國立交通大學

網路工程研究所

碩 士 論 文

整合靜態分析及動態分析結果
作為機器學習標準的
Android 惡意程式偵測系統

An Android Machine Learning
Malware Detection System
Using the Result of
Static Analysis and Dynamic Analysis
as the Features

研 究 生：蔡立倫

指導教授：曾文貴　教授

中 華 民 國　１０３　年　９　月

整合靜態分析及動態分析結果作為機器學習標準的 Android 惡意程式偵測系統

An Android Machine Learning Malware Detection System Using the Result of Static Analysis and Dynamic Analysis as the Features

研 究 生：蔡立倫　　Student：Li-Luen Tsai

指導教授：曾文貴　　Advisor：Wen-Guey Tzeng

國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

September 2014

Hsinchu, Taiwan, Republic of China

中華民國 103 年 9 月

整合靜態分析及動態分析結果作為機器學習標準的 Android 惡意程式偵測系統

學生：蔡立倫　　**指導教授：曾文貴**

國立交通大學網路工程研究所碩士班

摘　　要

現在的智慧型手機具有各式各樣強大的功能，因此有越來越多的人將智慧型手機當成隨身的個人電腦來使用。其中 Android 是一個擁有相當多使用者的智慧型手機系統，許多使用者都喜歡其開放性，然而惡意程式開發者也藉由其開放性，來危害使用者。

因為目前防毒軟體偵測惡意程式的方法，是藉由辨認特徵碼來判別惡意程式，然而目前 Android 手機惡意程式發展相當迅速，取得特徵碼的辨識方式過於緩不濟急，使用者無法藉由安裝防毒軟體以獲得真正的保障。

因此本論文提出一套結合靜態分析與動態分析的系統，藉由實行這兩種分析以取得應用程式多面相的特徵屬性，並且藉由機器學習演算法，將這些特徵屬性進行分類，以分辨該應用程式是否為惡意程式。

本論文於動態分析部分，實作一個可辨識使用者介面之自動行為觸發程式，以更擬真的模擬使用者操作應用程式之動作，以確實激發應用程式功能；並且提出一種新的特徵屬性，以利用應用程式執行期間所使用之 system call 順序，提升惡意程式分辨率。本論文也取得大量應用程式樣本，並且證實使用這種行為觸發程式和這些特徵屬性，可以準確地判斷惡意程式。

關鍵詞：Android、惡意程式偵測、機器學習、靜態分析、動態分析、行為觸發。

An Android Machine Learning Malware Detection System Using the Result of Static Analysis and Dynamic Analysis as the Features

Student: Li-Luen Tsai Advisor: Wen-Guey Tzeng

Institute of Network Engineering
College of Computer Science
National Chiao Tung University

ABSTRACT

Nowadays, there are a lot of functions on the smart phone, so more and more people take their smart phone like a portable personal computer. Android is one kind of smart phone system with a lot of users. Many users like its ability of installing apps from unverified sources, but attackers also use this ability to harm the users. Antivirus usually use signatures to detect malware, but Android malware develop too fast. This method is too slow, so users can not protect themselves with installing antivirus on their smart phone.

In this paper, we present an Android machine learning malware detection system. This system uses the result of static analysis and dynamic analysis as its features to do machine learning, and determine whether this application is malware or not.

In the part of the dynamic analysis, we propose an automatic behavior trigger which can identify the user interface on the screen. This behavior trigger simulates events from a user's interaction with this app to trigger the functions of this app. We also propose a new feature set, and this kind of feature set can record the sequence of the system calls to elevate the detection rate. We have got a lot of samples to prove our system can use this behavior trigger and this kind of feature set to distinguish malware from normal apps.

Keywords: Android, Malware Detection, Machine Learning, Static Analysis, Dynamic Analysis, Behavior Trigger.

誌　　謝

首先我要感謝我的指導教授曾文貴老師兩年來的指導，從一開始循序漸進的了解密碼學知識到後來的論文方向提供與各種指導，讓我的論文從無到有。即使老師去了美國仍舊每個禮拜撥空聽取我的進度，並且給予各種意見，假如沒有老師的 push 我真的不會做出這樣與他人不同的動態分析程式，真的十分感謝曾老師的指導，老師謝謝您！

同時也要感謝口試委員孫宏民老師和蔡錫鈞老師，能夠在這個快要開學的時期，撥空參加我們的口試，並且給予我們許多的意見，謝謝兩位老師！

然後是要感謝毅睿學長與宣佐學長，一直指導我如何報告、講解各種專有名詞、以及平常的論文進度關心，讓我從一開始連實驗室要報告的論文內容都搞不懂，到現在能夠好好地呈現給口試委員我的論文以及論文貢獻，真的很感謝兩位學長兩年的指導，謝謝學長！Amir 也是，雖然跟你相處的這一年，基本上真的都沒什麼對話，但是我真的很感謝你這幾天，還特地幫我們校對口試投影片，以免被教授笑話，真的很感謝你！還有維揚學長，感謝你將 Android 的各種知識傳授給我，以及最後還可以從你的論文作為出發點，好來完成我的碩士論文。

接下來是要感謝我在交大的各位同學與朋友，陪伴我度過這兩年的碩士生涯、陪伴我熬過每個生進度的日子與趕進度的夜晚、包容我的各種白目與耍蠢，假如沒有你們，我是不可能能夠熬過這兩年的，真的很感謝你們！

當然也要感謝我的父母，白手起家養育我 24 年，從沒讓我餓過、或者有所不滿足過，真的很感謝你們聽了我 24 年來的各種任性，讓我現在可以順利完成碩士論文，謝謝爸媽！

因為要感謝的人，幫助我度過這兩年碩士生涯的人實在太多了，所以這邊就不逐一列舉了。最後還是要節錄一句陳之藩先生寫過的話：「因為需要感謝的人太多了，就感謝天罷。」。

蔡立倫 謹誌

2014 年 9 月

目　　錄

圖目錄

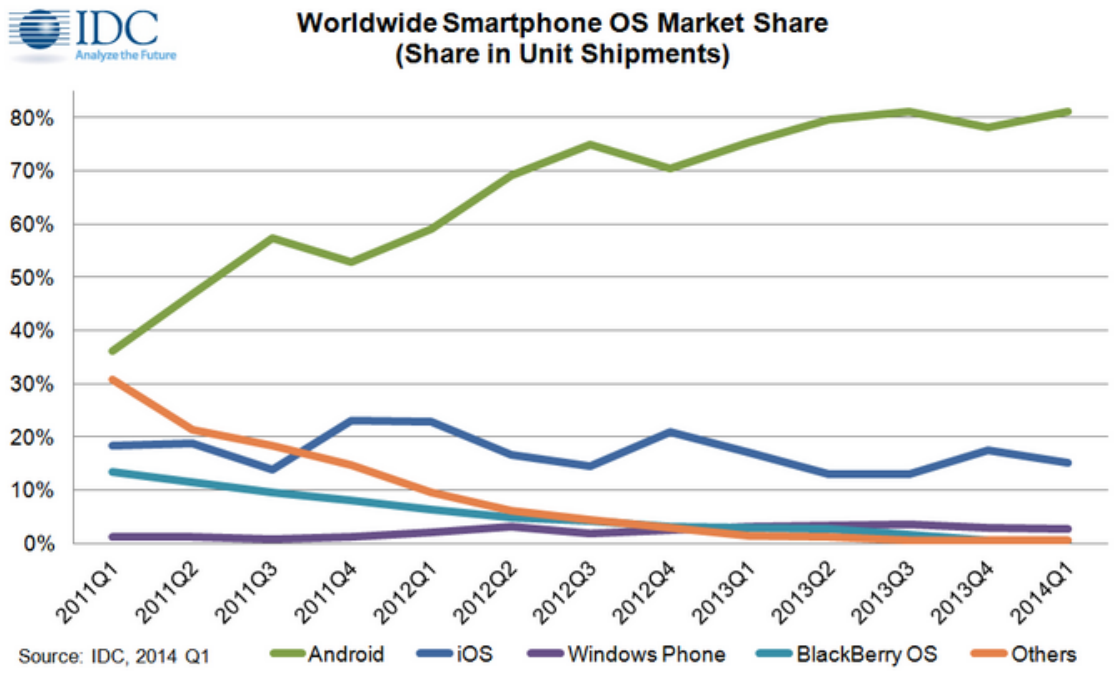
表目錄

第一章 介　　紹

1.1 背景簡介

隨著科技的日新月異，高階手機變得越來越平價，也有越來越多民眾以智慧型手機(smart phone)取代傳統手機。所謂的智慧型手機，即是具有開放式作業系統，不同於傳統手機在出售時已將所有功能都加載於手機上；智慧型手機可藉由有線或無線傳輸，自由地安裝或移除應用軟體(application)以擴充手機功能。

根據 IDC 調查結果[1]，Android 系統在 2014 年第一季已經達到全球智慧型手機系統市占率 81.1%，而第二名的 iOS 系統市占率則是 15.2%，該資料顯示目前全世界有八成以上智慧型手機使用者都在使用 Android 智慧型手機系統。



Period	Android	iOS	Windows Phone	BlackBerry OS	Others
Q1 2014	81.1%	15.2%	2.7%	0.5%	0.6%
Q1 2013	75.3%	17.1%	3.2%	2.9%	1.5%
Q1 2012	59.2%	23.0%	2.0%	6.3%	9.5%
Q1 2011	36.1%	18.3%	1.2%	13.6%	30.8%

Source: IDC, 2014 Q1

圖 1 過去三年各季全世界智慧型手機市占率折線圖

Android 系統主要優點在於其開放性。開發者可以編寫應用程式，並且將應用程式放置於網路上[2] ；使用者也可以自由地下載並安裝這些應用程式於手機上，因此在網路上有各式各樣大量的應用程式可供使用者下載使用。然而惡意程式(malware)也因為此開放性，藏身於一般的應用程式中，藉由提供其他一般熱門應用程式的功能，相當容易地散播於 Android 手機間，竊取使用者資訊，或者造成使用者財物上的損失。

Android 系統為了避免使用者受到惡意程式的危害，雖然沒有提供應用程式放置於官方下載網站前的檢測，取而代之的使用權限系統(permission)，來處理該問題。開發者在使用部分特定的 API 時，必須先在 AndroidManifest 檔案內宣告其對應的權限，方能使用該 API；Android 系統也會在使用者安裝該應用程式時，顯示該應用程式會取用的特定權限，希望使用者藉此自行過濾含有惡意行為的應用程式。然而大部分使用者並不知道系統顯示應用程式所需權限的意義，往往盲目地忽略掉並且按下接受以安裝應用程式，以致權限系統無法有效地限制惡意程式的散佈。

1.2 研究動機

目前防毒軟體偵測惡意程式的方法主要在於辨認該程式是否含有特定特徵碼(signature)，作為分辨該程式是否為惡意程式的依據。

然而必須要確認特定惡意行為後，才可進一步取得特徵碼，因此需要耗費一定時間才能取得特徵碼，但是目前 Android 手機惡意程式發展相當迅速，取得特徵碼的辨識方式過於緩不濟急，無法應對快速變化的惡意程式，使用者無法藉由安裝防毒軟體則可以獲得真正的保障。

1.3 貢獻

我們開發一個不僅利用靜態分析(static analysis)收集應用程式的特徵屬性(feature)，更藉由動態分析(dynamic analysis)與新提出的特徵屬性，自動且有效地觸發應用程式行為並且紀錄這些應用程式行為，然後將這些行為轉換成特徵屬性交由不同機器學習演算法(machine learning)，獲得分類結果，以告訴使用者該應用程式是否為惡意程式或者只是一般應用程式，來產生在第一時間就有能力分辨出惡意程式的防範系統。

1.4 各章節簡介

後續章節內容分別為：第二章簡介相關 Android 惡意程式研究、以及簡介兩種主要惡意程式分類方法。第三章介紹本系統收集的三大類特徵屬性、並且簡介建置本系統使用之工具。第四章詳細介紹本系統建置方法與過程。第五章為機器學習實驗結果與討論。第六章則為本論文結論。

第二章 相關研究

近年來，因為 Android 惡意程式越來越多，也逐漸危害到一般使用者，因此不斷有相關論文被發表，關於 Android 惡意程式各種家族研究較為著名的為這篇由 Zhou 和 Jiang 所發表的論文[7] 、以及 Felt 等人的這篇[8] ，Felt 等人這篇不僅研究 Android 的樣本，也有研究 iOS 和 Symbian 的樣本。

辨識 Android 惡意程式的方法大致上可以分為兩種：(1) 特徵碼比對、(2) 機器學習分類，以下會對這兩種方法做簡單介紹。

第一種方式是要事先定義惡意行為的程式碼片段，以製作成特徵碼，然後依照這些特徵碼以進行惡意程式比對，這種方法優點在於可以準確偵測惡意程式，因為只要比對相同即可代表該樣本為惡意程式；缺點在於需要時間及樣本以產生新的特徵碼，因此對於新的惡意程式反應較慢。

使用第一種方式的論文大致上有下面幾篇：Fuchs 等人提出 SCanDroid[9] ，從應用程式的 AndroidManifest 檔案萃取出該應用程式的安全規則，再藉由使用資料流分析(data flow analysis)驗證是否有任何資料流違反上述規則；Enck 等人提出 Kirin[10] ，藉由比對特定權限組合(permission combination)，以辨識潛在的惡意程式；Zhou 等人提出 DroidRanger[11] ，該系統藉由應用程式宣告的權限、和事先定義的惡意行為，可辨識出已知的惡意程式，再藉由是否會動態下載新的程式碼，以偵測未知的惡意程式。

第二種方法則是只要定義收集的特徵屬性，例如：宣告的權限、使用的 API…等特徵屬性，系統即可針對這些特徵屬性作收集，在藉由事先學習產生的分類模組(model)進行分類，產生偵測結果。優點是藉由具有象徵性的樣本學習產生的模組，有能力分辨出新的惡意程式；缺點則是有一定的誤報率，就是有可能會將一般的應用程式分類為惡意程式，不過為了應對 Android 惡意程式的大量且快速的變形，我們寧願誤報而不希望產生漏報，以盡可能防堵惡意程式接觸到使用者。

使用第二種方式的論文大致上有下面幾篇：Burguera 等人提出 Crowdroid[12] ，紀錄使用者使用 app 時產生的 system call 紀錄，並使用 partitional clustering 演算法進行分類；Peng 等人[13] 則是利用應用程式的權限以及 probabilistic learning 方法來辨識有危險性的應用程式；Aafer 等人提出 DroidAPIMiner[14] ，利用應用程式使用的 API 作為特徵屬性，並且為了分辨常見於一般應用程式和惡意程式的 API，他們還使用資料流分析以回復這些 API 參數，提升一般應用程式與惡意程式的分辨率，最後使用幾種不同的機器學習獲得分類結果；Wu 等人提出 Droidmat[15] ，利用從 AndroidManifest 檔案內宣告的權限、應用程式元件(App Component)、聽取的意圖(intent)、以及應用程式 bytecode 內受權限限制的 API 總共 4 大類的特徵屬性，並使用 k-means 以針對不同目的的惡意程式做分群，最後使用 KNN 進行分類；Arp 等人提出 Drebin[16] ，利用從

AndroidManifest 檔案收集到的使用的硬體、宣告的權限、應用程式元件、聽取的意圖以及從應用程式的 bytecode 反組譯出的程式碼內收集到受權限限制的 API、實際使用的權限、特定字串、網路位址總共 8 大類的特徵屬性，並使用 SVM 進行分類，以過濾使用者下載的 apk 檔案。

第三章 系統與工具簡介

3.1 Android permission 系統簡介

Android 系統提供各式各樣的函式，給予開發者極大的便利性以開發出千變萬化的應用程式，然而有些功能可能造成其他應用程式、Android 系統、甚至使用者的損害，例如：應用程式可以不受限制的讀取使用者的私密資料，並且可以自行開啟網路連線，將收集到的資料上傳到伺服器，危害到使用者隱私。

為避免應用程式獲得過多的能力，Android 因此提供權限(permission)系統，來讓使用者可以管控應用程式的行為。回到前一段例子，應用程式必須要先宣告讀取使用者私密資料與網路連線的權限(READ_PHONE_STATE & INTERNET)於 AndroidManifest 檔案，該應用程式方能讀取使用者私密資料與網路連線這些功能，並且 Android 系統會在使用者安裝該應用程式時，告訴使用者該應用程式要求使用這些權限，使用者即可判斷該應用程式是否確實需要這些功能。

例如：一些通訊應用程式可能需要讀取使用者的聯絡人電話，並且回傳到應用程式伺服器，以尋找是否存在這些聯絡人的帳號，以通知使用者，幫助使用者與這些帳號進行通訊；另外假如有一個外語辭典應用程式、或者應該不需要取得私密資料的應用程式類型，竟然向使用者要求讀取使用者私密資料與網路連線的權限，使用者則可以藉此懷疑該應用程式或許是假借外語辭典功能、或一般應用程式的外表，其實主要目的是想竊取使用者私密資料。

因為從 AndroidManifest 取得應用程式所宣告之權限相當容易，因此早期的 Android 惡意程式偵測，大部分皆以此作為觀察的特徵屬性，希望藉此取得準確的惡意程式分類結果，然而因為 Android 本身文件說明不夠明確[17] [18] ，導致部分開發者會宣告過多權限，容易對單純觀察權限之分類系統產生誤報(False Positive)；另外有其他論文指出[19] [20] ，有些惡意程式甚至可以在沒有宣告特定權限情況下，利用其他具有安全漏洞的一般應用程式，幫助惡意程式達到惡意行為。

3.2 特徵屬性簡介

本系統收集三大類特徵屬性分別為：應用程式宣告之權限、應用程式執行期間的網路封包資料洩漏行為、以及呼叫 system call 的順序產生之 n-gram，以下會逐一對這三類特徵屬性做詳加敘述。

3.2.1 應用程式權限

如同 3.1 章所述，Android 會要求應用程式在使用特定的 API 功能時，需要宣告權限，以便讓使用者知道該應用程式會使用這些權限，並且從使用者已知的該應用程式功能，考慮該應用程式擁有這些權限是否正常？是否宣告了一些從該應用程式名稱或簡介而言，不應該出現的權限？

因為目前 Android 惡意程式主要的惡意行為為：收集使用者隱私資料、收發付費簡訊...等危害使用者隱私、或者財產的行為。這些行為跟某些特定的權限有關，例如：讀取手機的 IMEI、IMSI、手機重要資料需要 READ_PHONE_STATE 權限；讀取、撰寫、收發簡訊分別需要這四種權限：READ_SMS、WRITE_SMS、RECEIVE_SMS、SEND_SMS。因此本系統認為應用程式權限為一個重要的特徵屬性，可以做為機器學習分類的重要依據。

目前大部分論文僅收集 Android 官方所公佈的標準權限[23] ，但是本系統認為應該收集應用程式所宣告之全部權限，即不管是 Android 官方公布的標準權限，本系統也收集其他特別的權限，或者開發者自行定義之權限，例如：Android 後來才釋出的新功能 Android Cloud to Device Messaging (C2DM)，最近改版為 Google Cloud Messaging for Android (GCM)[24] ，該框架幫助開發者將伺服器資料傳送給手機端的應用程式，例如通訊軟體的即時訊息、或者通知手機端的應用程式至伺服器端獲得資料，例如電子信箱軟體收到新郵件，伺服器通知手機端應用程式，以告訴使用者收到新郵件。開發者想使用這樣的功能，需要先行宣告 C2DM 的權限，本系統認為像這種權限有較高機率可以判斷該應用程式為非惡意的，因此本系統會收集應用程式的全部權限，並不會作特定的過濾。

因為 Android 官方並沒有詳細說明 API 與權限的對應關係，所以有些開發者不清楚其使用到的 API 需要那些權限，會造成應用程式有權限過度宣告的情形[17] [18] ；甚至有些惡意程式可以藉由其他一般無害的應用程式漏洞，幫助實行惡意行為，而這些惡意程式則可以不用宣告這些惡意行為的權限，遭受利用的應用程式有宣告這些權限即可[19] [20] 。因此只收集應用程式權限是不夠的，本系統想藉由收集以下兩種特徵屬性來補足這部分缺失，以更精確的辨識惡意程式。

