國立交通大學

電信工程研究所碩士班

碩士論文

利用權限過濾器及靜態分析之 Android 惡意軟體偵測

Two-tier Android Malware Detection with Permission Pre-filter and Static Analysis

研 究 生：姜家安

指導教授：李程輝　教授

中 華 民 國　一　佰 零 一 年 七 月

利用權限過濾器及靜態分析之 Android 惡意軟體偵測

Two-tier Android Malware Detection with Permission Pre-filter and Static Analysis

研 究 生：姜家安　　Student：Chia-An Chiang
指導教授：李程輝　　Advisor：Tsern-Huei Lee

國 立 交 通 大 學
電信工程研究所
碩 士 論 文

A Thesis
Submitted to Institute of Communications Engineering
College of Electrical Engineering and Computer Engineering
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer and Information Science

July 2012

Hsinchu, Taiwan, Republic of China

中華民國 一佰零一 年 七月

利用權限過濾器及靜態分析之 Android 惡意軟體偵測

學生：姜家安　　指導教授：李程輝

國立交通大學電信工程研究所碩士班

摘　要

Android 系統以開放的開發環境在近幾年來快速的發展。越來越多人使用 Android 系統的手持產品。現在智慧型手機的功能越來越全面，例如:網路銀行、NFC、GPS 等，都使得攻擊者有更多攻擊的模式。因為在 Google Market 上傳應用程式沒有限制所以惡意軟體的數量也隨之增加。

由於手機的效能還比不上電腦，若使用靜態分析的話對手機來講是一個負擔。所以這篇論文提出使用 Android 應用程式的權限先判斷應用程式，以減少使用靜態分析的頻率，若判斷結果是可疑的應用程式，再以靜態分析來偵測。

關鍵字：Android 權限、病毒偵測、靜態分析

Two-tier Android Malware Detection with Permission Pre-filter and Static Analysis

student：Chia-An Chiang Advisors：Prof. Tsern-Huei Lee

Institute of Communications Engineering

National Chiao Tung University

ABSTRACT

Android operation system has advanced rapidly through open develop environment in recently years. More and more people use Android operation system's mobile devices. The functionality of smartphone has became more comprehensive, ex: cyberbank, NFC, GPS, so that attacker has more different ways to attack the end user. Due to the unrestricted access of uploading application to Google Market, there is a noticeable increase in the number of Malware.

A cell phone has limited capacity comparing to a computer; as a result, the usage of a static analysis may overload a cell phone device. This thesis suggests an approach to efficient detection of malwares: the first part of detection involves catching malicious applications by inspecting the applicants' permissions, which reduces the need for a static analysis; a further static analysis is only needed if any application is identified as being suspicious.

Keywords: Android, permission, static analysis, malware detection, control flow graph

誌　　謝

能完成這篇論文，首先我要感謝我的指導教授－李程輝教授。他總是不辭辛勞地教導著我作研究的態度，在這兩年的研究所生活中，我不僅學習到專業知識還有獨立思考的能力，更重要的是我從老師身上學到了對研究以及做事的正確態度。這些處理事情的態度對我未來工作上會有很大的幫助。

感謝我的父母－姜永根先生與張碧玉女士。感謝父母一路上對我的鼓勵以及對我的養育之恩，讓我在求學生涯裡不需要為金錢煩憂。

感謝我的朋友，在我最焦慮徬徨無助時，給我建議和鼓勵，陪我紓解壓力，讓我不會被壓力給吞沒。

感謝實驗室的各位，在我研究的時候給我建議，感謝學長的指導，感謝一起跟我奮鬥的嗣儒，感謝陪我聊天解惑的冠佑，感謝給我很多意見和幫我一起找程式錯誤的廣煜、鐙標學弟，感謝其他我沒有提到的實驗室夥伴，陪我度過這兩年研究生的生活。

