實驗的結果顯示，本研究在於惡意程式的分群上，的確能比其他系統更快速且準確的進行分析。

本研究仍有一些有待改進的地方，本研究的特徵擷取只適用於 Linux 系統中的惡意程式，對於 Windows 系統的惡意程式而言，可能要再作特徵擷取的調整，像是 JAVA 程式碼檔和 PE 執行檔。

參考文獻

- [1] L. Gordon, M. Loeb, W. Lucyshyn, and R. Richardson. Computer Crime and Security Survey. Technical report, Computer Security Institute (CSI), 2005.
- [2] Symantec ISTR20 INTERNET SECURITY THREAT REPORT VOLUME 20. 2015 April.
- [3] Symantec ISTR20 INTERNET SECURITY THREAT REPORT VOLUME 20 APPENDICES. 2015 April.
- [4] Hao Bai, Chang-zhen Hu, Xiao-chuan Jing, Ning Li, Xiao-yin Wang “Approach for malware identification using dynamic behaviour and outcome triggering” Journal in *Information Security IET*, Vol. 8, Iss. 2, 2014, pp. 140-151
- [5] Sheng Wen, Wei Zhou, Jun Zhang, Yang Xiang, Wanlei Zhou, Weijia Jia, Cliff C. Zou “Modeling and Analysis on the Propagation Dynamics of Modern Email Malware” Journal in *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING*, VOL. 11, Iss. 4, JULY/AUGUST 2014, pp. 361–374
- [6] Lei Cen, Christoher S. Gates, Luo Si, and Ninghui Li “A Probabilistic Discriminative Model for Android Malware Detection with Decompiled Source Code” journal in *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING*, VOL. 12, Iss. 4, JULY/AUGUST 2015, pp. 400-412
- [7] Deniz Yuret “FASTSUBS: An Efficient and Exact Procedure for Finding the Most Likely Lexical Substitutes Based on an N-Gram Language Model” journal in *IEEE SIGNAL PROCESSING LETTERS*, VOL. 19, Iss. 11, NOVEMBER 2012, pp. 725-728
- [8] Nicole L. Beebe, Laurence A. Maddox, Lishu Liu, Minghe Sun “Sceadan: Using Concatenated N-Gram Vectors for Improved File and Data Type Classification” journal in *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY*, VOL. 8, Iss. 9, SEPTEMBER 2013, pp. 1519-1530
- [9] Nikita Jain, Rashi Garg, Indu Chawla “Concept Localization using n-gram

Information Retrieval Model and Control Flow Graph”in proceedings of *Confluence 2013: The Next Generation Information Technology Summit (4th International Conference)* , 26-27 Sept. 2013, pp. 29-34

[10] M. Shankarapani, K. Kancherla, S. Ramammoorthy, R. Movva and S. Mukkamala, “Kernel machines for malware classification and similarity analysis,” in Proceedings of *International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1-6.

[11] S. Cesare, Y. Xiang and W. Zhou, “Malwise—an effective and efficient classification system for packed and polymorphic malware,” Journal in *IEEE Transactions on Computers*, 2013, pp. 1193-1206

[12] B. Kang, H.S. Kim, T. Kim, H. Kwon and E.G. Im, “Fast malware family detection method using Control Flow Graphs ,” in Proceedings of *the 2011 ACM Symposium on Research in Applied Computation*, 2011, pp.287-292.

[13] H. Agrawal, L. Bahler, J. Micallef, S. Snyder and A. Virodov, “Detection of global, metamorphic malware variants using Control and Data Flow Analysis ,” in Proceedings of *MILITARY COMMUNICATIONS CONFERENCE*, 2012, pp. 1-6.

[14] G. Conti, S. Bratus and A. Shubinay, “ A visual study of primitive binary fragment types ,” *Black Hat USA*, 2010.

[15] L. Nataraj, S. Karthikeyan, G. Jacob and B.S. Manjunath, “ Malware images: visualization and automatic classification,” in proceedings of *the 8th International Symposium on Visualization for Cyber Security*, 2011.

[16] Maya Louk, Hyotaek Lim, HoonJae Lee , Mohammed Atiquzzaman “An Effective Framework of Behavior Detection- Advanced Static Analysis for Malware Detection”in proceedings of *2014 International Symposium on Communications and Information Technologies (ISCIT)*, 24-26 Sept. 2014, pp. 361 – 365

[17] Young Han Choi, Byoung Jin Han, Byung Chul Bae, Hyung Geun Oh, Ki Wook Sohn “Toward Extracting Malware Features for Classification using Static and Dynamic Analysis”in proceedings of *2012 8th International Conference on Computing and Networking Technology (ICCNT)*, 27-29 Aug. 2012, pp. 126 – 129

- [18] M.K. Shankarapani, S. Ramamoorthy, R.S. Movva and S. Mukkamala, “Malware detection using assembly and API call sequences,” *Journal in Computer Virology*, 2011, pp.107-119.
- [19] Hesham Mekky, Aziz Mohaisen, Zhi-Li Zhang. Blind Separation of Benign and Malicious Events to Enable Accurate Malware Family Classification. *ACM 978-1-4503-2957-6/14/11*.2014.
- [20] Jeff Gennari, David French. Defining Malware Families Based on Analyst Insights. *IEEE 978-1-4577-1376-7/11/*. 2011.
- [21] Yang Zhong, Hirofumi Yamaki, Yukiko Yamaguchi, Hiroki Takakura. ARIGUMA Code Analyzer: Efficient Variant Detection by Identifying Common Instruction Sequences in Malware Families. *IEEE DOI 10.1109/COMPSAC*. 2013.
- [22] Gregory Blanc, Ruo Ando, Youki Kadobayashi “Term-Rewriting Deobfuscation for Static Client-Side Scripting Malware Detection”in proceedings of 2011 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 7-10 Feb. 2011, pp. 1 - 6