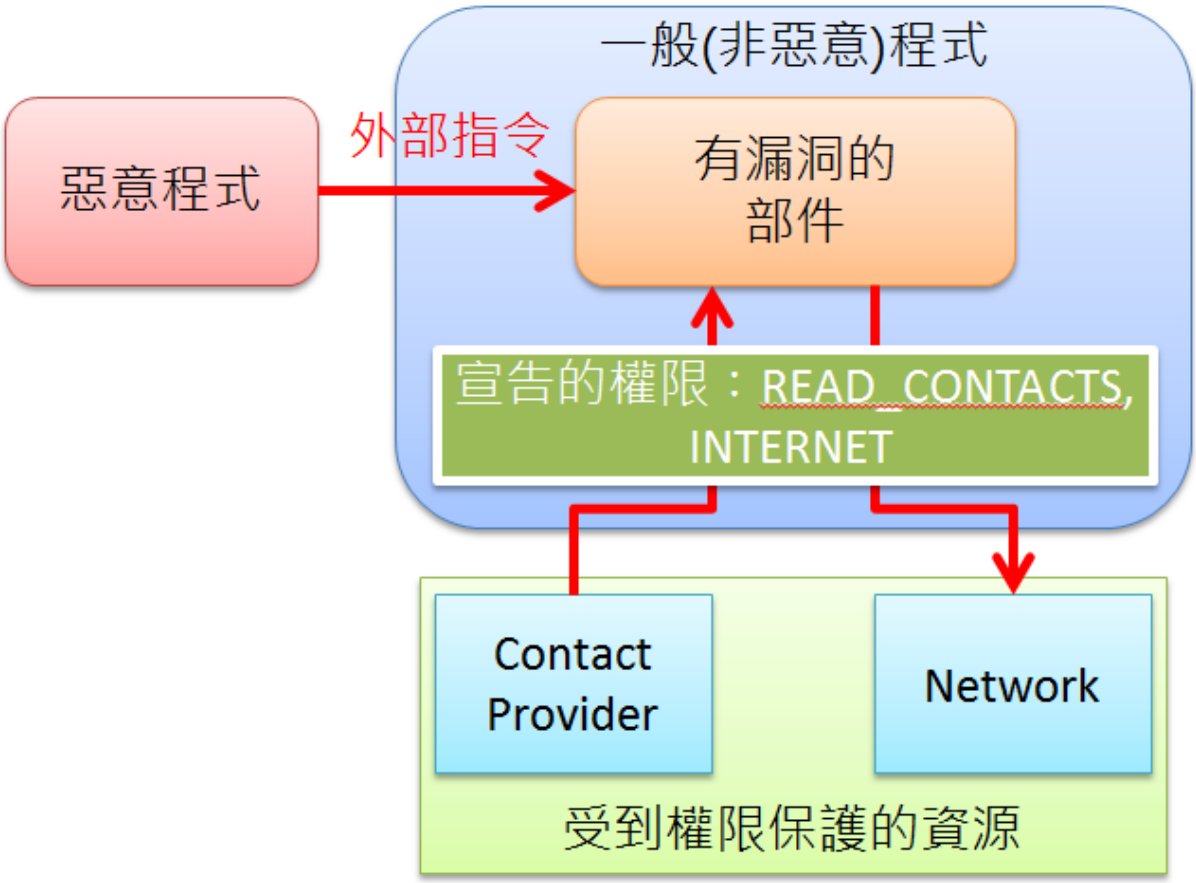


圖 2 惡意程式無須宣告權限即可達成惡意行為

3.2.2 網路封包資料洩漏

該特徵屬性為先前實驗室學長所提出[25] ，主要定義應用程式的七種將手機內資料上傳到網路的行為：傳送通訊錄、傳送 IMEI、傳送 IMSI、傳送 ISO、傳送手機本身號碼、傳送手機 SIM 卡號碼、傳送手機內簡訊。因為本系統認為當觀察到應用程式實際執行上述行為，該應用程式有很高機率為惡意程式，因此選擇將這七種行為加入到本系統的特徵屬性集合內，更多關於這些特徵屬性的詳細資訊，可以參考[25] 。

3.2.3 system call n-gram

應用程式在執行期間，會使用到許多種不同 system call，先前有論文是單純觀察應用程式執行期間的 system call 次數統計，本系統藉此聯想，是否可以觀察 system call 的使用次序，以更接近應用程式的行為？因此本論文提出 system call n-gram 作為觀察 system call 執行順序的特徵屬性。

所謂的 n-gram 即是在一整段文章內、或一個字串內，由 n 個連續項目所組成的序列，舉例來說：對這串字串” 整合靜態分析及動態分析結果作為機器學習標準”做 n=3 的 n-gram，會產生” 整合靜”、” 合靜態”、” 靜態分”、” 態分析”、…、” 學習標”、” 習

標準”的 n-gram。

在 system call 使用順序上，例如現有一個應用程式的 system call 使用順序：open、read、close、open、write、close、exit，本論文對此順序做 n=3 的 n-gram 則會產生”open-read-close”、”read-close-open”、”close-open-write”、”open-write-close”、”write-close-exit”的 n-gram。對原始的 system call 直接做統計會產生 open 跟 close 使用兩次、其他 system call 使用一次，直接統計結果與取得 n=1 的 n-gram 結果相同；使用 n=3 的 n-gram 對目前的使用順序而言，則是產生每個 n=3 的 n-gram 都使用一次的結果。本系統想藉由 n-gram 觀察，以 system call 的執行順序作為特徵屬性，是否可以增加惡意程式的辨識率？

3.3 系統簡介

本系統會先藉由受測的應用程式安裝檔(apk)取得其宣告的權限(permission)，接著安裝受測的應用程式以進行自動化應用程式行為觸發，並且記錄受測應用程式執行期間的網路封包資料洩漏行為、以及呼叫 system call 的順序產生之 n-gram，最後將上述收集到的行為轉成向量，交由不同的機器學習演算法辨認受測應用程式是否為惡意程式，因此會用到以下工具。此處僅簡介工具功能以及本系統使用該工具之情況，詳細使用過程描述在第四章。

3.4 使用工具簡介

3.4.1 Robotium

Robotium[3] 是一個 Android 測試自動化框架，當開發者需要自動化測試應用程式的使用者介面(User Interface)是否正常、是否會產生預期以外的錯誤時，開發者可以利用 Robotium 提供的函式，撰寫測試程式，即可自動化測試應用程式，以節省開發者手動測試的負擔，並且可以在測試過程產生提示文字，幫助開發者了解測試過程及測試結果。

因為本系統需要自動化模擬使用者操作應用程式的事件，以激發出應用程式行為，藉此作為機器學習的特徵屬性，因此我們需要一個工具可以確實了解目前使用者介面上有多少按鍵，並且可以確實點擊到這些按鍵的工具，Robotium 有提供這些功能，可以幫助我們確切地點擊到使用者介面上的按鍵，於是我們選擇 Robotium 以激發應用程式行為。

Monkey[4] 為一個 Android 提供的隨機自動產生使用者事件，例如：點擊、觸摸、或者滑動...等使用者事件的工具，雖然本系統也可以藉此激發出應用程

式行為，但是 Monkey 並不知道實際上使用者介面上有那些按鍵、或者那些圖片可以提供使用者做點擊，因此有可能執行 Monkey 數分鐘，卻遲遲點選不到進入應用程式的”START”按鍵，可見在於有效激發應用程式行為上，相較於 Monkey，Robotium 明顯有效。

使用 Robotium 的缺點在於，系統必須在測試程式的程式碼內，撰寫上受測應用程式的資訊，即受測應用程式的套件名稱(package name)與受測應用程式的啟動 activity(launcher activity)，以便測試程式執行受測程式。開發者所撰寫的測試程式只對應單一受測應用程式，但是本系統需要檢測不同的應用程式，即有多個受測應用程式，因此本系統需要額外功能以取得受測應用程式資訊、修改測試程式程式碼、並且自動編譯，以繼續執行惡意程式偵測，於是需要以下取得受測應用程式資訊工具 aapt、和自動編譯工具 ANT。

另外，受測應用程式必須和 Robotium 所撰寫的測試程式，擁有相同簽章，因為系統所處理的受測應用程式其原本簽章不可能與測試程式相同，因此系統必須刪除原有簽章，並且簽署與測試程式相同簽章，以進行後續受測應用程式的執行，於是需要以下工具 aapt 和 jarsigner。

3.4.2 Another Neat Tool(ANT)

ANT[5] 是一個可以將軟體編譯、測試、部署等步驟聯繫在一起的自動化構建工具，該工具多用於 Java 程式開發，因此 Android 也可以藉由 ANT 簡化專案編譯過程[6] 。

藉由這篇文章的介紹[21] ，可以知道當系統已經有一份完整的 Android 專案以後，系統需要做到以下事情以便產生測試程式：
(1) 編譯程式碼, (2) 產生 DEX 檔案, (3) 產生 APK 檔案。需要完成以上步驟並且給予繁複參數，才能夠產生可以安裝並且執行的測試程式；不過當系統運用 ANT 工具，系統只需給予 build.xml 告訴 ANT 如何建置該 Android 專案，然後在執行 ANT 即可自動產生 APK 檔案。

3.4.3 android 工具

該章節的 android，並不是作業系統 Android，而是 Android 提供的開發者工具[26] 。android 可以幫助開發者使用指令產生 Android 專案、更新現有專案、或者產生函式庫專案等專案相關功能，因為本系統需要 build.xml 以便讓 ANT 產生測試程式 apk 檔案，因此本系統可以利用 android 工具產生 build.xml，以接續給 ANT 產生 apk 檔案。

3.4.4 Android Asset Packaging Tool (aapt)

aapt 是 Android software development kit(SDK)所提供的眾多開發者工具中的其中一個，aapt 可以幫助開發者新增、更新、以及檢視 ZIP 兼容格式(zip, jar, apk)內的檔案。因為 AndroidManifest 以加密的格式存在於應用程式安裝檔內，我們可以藉由 aapt 獲得 AndroidManifest 的內容以取得應用程式的套件名稱、啟動 activity、以及所宣告的權限；另外我們需要刪除應用程式安裝檔內的簽章，以重新簽署和本系統相同之簽章，也可以藉由 aapt 刪除現有簽章。

3.4.5 jarsigner 和 zipalign

Android 使用簽章作為辨認應用程式開發者的依據，可以依據這個簽章建立相同簽章的應用程式間資料分享關係。因為本系統測試程式與受測應用程式需要相同簽章，於是本系統會需要用到簽署簽章工具，Android 官方推薦使用 jarsigner[22] 。另外官方建議[22] 簽署簽章後的 apk 檔案需要使用 zipalign 進行優化處理，此動作可以有效提升應用程式執行速度，因此對於本系統的編譯測試程式、以及重新簽署後的受測應用程式，本系統皆會用 zipalign 加以優化。

3.4.6 strace

strace 是一個程式檢視以及除錯工具，系統管理員可以藉由此工具來了解沒有程式碼的程式其運行過程，藉由 strace 可以看到程式的 system call 呼叫，以及使用的參數等資訊。本系統藉由此工具可以收集到應用程式使用的 system call 順序，產生 system call n-gram，以作為機器學習的特徵屬性。

3.4.7 Weka

Weka 是一個著名的機器學習套裝軟體，支持多種標準的資料探勘功能，例如：資料前處理、分群、分類、迴歸...等功能。本系統藉由機器學習演算法分辨惡意程式，以取代利用特徵碼比對惡意行為，因此選擇利用 Weka 作為機器學習的分類工具。

第四章 系統設計與實作

4.1 系統架構圖

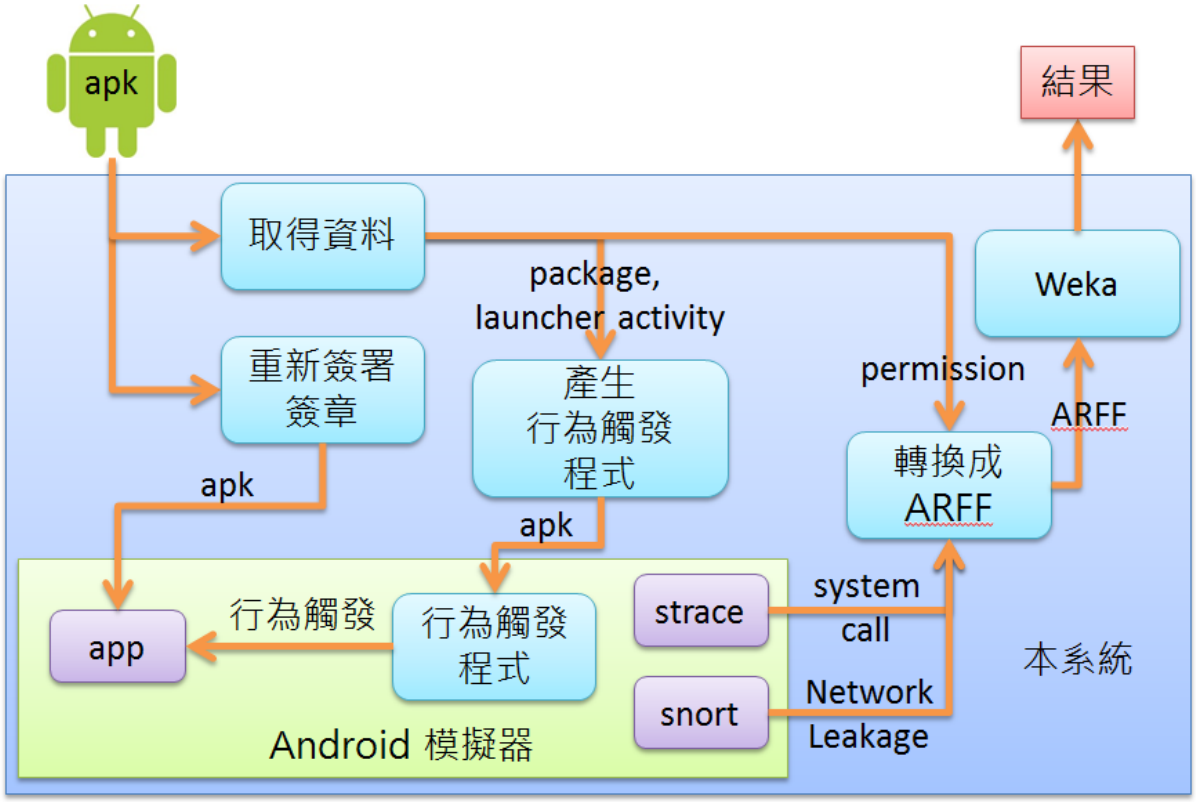


圖 3 系統架構圖

本章節先對本系統執行流程作簡介，詳細作法將會於後續章節解釋。

i. 從受測應用程式安裝檔(apk)取得，受測應用程式宣告之權限(permission)，作為其中一種特徵屬性、以及套件名稱(package name)與啟動組件(launcher activity)，以修改行為觸發程式，令行為觸發程式執行於該應用程式上。

ii. 刪除受測應用程式安裝檔原有簽章，簽署與行為觸發程式相同簽章。

iii. 於行為觸發程式程式碼內加入受測應用程式的套件名稱與啟動組件，並且將行為觸發程式編譯成安裝檔。

iv. 安裝行為觸發程式與受測應用程式於 Android 模擬器上，進行行為觸發，藉由 strace 工具與 snort 工具，紀錄受測應用程式於執行期間產生的 system call 順序與網路封包資料洩漏行為。

v. 將受測應用程式宣告之權限、system call 順序、與網路封包資料洩漏行為等三類特徵屬性，轉換成特徵屬性向量，以讓 Weka 讀入。

vi. Weka 對上述特徵屬性產生分類結果，判斷該受測應用程式是否為惡意程式。

4.2 收集特徵屬性集(Feature sets)

本系統分別從靜態分析，收集到應用程式所宣告的權限；並且從動態分析，收集到應用程式的網路封包資料洩漏行為、以及呼叫 system call 的順序產生之 n-gram，以下將逐一介紹如何收集這三類特徵屬性。

4.2.1 靜態分析(Static Analysis)

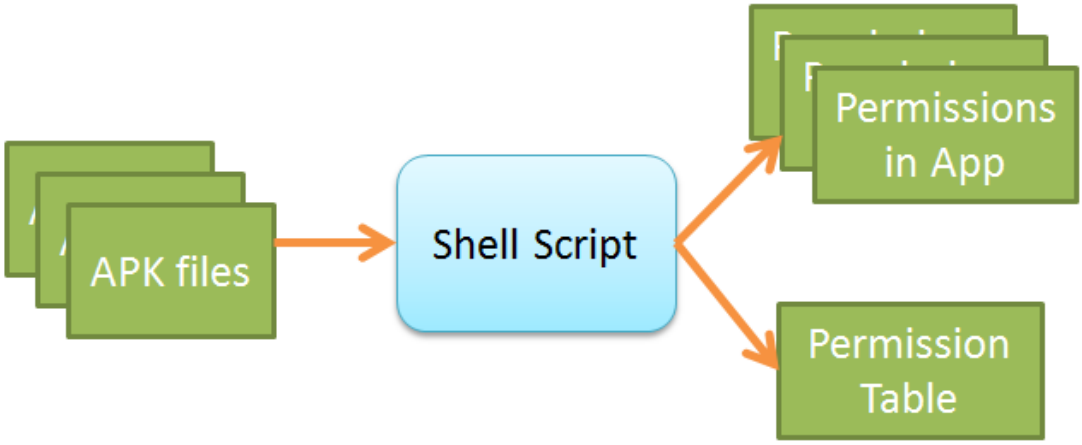


圖 4 靜態分析架構圖

在靜態分析的部分，首先系統藉由 shell script aapt_get_list.sh，逐一對所有應用程式安裝檔取得其宣告之權限，並且產生權限列表，即為在目前所有應用程式樣本中出現過的權限，系統會將這些權限條列在權限列表，以便後續產生 ARFF 檔案，即可以讓 Weka 讀入之輸入檔。

該 shell script 主要使用 aapt 工具的指令，收集應用程式在安裝檔內 AndroidManifest 檔案所宣告的權限，該 shell script 去除應用程式套件名稱後，將應用程式宣告的權限，依照不同的應用程式個別存放，以便後續與網路封包資料洩漏行為、以及 system call n-gram 等特徵屬性做結合。

aapt d[ump] permissions <file.apk>

```
dc@ccisCHTServer:~$ aapt dump permissions DroidDream_sample_1.apk
package: cn.com.opda.android.cleanup1
uses-permission: android.permission.RESTART_PACKAGES
uses-permission: android.permission.KILL_BACKGROUND_PROCESSES
uses-permission: android.permission.RECEIVE_BOOT_COMPLETED
uses-permission: android.permission.INTERNET
uses-permission: android.permission.ACCESS_NETWORK_STATE
uses-permission: android.permission.ACCESS_WIFI_STATE
uses-permission: android.permission.WRITE_EXTERNAL_STORAGE
uses-permission: android.permission.READ_LOGS
uses-permission: android.permission.READ_PHONE_STATE
uses-permission: android.permission.CHANGE_WIFI_STATE
```

圖 5 aapt 取得應用程式安裝檔所宣告之權限

4.2.2 動態分析(Dynamic Analysis)

在動態分析部分，本篇論文會於第五章逐一詳細介紹下列兩大部分：行為觸發程式建置與自動化分析 shell script 建置，本小節僅作先行簡介。

本系統想藉由動態分析，取得應用程式於執行期間的網路封包資料洩漏行為[25]、以及呼叫 system call 的順序產生之 n-gram，為了收集到上述兩類資料，本系統需要一個能夠有效觸發應用程式行為，即能夠確實點擊到應用程式畫面上物件的應用程式行為觸發程式。

先前本實驗室的自動化應用程式行為觸發程式[25] ，是使用 Android 提供的 Monkey 工具來完成這部分功能，問題是 Monkey 僅會隨機產生使用者行為，如：點擊、滑動等行為，於應用程式畫面上的任一點，Monkey 無法確實辨識應用程式畫面上有那些可點選的物件，因此有可能讓 Monkey 執行數分鐘，卻無法點擊到任何按鍵、觸發任何程式行為，並不是一個真正的應用程式行為觸發程式。

為了確實觸發應用程式行為，本系統想藉由 Robotium 提供的函式，開發出能夠確實點擊到應用程式使用者介面的測試程式，以取代先前藉由 Monkey 隨機點選模擬器畫面的行為觸發程式；而且建置行為觸發程式後，希望可以使用 shell script 輔助測試程式以對不同應用程式觸發行為，因此該 shell script 需要做到自動化重新簽署簽章、取得受測應用程式資訊、修改測試程式測試對象、編譯測試程式產生安裝檔並且簽署簽章、最後再執行測試程式，而不只侷限於原始 Robotium 測試程式只可針對單一應用程式執行。為了達到上述行為，本論文撰寫特別的函式，以致由 Robotium 開發出的測試程式，可以針對多種應用程式自動執行行為觸發程式，以及撰寫輔助測試程式，指定受測應用程式資料的 shell script auto_trigger.sh，以完成系統自動化。

當本系統產生靜態分析與動態分析之結果紀錄檔，本系統需要將這些結果紀錄檔轉成特徵屬性，供 Weka 程式讀入。

4.3 產生特徵屬性向量(Feature vector)

本論文於上一個章節詳細解釋，如何分別從靜態分析，收集到應用程式所宣告的權限；並且從動態分析，收集到應用程式的網路封包資料洩漏行為、以及呼叫 system call 的順序，本章節將詳細介紹如何將上述三類特徵屬性轉換成特定格式的向量空間(vector space)，供 Weka 讀入。

Weka 所能夠讀入的是一種名為 ARFF 的檔案格式[28] ，在此作簡單介紹，一開始可以以%為開頭作該行註解，然後依序是@RELATION、@ATTRIBUTE、@DATA 三部分，如圖 6。

```
1  @RELATION network_leak
2
3  @ATTRIBUTE CONTACT_NET NUMERIC
4  @ATTRIBUTE IMEI_NET NUMERIC
5  @ATTRIBUTE IMSI_NET NUMERIC
6  @ATTRIBUTE ISO_NET NUMERIC
7  @ATTRIBUTE PHONE_NUMBER_NET NUMERIC
8  @ATTRIBUTE SIM_NET NUMERIC
9  @ATTRIBUTE SMS_NET NUMERIC
10 @ATTRIBUTE class {MALWARE,NORMAL}
11
12 @DATA
13 0,0,0,9,0,0,0,MALWARE
14 0,0,0,9,0,0,0,MALWARE
15 0,0,0,9,0,0,0,MALWARE
16 0,0,0,9,0,0,0,MALWARE
17 0,0,0,748,0,0,0,NORMAL
18 0,4,0,912,0,0,0,NORMAL
19 0,0,0,1204,0,0,0,NORMAL
```