感謝嘉琦不厭其煩的幫我訂正論文及給我建議。

最後謹將此論文獻給身邊所有愛我的人及我愛的人。

2012/07　姜家安

目　　錄

圖　目　錄

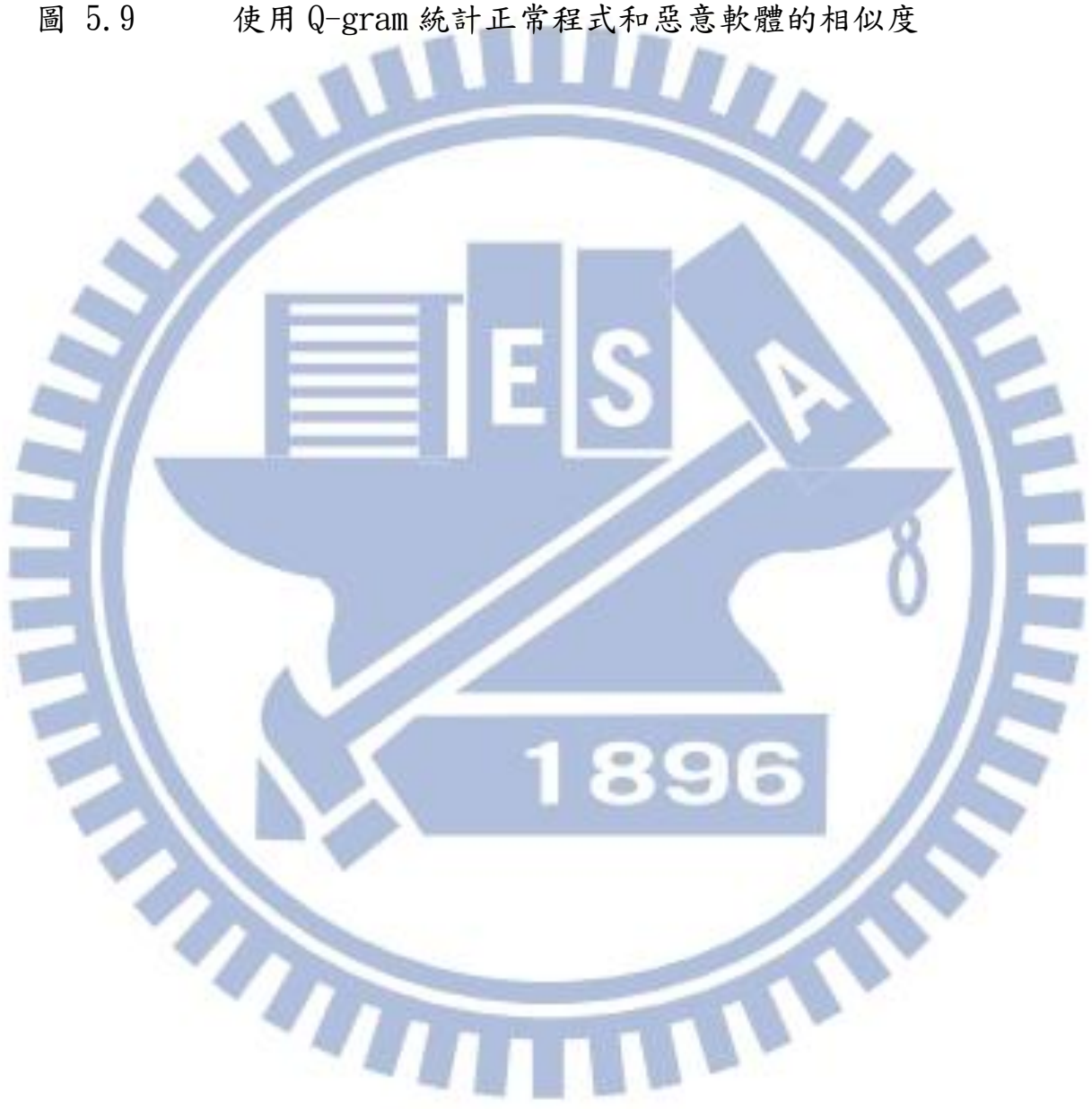


表　目　錄

第一章

簡介

1.1 研究背景與動機

由於通訊網路技術和微處理器的進步，智慧型手機在近幾年來已成為非常熱門的話題。甚至有很多人認為智慧型手機是生活中不可或缺的東西。各家大廠都意識到這是一個龐大的商機，因此推出了各自的智慧型手機的作業系統。第一支搭載 Symbian 的手機是由 Ericsson 在 2000 年所推出。Symbian 作業系統較早推出，所以銷售量也領先其他的作業系統。根據 Gartner 的統計[1]，在 2010 年第 3 季，搭載 Symbian 作業系統的手機的銷售量佔所有智慧型手機市場的 36.3%，Android 佔 25.3，RIM 佔 15.4，而 iOS 佔 16.6%。然而 Symbian 作業系統對新興的社群網路和 web 2.0 內容支援欠佳，Symbian 的市場佔有率日益萎縮。2009 年到 2011 年，Nokia、Motorola、Samsung、LG 和 Sony Ericsson 等廠商宣布停止生產 Symbian 的手機，轉而生產 Google 的 Android、Microsoft 的 Windows Phone 和 Samsung 的 bada[2]。

Google 在 2005 年 8 月 17 日收購 Android 公司，Android 成為了 Google 的子公司。在 2007 年 10 月 5 日，由 Google 主導成立開放手持設備聯盟(Open Handset Alliance)[3]共同開發 Android 作業系統。市場上第一款採用 Android 作業系統的智慧型手機是 HTC 生產的 T-Mobile G1，於 2008 年 10 月 22 日在美國上市。由於 Android 作業系統是完全免費開源的，任何廠商都可以不經過 Google 和開放手持設備聯盟的授權隨意使用

Android 作業系統。所以有很多的手機製造商選擇 Android 來當手機的作業系統。根據圖 1.1，在 2011 年第 3 季 Android 佔所有智慧型手機銷售量的 52.5%，是前一年的兩倍。

Operating System	3Q11 Units	3Q11 Market Share (%)	3Q10 Units	3Q10 Market Share (%)
Android	60,490.4	52.5	20,544.0	25.3
Symbian	19,500.1	16.9	29,480.1	36.3
iOS	17,295.3	15.0	13,484.4	16.6
Research In Motion	12,701.1	11.0	12,508.3	15.4
Bada	2,478.5	2.2	920.6	1.1
Microsoft	1,701.9	1.5	2,203.9	2.7
Others	1,018.1	0.9	1,991.3	2.5
Total	**115,185.4**	**100**	**81,132.6**	**100**

圖 1.1 智慧型手機的市佔率

最有效減少病毒的方法就是只有經過認證的應用程式才可以被使用者下載並且安裝。Apple 的智慧型手機作業系統是 iOS，官方的 App Store 提供 iOS 用戶應用程式的下載。開發者在上傳應用程式後，並不會馬上上架，而是經過一連串的檢查，測試人員沒有發現程式有惡意行為或沒有程式錯誤才會讓這個應用程式上架。反觀 Android 作業系統，官方的 Android Market 提供 Android 用戶應用程式的下載，但是沒有提供應用程式源碼(source code)的檢測。開發者只要花 25 元美金就可以申請一個 Android 程式開發者帳號，一旦申請好帳號就可以無限制的上傳應用程式。開發者上傳應用程式之後 Android Market 並不會去檢查這個應用程式有沒有惡意行為，會直接讓這個應用程式上架。但是 Android 應用程式可以執行的某些事情對使用者來說太危險了。舉例來說，應

用程式可以在使用者不知情的情況下打電話或寄簡訊給某個電話號碼，並不會有使用者介面去取得使用者的認可或通知使用者。

由於使用 Android 作業系統的使用者越來越多，而且 Android Market 上並不會去檢查開發者上傳的應用程式，Android 作業系統變成病毒攻擊的主要目標。如圖 1.2[5]，Lookout 統計 2011 一月到 2011 年七月 Android Market 上惡意軟體的數量。我們可以發現惡意軟體的數量正在迅速增加當中。

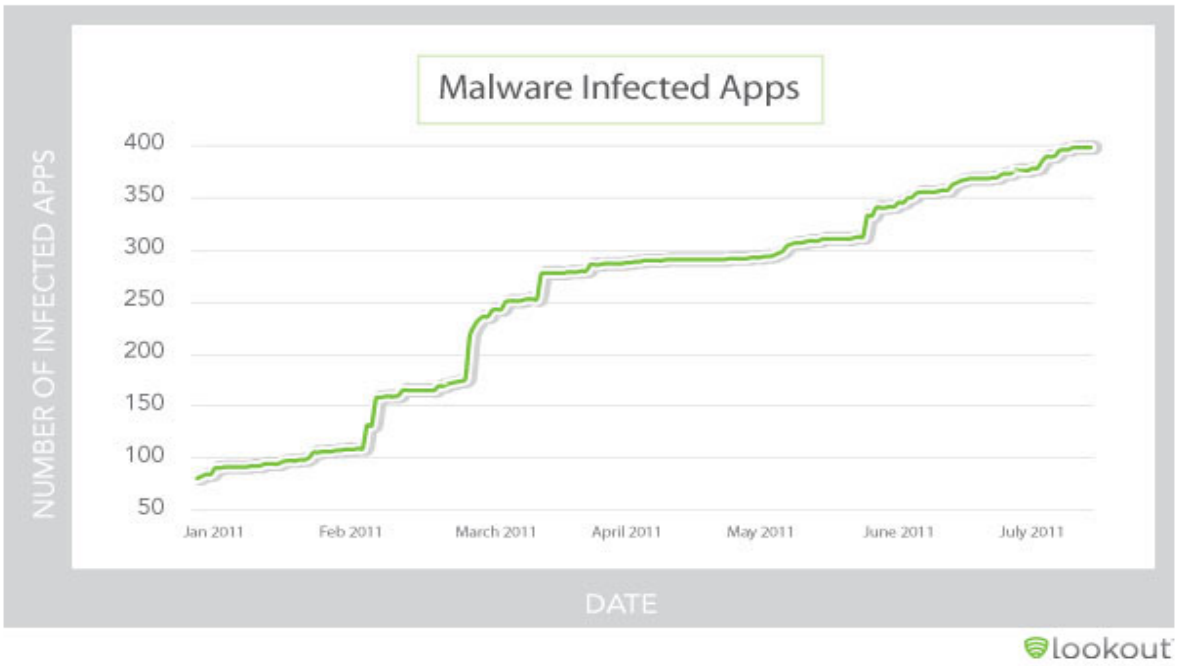


圖 1.2 受感染的應用程式的數量

本篇論文針對 Android 作業系統主要原因是因為(1)一半以上的智慧型手機都是使用 Android 作業系統(2)Android Market 沒有提供應用程式源碼檢查，程式開發者可以任意的上傳應用程式，使得 Android Market 上的惡意軟體數量越來越多(3)Android 作業系統是開放原始碼(open source)。