圖 6 ARFF 格式範例

@RELATION 表示該 ARFF 是關於什麼的向量空間，其格式為

@RELATION <relation-name>

後面的<relation-name>不可以有空白，否則會導致 Weka 讀取錯誤；@ATTRIBUTE 表示該向量空間內某一個項目的詳細特徵屬性名稱，其格式為

@ATTRIBUTE <attribute-name> <datatype>

中間的<attribute-name>即上述的詳細特徵屬性名稱，<datatype>則是表示該特徵屬性名稱的單位，有固定格式，本篇論文的所有特徵屬性皆為次數統計，所以此處標示皆為 NUMERIC，除了最後一項@ATTRIBUTE 表示該向量空間的分類，所以並不是標示

成 NUMERIC，如圖 6 第 3 行到第 10 行，我們可以知道該向量空間總共有 8 種特徵屬性；@DATA 則是為該筆 ARFF 檔案內所有樣本的向量空間，如圖 6 第 13 行到第 19 行，每一行都是代表一個應用程式(樣本)的向量空間，所以該資料集合總共有 7 筆樣本，依照先前第 3 行到第 10 行，我們可以將該向量空間的每一個元素對應到先前的 8 種特徵屬性，表示該應用程式每種特徵屬性的大小，以第 6 筆樣本為例，即圖中第 18 行，其向量空間為 0,4,0,912,0,0,0,NORMAL，再藉由第 3 行到第 10 行的特徵屬性，可以產生表 1 的結果，因此我們可以知道該樣本是一般的應用程式(NORMAL)、有 4 次 IMEI_NET 行為、有 912 次 ISO_NET 行為、於執行期間沒有觀察到其他任何特徵屬性行為。

表 1 特徵屬性表示範例

CONTACT_NET	0
IMEI_NET	4
IMSI_NET	0
ISO_NET	912
PHONE_NUMBER_NET	0
SIM_NET	0
SMS_NET	0
class {MALWARE,NORMAL}	NORMAL

從先前收集到的三類特徵屬性，可產生三類的單一特徵屬性輸入檔，分別是應用程式權限的 ARFF、網路封包資料洩漏行為的 ARFF、system call n-gram n=1~n=4，其中權限宣告只分為有宣告，其值為 0；沒有宣告，其值為 1，雖然有些應用程式會重複宣告相同權限，但並不會產生特別影響，另外 system call 則為了觀察其呼叫順序對於分類的準確率是否有影響，因此本論文產生 n=1 到 n=4 的 ARFF，以進行實驗。本論文也將上述三類兩兩做聯集、以及將三類特徵屬性作聯集，產生其他 ARFF，以觀察那些特徵屬性可以產生較高之準確率。

4.4 Weka 機器學習(Machine Learning)

本論文分別使用不同類型機器學習演算法，以測試不同演算法對於上述特徵屬性資料之效果，總共執行了以下表 2 幾種機器學習演算法。

表 2 本論文使用之機器學習演算法

SVM	SimpleLogistic
SMO	NaiveBayes
J48	Adaboost
REPTree	AdditiveRegression
RandomTree	LogitBoost
RandomForest	MultiBoostAB
KNN	PCA

SVM(Support Vector Machine)：當資料有 n 個特徵屬性，我們將資料散佈於 n 維的空間，SVM 就是要尋找是否存在一個 n-1 維的空間，稱作超平面(hyperplane)，能夠順利將兩類資料給分開，或者先利用核函數(Kernel function)，將資料轉到更高維度的空間，以嘗試找到超平面，分類資料。

SMO (Sequential Minimal Optimization)：SMO 是一種將 SVM 的問題簡單化的演算法，SMO 將原先 SVM 計算超平面之問題，分解成若干個於二維空間可以被解析的子問題，這樣避開了需要解決數值最優化問題的難度。

決策樹(Decision tree)的產生方法其實就是每次選擇一個好的特徵屬性作為分裂點，以逐步將資料作分類。以下 J48、REPTree、RandomTree 都是決策樹的一種，RandomForest 則是 RandomTree 的衍生。

J48：又稱 C4.5，是經由 ID3 修改而來的決策樹(Decision Tree)演算法，ID3 是藉由計算每個特徵屬性的資訊獲利(Information Gain)，來決定從哪一個特徵屬性作為節點下去產生分支；而 J48 則是計算資訊獲利率(Information Gain Ratio)，產生的決策樹。

當系統在選擇特徵的時候 對於所選擇的每一個不同的特徵 會得到一個資訊獲利，資訊獲利越大 則表示使用該特徵作分類後 可以較大量降低資料凌亂程度 也就是說可以將樣本分得比較清楚，於是可以利用該特徵來分類資料，問題是這樣的作法會傾向於選擇擁有許多不同數值的特徵，因此要先將資訊獲利做正規化，獲得資訊獲利率，才可以解決這種問題。

產生決策樹的過程中，可能於訓練資料階段產生之節點過多，雖然該階段顯示之錯誤較小，但這樣節點過多的決策樹會於測試資料階段，觀察到變大的錯誤，即過度學習之情況(overfitting)，因此可以利用適度的剪枝以緩解過度學習之問題。

REPTree：REPTree 可以說是 ID3 之變形，REPTree 與 ID3 同樣使用資訊獲利決定樹節點，後來再利用 Reduced Error Pruning 進行剪枝的動作。REP 即為 Reduced Error Pruning 的簡稱，其做法為：對於該決策樹的每棵非葉節點的子樹，使用葉節點代替該子樹，如果該子樹被葉節點替代後形成的新樹之誤差等於或小於原先決策樹之誤差，則用該葉節點替代子樹。

RandomTree：RandomTree 顧名思義，該決策樹是一個於產生節點階段，隨機挑選 K 個特徵屬性以決定該節點的決策樹，並且最後產生之決策樹沒有進行剪枝的動作。

RandomForest：RandomForest 則是藉由，許多的 RandomTree 所產生的決策樹群，執行 RandomForest 最後產生的分類結果則是藉由這些決策樹群產生的結果進行多數決。

KNN(K-Nearest Neighbors)：將未知類別的資料點放置於已知資料集合的空間中，計算其周圍距離最近的 K 個鄰居，依照 K 個鄰居的類別作多數決投票(majority vote)，藉此多數決的結果決定未知類別資料點的類別。

SimpleLogistic：是簡單線性回歸(simple linear regression)與 LogitBoost 的組合。所謂的回歸簡單來說就是在一個資料集合中，我們能夠找到一條曲線去嘗試擬和這些資料集合，如果該曲線是一條直線，則被稱作線性回歸；如果該曲線是一條二次曲線，就被稱為二次回歸。

NaiveBayes：計算某一個樣本 x 對於不同類別的條件機率，因為 NaiveBayes 假設各個特徵屬性之間沒有關聯，是獨立的，所以我們可以將該樣本 x 對於類別 K 的條件機率問題，轉變成計算已知類別 K 對於該樣本 x 的條件機率，即已知類別 K 對於所有特徵屬性的條件機率。最後將這些條件機率相乘，即可得到該樣本 x 對類別 K 的條件機率，其中最大者，樣本 x 即屬於該類別。

Boosting 是一種迭代演算法，其核心思想是針對同一個訓練集合訓練產生不同的分類器(弱分類器)，然後把這些弱分類器集合起來，構成一個更強的最終分類器(強分類器)，其演算法本身是通過改變數據分佈來實作的，它根據每次訓練集之中每個樣本的分類是否正確，以及上次的總體分類的準確率，來確定每個樣本的權值，再將修改過權值的新數據集合送給下層分類器進行訓練，最後將每次訓練得到的分類器最後融合起來，作為最後的決策分類器。使用 Adaboost 分類器可以排除一些不必要的訓練數據，並將重點放在關鍵的訓練樣本上面。

AdditiveRegression：即為 Weka 內實作之 Gradient Boosting 演算法，與 Adaboost 不同的在於 Gradient Boosting 藉由對每次產生模型之損失函數(loss function)進行微分，以得到其梯度下降(gradient descent)方向，以產生下一個分類模型。

LogitBoost：將 Adaboost 想成是 generalized additive model，然後將其損失函數改成 logistic regression 則可產生 LogitBoost。

MultiBoostAB：可以把 MultiBoost 想成是 Adaboost 和 wagging 之組合，

MultiBoost 同時擁有兩者的特性，即擁有 Adaboost 的高 bias 以及使用 wagging 的 superior variance reduction 以實作 variance reduction。

PCA(Principal Component Analysis)：主成分分析藉由計算特徵屬性的共變異數矩陣(covariance matrix)，然後對共變異數矩陣進行特徵分解，以得出特徵屬性的主成分，即特徵向量(eigenvector)，與它們的權值，即特徵值(eigenvalue)，最後再藉由特徵值的排序，取得較重要之特徵屬性，以降低輸入資料集合的特徵屬性數量，以提高後續執行機器學習演算法的速度。在此會提到此演算法，是因為本論文偶然發現，將執行 PCA 的結果交由 SVM，可提高 SVM 的準確率，詳細內容會在下一章作更多討論。

第五章 行為觸發程式與輔助 shell script

以下將逐一介紹本論文如何利用 Robotium 的函式，作為應用程式行為觸發程式，本系統的行為觸發程式與一般利用 Robotium 所開發的應用程式有何不同；以及 shell script 如何輔助上述行為觸發程式，以幫助行為觸發程式處理所有應用程式樣本。

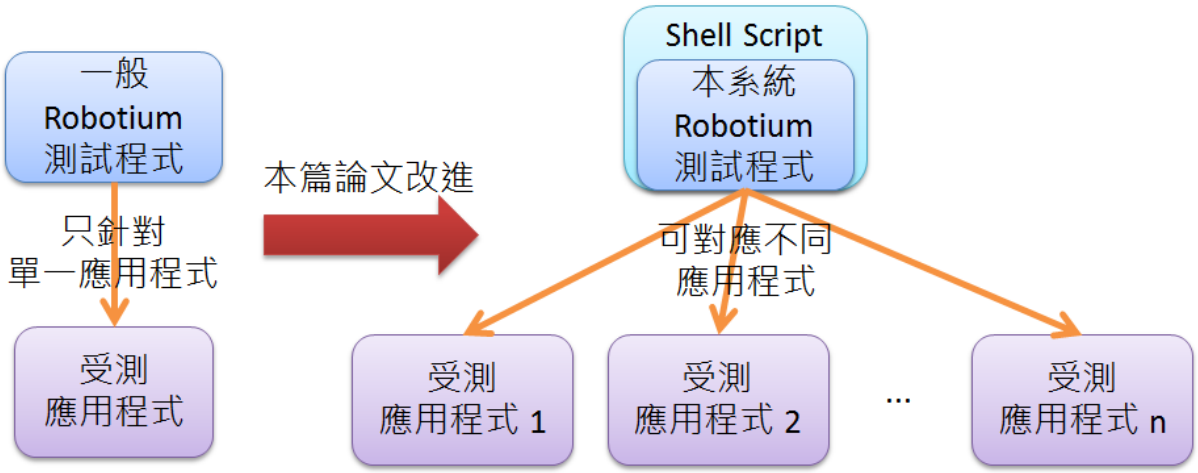


圖 7 行為觸發程式示意圖

5.1 行為觸發程式建置

5.1.1 行為觸發程式基本設定

詳細 Robotium 測試程式撰寫，在官方網站有提供 Robotium 程式的範例[27] ，可以至官方網站實際下載並執行。

一開始開發者必須在測試程式專案資料夾下 AndroidManifest.xml 檔案內指定受測應用程式的套件名稱(package name)，以指定受測程式，如圖 8 內第 11 行，該測試程式指定 com.example.android.notepad 為其受測程式。

```
 8 <uses-library android:name="android.test.runner" />
 9 </application>
10 <uses-sdk android:minSdkVersion="10" />
11 <instrumentation android:targetPackage="com.example.android.notepad"
```

圖 8 指定受測應用程式套件名稱

藉由 Robotium 函式庫所撰寫而成的測試程式，與一般 java 或者 Android 程式有些許不同，並沒有如同 java 程式的 main()或者 Android 的 onCreate()作

為程式進入點，其測試程式需要一個主要的 java 程式碼，以執行測式行為。該程式碼可分為四個部分。首先需要該 Class 的建構子，並且繼承受測應用程式的啟動組件的 Class，如圖 9 內第 33 行的 RobotiumTest 函式繼承常數 LAUNCHER_ACTIVITY_FULL_CLASSNAME 的 Class；在 Robotium 的測試程式中，主要的測試行為是藉由一個名為 Solo 的 Class 下去執行，因此第二部分即為創建 Solo 物件，如圖 9 內第 37 行的 SetUp()函式；第三部分則為實際執行測試行為的函式，此部分並沒有限制可以撰寫幾個測試函式，只是需要函式名稱開頭皆為 test，如圖 9 內第 51 行的 testAddNote()函式，開發者可以撰寫多個 test 函式，其執行順序即為程式碼內撰寫順序，如圖 10 測試程式會依序執行 testAddNote()、再執行 testEditNote()、最後再執行 testRemoveNote()；第四部份則為 Solo 物件的結束，如圖 9 內的第 44 行 tearDown()，該函式內的 solo.finishOpenedActivities();可以將測試程式執行期間所啟動的受測應用程式組件，悉數結束，以順利執行下次測試。

```
31 @SuppressWarnings("unchecked")
32 public RobotiumTest() throws ClassNotFoundException {
33     super(Class.forName(LAUNCHER_ACTIVITY_FULL_CLASSNAME));
34 }
35
36 @Override
37 public void setUp() throws Exception {
38     //setUp() is run before a test case is started.
39     //This is where the solo object is created.
40     solo = new Solo(getInstrumentation(), getActivity());
41 }
42
43 @Override
44 public void tearDown() throws Exception {
45     //tearDown() is run after a test case has finished.
46     //finishOpenedActivities() will finish all the activities
47     //that have been opened during the test execution.
48     solo.finishOpenedActivities();
49 }
50
51 public void testAddNote() throws Exception {
```

圖 9 Robotium 測試程式主要四個部分

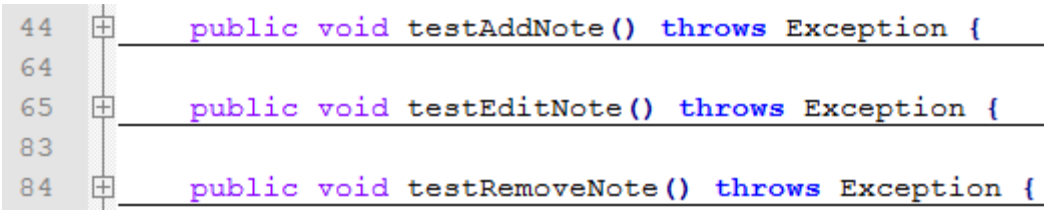


圖 10 測試程式的執行順序為由前到後

在撰寫 Robotium 主要程式碼過程中，上述四個部分缺一不可，需要第一部分以指定受測應用程式的啟動組件；需要第二與第四部分的函式覆寫父類別函式，以創建與關閉 Solo 物件以及產生的組件；需要第三部分以實際執行測試行為。另外，因為使用到 Robotium 提供的函式庫，因此需要掛載 Robotium 的 jar 檔案於程式專案上。

本系統為了執行行為觸發程式於受測程式上，於每次執行行為觸發程式前，必須修改行為觸發程式專案內兩個部分，即為上述的指定受測應用程式套件名稱、與指定受測應用程式啟動組件名稱。首先需要在專案資料夾下 AndroidManifest.xml 檔案內指定受測應用程式的套件名稱(package name)，如圖 8 內的第 11 行。

```
<instrumentation android:targetPackage="受測應用程式套件名稱"
```

接下來則需要在主要的 java 程式碼內，指定受測應用程式啟動組件(launcher activity)，如圖 9 內的第 33 行 RobotiumTest 函式繼承常數 LAUNCHER_ACTIVITY_FULL_CLASSNAME 的 Class，以執行常數 LAUNCHER_ACTIVITY_FULL_CLASSNAME 該啟動組件。

```
super(Class.forName("受測應用程式啟動組件名稱"));
```

5.1.2 行為觸發程式流程

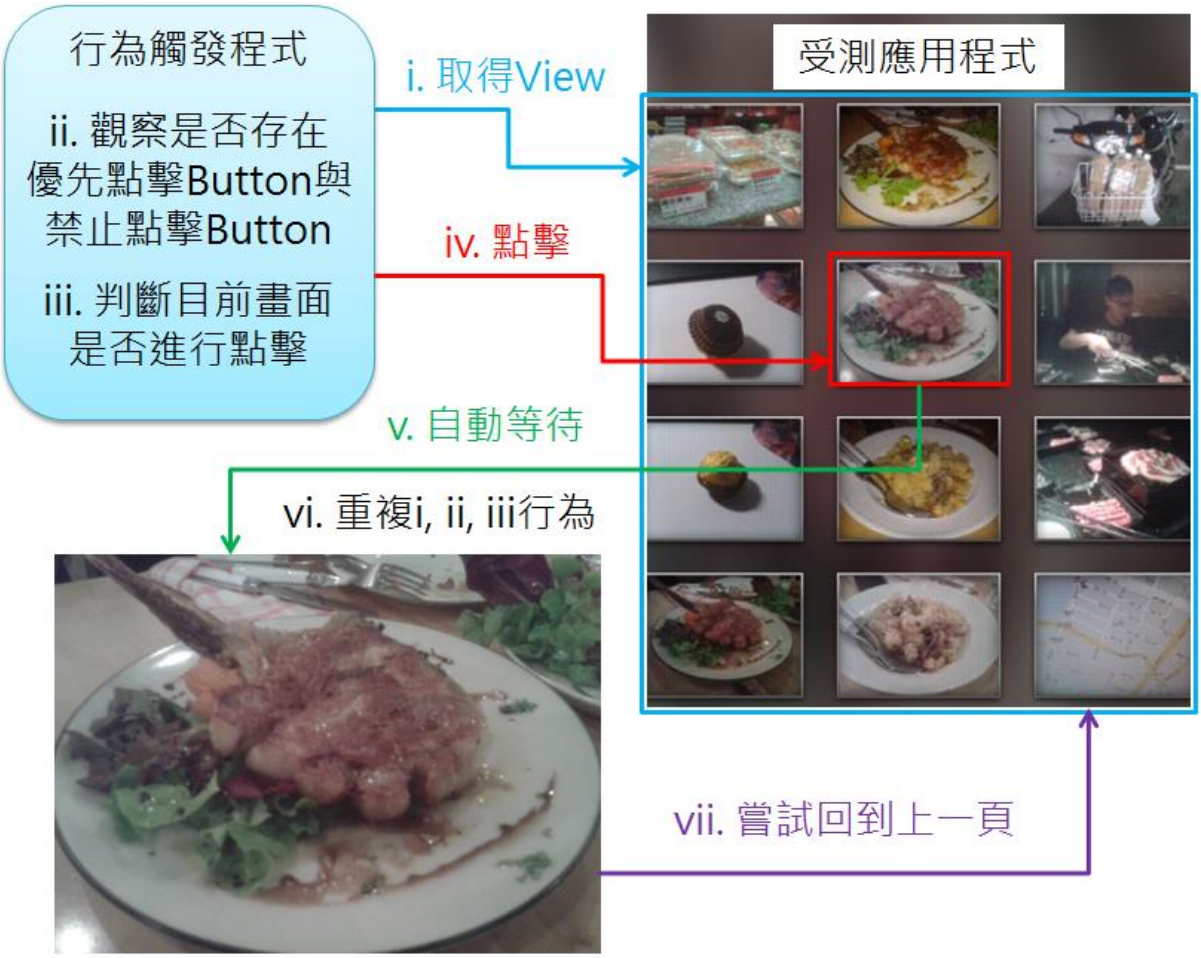


圖 11 行為觸發程式流程圖

本小節先對本行為觸發程式流程作簡介，詳細作法將會於後續小節解釋。

i. 從受測應用程式目前畫面，取得畫面上所有的 View 物件。
ii. 觀察是否現在畫面存在可以優先點擊之 Button 物件，與禁止點擊之 Button 物件。
iii. 判斷是否於現在畫面執行點擊。
iv. 對於特定 View 物件執行點擊動作。
v. 等待程式是否完成資料下載動作。
vi. 重複前述 i, ii, iii 行為。
vii. 判斷沒有可點擊 View 物件，並回到上一頁。

5.1.3 取得 View 物件與點擊 View

本行為觸發程式主要藉由取得畫面中可以點選的物件，逐一點擊這些物件以模擬使用者使用應用程式的行為，進而觸發應用程式執行 system call 或者傳輸網路封包的行為。在 Android 裡，View 對於使用者介面而言是一個最基本的部件，許多可以點選的物件，例如：ImageView、ListView、GridView、Button、EditText…等物件，皆為 View 的子類別，換句話說，該行為觸發程式無須對每種可點擊的物件都撰寫函式，分門別類處理；本行為觸發程式只要針對目前畫面上可點擊的 View，逐一作點擊即可。因此本部分主要利用到 Robotium 提供的函式為 solo.getCurrentViews()以及 RobotiumUtils.removeInvisibleViews()。

Android 程式可以在同一個 Activity 上放置許多物件，視現在程式使用程度或者使用情況，可以顯示或者隱藏物件，如圖 12 與圖 13 皆為同一個程式的 Activity，卻可以於不同狀態分別顯示以 TextView 所構成的影片資訊；以及以 ListView 所構成的其他使用者留言，其實該程式同時在右下角紅框部分放置上許多 TextView 以及 ListView，該程式則視情況顯示 TextView 隱藏 ListView；反之則顯示 ListView 隱藏 TextView，程式無須作整個 Activity 畫面的切換，額外浪費系統資源。

圖 12 YouTube 影片撥放程式，顯示影片資訊畫面

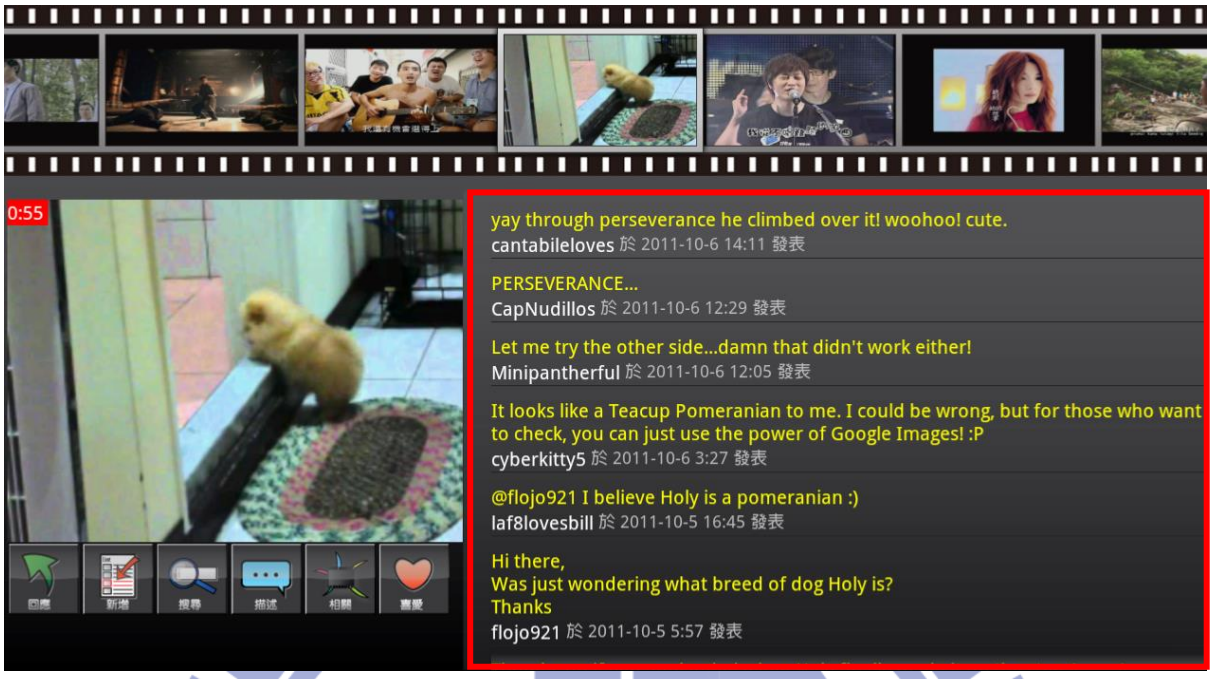


圖 13 YouTube 影片撥放程式，顯示其他使用者留言畫面

使用 Robotium 提供的 solo.getCurrentViews()函式會回傳目前使用者介面上的所有 View，無論隱藏或非隱藏，然後再利用 solo.clickOnView(view)對特定的 view 作點擊的動作。問題是 solo.clickOnView(view)與使用者相同，無法點擊隱藏的 View，因此當傳給 solo.clickOnView(view)的 view 是隱藏的，程是會出現問題。此處即可利用 RobotiumUtils.removeInvisibleViews()，回傳所有非隱藏的 View，因此此處程式碼可以如此撰寫

```
List<View> viewList = RobotiumUtils.removeInvisibleViews(solo.getCurrentViews());
```

即可得到目前畫面上可見的 View List，且每一個 View 都會設定是否可以點擊，因此再撰寫簡單的判斷函式即可取得目前畫面上所有可見可點擊的 View List 以作更進一步的點擊或處理。

5.1.4 優先點擊與禁止點擊 Button

當使用者初次使用程式時，有些應用程式會顯示使用者條款請使用者閱讀並且點擊接受，使用者需要點擊接受方能繼續使用程式，當使用者點擊拒絕則往往程式會直接結束回到 Android 系統畫面；抑或有些應用程式會要求使用者至 Google Play 等應用程式發布平台，對該應用程式進行評論，或者有些免費程式會詢問使用者是否有意願進行捐款，如圖 14，以資助開發者開發更加便利的應用程式。問題是當使用者點擊評論或者捐款按鍵，往往會離開應用程式啟動瀏覽器或者 Google Play 應用程式，使用者需要回到 Android 系統，重新點擊應用程式，方能繼續使用應用程式。如同上述這些情況，本行為觸發程式

於執行期間有可能會點擊到這些按鍵，以致提早結束行為觸發過程，因此本行為觸發程式需要撰寫特殊的函式，以處理這些情形，確保行為觸發程式不會提早結束，因此特地撰寫 forbidClickableVutton()判斷畫面上是否有特定按鍵，會造成上述離開應用程式的情況，並且忽略這些按鍵，選擇點擊其他按鍵；以及 findPreviousClickButton()則是判斷畫面上是否有點擊後應該不會提早結束行為觸發的按鍵，並且進行點擊。

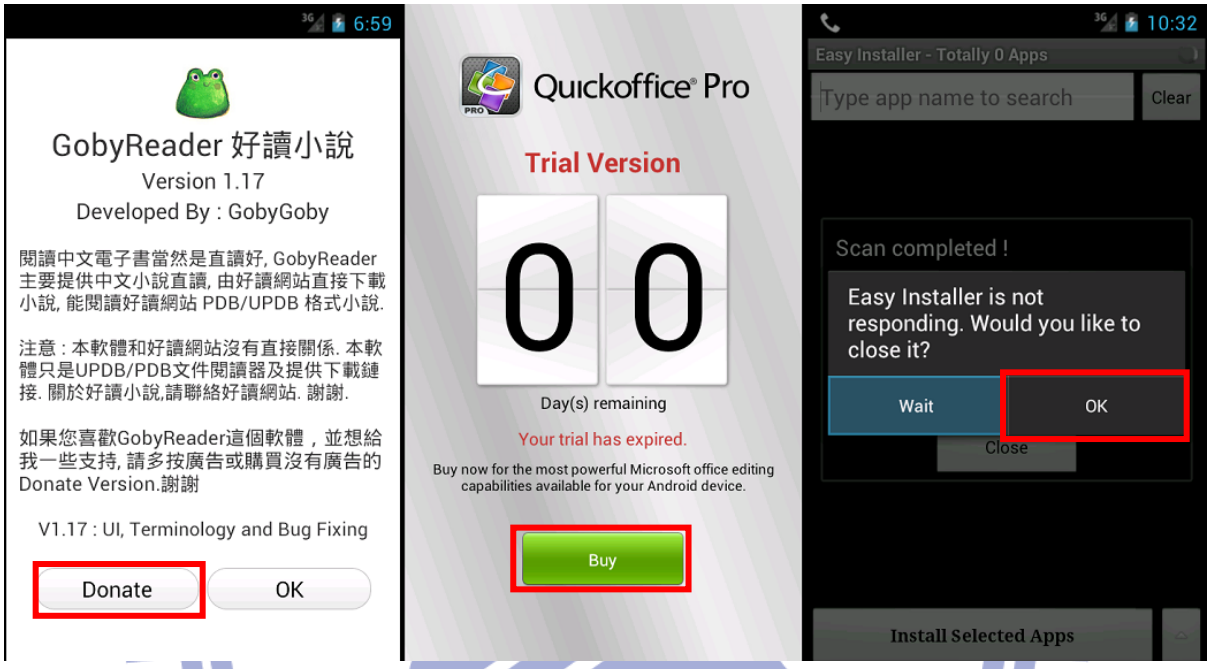


圖 14 會提早結束或離開應用程式的按鍵範例

5.1.5 畫面判斷

本行為觸發程式於執行過程，當遇到先前見過的畫面，則不會再次點擊先前點擊過的 View，並且會點擊剩餘未點擊過的 View，以觸發更多應用程式行為。問題是，如前所述，相同 Activity 於不同狀態下，有不同可見可點擊的 View，因此本行為觸發程式無法單純以先前是否見過該 Activity 作為標準，判斷是否已經點擊過該畫面上 View，需要其他判斷方法。因此本行為觸發程式，取得所有畫面上的 View，並且逐一比對其物件名稱、與物件位置，作為判斷是否見過目前畫面之基準。

5.1.6 自動等待

一般應用程式於讀取或下載資料時，為了讓使用者知道該應用程式處於讀取資料中的狀態，並非程式當掉的狀態，都會有提示字元或者提示畫面告訴使用者，如圖 15。

圖 15 下載或者讀取資料提示字元與畫面

使用 Robotium 函式庫建置的測試程式，其原本的作用是幫助開發者檢查自己開發的程式功能是否正確、是否會產生不如預期的錯誤，因此當該測試程式期間產生不如預期的結果，測試程式會立即中止並且顯示執行錯誤資訊。然而本行為觸發程式，無法預知受測應用程式可能的執行過程，當遇到受測應用程式為讀取資料的狀態，本行為觸發程式必定不可以繼續執行、或者隨便點擊按鍵，以免提早結束行為觸發，因此本觸發程式需要可以自動判斷程式是否處於資料讀取狀態的判斷函式。

於是本論文撰寫一個自動等待程式資料讀取完成的函式 autoWait()，當行為觸發器點擊一個 View 後，立即執行 autoWait()，autoWait()會每隔一段特定時間就判斷目前畫面上是否有可以點擊的 View，而且因為有可能雖然已經出現可點擊的 View，但是程式尚未載入完成，因此 autoWait()會在等一個特定時間，並且判斷前一個畫面與現在畫面上所含的 View 是否相同，如果相同則可繼續執行行為觸發，若不同則代表程式尚未載入完成，繼續作等待，直到等待到特定時間上限，autoWait()才會結束等待。

5.1.7 回到上一頁

有些應用程式，例如：圖片瀏覽器，如圖 16，當使用者點擊圖片集內特定圖片作觀看，觀看後使用者必須點擊上一頁按鍵回到圖片集畫面，否則無法繼續作圖片瀏覽的行為。本行為觸發器於執行期間也有可能造成這種情況，因此也有針對這種情況執行點擊上一頁的動作，即呼叫 solo.goBack()函式。

圖 16 圖片瀏覽器使用流程

當本行為觸發程式判斷該畫面上可點擊的 View 已經全數點擊完畢，本行為觸發程會任意點擊可點擊的 View，以嘗試進入其他畫面，尋找其他尚未點擊的 View。原先遭遇這種情況，打算呼叫 solo.goBack()函式以回到上一頁，問題是此時該應用程式可能已經在主頁面，換句話說，當使用者在主頁面點擊回到上一頁的按鍵，表示使用者想離開該應用程式。因此本行為觸發程式為了避免提早結束行為觸發過程，本行為觸發器不會在這種情況呼叫 solo.goBack()函式，而是以任意點擊畫面上可點擊之 View，期望可以進入其他尚未完成點擊的畫面。

5.1.8 結束行為觸發程式

當行為觸發程式開始執行，就需要執行至 testFuction 結束，行為觸發程式才會結束，問題是有些應用程式有無止盡的畫面可點選，例如當使用者使用 Facebook 的應用程式，該程式會一次下載固定數量貼文給使用者瀏覽，當使用者瀏覽完目前顯示的所有貼文，程式才會再下載其他貼文。當使用者換成本

行為觸發程式去執行該應用程式，本行為觸發程式遭遇這種應用程式將無法結束行為觸發。因此原先的想法是計算所點選的 View 次數，即點選一定 threshold 次數後，程式自動結束，問題是該作法無法保證執行時間長短，於是最後修改成在啟動行為觸發函式時，產生一個 thread 並且 sleep 一段時間，當該 thread sleep 完則呼叫 System.exit(0)，即可結束行為觸發程式，確保該行為觸發程式可於特定時間內結束行為觸發過程。

5.2 自動化輔助 shell script 建置

為了對受測應用程式使用 Robotium 開發的行為觸發程式，受測應用程式需要和行為觸發程式擁有相同簽章，因此系統需要先刪除受測應用程式安裝檔內原有簽章，接著簽署與行為觸發程式相同之簽章，最後再對受測應用程式安裝檔進行優化處理。

系統首先使用 aapt 刪除受測應用程式安裝檔內的簽章，即刪除 META-INF 資料夾下的所有檔案，於是系統條列 META-INF 資料夾內的檔案，每個應用程式其 META-INF 資料夾內的檔案不一定會擁有相同名稱，因此需要列出 META-INF 資料夾內的檔案名稱，如圖 17，再逐一作刪除，如圖 18。

aapt l[ist] <file.apk> | grep ^META-INF/

```
dc@ccisCHTServer:~$ aapt l DroidDream_sample_1.apk | grep ^META-INF/
META-INF/MANIFEST.MF
META-INF/CERT19.SF
META-INF/CERT19.RSA
```

圖 17 aapt 條列應用程式安裝檔內 META-INF 資料夾內所有簽章

aapt r[emove] <file.apk> <signature_file>

```
dc@ccisCHTServer:~$ aapt r DroidDream_sample_1.apk META-INF/MANIFEST.MF
dc@ccisCHTServer:~$ aapt r DroidDream_sample_1.apk META-INF/CERT19.SF
dc@ccisCHTServer:~$ aapt r DroidDream_sample_1.apk META-INF/CERT19.RSA
dc@ccisCHTServer:~$ aapt l DroidDream_sample_1.apk | grep ^META-INF/
dc@ccisCHTServer:~$
```

圖 18 aapt 逐一刪除應用程式安裝檔內 META-INF 資料夾內所有簽章

接著利用 jarsigner 工具，對受測應用程式安裝檔簽署與行為觸發程式相同之簽章，最後再使用 zipalign 工具對安裝檔進行優化，詳細指令如下。

jarsigner -keystore <key.keystore> -storepass <password> -keypass <password> -sigalg MD5withRSA -digestalg SHA1 <file.apk> <alias>

zipalign -v 4 <file.apk> < destination.apk>

修改受測應用程式安裝檔後，系統必須在行為觸發程式的 AndroidManifest.xml 檔案內 targetProject 項目上，指定受測應用程式之套件名稱(package name)、以及在行為觸發程式的主要 java 程式碼內指定受測應用程

式的啟動組件(launcher activity)，行為觸發程式才可以知道哪一個是受測應用程式、以及從哪一個組件啟動。系統同樣可以藉由 aapt 取得受測應用程式的套件名稱及啟動組件名稱，如圖 19 的紅色方框內為該應用程式的套件名稱、藍色方框內則為啟動組件名稱。

aapt d[ump] badging <file.apk>

```
dc@ccisCHTServer:~$ aapt dump badging DroidDream_sample_1.apk
package: name='cn.com.opda.android.cleanup1' versionCode='1' versionName='1.0'
application-label:'开机加速'
application-icon-120:'res/drawable/speed.png'
application-icon-160:'res/drawable/speed.png'
application-icon-240:'res/drawable/speed.png'
application: label='开机加速' icon='res/drawable/speed.png'
application-debuggable
launchable-activity: name='com.android.root.main'  label='' icon=''
sdkVersion:'4'
uses-permission:'android.permission.RESTART_PACKAGES'
```

圖 19 aapt 取得應用程式安裝檔內套件名稱及啟動組件名稱

系統修改行為觸發程式其程式碼後，需要重新編譯程式碼、產生行為觸發程式的 apk 檔案、並且簽署簽章及優化 apk 檔案，執行完上述動作後方可產生完整的行為觸發程式 apk 檔案，以執行後續測試。首先系統需要更新行為觸發程式的專案，產生 Build.xml 檔案，最後再使用 ANT 依據 Build.xml 檔案幫助系統編譯程式碼產生 apk 檔案。

android update project -p <trigger_project>

ant clean release

因為行為觸發程式使用到 Robotium 的函式庫，因此需要將 Robotium 的 jar 檔案放置於行為觸發程式專案下的 libs 資料夾內，方可編譯成功。

系統將應用程式與行為觸發程式安裝於本實驗室先前修改的 Android 模擬器上，以記錄應用程式的網路傳輸行為，關於此模擬器詳細開發過程請閱讀此論文[25] ，安裝指令如下

adb install <tested_app.apk>

adb install <trigger_app.apk>

接著執行以下指令以執行行為觸程式，行為觸發程式會啟動受測應用程式，並且執行先前撰寫之行為觸發動作

adb shell am instrument -w <trigger_package>/android.test.InstrumentationTestRunner

開始觸發受測應用程式後，取得受測應用程式的 pid，並且利用 strace 記錄受測應用程式執行期間所呼叫的 system call 順序

adb shell strace -r -f -F -p <pid_of_tested_app >

執行行為觸發後，執行期間行為觸發程式可能會點擊外部連結，啟動瀏覽器；或者觸發受測應用程式下載應用程式更新檔、甚至下載其他應用程式，因此本系統測試完成後、不僅會移除受測應用程式與測試程式、也會強制停止瀏覽器、以及強制停止下載程式，因此需要用到以下指令。

```
adb uninstall <tested_package>
adb uninstall <trigger_package>
adb shell am force-stop com.android.providers.downloads
adb shell am force-stop com.android.browser
```

重複以上動作，即可將訓練資料集合(training dataset)內的受測應用程式，全數執行完畢，產生本系統所需的各種特徵屬性的紀錄檔。

第六章 實驗與討論

6.1 樣本集合

非常感謝 Yajin Zhou 與 Xuxian Jiang 提供大量的惡意程式安裝檔[7] ，以及陶嘉仁[29] 提供大量一般應用程式安裝檔，以幫助本實驗順利進行。

一般應用程式是根據 Google Play 對於應用程式的分類，總共分為 18 類，詳細如表 3，本論文使用之一般應用程式集合，是從各類免費熱門下載排行榜下載而來。

表 3 一般應用程式樣本各分類數量表

類別名稱	應用程式樣本數量
Books	17
Business	44
Communication	70
Education	26
Entertainment	50
Finance	50
Game	46
Media	39
Music	54
News	27
Personalization	44
Photography	39
Productivity	78
Shopping	35
Social	48
Tool	41
Transportation	66
Travel	51
合計	825

惡意程式則是根據攻擊方法與攻擊目的不同，總共分為 49 個家族，詳細如表 4，每一個家族的名稱主要是以該家族中首先被發現的惡意程式名稱以此命名。

表 4 惡意程式樣本各家族數量表

家族名稱	惡意程式樣本數量
ADRD	22
AnserverBot	187
Asroot	8
BaseBridge	122
BeanBot	8
Bgserv	9
CoinPirate	1
CruseWin	2
DogWars	1
DroidCoupon	1
DroidDeluxe	1
DroidDream	16
DroidDreamLight	46
DroidKungFu1	34
DroidKungFu2	30
DroidKungFu3	309
DroidKungFu4	96
DroidKungFuSapp	3
DroidKungFuUpdate	1
EndofDay	1
FakeNetflix	1
FakePlayer	6
GamblerSMS	1
Geinimi	69
GGTracker	1
GingerMaster	4
GoldDream	47
Gone60	9
GPSSMSSpy	6
HippoSMS	4
Jifake	1
jSMSHider	16

Kmin	52
LoveTrap	1
NickyBot	1
NickySpy	2
Pjapps	58
Plankton	11
RogueLemon	2
RogueSPPush	9
SMSReplicator	1
SndApps	10
Spitmo	1
Tapsnake	2
Walkinwat	1
YZHC	22
zHash	11
Zitmo	1
Zsone	12
合計	**1260**

本實驗主要是以監督式學習(Supervised Machine Learning)的方式進行，即演算法可以於訓練資料集合中得知該樣本的類別；並且利用 10 折交叉驗證(10-fold cross-validation)的方式，將全部資料集合分成 10 等分，每次取其中 1 份作為測試資料集合(testing set)，其他 9 份作為訓練資料集(training set)，以交互作測試資料與訓練資料。

6.2 分類結果定義

首先說明本論文機器學習結果的評估方式。本論文對於分類結果的定義，如下圖 20，本論文將原本為惡意程式而系統分成惡意程式的結果稱作真陽性(True Positive)，簡稱 TP；將原本為一般程式而系統分成一般程式的結果稱作真陰性(True Negative)，簡稱 TN；將原本為惡意程式而系統分成一般程式的結果稱作偽陰性(False Negative)，簡稱 FN，因為應該將該樣本標記成惡意程式，卻遺漏了該樣本，因此又稱漏報；將原本為一般程式而系統分成惡意程式的結果稱作陽性(False Positive)，簡稱 FP，因為應該將該樣本標記成一般程式，卻分類錯誤了該樣本，因此又稱誤報。

分類結果	實際類別：惡意程式	實際類別：一般程式
惡意程式	True Positive (TP)	False Positive (FP)
一般程式	False Negative (FN)	True Negative (TN)

圖 20 分類結果名詞定義

我們可以藉由上述四個值 TP、FP、FN、TN，可以計算得出以下三個數值。

$$\text{真陽性率(True Positive Rate)：} TPR = \frac{TP}{TP+FN} \quad (1)$$

$$\text{偽陽性率(False Positive Rate)：} FPR = \frac{FP}{FP+TN} \quad (2)$$

$$\text{準確率(Accuracy)：} Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \quad (3)$$