1.2 背景知識介紹

1.2.1 Android

Android系統是基於Linux核心的軟體平台和作業系統，如圖1.3。這個作業系統和以往由Apple、Microsoft等大廠最大不一樣的地方在於它開放原始碼，讓一般人可以輕易地利用SDK(Software Development Kit)開發各式各樣的軟體，另外也結合了Google所提供的各項服務功能，使得Android作業系統的使用者在近幾年來迅速的增加[6]。

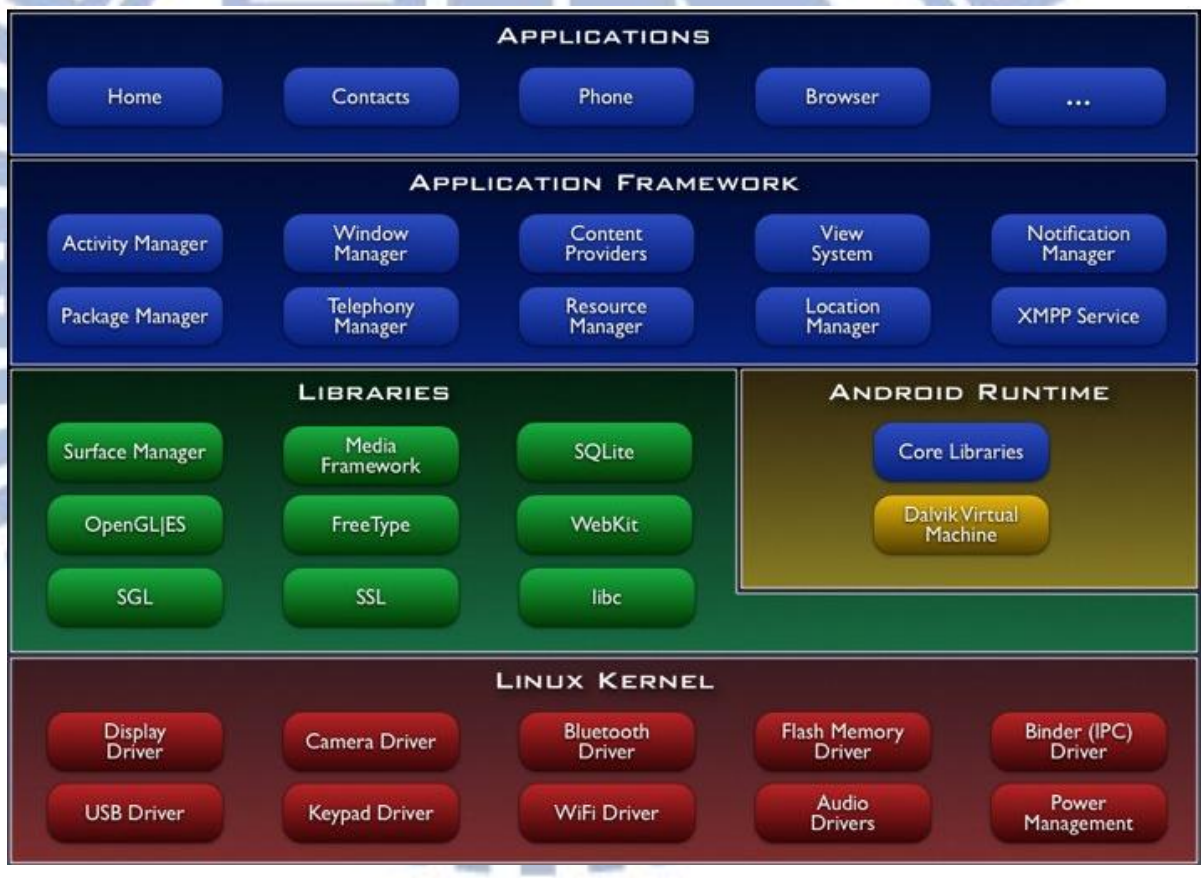


圖 1.3 Android 作業系統架構

通常 Android 的應用程式都是由 Java 程式語言來開發，接著在編譯成字節碼(bytecode)，傳統的 Java 虛擬機器(Virtual Machine)是一種堆疊機器(stack machine)而 Dalvik 虛擬機器則是基於暫存器(register)的架構。Dalvik 是 Android 系統的進程虛擬機器(Process Virtual Machine)，它可以運行已轉換為.dex(Dalvik Executable)格式的檔案，.dex 檔案是專門為 Dalvik 設計的一種壓縮格式，適合內存和記憶體有限的系統[7]。

一個 Android 的應用程式是由很多個元件(component)所組成。每一個元件對系統而言都是一個進入應用程式的點。總共有四種不同形式的元件。每一種元件的功能都不一樣，而且有自己的生命周期(lifecycle)，生命週期定義該種元件何時將被創造或關閉。

Activities:

一個 Activity 代表一個畫面，這個畫面有可能是要給使用者下指令、和使用者溝通或是呈現某些東西給使用者看，一個應用程式可能會有很多個 Activity。例如:一個網路銀行的應用程式會有一個 Activity 給使用者登入帳號，也可能會有另一個 Activity 提供使用者轉帳的介面，還會另有一個 Activity 是用來顯示這個帳號還有剩多少錢。每一個 Activity 是彼此獨立的，而且在同一個時間也只會有一個 Activity 呈現在螢幕上。

Services:

Service 就像沒有介面的 Activity，是可以長時間在背景運行的。但是系統一次只能顯示一個 Activity，當有新的 Activity 要執行時，目前這個 Activity 就會先被暫停。然而系統一次可以執行很多個 Service，這些 Service 在背景中運作，並不會有使用者介面。例如:一個 Service 有可能是一個音樂撥放器，當使用者在執行其他應用程式時，這個 Service 可以在背景撥放音樂。其他的元件也可以啟動 Service 或對 Service 下達指令，程式開發者就可以藉由這樣把使用者的資料上傳去遠端的伺服器。

Content Providers:

在Android系統裡，應用程式和應用程式之間是無法直接分享資料的，為了解決應用程式之間彼此要分享資料的需求，Android提供了Content Provider這個機制。Content Provider就像一個資料庫一樣，可以提供其他應用程式使用。假如開發者沒有需要分享資料給其他應用程式的話就不需要用到Content Provider。

Broadcast Receiver:

Broadcast receiver 是負責接收廣播的訊息。例如:系統在開機後會有系統啟動完成的廣播,應用程式可以用一個Broadcast Receiver去接收這項訊息,然後啟動Service來執行想做的事情。大部分的廣播都來自系統本身，像是開機完成(boot completed)、低電量(battery low)等，開發者也可以自己定義想要廣播的事件。

Android 系統並沒有規定一個應用程式要有幾個元件。 每一個元件都應該被宣告在清單檔案(manifest file)裡面。 每一個 Android 的應用程式都會有一個清單檔案，這個檔案就像清單一樣登記著此檔案的資訊。例如:一個應用程式有三個 Activity、一個 Service 和使用 INTERNET 權限(稍後介紹),那麼程式開發者就必須把這些東西一一的列在清單檔案裡面。

在 Android 作業系統中，每一個應用程式都會有自己的 user ID 即在自己的程序中執行。這樣做的好處是可以保護系統及每一個應用程式，不會被其他不正常的應用程式所影響。可以把每一個程序想像成是一個個的黑盒子，有自己的記憶體，彼此之間不會互相影響。然而應用程式和系統或其他應用程式之間還是有可能需要互相分享資訊或資源,為了要讓不同程序之間可以互通,Android 作業系統使用權限(permission)來達成。程式開發者必須在應用程式的清單檔案(manifest file)中宣告需要使用那些權限。一

旦這個應用程式安裝完成以後，他的權限就固定了不能再增加或刪除，除非解除安裝這個應用程式。

Android 作業系統中的權限有兩種功能:

(1) Android 作業系統中每一個應用程式都在不同的程序中執行，但是也可以藉由定義權限的方式讓其他的應用程式使用組件。例如:開發者可以為一個組件定義一個權限，其他的應用程式就可以透過進程間通訊(inter process communication, IPC)來和這個組件溝通。

(2) 權限也可以用來鎖住某些特定的應用程式設定介面(application programming interface, API)，也就是說當開發者需要使用到那些鎖住的應用程式設定介面時，就必須在清單檔案中宣告權限。例如:使用者要使用到網路相關的應用程式介面時就必須在清單檔案中宣告 android.permission.INTERNET 這個權限。

程式開發者可以依自己的需求定義權限。Android 作業系統中已經定義好很多系統權限(standard permission)，這些權限都是跟手機的軟體、硬體資源相關。例如:可以使手機震動、可以讀取手機的位置、可以收發簡訊等等。這些系統權限對手機安全的威脅都不一樣，像是設定桌布的權限跟寄簡訊的權限比起來威脅度就比較小。所以 Google 定義了保護等級(protection level)，由於系統權限是 Google 定的，所以每一個系統權限的保護等級已經被定義好了。程式開發者可以選擇自己定義的權限是甚麼等級，假如使用者沒有定義保護等級的話，系統會自動給定 normal 等級[8]。

權限等級(protection level)總共分為四類:

(1) Normal:

被歸類在 Normal 等級的權限對系統或其他應用程式而言不會有太大的威脅。像是使手機震動、開啟照相機的閃光燈、或者是讀取電池的電量資訊等等。當使用者要安裝應用程式的時候，手機會提醒使用者這個應用程式需要用到哪些系統權限。Normal 等級的權限並不會主動顯示給使用者知道，而是提供一個下拉式選單，當使用者點選時才會顯示這個應用程式使用了哪些 Normal 等級的權限。

(2) Dangerous:

Dangerous 等級的權限對系統安全的威脅較大。這個等級的權限可以允許應用程式做可能影響系統安全的事情。合法的應用程式用這些權限去做正當的事情，但是惡意軟體也有可能用這個等級的權限去讀取使用者的隱私資料或是造成使用者的額外花費。例如:應用程式有 READ_PHONE_STATE 權限的話就可以去讀取手機的資料像是電話號碼或是讀取國際移動設備辨識碼(International Mobile Equipment Identity number)。因為這個等級的權限有潛在性的危險，所以在安裝的時候系統會顯示此應用程式需要使用那些 Dangerous 等級的系統權限讓使用者知道。

(3) Signature:

所有的 Android 應用程式都會有一個憑證，這個憑證是用來辨識開發者是否相同。開發者在撰寫好應用程式後必須使用私人鑰匙來產生出 apk(Android Package)檔，系統會根據那把鑰匙來產生這個應用程式的憑證，如果鑰匙一樣就會產生出相同的憑證。當應用程式要使用這個等級的權限，系統會檢查應用程式的憑證和定義這個權限的應用程式有沒有相同，如果憑證一樣系統不會通知使用者，而自動允許使用這個權限。

(4) SignatureOrSystem:

SignatureOrSystem 這個等級的權限除了跟 Signature 等級的權限一樣需要有相同的憑證才可以使用以外，還多加了一項條件，只要是在 system/app 這個資料夾裡面的

應用程式都可以使用這個等級的權限。但是，system/app 這個資料夾被系統保護，一般的程式開發者無法擅自將應用程式安裝在這個資料夾底下。

1.2.2 控制流程

傳統的病毒偵測系統使用一個或多個固定的字串來代表一個病毒。這些字串可能是用 MD5 或是雜湊函數(Hash function)來產生的。如果要比對的檔案裡面有跟資料庫一樣的字串就判斷成病毒。但是傳統病毒偵測的方法在面對多型的病毒變種(polymorphic malware variant)偵測效率會很差[12]。例如:在程式碼中修改一下變數的名字或者是把程式中的 for 迴圈改成 while 迴圈。經過上述的更改，程式片段經過雜湊函數的字串結果將會不一樣，但是程式片段的功能還是一樣。由於字串不一樣病毒偵測系統將不會認為修改後的病毒和原本的病毒一樣。

控制流程(control flow)就是被用來克服這種程式碼改變但是程式的架構或語意沒有改變的情形。控制流程代表一個程式可能執行的路徑主要有兩種形式來表示。一個是控制流程圖(control flow graph)另一個是函數調用關係圖(call graph)。將程式中函數可能的執行路徑都分析並且記錄，該結果就是控制流程圖。分析程式中函數的呼叫關係並記錄，該結果就是函數調用關係圖。圖 1.4 和圖 1.5 描述控制流程圖和函數調用關係圖的不同。

```
if (x > y){
    max = x;
}
else{
    max = y;
}
return max;
```

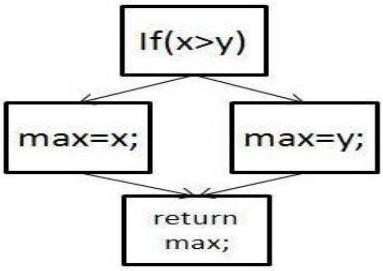


圖 1.4 控制流程圖的例子

```
int main(int state){
   start(state);
}
void start(int number){
   if(number){
      number = number  1;
      start(number);
   }
   else{
      wait()
   }
}
void wait(){

}
```

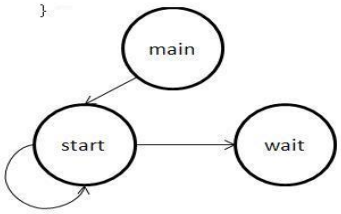


圖 1.5 函數調用關係圖的例子

1.2.3 手機病毒的威脅

就像個人電腦一樣，手機上有很多威脅可能會影響到系統或使用者。但是手機跟電腦不一樣的地方是手機有一些電腦沒有的功能，例如打電話或傳簡訊等。使得現在手機上的攻擊方式和以前電腦上攻擊方式不一樣。在以前很多攻擊者攻擊的目的只是為了名

譽，但是現在手機上的攻擊者主要都是為了竊取使用者的隱私資料或者是從使用者身上賺取金錢。在[5]這份報告中將基於應用程式的威脅分為三類:

1） 惡意軟體:

惡意軟體是想要在使用者的手機上做一些惡意行為。以 TrojanSMS 為例，這是第一個被發現經由寄簡訊而賺取獲利的惡意軟體。首先，攻擊者先跟電信公司申請一個高費率的號碼，只要有人打電話或傳簡訊到這個號碼，電信公司就會跟申請人分錢。接著攻擊者在開發一個惡意軟體，這個惡意軟體會寄簡訊到攻擊者申請的號碼。最後再把這個惡意軟體上傳去 Android Market，讓使用者下載安裝到這個應用程式，一旦使用者安裝了以後，這個惡意軟體會在使用者不知道的情況下寄簡訊到那個號碼。

2） 間諜軟體:

間諜軟體主要是在使用者不知道的情況下來竊取使用者的隱私資料。這些隱私資料包括簡訊內容、位置資訊、電話簿、手機號碼等等。Android.Tapsnake[9]就是一個例子。這個應用程式表面上看起來就只是一個很普通的貪食蛇遊戲。但是這個應用程式有個 Service 在背景中運作，並且每 15 分鐘就會上傳使用者的位置給遠端的伺服器。

3） 攻擊系統漏洞的應用程式:

攻擊者也會藉由系統的漏洞來攻擊。但是這些漏洞通常會在新的系統版本中被修復。以 z4root 為例，這個應用程式是藉由 Linux 的漏洞 rage against the cage，來取的管理員的權限。取得管理員權限的手機存在著很大的風險，惡意軟體可以藉此做更多危險的事情。但是這個漏洞在 Android 2.2 版本以後已經被修復了。Android 2.2 版本之後這個應用程式已經沒辦法去取得管理權限了。

1.3 論文架構

在第二章我們介紹相關的研究。2.1 節中我們介紹使用權限的惡意軟體測，接著在 2.2 節介紹靜態分析。在第三章主要介紹我們提出的系統架構。3.1 節中討論我們想解決的問題，3.2 節中說明使用權限來偵測惡意軟體的優缺點，3.3 節中詳細的說明我們的想法和提出的系統架構，3.4 節中說明惡意軟體偵測系統是怎麼運作的。在第四章中說明權限過濾器和靜態分析。4.1 節中說明我們的想法和權限過濾器的安全規則，4.2 節中說明 MethSI 怎麼比對惡意軟體。在第五章顯示模擬的結果，5.1 節中比較權限過濾器和 Kirin 的安全規則，5.2 節比較 MethSI 和 Q-gram 的時間和偵測能力。最後我們在第六章做結論。

第二章

相關研究

2.1 基於權限的惡意軟體檢測

2.1.1 Kirin

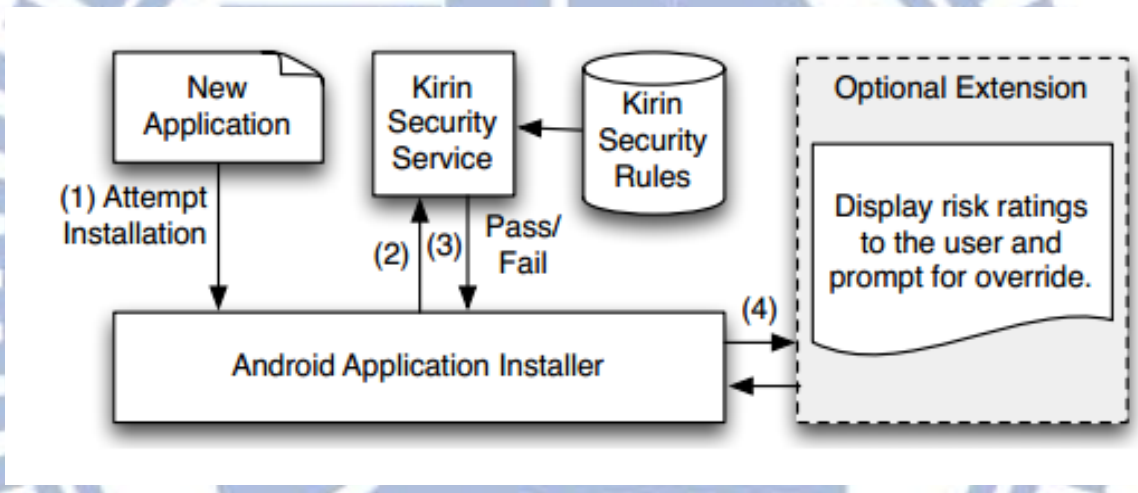


圖 2.1 基於 Kirin 的軟體安裝器

圖 6 描述基於 Kirin 的軟體安裝器[11]。首先，基於 Kirin 的軟體安裝器先檢查應用程式使用了哪些系統權限，看有沒有符合 Kirin 安全規則。假如通過的話就直接安裝應用程式，沒有通過的話軟體安裝器可以有兩種設計，軟體安裝器可以直接拒絕安裝這個應用程式，或者是提供一個使用者介面詢問使用者是否要安裝這個應用程式。

Kirin 安全規則定義如下:

(1) 應用程式不能有 SET_DUGBUG_APP 這個權限。

(2) 應用程式不能有 PHONE_STATE、RECORD_AUDIO 和 INTERNET 這三個權限。

(3) 應用程式不能有 PROCESSOUTGOING_CALL、 RECORD_AUDIO 和 INTERNET 這三個權限。

(4) 應用程式不能有 ACCESS_FINE_LOCATION 和 RECEIVE_BOOT_COMPLETE 這兩個權限。

(5) 應用程式不能有 ACCESS_COARSE_LOCATION、INTERNET 和 RECEIVE_BOOT_COMPLETE 這三個權限。

(6) 應用程式不能有 RECEIVE_SMS 和 WRITE_SMS 這兩個權限。

(7) 應用程式不能有 SEND_SMS 和 WRITE_SMS 這兩個權限。

(8) 應用程式不能有 INSTALL_SHORTCUT 和 UNINSTALL_SHORTCUT 這兩個權限。

(9) 應用程式不能有 SET_PREFERRED_APPLICATION 權限和接收 CALL 的 intent。

2.1.2 ASESD

另一個跟 Android 的權限相關的研究是 Android Security Enforcement with a Security Distance model (ASESD)[12]。跟 Kirin 一樣主要是想要使用 Android 的權限來讓降低使用者安裝到惡意軟體的機會。

ASESD 定義了四種安全距離(security distance, SD):

-Safe SD 表示這兩個權限的組合是安全的，Safe SD 的威脅分數是 0。

-Normal SD 表示這兩個權限的組合沒有明顯的威脅，Normal SD 的威脅分數是 1。

-Dangerous SD 表示這兩個權限的組合可能會有危險，Dangerous SD 的威脅分數是 5。

-Severe dangerous SD 表示這兩個權限的組合對系統而言非常危險，Severe dangerous SD 的威脅分數是 25。

ASESD 檢查應用程式的清單檔案，把這個應用程式用到的所有權限記錄下來。接著運用剛剛定義的安全距離，算出兩兩權限的安全距離，再用下列公式算出這個應用程式的風險。

$$R = (\sum d_c + \sum d_{ij} \times d_{jk}) \times G$$

在[12]中，作者認為一個應用程式被惡意軟體感染以後會使用更多的權限，來達成那些惡意行為。兩個權限的組合越危險分數也越高，最後算出這個應用程式的風險度，風險度越高代表這個應用程式越有可能是惡意軟體。