TPR 即為所有惡意程式樣本正確地分成惡意程式的比例；FPR 則為所有一般程式樣本分成惡意程式的比例；Accuracy 則是所有樣本，正確被分成惡意程式或者一般程式的比例。因此我們知道，當 TPR 與 Accuracy 越高，表示該特徵屬性與該演算法較能準確分類樣本；而當 FPR 越高，則表示該特徵屬性與該演算法較不能準確分類樣本，因此我們期望看到 TPR 和 Accuracy 較高、與 FPR 較低。

另外本論文執行不同機器學習演算法、與該演算法所搭配之參數，如表 5，為了簡化後續圖表，因此後續圖表將以大寫英文字母代號代替該演算法與參數。

表 5 各大寫英文字母代號對應之演算法及其參數

(A)	SVM	(N)	KNN
(B)	SMO Poly	(O)	Adaboost(KNN)
(C)	SMO NPoly	(P)	GB(KNN)
(D)	REPTree	(Q)	LogitBoost(KNN)
(E)	Adaboost(REPTree)	(R)	MultiBoostAB(KNN)
(F)	GB(REPTree)	(S)	KNN 3
(G)	LogitBoost(REPTree)	(T)	KNN 5
(H)	MultiBoostAB(REPTree)	(U)	KNN 10
(I)	J48	(V)	SimpleLogistic
(J)	RandomTree	(W)	NaiveBayes
(K)	RandomForest 10	(X)	Adaboost(NaiveBayes)
(L)	RandomForest 50		
(M)	RandomForest 100		

6.3 Monkey 與本行為觸發程式比較

首先，本論文使用 Robotium 函式庫所提供的功能，開發可以辨識應用程式畫面上物件的行為觸發程式，以更準確地觸發應用程式行為，取代先前 Monkey 作為行為觸發程式的系統。為了觀察使用本行為觸發程式，相對於使用 Monkey，是否能確實提升機器學習準確率，本章節分別以 Monkey 與本行為觸發器，去執行先前的 2085 個樣本，觀察其藉由這兩種程式所產生之機器學習結果，是否有任何差別。

以下是分別以不同特徵屬性集合，執行不同機器學習所產生的結果。本章節將原本的三類特徵屬性作不同的聯集，產生不同的特徵屬性集，即權限集合(簡稱 pm)、網路行為統計(簡稱 nw)、system call 使用次數統計(簡稱 sys)；以及上述三類特徵屬性集合所產生的兩兩聯集，即網路行為與權限的聯集、網路行為與 system call 統計的聯集、system call 與權限的聯集、和全部的特徵屬性所產生的聯集，本論文將依序呈現以上特徵屬性集合所產生之機器學習結果。

為了幫助解讀機器學習產生之數據結果，本論文會於各表中作藍色文字之特別標記，即為每一行結果之數值前五高者，以利本論文判斷哪些演算法較適合那些特徵屬性，例如以 Accuracy 部分，表 6 所有演算法分類結果 Accuracy 部分由大到小依序為 95.40% (RandomForest 50)、95.38% (RandomForest 100)、95.01% (SMO NPoly)、94.44% (RandomForest 10)、94.34% (Adaboost(REPTree))、94.10% (SMO Poly)、94.00% (LogitBoost(KNN))、…、82.97% (NaiveBayes)，因此將 RandomForest 50、RandomForest 100、SMO NPoly、RandomForest 10、Adaboost(REPTree)這五個 Accuracy 資料標記成藍字。

使用不同行為觸發器，只會影響動態分析之結果，不會影響靜態分析之結果。因此僅以宣告權限作為特徵屬性之結果，於不同的行為觸發器，並不會產生不同之結果。因此本論文於宣告權限作為特徵屬性之結果，不作 Monkey 與 Robotium 之比較，其產生的機器學習結果如表 6。

表 6 以宣告權限作為特徵屬性之結果

Algorithm	TPR	FPR	Accuracy
SVM	0.887	0.091	89.54%
SMO Poly	0.937	0.052	94.10%
SMO NPoly	0.941	0.036	95.01%
REPTree	0.928	0.095	91.89%
Adaboost(REPTree)	0.947	0.062	94.34%
GB(REPTree)	0.932	0.091	92.28%
LogitBoost(REPTree)	0.943	0.067	93.91%
MultiBoostAB(REPTree)	0.934	0.069	93.29%
J48	0.936	0.079	93.00%
RandomTree	0.933	0.093	92.23%
RandomForest 10	0.954	0.07	94.44%
RandomForest 50	0.954	0.046	95.40%
RandomForest 100	0.952	0.044	95.38%
KNN 1	0.954	0.085	93.86%
Adaboost(KNN)	0.952	0.084	93.81%
GB(KNN)	0.954	0.085	93.86%
LogitBoost(KNN)	0.954	0.081	94.00%
MultiBoostAB(KNN)	0.952	0.091	93.53%
KNN 3	0.952	0.095	93.33%
KNN 5	0.948	0.092	93.19%
KNN 10	0.92	0.093	91.46%
SimpleLogistic	0.938	0.075	93.29%
NaiveBayes	0.776	0.088	82.97%
Adaboost(NaiveBayes)	0.929	0.136	90.36%

經由表 6 的實驗數據我們可以發現，其實藉由宣告的權限，我們就可以得到很好的準確率結果，會有這樣的結果可能是大部分的惡意程式會宣告類似的權限，而一般程式則較少會宣告這些權限。

表 7 則是單以網路封包資料洩漏行為作為特徵屬性，產生的分類結果，左邊是以 Monkey 作為行為觸發器的結果、右邊則是本論文藉由 Robotium 函式庫所開發之行為觸發器的結果，圖 21、圖 22、圖 23 則分別是依照表 7 產生的 Monkey 與本論文行為觸發器之比較折線圖，nw(mk)代表以 Monkey 產生的網路封包資料洩漏行為、nw(rb) 代表以本論文行為觸發器產生的網路封包資料洩漏行為，橫軸的各英文字母代號請參閱表 5。

表 7 以網路封包資料洩漏行為作為特徵屬性之結果

monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.875	0.096	88.63%	SVM	64.80%	0.767	0.918
0.983	0.76	68.87%	SMO Poly	60.72%	0.996	1
0.996	0.975	61.20%	SMO NPoly	62.83%	0.172	0.498
0.965	0.125	92.95%	REPTree	75.67%	0.473	0.906
0.947	0.113	92.33%	Adaboost(REPTree)	75.38%	0.388	0.846
0.965	0.133	92.61%	GB(REPTree)	75.67%	0.467	0.902
0.961	0.127	92.61%	LogitBoost(REPTree)	75.53%	0.463	0.898
0.954	0.109	92.90%	MultiBoostAB(REPTree)	75.24%	0.445	0.881
0.97	0.131	93.00%	J48	76.06%	0.462	0.906
0.945	0.133	91.41%	RandomTree	70.96%	0.438	0.806
0.941	0.118	91.80%	RandomForest 10	71.06%	0.45	0.815
0.941	0.116	91.85%	RandomForest 50	70.87%	0.44	0.806
0.941	0.115	91.89%	RandomForest 100	70.43%	0.44	0.798
0.943	0.133	91.27%	KNN	70.77%	0.444	0.806
0.941	0.132	91.22%	Adaboost(KNN)	70.77%	0.444	0.806
0.943	0.133	91.27%	GB(KNN)	70.77%	0.444	0.806
0.943	0.133	91.27%	LogitBoost(KNN)	70.77%	0.444	0.806
0.936	0.124	91.22%	MultiBoostAB(KNN)	69.66%	0.435	0.783
0.945	0.12	91.94%	KNN 3	73.56%	0.43	0.844
0.945	0.122	91.85%	KNN 5	74.76%	0.427	0.861
0.96	0.124	92.71%	KNN 10	74.86%	0.467	0.889
0.981	0.364	84.46%	SimpleLogistic	60.72%	0.977	0.987
0.95	0.92	60.58%	NaiveBayes	47.79%	0.389	0.391
0.95	0.92	60.58%	Adaboost(NaiveBayes)	49.52%	0.423	0.442

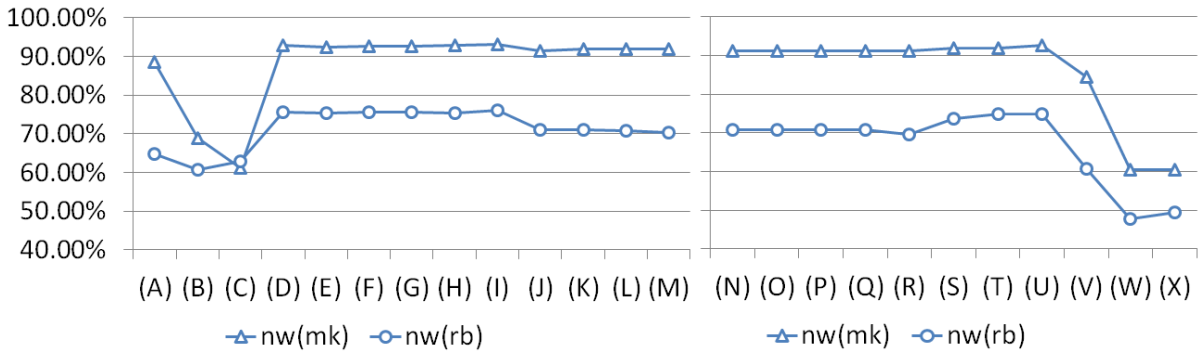


圖 21 網路封包資料洩漏行為作為特徵屬性之 Accuracy 比較折線圖

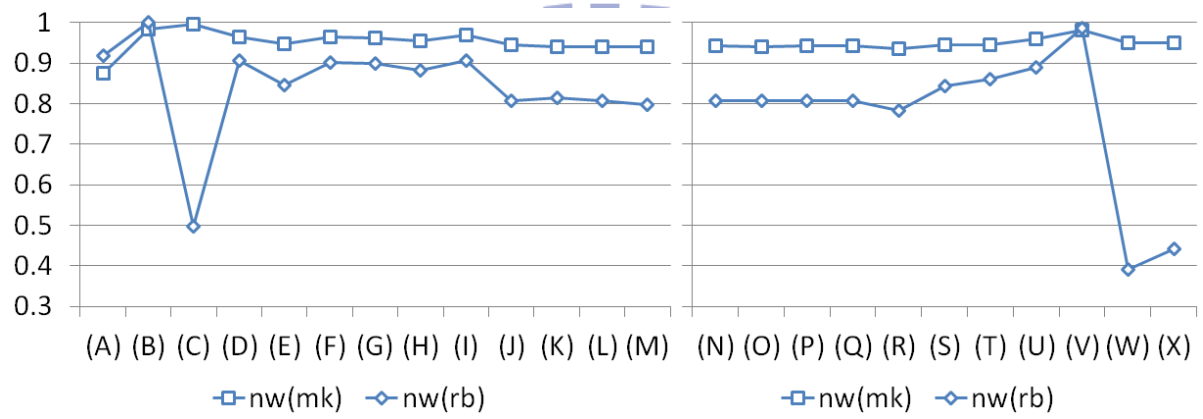


圖 22 網路封包資料洩漏行為作為特徵屬性之 TPR 比較折線圖

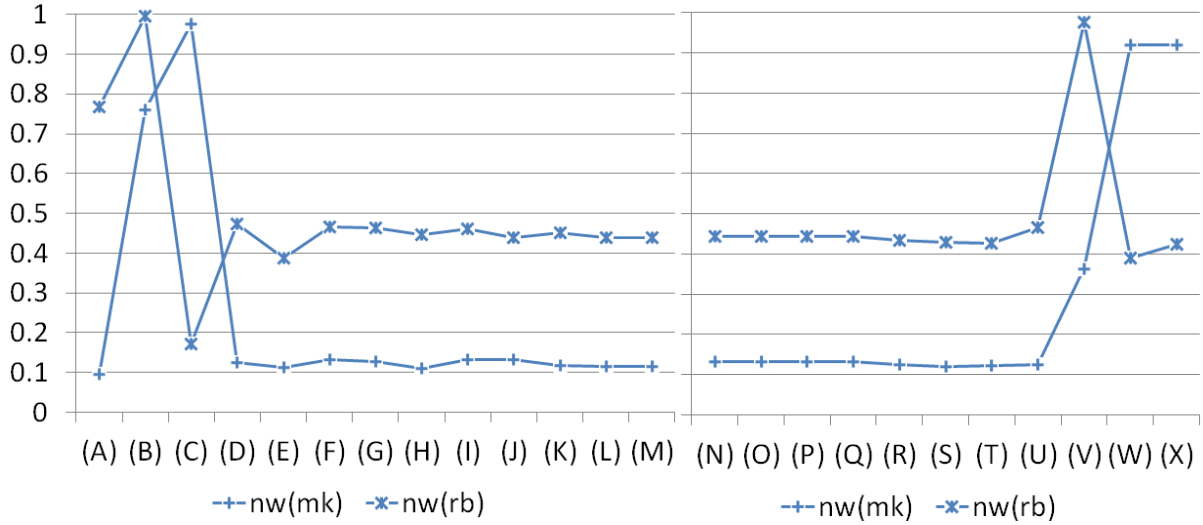


圖 23 網路封包資料洩漏行為作為特徵屬性之 FPR 比較折線圖

經由表 7 與圖 21、圖 22、圖 23，我們可以很明顯看出原先 Monkey 的結果，比本論文所開發的行為觸發程式來的準確。其實仔細考慮後，會發現這樣的結果是有可能的，因為不管使用者有沒有使用惡意程式，惡意程式仍舊會執行這些網路封包資料洩漏行為，而一般程式雖然也會有網路封包洩漏行為，但是卻是藉由使用者執行該一般程式所產生的，因此當系統使用 Monkey 作為行為觸發程式時，Monkey 不管有沒有正確地點擊到惡意程式畫面上的物件，惡意程式仍舊會有網路封包洩漏行為；一般程式則需要 Monkey 點擊到，方能產生該行為，於是演算法可以藉由這

樣的特徵屬性進行分類。換成本論文之行為觸發程式，因為本程式能夠確實分辨應用程式畫面上物件並進行點擊，反而觸發一般程式的網路封包洩漏行為，導致最後不管是惡意程式或者一般程式，皆會觀察到網路封包洩漏行為，以致這類特徵屬性之結果 Monkey 較本行為觸發程式準確。

表 8 以 system call 次數統計作為特徵屬性之結果

monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.933	0.79	64.65%	SVM	63.21%	1	1
0.929	0.567	73.29%	SMO Poly	81.07%	0.407	0.938
0.828	0.24	80.10%	SMO NPoly	82.82%	0.342	0.927
0.821	0.221	80.43%	REPTree	81.69%	0.291	0.88
0.851	0.15	85.04%	Adaboost(REPTree)	87.38%	0.193	0.913
0.825	0.21	81.15%	GB(REPTree)	83.21%	0.266	0.889
0.853	0.177	84.12%	LogitBoost(REPTree)	86.20%	0.213	0.906
0.856	0.19	83.79%	MultiBoostAB(REPTree)	87.44%	0.219	0.929
0.819	0.19	81.53%	J48	83.21%	0.224	0.865
0.813	0.208	80.48%	RandomTree	83.66%	0.254	0.889
0.879	0.217	84.12%	RandomForest 10	87.27%	0.245	0.941
0.885	0.159	86.76%	RandomForest 50	88.73%	0.201	0.939
0.877	0.154	86.47%	RandomForest 100	89.07%	0.194	0.94
monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.818	0.211	80.67%	KNN	85.63%	0.225	0.904
0.818	0.211	80.67%	Adaboost(KNN)	85.63%	0.225	0.904
0.818	0.211	80.67%	GB(KNN)	85.63%	0.225	0.904
0.818	0.211	80.67%	LogitBoost(KNN)	85.63%	0.225	0.904
0.818	0.211	80.67%	MultiBoostAB(KNN)	85.63%	0.225	0.904
0.839	0.221	81.53%	KNN 3	83.66%	0.277	0.903
0.838	0.242	80.62%	KNN 5	83.66%	0.3	0.916
0.86	0.314	79.14%	KNN 10	81.92%	0.387	0.939
0.902	0.444	76.55%	SimpleLogistic	82.48%	0.331	0.915
0.304	0.064	55.40%	NaiveBayes	66.65%	0.191	0.584
0.402	0.124	58.94%	Adaboost(NaiveBayes)	73.92%	0.323	0.775

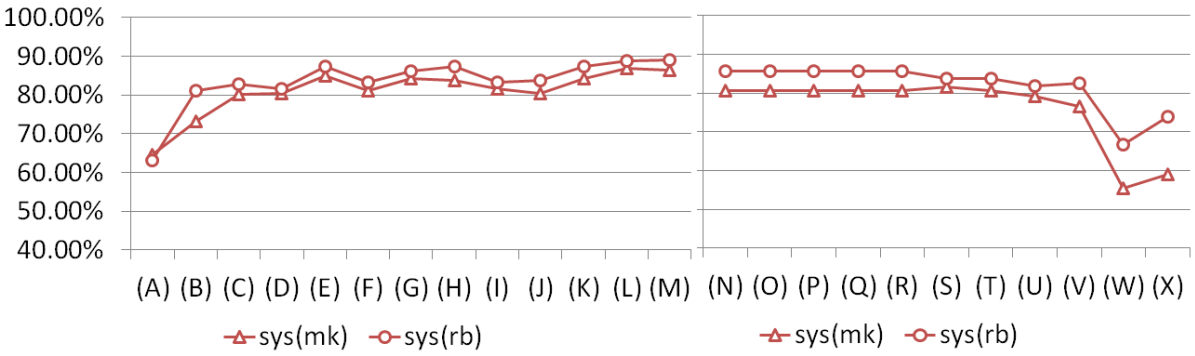


圖 24 以 system call 次數統計作為特徵屬性之 Accuracy 比較折線圖

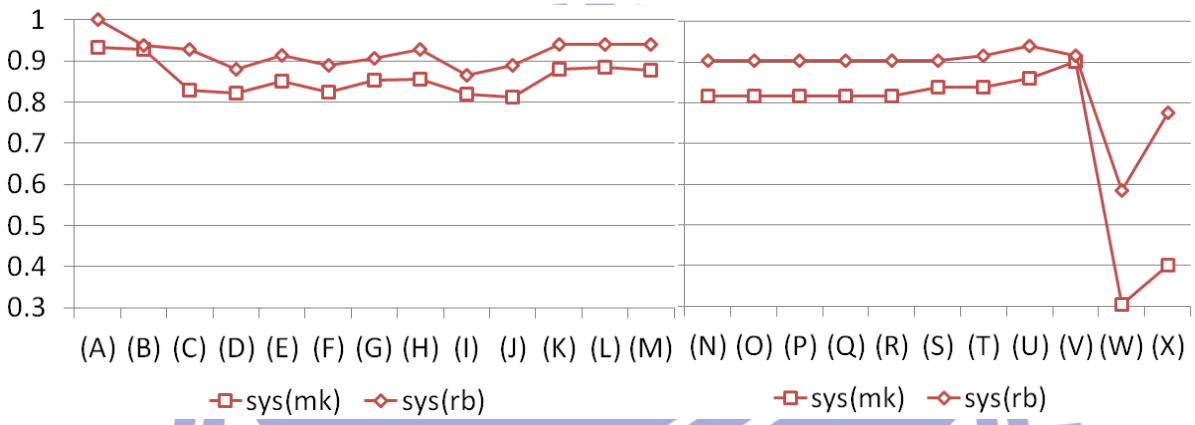


圖 25 以 system call 次數統計作為特徵屬性之 TPR 比較折線圖

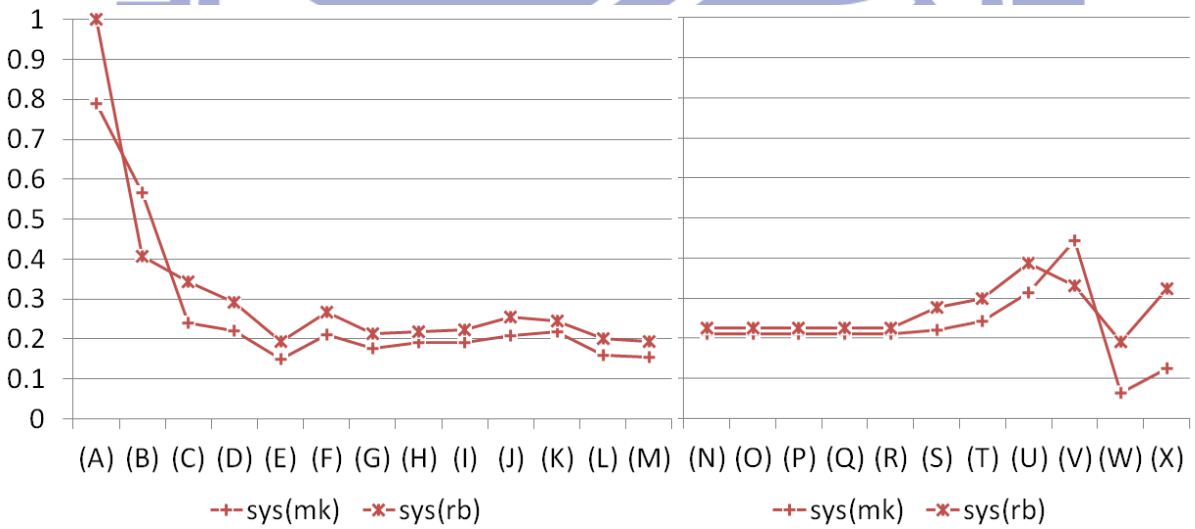


圖 26 以 system call 次數統計作為特徵屬性之 FPR 比較折線圖

至於使用 system call 統計之部分，藉由表 8 和圖 24、圖 25、圖 26 我們可以看到，可能是因為這類特徵屬性收集到其他程式行為，而不僅只是網路封包行為，而且本行為觸發程式也能夠觸發應用程式以表現出這些 system call 行為，造成演算法可以藉由這些行為進行分類，因此可以看到在這個部分，本行為觸發器的結果比 Monkey 來的好。