2.2 靜態分析方法

在[10]中，作者提出先把一個程式經由控制流程圖來分析，接著把這控制流程圖經過結構演算法，把控制流程圖轉換成結構形式，如圖 2.2 的第一步到第二步。接著根據這個程式的結構形式藉由圖 2.3 的文法將結構形式轉換成一個字串，如圖 2.2 的第二步到第三步。轉換成字串以後再把字串藉由 Q-Gram 的方式把字串轉換成代表這個程式的特性向量(feature vector)，以特性向量來分析程式之間的相似度。此方法在資料庫中存放代表病毒的特性向量，當一個要詢問的程式近來此系統時，先將這個程式轉換成代

表這個程式的特性向量，在跟資料庫中的特性向量比對相似度，當相似度大於臨界值（threshold），就將詢問的程式判斷為病毒。

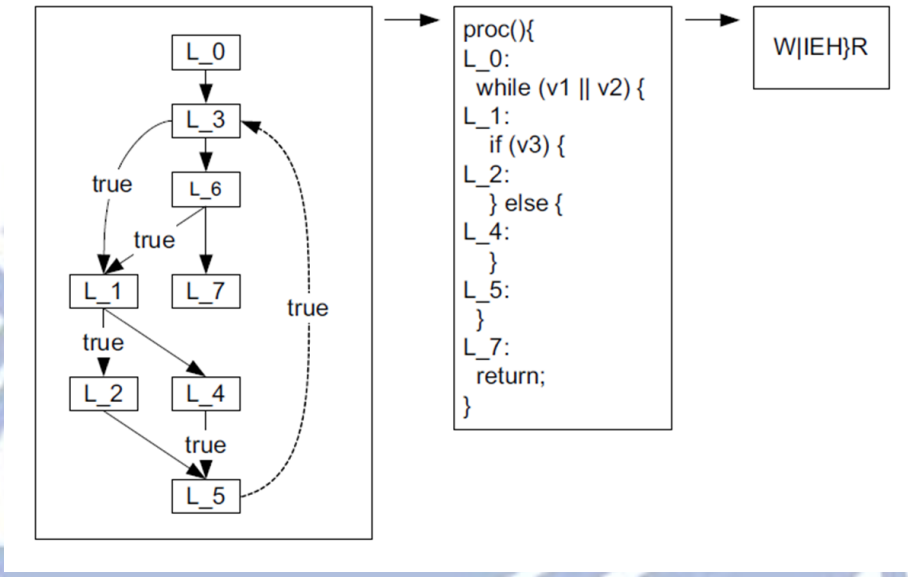


圖 2.2 控制流程圖、結構圖和字串表示的關係

Procedure	::= StatementList
StatementList	::= Statement | Statement StatementList
Statement	::= Return | Break | Continue | Goto | Conditional | Loop | BasicBlock
Goto	::= 'G'
Return	::= 'R'
Break	::= 'K'
Continue	::= 'C'
BasicBlock	::= SubRoutineList
SubRoutineList	::= 'S' | 'S' SubRoutineList
Condition	::= | ConditionTerm | ConditionTerm NextConditionTerm
NextConditionTerm	::= '!' Condition | Condition
ConditionTerm	::= '&' | '|'
IfThenCondition	::= Condition | '!' Condition
Conditional	::= IfThen | IfThenElse
IfThen	::= 'I' IfThenCondition StatementList 'H'
IfThenElse	::= 'I' IfThenCondition StatementList 'E' StatementList 'H'
Loop	::= PreTestedLoop | PostTestedLoop | EndlessLoop
PreTestedLoop	::= 'W' Condition 'StatementList '}'
PostTestedLoop	::= 'D' StatementList '}' Condition
EndlessLoop	::= 'F' StatementList '}'

圖 2.3 字串結構的文法

作者使用了一個叫做 Q-Gram 的方法將字串轉換成特性向量。一個 q-gram 特性表示在代表程式的字串中一個 q 長度的字串。每一個可能的 q 長度的字串代表一個特性。為了要建立代表程式的特性向量，所以要先選擇基底。作者從好的程式和壞的程式中選出最多出現 500 次的特性當作基底。接著統計每一個基底代表一個程式結構的字串中總共出現的次數，來代表那一個分量的值。例如:在一個字串中第一個基底總共出現 5 次，那第一個分量的值就是 5。所以就可以把一個字串轉換成一個特性向量。

兩個特性向量之間的距離:

$$d(p,q) = \| p - q \|_1 = \sum_{i=1}^{n} | p_i - q_i |$$

資料庫中的 p 向量和詢問的向量 q 之間的相似度:

$$sim(p,q) = 1 - \frac{d(p,q)}{|q|}$$

兩個程式轉換成字串，這兩個字串越像的話代表這兩個程式的結構越像，也代表這兩個程式的行為越像。所以這個相似度的公式代表兩個字串越像的話兩個向量的距離也會越小，所以相似度會越大。

第三章

問題描述與系統架構

3.1 問題描述

智慧型手機會迅速竄起的一個很重要的原因是應用程式服務。不像以前電腦的應用程式幾乎都是廠商在開發，Android 的程式開發者可以用開放且免費的軟體開發套件來開發應用程式。使得應用程式多元又創新，從遊戲到記帳軟體各式各樣的應用程式都有。手機使用者對應用程式的依賴性也越來越高，像是以前都還要拿張便條紙來記錄事情，現在可以將待辦事情記在手機的應用程式，還可以定時去提醒使用者。

根據[13]，官方的 Android Market 統計在 2011 年底總下載次數已經超過一百億次。值得注意的是在 2011 年 7 月，總下載次數大約六十億，但是在接下來的六個月總下載次數增加了四十億，如圖 3.1 所示。主要原因是 Android 系統的使用者快速的增加而且使用者對對應用程式的依賴性越來越大。但是在開發者在把應用程式上架的時候，Android Market 並不會去做程式源碼的檢測，這使得 Android Market 成為了攻擊者散播病毒的最佳途徑。

由於上述的原因，使得基於應用程式的攻擊方式很盛行。本篇論文針對使用者從 Android Market 或者是第三方來源取得的應用程式進行檢測。我們提出了新的架構和方法來偵測惡意軟體。

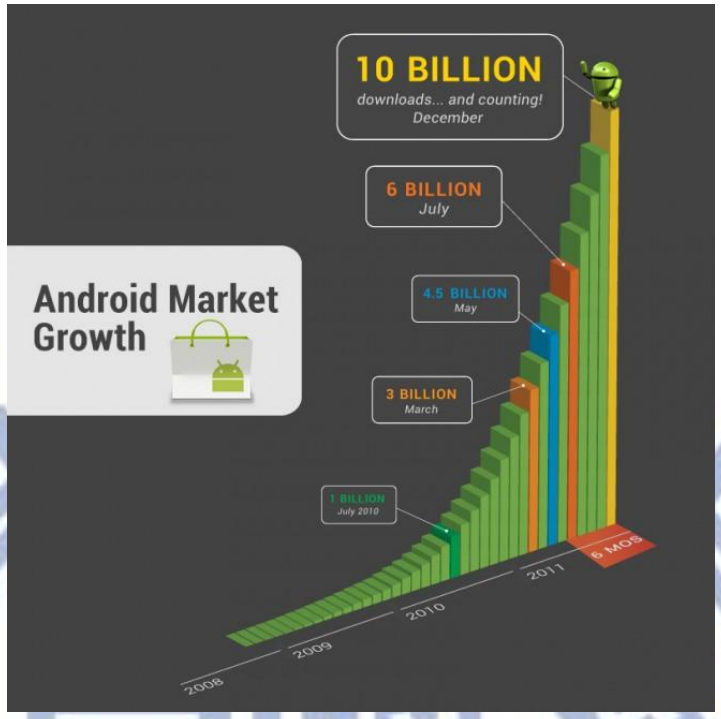


圖 3.1 Android Market 下載量

3.2 使用權限偵測惡意軟體的優點及缺點

權限是 Android 系統特有的架構。如果一個應用程式需要使用到網路的功能就必須宣告網路的權限，如果一個應用程式需要寄簡訊就必須要宣告寄簡訊的權限。看一個應用程式宣告的權限可以知道這個應用程式需要用到手機的哪些功能。使用權限來偵測病毒只需要在安裝應用程式的時候去讀取應用程式的清單檔案，並且把宣告的權限記錄下來，接著做些簡單的判斷像是在[11]中運用規則去判斷或是在[12]中去計算這些權限的風險度來判斷惡意軟體。這樣做的好處是程式很小而且檢查的速度很快。

但是只從清單檔案檢查應用程式的權限的話也有缺點。以一個非常有名的即時通訊應用程式 WhatsApp 為例子。這個應用程式需要用到很多權限如:READ_PHONE_STATE、

INTERNET 、 RECORD_AUDIO 、 RECEIVE_SMS 和 SEND_SMS 等 等 。 WhatsApp 有 READ_PHONE_STATE、INTERNET 和 RECORD_AUDIO 會違反 Kirin 的二條安全規則。而且 WhatsApp 有使用到很多惡意軟體可能會用到的權限，在 ASESD 中認為 RECEIVE_SMS 和 SEND_SMS 這兩個權限的組合是 Severe dangerous SD，所以若使用 ASESD 來判斷 WhatsApp 的話風險度會很高。這兩種方法都會把 WhatsApp 判斷為惡意軟體。

另一方面，攻擊者可以開發一個簡單的惡意軟體，表面上看起來是一個音樂撥放器，但是實際上這個應用程式會每間隔一段時間就寄簡訊到已經申請好的高費率號碼。這個應用程式只需要宣告 SEND_SMS 這個權限在清單檔案裡面，Kirin 和 ASESD 都會認為這個應用程式對系統或使用者不會造成威脅。

3.3 系統架構

在新的系統架構之下，應該要有新的惡意軟體檢測方式，而不是使用舊有電腦上或者是其他手機系統中的惡意軟體偵測系統。Android 系統中提供了權限的框架，任何應用程式都必須要宣告權限才可以去使用想要得到的功能。我們可以從權限中獲得應用程式的安全資訊。在這一節將介紹我們提出的惡意軟體檢測的系統架構。

在 Android 的系統中，當使用者要安裝應用程式的時候 Android 作業系統會顯示這個應用程式使用了那些 Normal、Dangerous 等級的權限。主要目的是希望使用者可以根據自己的認知來判斷是否要安裝這個應用程式。例如:有一個應用程式是音樂播放器，但是在清單檔案中宣告了寄簡訊的權限，使用者就可以合理的懷疑這個應用程式，並且不安裝此應用程式。但是一般的使用者並沒有這種知識來判斷是否要安裝應用程式。使用者可能覺得這個應用程式的畫面很漂亮或是功能很吸引人，就去下載安裝該應用程式而不管它需要用到甚麼權限。那麼權限的架構對使用者來說就等於沒有一樣。所以開始

有一些研究是使用 Android 的權限來判斷惡意軟體。因為我們可以使用應用程式的權限初步判斷這個應用程式所提供的服務。例如:應用程式在清單檔案中有宣告 INTERNET 權限的話，這個應用程式需要用到網路功能。如果另一個應用程式在清單檔案中沒有宣告任何權限，代表說這個應用程式沒辦法取得網路或寄簡訊等系統功能也無法讀取使用者或是系統的資料。這個應用程式只能提供遊戲或者是計算機等簡單的服務，對系統或使用者來說並沒有威脅性。使用權限來判斷應用程式的好處就是速度，因為只要讀取應用程式的清單檔案再根據已訂好的規則來判斷就好。但是我們無法只根據應用程式所用的權限有沒有違反安全規定或是使用權限來計算應用程式的風險度來判斷應用程式是否為惡意程式。因為應用程式可以只宣告一個 SEND_SMS 權限來做惡意行為。

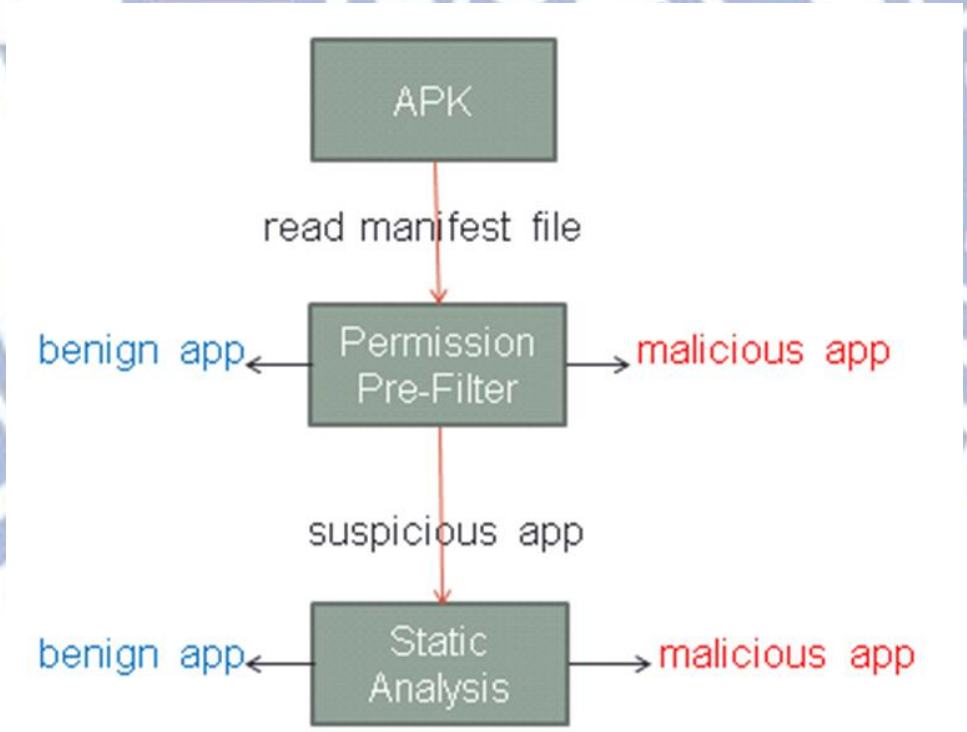


圖 3.2 系統架構

因為上述的原因，我們提出了兩層的 Android 惡意軟體檢測架構。第一層是權限過濾器(permission pre-filter)，是根據應用程式所宣告的權限來判斷。使用權限來判斷的原因是清單檔案中宣告的權限含有關於這個應用程式的一些安全資訊。而且相對於電腦，手機的效能比較差，而且在背景中會一次運行很多個其他應用程式的 Service 這將會耗費很多的記憶體，若執行惡意軟體偵測的話可能會影響使用者使用其他應用程式。如果我們先判斷應用程式所使用的權限有需要的話再去分析並且偵測，將可以降低手機的負擔。系統架構如圖 3.2，如果應用程式經由權限過濾器判斷為良好的應用程式的話，惡意軟體偵測系統將不會做任何事情。如果應用程式被判斷成可疑的應用程式的話，我們將會自動對這個應用程式執行第二層的分析，此分析將會告知使用者此應用程式是否為惡意軟體。如果應用程式被權限過濾器判斷為惡意的軟體，會顯示一個使用者介面，告知使用者此應用程式有風險存在，要不要執行第二層的分析。