以下則是上述三類特徵屬性集合所產生的兩兩聯集，接下來表格與圖片會使用

簡稱表示即網路行為與權限的聯集(簡稱 nw+pm)、網路行為與 system call 統計的聯集(簡稱 nw+sys)、system call 與權限的聯集(簡稱 sys+pm)、和全部的特徵屬性所產生的聯集(簡稱 nw+sys+pm)。

表 9 以 nw+pm 作為特徵屬性之結果

monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.944	0.112	92.23%	SVM	66.57%	0.573	0.822
0.937	0.051	94.20%	SMO Poly	94.15%	0.053	0.938
0.942	0.033	95.20%	SMO NPoly	95.01%	0.039	0.943
0.967	0.036	96.59%	REPTree	92.42%	0.093	0.936
0.977	0.033	97.31%	Adaboost(REPTree)	94.67%	0.072	0.959
0.968	0.042	96.40%	GB(REPTree)	93.62%	0.074	0.943
0.979	0.029	97.55%	LogitBoost(REPTree)	94.24%	0.084	0.96
0.978	0.032	97.41%	MultiBoostAB(REPTree)	94.39%	0.073	0.955
0.974	0.05	96.45%	J48	94.29%	0.064	0.948
0.964	0.068	95.16%	RandomTree	91.94%	0.12	0.945
0.98	0.041	97.17%	RandomForest 10	95.40%	0.067	0.967
0.981	0.025	97.84%	RandomForest 50	96.55%	0.039	0.968
0.979	0.023	97.84%	RandomForest 100	96.50%	0.038	0.967
monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.974	0.053	96.31%	KNN	94.39%	0.09	0.966
0.973	0.053	96.30%	Adaboost(KNN)	94.39%	0.09	0.966
0.974	0.053	96.31%	GB(KNN)	94.39%	0.09	0.966
0.974	0.053	96.31%	LogitBoost(KNN)	94.39%	0.09	0.966
0.974	0.053	96.31%	MultiBoostAB(KNN)	94.39%	0.09	0.966
0.972	0.076	95.30%	KNN 3	94.05%	0.091	0.961
0.97	0.079	95.06%	KNN 5	94.00%	0.087	0.958
0.967	0.096	94.24%	KNN 10	92.09%	0.112	0.942
0.946	0.059	94.39%	SimpleLogistic	93.19%	0.08	0.94
0.871	0.107	88.01%	NaiveBayes	83.26%	0.085	0.779
0.921	0.08	92.04%	Adaboost(NaiveBayes)	89.77%	0.126	0.913

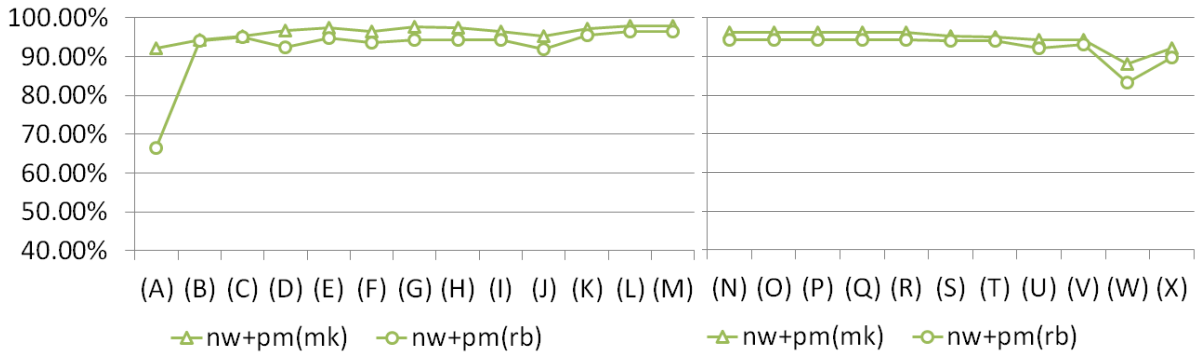


圖 27 以 nw+pm 作為特徵屬性之 Accuracy 比較折線圖

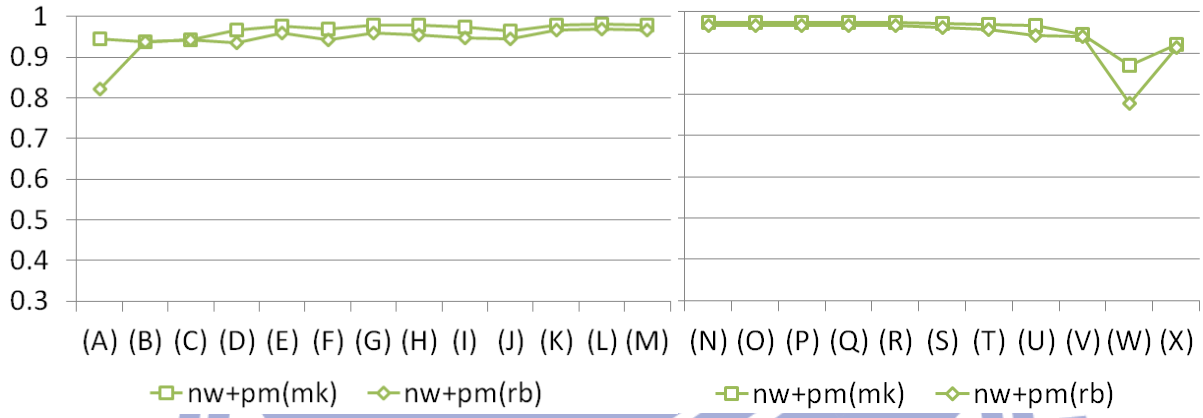


圖 28 以 nw+pm 作為特徵屬性之 TPR 比較折線圖

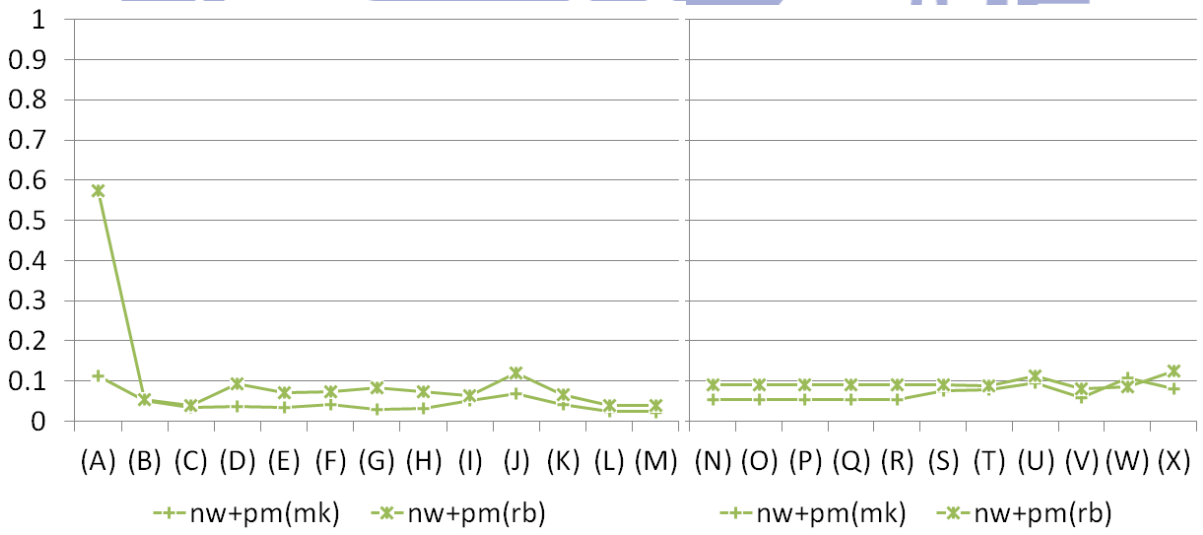


圖 29 以 nw+pm 作為特徵屬性之 FPR 比較折線圖

網路行為與權限的聯集部分，因為 pm 不會造成 Monkey 與本行為觸發程式結果不同，因此這部分只有 nw 會造成影響，於是結果類似單看 nw 部分。

表 10 以 nw+sys 作為特徵屬性之結果

monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.998	0.805	68.01%	SVM	63.46%	0.918	0.994
0.941	0.457	78.67%	SMO Poly	76.15%	0.52	0.944
0.871	0.141	86.62%	SMO NPoly	83.17%	0.276	0.902
0.972	0.064	95.78%	REPTree	82.84%	0.255	0.883
0.975	0.046	96.69%	Adaboost(REPTree)	87.88%	0.171	0.911
0.965	0.052	95.83%	GB(REPTree)	84.57%	0.223	0.89
0.974	0.044	96.69%	LogitBoost(REPTree)	86.97%	0.189	0.908
0.974	0.044	96.69%	MultiBoostAB(REPTree)	88.27%	0.172	0.918
0.967	0.058	95.68%	J48	85.67%	0.218	0.906
0.898	0.156	87.67%	RandomTree	82.74%	0.237	0.869
0.964	0.136	92.47%	RandomForest 10	88.17%	0.204	0.937
0.966	0.069	95.20%	RandomForest 50	89.38%	0.168	0.934
0.968	0.063	95.59%	RandomForest 100	89.95%	0.163	0.94

monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.918	0.173	88.20%	KNN	85.77%	0.213	0.904
0.917	0.172	88.20%	Adaboost(KNN)	85.82%	0.212	0.904
0.918	0.173	88.20%	GB(KNN)	85.77%	0.213	0.904
0.918	0.173	88.20%	LogitBoost(KNN)	85.72%	0.213	0.903
0.918	0.173	88.20%	MultiBoostAB(KNN)	85.77%	0.213	0.904
0.936	0.184	88.82%	KNN 3	84.52%	0.243	0.902
0.939	0.192	88.73%	KNN 5	83.70%	0.27	0.906
0.952	0.265	86.57%	KNN 10	81.68%	0.34	0.919
0.983	0.101	94.96%	SimpleLogistic	78.61%	0.265	0.819
0.317	0.067	56.07%	NaiveBayes	58.38%	0.123	0.41
0.468	0.155	61.73%	Adaboost(NaiveBayes)	66.30%	0.23	0.594

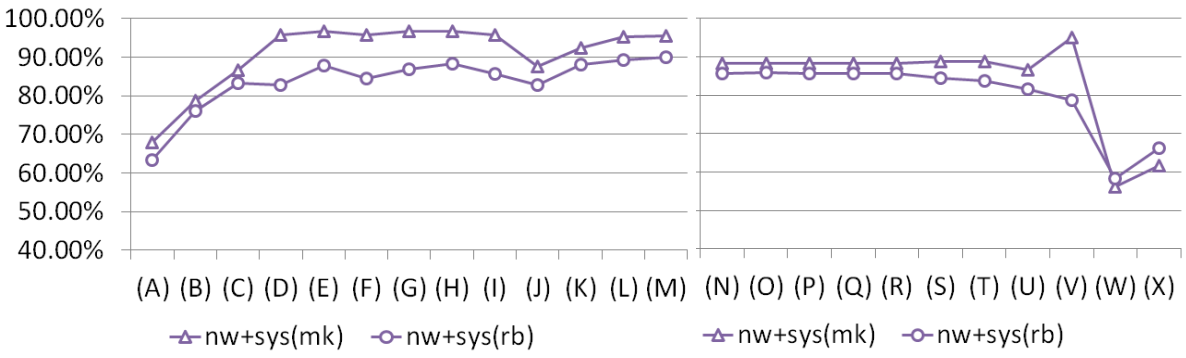


圖 30 以 nw+sys 作為特徵屬性之 Accuracy 比較折線圖

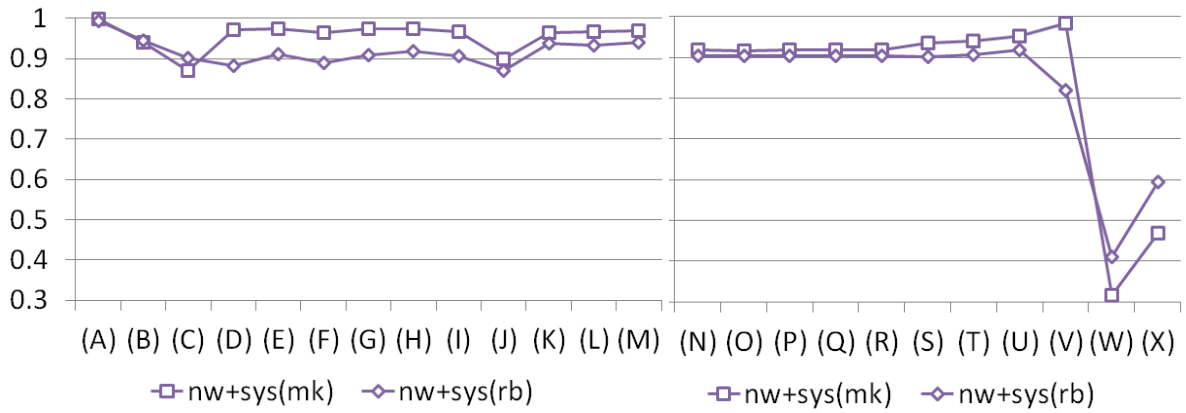


圖 31 以 nw+sys 作為特徵屬性之 TPR 比較折線圖

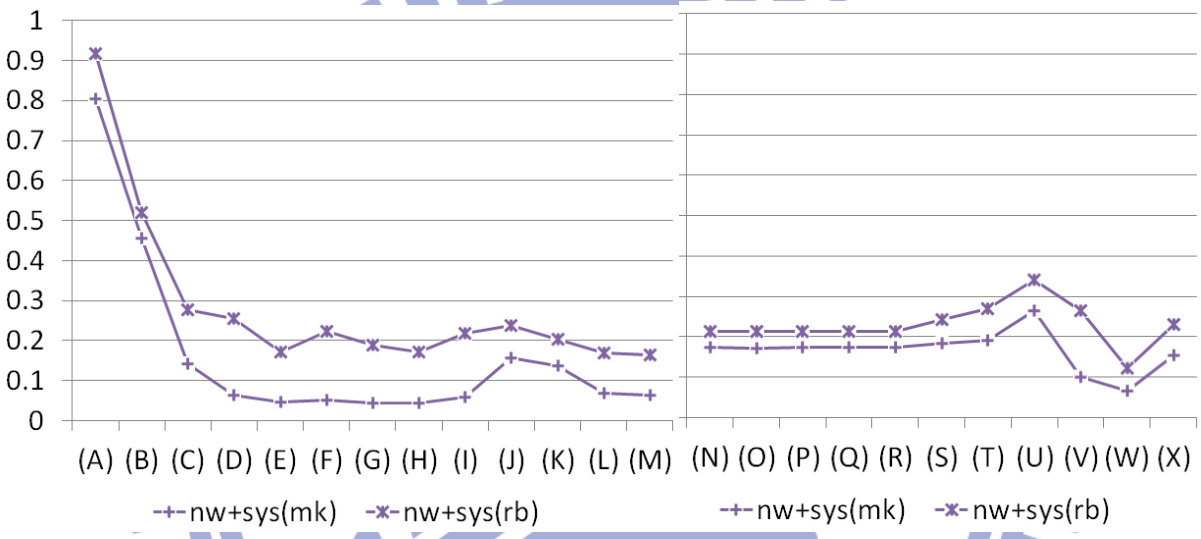


圖 32 以 nw+sys 作為特徵屬性之 FPR 比較折線圖

網路行為與 system call 統計的聯集部分，則是因為 Monkey 於 nw 部分，數值高過本行為觸發器太多，即使加上 sys 部分仍舊無法贏過 Monkey，因此這部分依舊是 Monkey 勝出。

表 11 以 sys+pm 作為特徵屬性之結果

monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.965	0.789	66.67%	SVM	66.19%	0.79	0.958
0.944	0.058	94.34%	SMO Poly	94.58%	0.051	0.944
0.953	0.035	95.78%	SMO NPoly	95.64%	0.036	0.952
0.937	0.112	91.80%	REPTree	90.46%	0.126	0.925
0.957	0.081	94.20%	Adaboost(REPTree)	93.62%	0.09	0.953
0.947	0.101	92.81%	GB(REPTree)	91.41%	0.107	0.928
0.96	0.062	95.11%	LogitBoost(REPTree)	93.67%	0.086	0.952
0.948	0.074	93.96%	MultiBoostAB(REPTree)	93.77%	0.073	0.944
0.95	0.104	92.85%	J48	92.66%	0.092	0.939
0.896	0.171	86.95%	RandomTree	87.39%	0.176	0.906
0.962	0.121	92.90%	RandomForest 10	93.43%	0.099	0.956
0.967	0.082	94.72%	RandomForest 50	95.16%	0.069	0.965
0.967	0.078	94.96%	RandomForest 100	95.44%	0.057	0.962

monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.962	0.085	94.34%	KNN	93.96%	0.097	0.963
0.962	0.085	94.34%	Adaboost(KNN)	93.96%	0.097	0.963
0.962	0.085	94.34%	GB(KNN)	93.96%	0.097	0.963
0.962	0.085	94.34%	LogitBoost(KNN)	93.96%	0.097	0.963
0.962	0.085	94.34%	MultiBoostAB(KNN)	93.72%	0.103	0.963
0.971	0.098	94.34%	KNN 3	93.48%	0.11	0.964
0.967	0.113	93.53%	KNN 5	92.71%	0.118	0.956
0.948	0.115	92.33%	KNN 10	91.70%	0.126	0.945
0.95	0.072	94.15%	SimpleLogistic	94.29%	0.065	0.948
0.505	0.032	68.82%	NaiveBayes	78.08%	0.052	0.671
0.744	0.048	82.59%	Adaboost(NaiveBayes)	81.82%	0.088	0.757

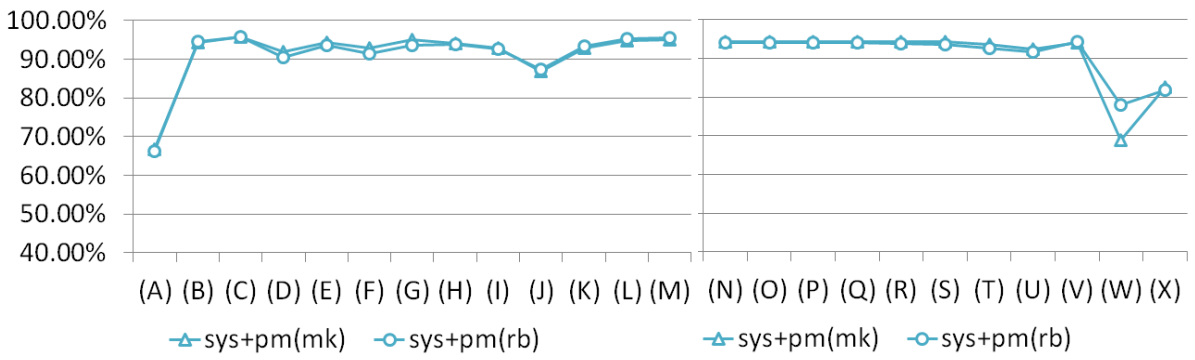


圖 33 以 sys+pm 作為特徵屬性之 Accuracy 比較折線圖

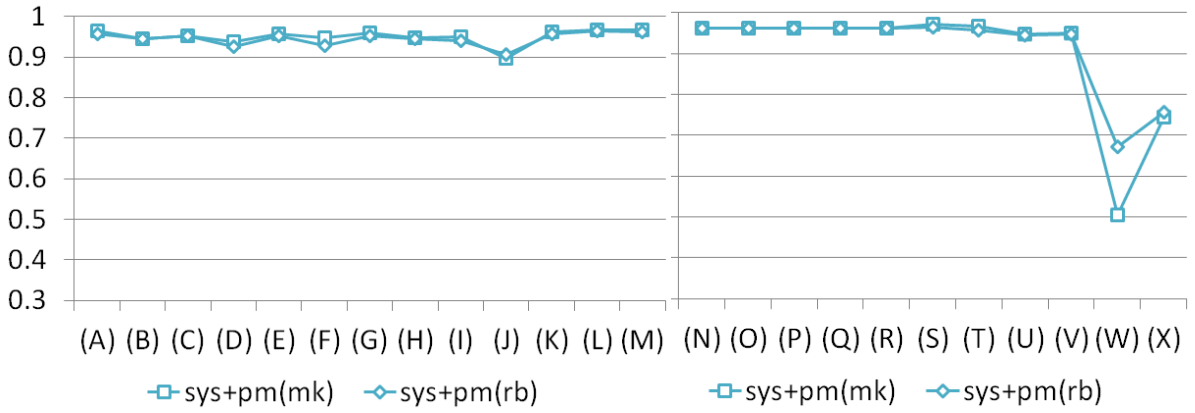


圖 34 以 sys+pm 作為特徵屬性之 TPR 比較折線圖

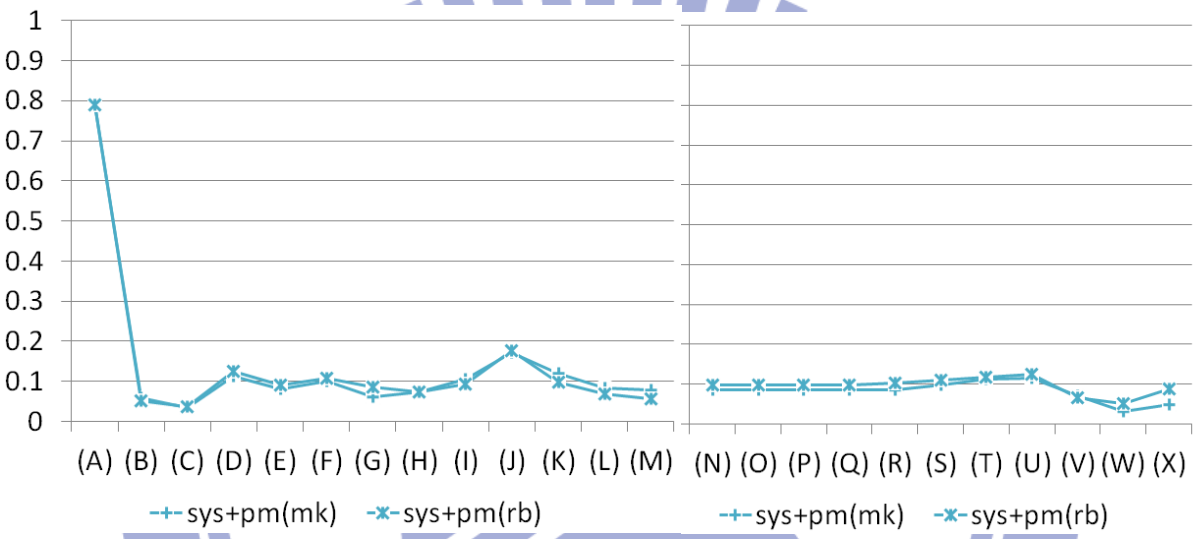


圖 35 以 sys+pm 作為特徵屬性之 FPR 比較折線圖

至於 system call 與權限的聯集部分，我們可以從表 6 與表 8 看到，因為 pm 對於各演算法的分類結果影響大於 sys，所以僅可以從表 11 和圖 33、圖 34、圖 35 觀察到，本行為觸發程式之結果比 Monkey 之結果來的有些微改善。

表 12 以 nw+sys+pm 作為特徵屬性之結果

monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.997	0.805	67.96%	SVM	64.12%	0.893	0.991
0.945	0.061	94.29%	SMO Poly	94.58%	0.051	0.944
0.956	0.036	95.92%	SMO NPoly	95.78%	0.038	0.955
0.968	0.046	96.26%	REPTree	91.75%	0.118	0.94
0.983	0.024	98.03%	Adaboost(REPTree)	94.77%	0.08	0.966
0.971	0.042	96.59%	GB(REPTree)	92.47%	0.095	0.937
0.983	0.027	97.89%	LogitBoost(REPTree)	94.39%	0.076	0.957
0.984	0.028	97.94%	MultiBoostAB(REPTree)	94.34%	0.081	0.96
0.974	0.046	96.59%	J48	93.38%	0.085	0.946
0.914	0.137	89.40%	RandomTree	88.35%	0.153	0.907
0.98	0.118	94.15%	RandomForest 10	93.82%	0.103	0.965
0.981	0.056	96.64%	RandomForest 50	95.25%	0.065	0.964
0.984	0.051	97.03%	RandomForest 100	95.54%	0.062	0.967

monkey				robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.963	0.085	94.39%	KNN	94.15%	0.099	0.968
0.963	0.085	94.39%	Adaboost(KNN)	94.15%	0.099	0.968
0.963	0.085	94.39%	GB(KNN)	94.15%	0.099	0.968
0.963	0.085	94.39%	LogitBoost(KNN)	94.15%	0.099	0.968
0.963	0.085	94.39%	MultiBoostAB(KNN)	94.15%	0.099	0.968
0.97	0.097	94.34%	KNN 3	93.67%	0.113	0.969
0.967	0.108	93.72%	KNN 5	93.19%	0.116	0.963
0.96	0.114	93.05%	KNN 10	91.70%	0.143	0.956
0.99	0.036	97.94%	SimpleLogistic	94.10%	0.072	0.949
0.552	0.032	71.65%	NaiveBayes	77.27%	0.055	0.66
0.771	0.076	83.12%	Adaboost(NaiveBayes)	82.88%	0.09	0.775

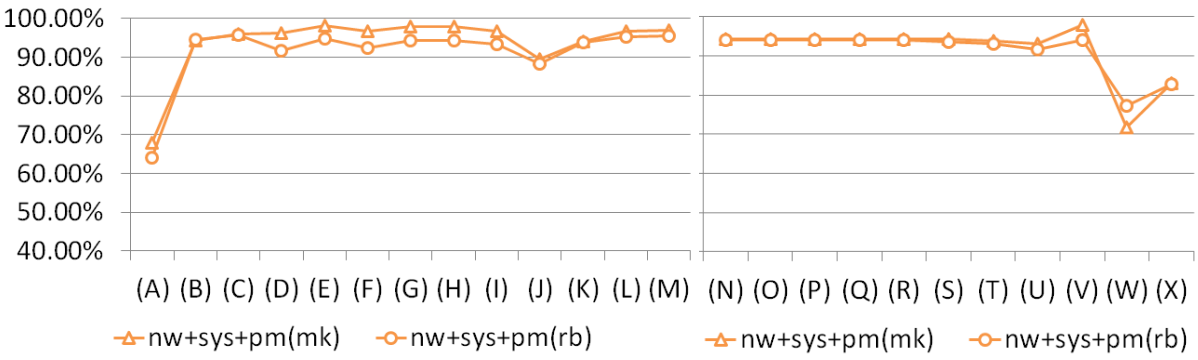


圖 36 以 nw+sys+pm 作為特徵屬性之 Accuracy 比較折線圖

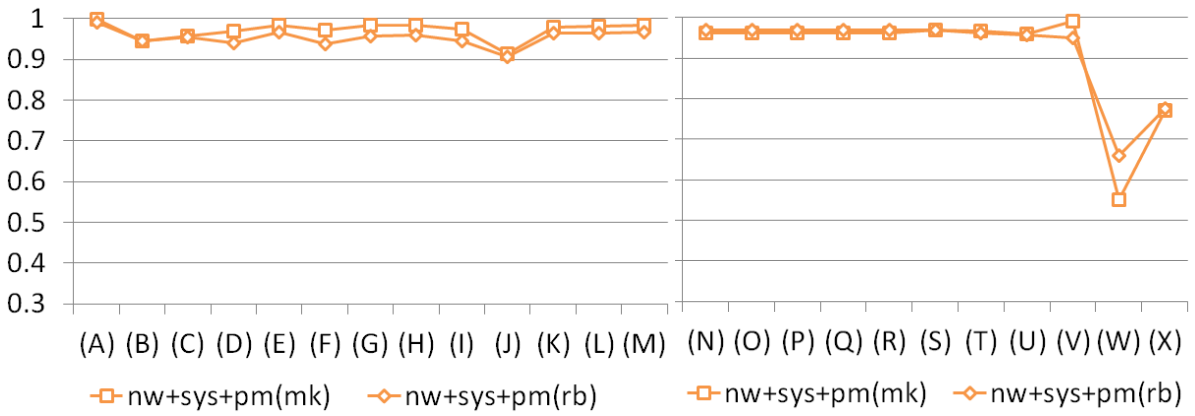


圖 37 以 nw+sys+pm 作為特徵屬性之 TPR 比較折線圖

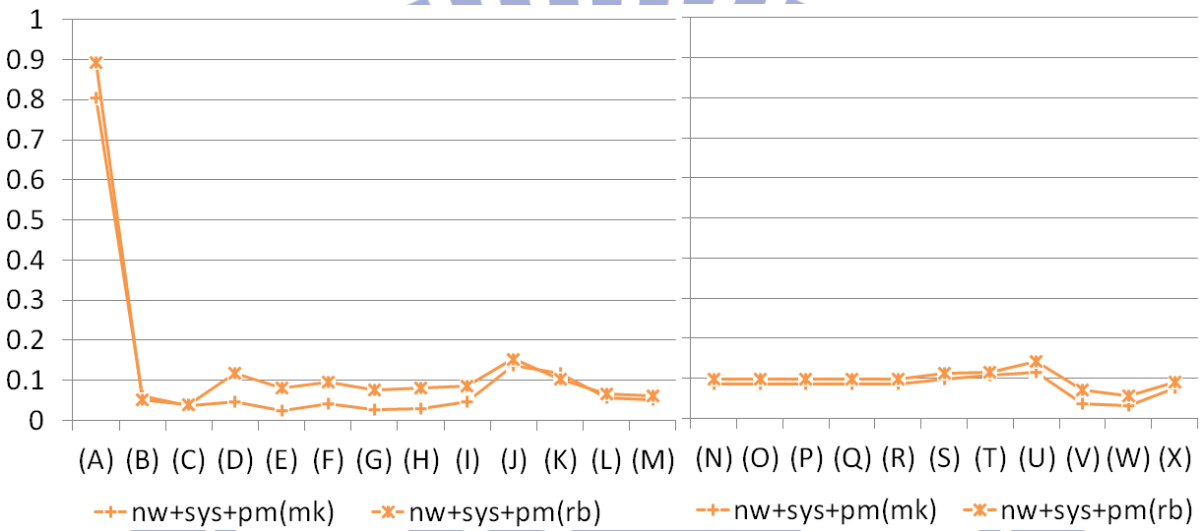


圖 38 以 nw+sys+pm 作為特徵屬性之 FPR 比較折線圖

全部三類特徵屬性之結果部分，則是因為 nw 部分特徵屬性，造成最後仍舊是 Monkey 勝過本行為觸發程式。

雖然從這些圖表，我們可以發現本行為觸發程式產生之結果，並不會高於 Monkey 太多，但其實仔細考慮惡意程式與一般程式行為後，本論文應該確實製作了一個不錯的行為觸發程式，能夠確實觸發一般程式行為，造成機器學習演算法無法藉由本行為觸發程式產生之行為，對惡意程式與一般程式準確分類。

下圖是以 nw+sys+pm 作為特徵屬性，RandomForest 100 的機器學習結果產生出的 ROC 曲線圖。

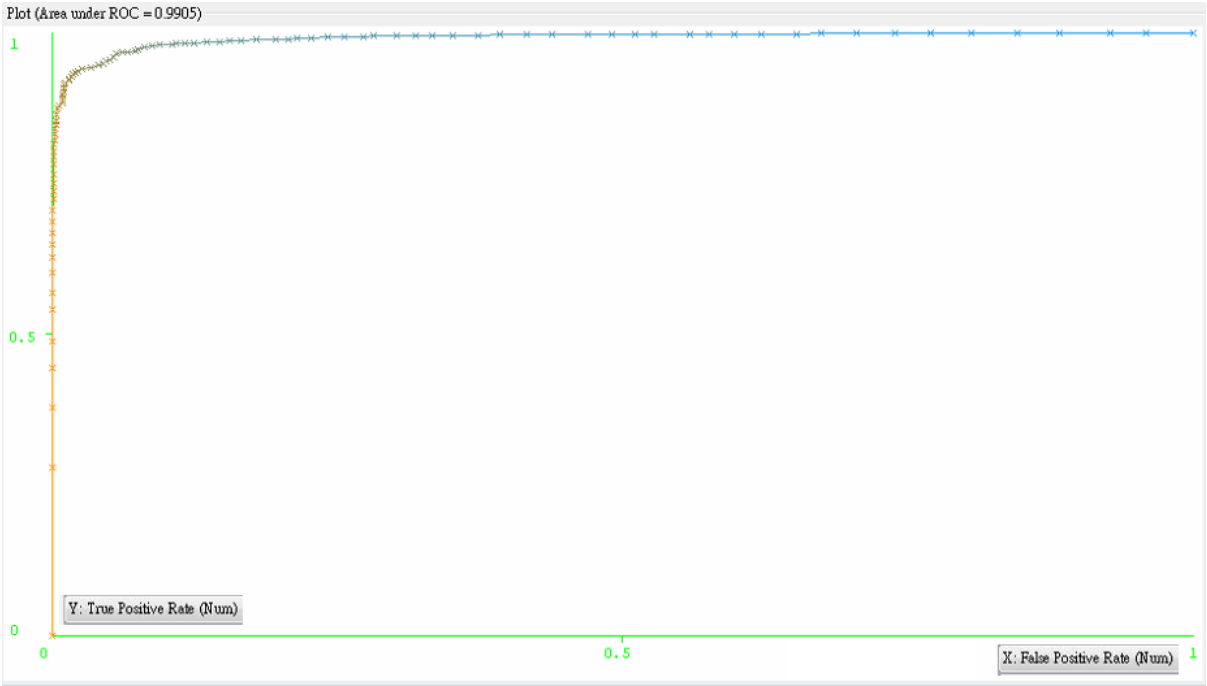
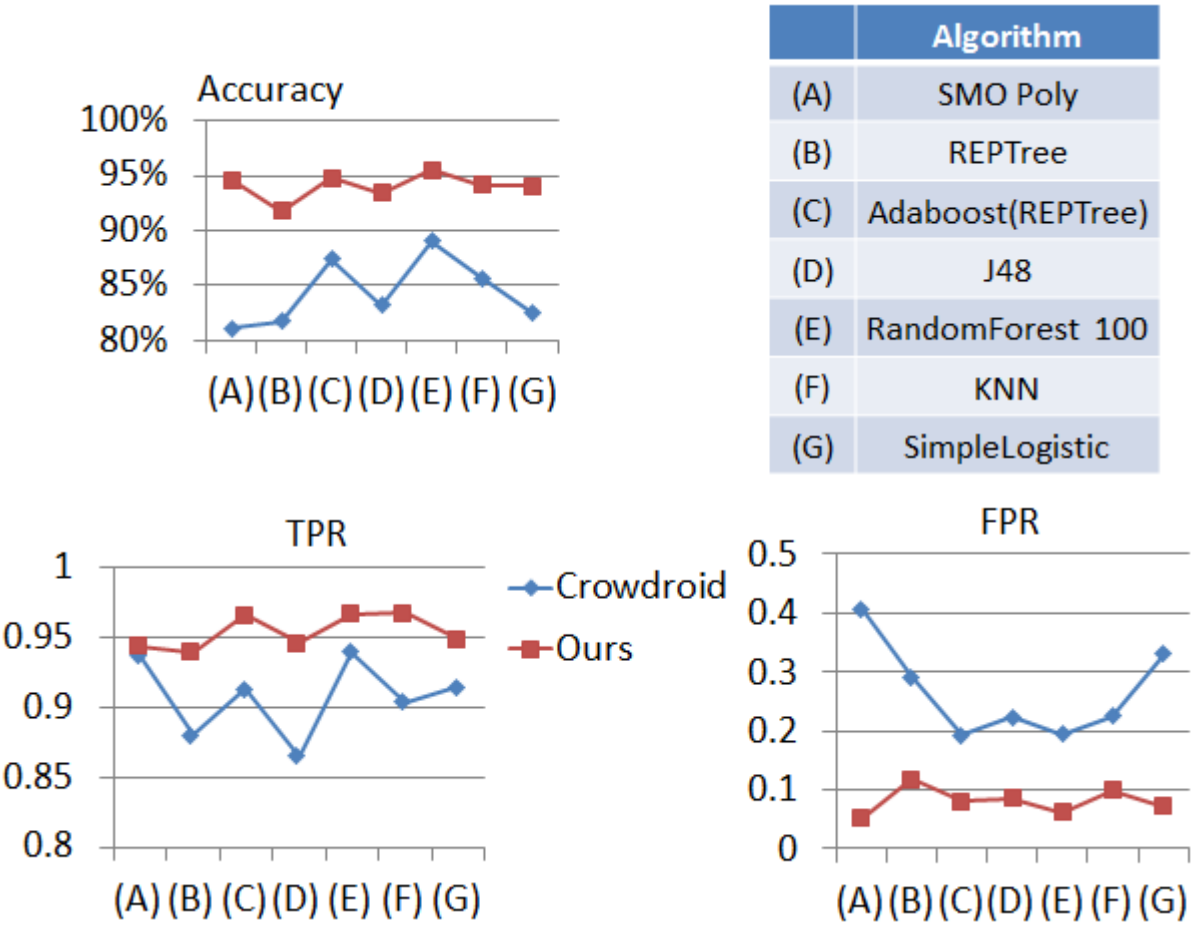


圖 39 以 nw+sys+pm 作為特徵屬性 RandomForest 100 的 ROC 曲線圖

6.4 相關論文比較

本章節利用三類特徵屬性，並且使用一些機器學習演算法，與相關論文[12] 作比較，圖中紅線是我們的三類特徵屬性聯集之結果，藍線則是論文[12] 的結果，X軸為 7 種不同的機器學習演算法，請對照下表。由此圖可見本論文選擇的三類特徵屬性，對於分類惡意程式有顯著的效果。



	Algorithm
(A)	SMO Poly
(B)	REPTree
(C)	Adaboost(REPTree)
(D)	J48
(E)	RandomForest 100
(F)	KNN
(G)	SimpleLogistic

圖 40 相關論文偵測率比較

6.5 System Call 與 System Call n-gram 比較

本章節探討是否能夠藉由使用 system call n-gram，相對於單純統計各個 system call 使用次數，提升整體系統分辨率。因此本論文先單純以 system call 的特徵屬性作比較，不加入網路與權限的特徵屬性，得到以下表 13 之結果，表 13 只顯示在 n 大於 2 的情況下，Accuracy 有提升的演算法，n=1 即單純統計各個 system call 使用次數之情況。

表 13 system call n-gram 比較表

Algorithm		n=1	n=2	n=3	n=4
SVM	Accuracy	63.21%	63.55%	64.56%	67.61%
	TPR	1	0.999	0.993	0.966
	FPR	1	0.989	0.951	0.822
SMO Poly	Accuracy	81.07%	85.75%	85.01%	88.17%
	TPR	0.938	0.935	0.908	0.928
	FPR	0.407	0.276	0.25	0.198
SMO NPoly	Accuracy	82.82%	86.20%	85.86%	85.63%
	TPR	0.927	0.942	0.952	0.95
	FPR	0.342	0.276	0.302	0.305
REPTree	Accuracy	81.69%	83.04%	82.14%	81.01%
	TPR	0.88	0.873	0.881	0.879
	FPR	0.291	0.242	0.282	0.308
NaiveBayes	Accuracy	66.65%	76.85%	78.20%	76.79%
	TPR	0.584	0.858	0.86	0.869
	FPR	0.191	0.386	0.352	0.406

可以看到確實有部分演算法在 n > 2 的情況，Accuracy 有確實提升，例如 SMO Poly、SMO NPoly、和 NaiveBayes，證實本論文的確可以藉由 system call 順序產生之 n-gram，提升分類準確率。

本論文原先打算產生 n=1 至 n=10 之結果，然而當 n=3 特徵屬性數量已達 48,760，至 n=4 更高達 244,850，因為記憶體不足與計算時間過長，本論文只能進行到 n=4 而已，就沒有繼續產生 n=5 以上之 ARFF。

因此我們嘗試將 system call n-gram 的特徵屬性與其他兩種特徵屬性作結合，因此產生以下結果。

表 14 nw+sys n-gram 之比較表

Algorithm		n=1	n=2	n=3	n=4
SVM	Accuracy	63.46%	63.70%	64.18%	66.25%
	TPR	0.994	0.986	0.974	0.963
	FPR	0.918	0.899	0.868	0.799
SMO Poly	Accuracy	76.15%	80.77%	79.86%	81.68%
	TPR	0.944	0.829	0.794	0.813
	FPR	0.52	0.226	0.195	0.177
SMO NPoly	Accuracy	83.17%	82.5%	82.64%	81.11%
	TPR	0.902	0.911	0.926	0.923
	FPR	0.276	0.307	0.327	0.361
REPTree	Accuracy	82.84%	82.93%	83.89%	82.21%
	TPR	0.883	0.875	0.879	0.868
	FPR	0.255	0.241	0.223	0.249
NaiveBayes	Accuracy	58.38%	71.88%	74.04%	72.74%
	TPR	0.41	0.846	0.869	0.871
	FPR	0.123	0.477	0.457	0.494

表 15 sys n-gram+pm 之比較表

Algorithm		n=1	n=2	n=3	n=4
SVM	Accuracy	66.19%	62.25%	61.82%	64.84%
	TPR	0.958	0.887	0.87	0.835
	FPR	0.79	0.782	0.766	0.636
SMO Poly	Accuracy	94.58%	94.58%	93.91%	93.86%
	TPR	0.944	0.958	0.948	0.947
	FPR	0.051	0.073	0.075	0.074
SMO NPoly	Accuracy	95.64%	95.64%	94.15%	92.76%
	TPR	0.952	0.948	0.927	0.921
	FPR	0.036	0.032	0.036	0.062
REPTree	Accuracy	90.46%	90.65%	90.98%	91.18%
	TPR	0.925	0.925	0.929	0.937
	FPR	0.126	0.121	0.12	0.126
NaiveBayes	Accuracy	78.08%	72.13%	73.57%	73.14%
	TPR	0.671	0.853	0.863	0.877
	FPR	0.052	0.48	0.459	0.491

表 16 nw+sys n-gram+pm 之比較表

Algorithm		n=1	n=2	n=3	n=4
SVM	Accuracy	64.12%	63.36%	64.22%	66.33%
	TPR	0.991	0.981	0.976	0.962
	FPR	0.893	0.897	0.868	0.793
SMO Poly	Accuracy	94.58%	94.53%	93.91%	93.91%
	TPR	0.944	0.958	0.948	0.947
	FPR	0.051	0.074	0.075	0.073
SMO NPoly	Accuracy	95.78%	95.73%	94.24%	93.29%
	TPR	0.955	0.952	0.929	0.929
	FPR	0.038	0.034	0.036	0.062
REPTree	Accuracy	91.75%	90.98%	91.37%	91.70%
	TPR	0.94	0.93	0.938	0.94
	FPR	0.118	0.121	0.124	0.119
NaiveBayes	Accuracy	77.27%	72.23%	73.67%	73.29%
	TPR	0.66	0.854	0.864	0.878
	FPR	0.055	0.479	0.458	0.488

為了更確實的了解，各演算法於 n 增加的時候，是否 Accuracy 和 TPR 也隨之增加，FPR 也隨之遞減，因此本論文另外繪製各演算法於不同輸入的特徵屬性產生之折線圖。小寫英文分別代表不同輸入的特徵屬性，如表 17。

表 17 各小寫英文字母代號對應之特徵屬性集合

(a)	sys
(b)	nw+sys
(c)	sys+pm
(d)	nw+sys+pm

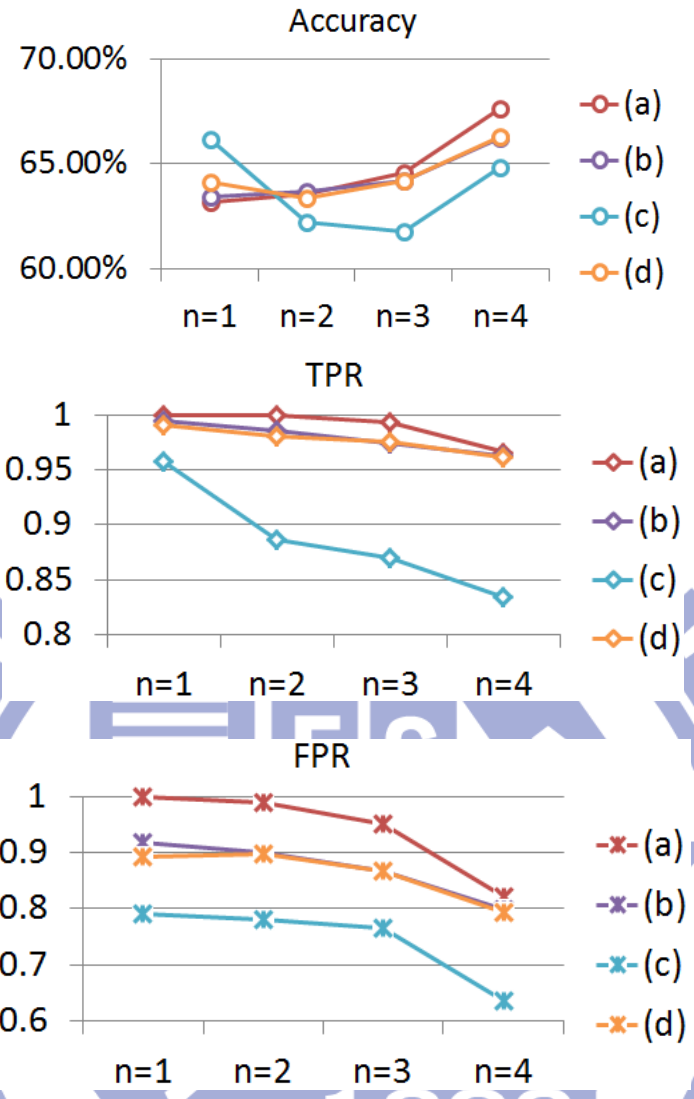


圖 41 SVM 各特徵屬性走勢圖

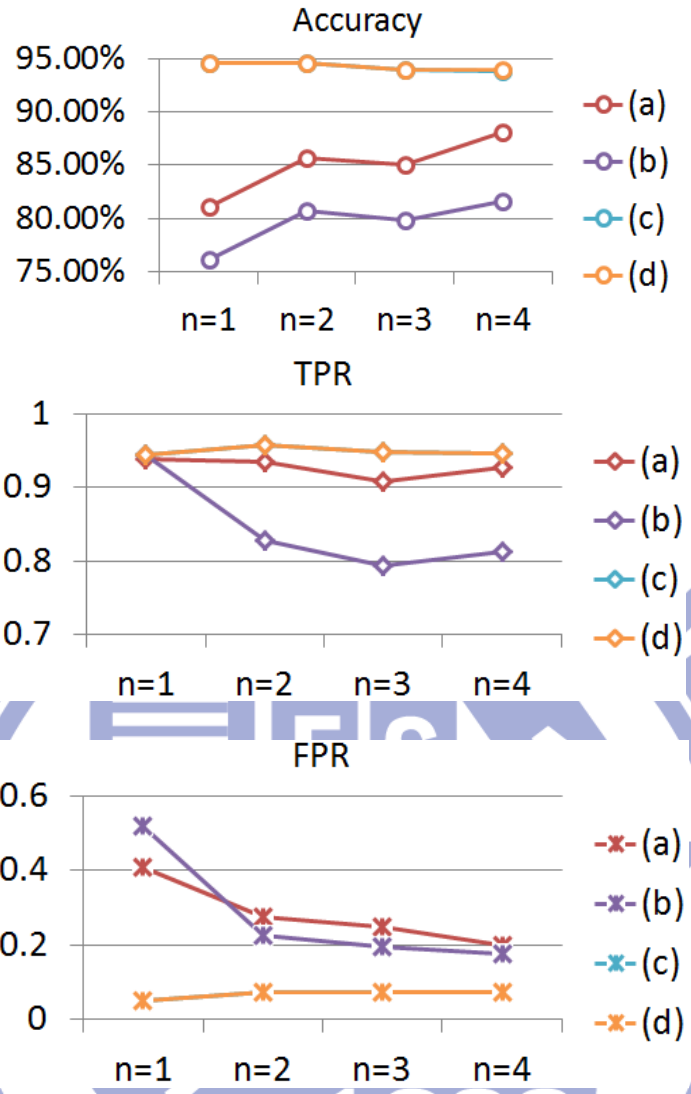


圖 42 SMO Poly 各特徵屬性走勢圖

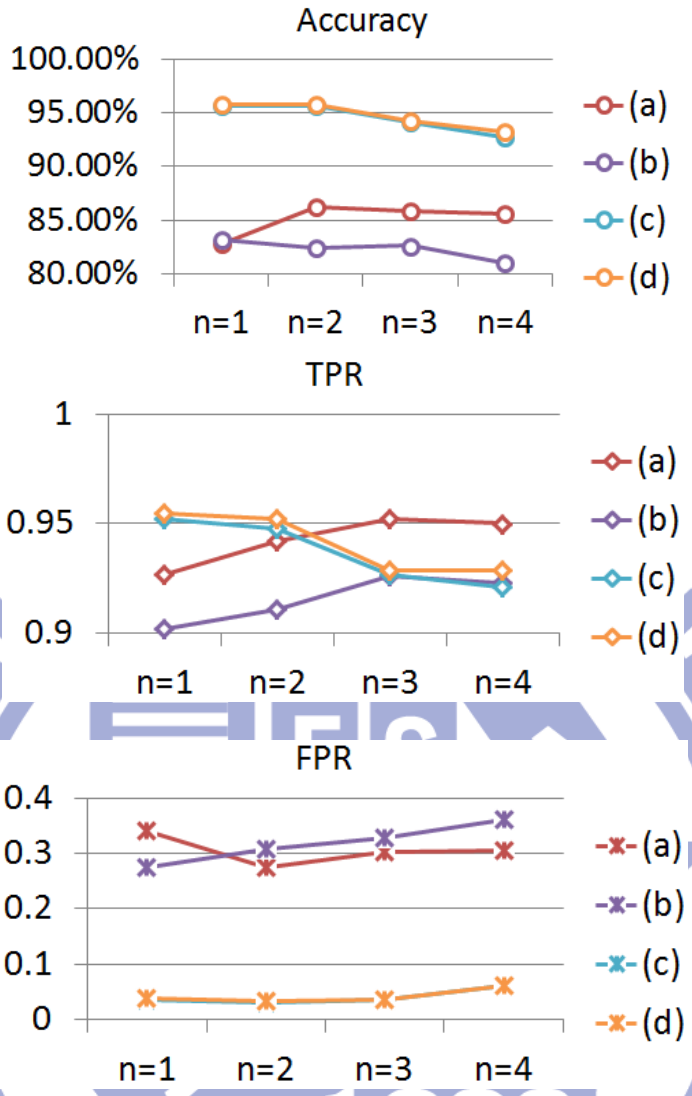


圖 43 SMO NPoly 各特徵屬性走勢圖

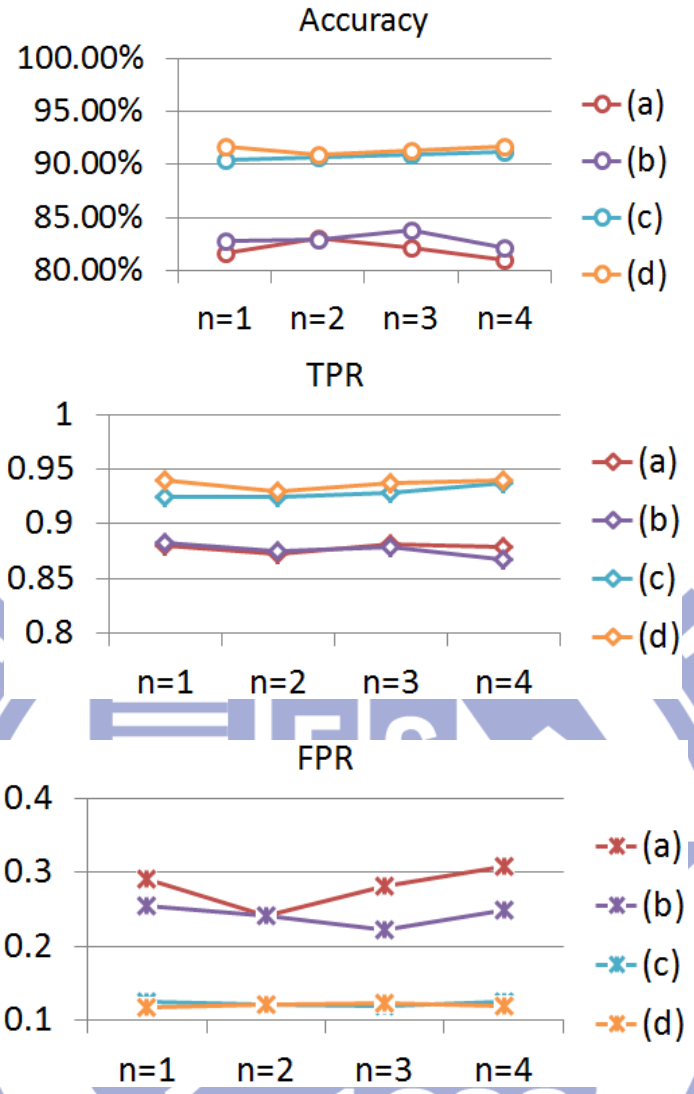


圖 44 REPTree 各特徵屬性走勢圖

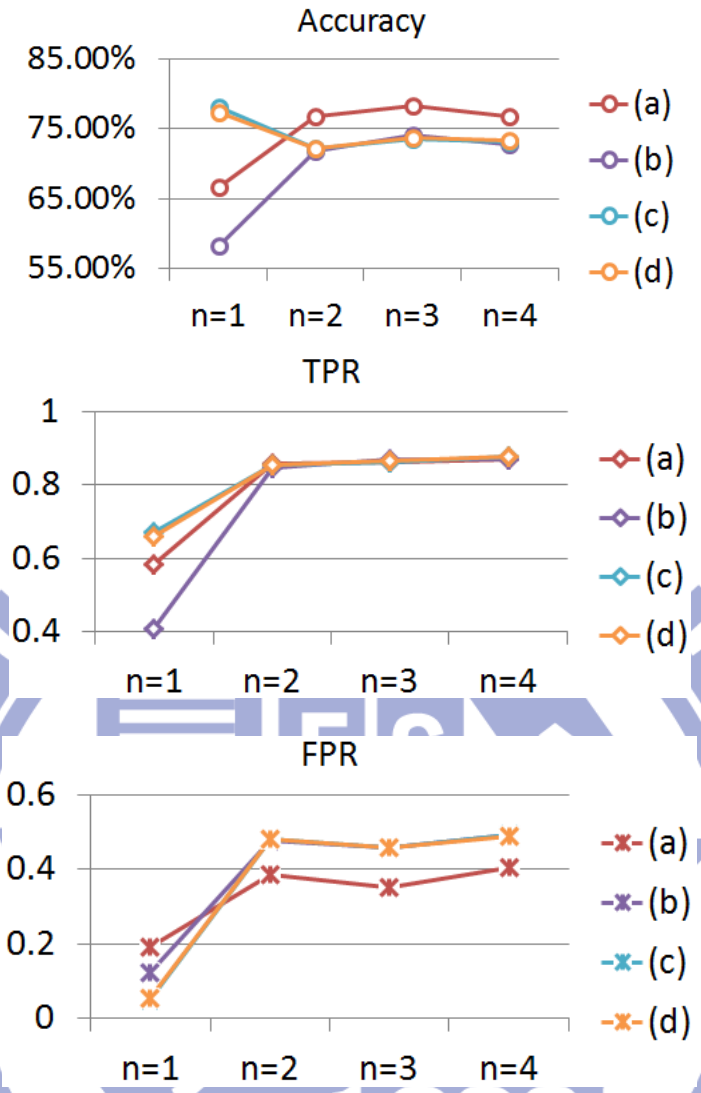


圖 45 NaiveBayes 各特徵屬性走勢圖

原先 nw、sys、與 pm 即可產生不錯的分類結果，然而卻可能因為 system call n-gram 之過多特徵屬性會影響原先之分類結果，產生較差之機器學習學習結果。不過我們可以從圖 41、圖 42、圖 43、圖 44 和圖 45，可以大致上看出其趨勢大致相同。

6.6 PCA SVM 之觀察

本論文在曾教授建議下，嘗試使用 PCA 進行降維，因為 system call n-gram 的維度相當高，當 n=3 特徵屬性數量已達 48,760，至 n=4 更高達 244,850。本論文進行降維後，發現 SVM 準確率有大幅度改善，詳細數據如表 18。

表 18 使用 PCA 前後之 SVM 結果比較表

SVM				PCA SVM		
TPR	FPR	Accuracy	Attributes	Accuracy	FPR	TPR
0.887	0.091	89.54%	permission	93.57%	0.045	0.923
0.918	0.767	64.80%	network	66.88%	0.784	0.963
1	1	63.21%	syscall n=1	83.89%	0.312	0.927
0.999	0.989	63.55%	syscall n=2	78.87%	0.279	0.828
0.822	0.573	66.57%	nw + pm	93.62%	0.042	0.922
0.994	0.918	63.46%	nw + syscall n = 1	80.19%	0.226	0.82
0.986	0.899	63.70%	nw + syscall n = 2	71.35%	0.51	0.859
0.958	0.79	66.19%	syscall n = 1 + pm	93.77%	0.046	0.927
0.887	0.782	62.25%	syscall n = 2 + pm	83.21%	0.063	0.763
0.991	0.893	64.12%	nw + syscall n = 1 + pm	93.62%	0.047	0.925
0.981	0.897	63.36%	nw + syscall n = 2 + pm	83.26%	0.063	0.764

雖然 SVM 可經由核函數將低維度線性不可分之訓練資料映射到高維度空間，以嘗試找出可分割兩分類的超平面，但實際結果就是即使使用核函數，仍會受到不重要的特徵屬性影響，因而找不到超平面；藉由 PCA 先行去除不重要的特徵屬性後，可大幅提升 SVM 準確率。

第七章 結果與討論

本論文藉由 Robotium 函式庫產生了一個新的行為觸發程式，且藉由 shell script 之輔助，本論文可以將原先只可使用於單一應用程式上的測試程式，衍生成為自動安裝並且執行大量應用程式的行為觸發器；另外本論文提出藉由 system call 使用順序產生之 system call n-gram 作為新的特徵屬性。

雖然最後證實，本論文之行為觸發器雖然於 system call 部分的確有較好之表現，但是本行為觸發器卻無法使用於網路封包資料洩漏，不過本論文也因此驗證研究者對於惡意程式行為的想法：不管使用者執行什麼行為，甚至只是安裝後不使用，惡意程式仍舊會背地裡執行惡意行為，與使用者的操作無關。

以及本論文提出之 system call n-gram，雖然相對於原先單純的 system call 次數統計，準確率的確有提高，然而當 system call n-gram 與其他特徵屬性作結合，卻會因為 system call n-gram 的 n 太大，有過多不必要的特徵屬性，甚至於影響其他特徵屬性之結果。其實我們可以藉此了解到，機器學習只要擁有少量可以準確分類的特徵屬性就夠了，過多沒有意義的特徵屬性反而會降低機器學習之準確率。

另外，本論文也藉由 SVM 降維前後之結果，同樣證實上一段的結論：機器學習只要擁有少量可以準確分類的特徵屬性就夠了，過多沒有意義的特徵屬性反而會降低機器學習之準確率。

第八章 參考文獻

[1] Smartphone OS Market Share, Q2 2014. http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[2] Android 應用程式發布方法， https://code.google.com/p/androidbmi/wiki/AndroidMarket

[3] Robotium, https://code.google.com/p/robotium/

[4] Monkey, http://developer.android.com/tools/help/monkey.html

[5] ANT, http://ant.apache.org/

[6] Android ANT, http://developer.android.com/tools/building/building-cmdline.html

[7] Y. Zhou, and X. Jiang, "Dissecting android malware: Characterization and evolution," In Proc. of the 33rd IEEE Symposium on Security and Privacy, San Francisco, USA, May 2012.

[8] A. P. Felt, et al, "A survey of mobile malware in the wild," In Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 3–14, Chicago, USA, October 2011.

[9] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "SCanDroid: Automated Security Certification of Android Applications," Technical Report CSTR-4991, Department of Computer Science, University of Maryland, College Park, USA, November 2009.

[10] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," In Proc. of the 16th ACM conference on Computer and communications security, pp. 235–245, New York, USA, November 2009.

[11] Y. Zhou, et al, "Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets," In Proc. of the 19th Network & Distributed System Security Symposium, San Diego, USA, February 2012.

[12] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," In Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 15–26, Chicago, USA, October 2011.

[13] H. Peng, et al, "Using probabilistic generative models for ranking risks of android apps," In Proc. of the 2012 ACM Conference on Computer and Communications Security, pp. 241–252, Raleigh, NC, USA, October 2012.

[14] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in android," In Proc. of the 9th International Conference on Security and Privacy in Communication Networks, Sydney, Australia, September 2013.

[15] D.-J. Wu, et al, "Droidmat: Android malware detection through manifest and API calls tracing," In Proc. of Asia Joint Conference on Information Security, pp. 62–69, Tokyo, Japan, August 2012.

[16] D. Arp, et al, “Drebin: Effective and Explainable Detection of Android Malware in Your Pocket,” In Proc. of the 2014 Network and Distributed System Security Symposium, San Diego, USA, February 2014.
[17] A. P. Felt, et al, “Android permissions demystified,” In Proc. of the 18th ACM Conference on Computer and Communications Security, pp. 627-638, Chicago, USA, October. 2011.
[18] K. W. Y. Au, et al, “PScout: Analyzing the Android Permission Specification,” In the Proc. of the 19th ACM Conference on Computer and Communications Security. October 2012.
[19] L. Lu, et al, “CHEX: statically vetting Android apps for component hijacking vulnerabilities,” In Proc. of the 2012 ACM Conference on Computer and Communications Security, 2012, pp.229-240, Raleigh, NC, USA, October 2012.
[20] Y. Zhou, and X. Jiang, “Detecting Passive Content Leaks and Pollution in Android Applications,” In Proc. of the 20th Network and Distributed System Security Symposium, San Diego, USA, February 2013.
[21] Building Android programs on the command line, http://geosoft.no/development/android.html
[22] Signing Your Applications, http://developer.android.com/tools/publishing/app-signing.html
[23] Manifest permission, http://developer.android.com/reference/android/Manifest.permission.html
[24] Google Cloud Messaging for Android (GCM), http://developer.android.com/google/gcm/gcm.html
[25] 謝維揚，“MalCatcher:以存取以及網路洩漏隱私資料行為為基礎的 Android 上惡意程式行為偵測”，國立交通大學，碩士論文，民國 102 年 6 月。
[26] android tool, http://developer.android.com/tools/projects/projects-cmdline.html
[27] Robotium 程式範例，https://code.google.com/p/robotium/wiki/Getting_Started
[28] ARFF 格式解釋，http://www.cs.waikato.ac.nz/ml/weka/arff.html
[29] 陶嘉仁，“Android 程式權限分析”，國立交通大學，碩士論文，民國 101 年 8 月