程式分析大致上可以分成兩種，一個是靜態分析(static analysis)，另一個是動態分析(dynamic analysis)。靜態分析和動態分析最大的差別在於需不需要執行程式。靜態分析去執行程式原始碼的分析，但是並不會直接去執行程式。相反的，動態分析則是會去執行程式並且觀察程式有沒有惡意行為，但是通常會在沙盒(sand box)中執行。沙盒是為來源不可信或是無法判定程序意圖的程式所提供的測試環境。然而，沙盒是和作業系統隔離的，因此沙盒中的所有更動對作業系統並不會造成任何損失。動態分析因為要實際去執行程式且需要配合測試資料才可以分析可能會耗費許多的資源和時間。所以我們不考慮使用動態分析的方式來分析應用程式。而靜態分析剛好可以滿足我們的需求—在執行應用程式之前進行程式碼的檢測。所以在我們系統的第二層選擇用靜態分析來偵測惡意程式。

3.4 惡意軟體偵測服務

BaseBridge 是 Android 系統中的一個惡意程式。這個惡意軟體在清單中只有宣告 WAKE_LOCK 權限保護等級是 Normal，此權限可以使螢幕一直亮著，防止手機進入休眠狀態的權限。光看權限會覺得此應用程式對手機並不會有威脅，安裝了以後發現此應用程式只是簡單的顯示圖片給使用者觀看。但是在使用者觀看圖片的時候會跳出一個通知，如圖 3.3 的左圖，要求使用者更新此應用程式。值得注意的是以安裝的應用程式並沒有宣告 INTERNET 權限，所以不是藉由網路來更新的。當使用者選擇更新應用程式的話，此應用程式會對內 apk 內嵌的一個檔案進行解碼，並且解碼出一個新的 Android 安裝檔。接著呼叫 Android 的應用程式安裝器，安裝這個解碼出來的安裝檔。這個新的應用程式宣告了很多權限，如圖 3.3 的右圖，而且這個新安裝的應用程式會執行惡意行為，像是打電話或寄簡訊到高費率的號碼。

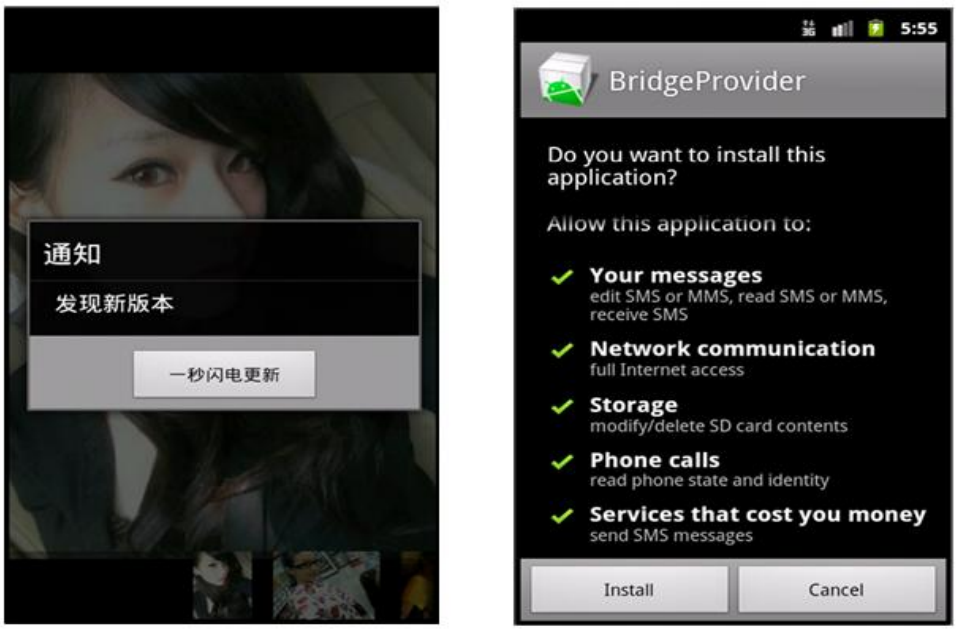


圖 3.3 Android 惡意軟體-BaseBridge

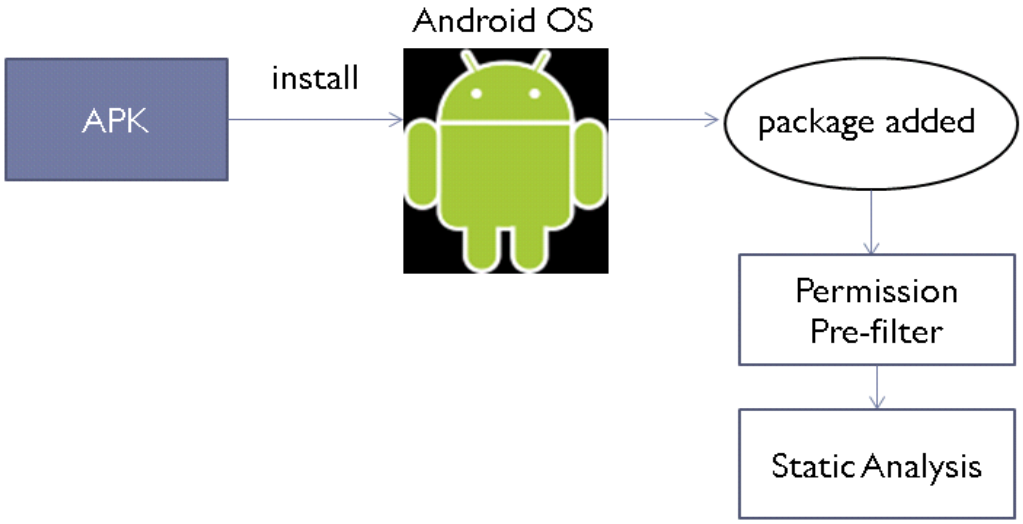


圖 3.4 偵測系統服務

我們的惡意軟體偵測系統會在系統的背景中運作，圖 3.4 描繪了系統運作的方式。當使用者安裝好應用程式之後，Android 作業系統會藉由廣播發出 package added 這個 intent，所以只要有任何的應用程式被 Android 的程式安裝器安裝都會啟動我們惡意軟體檢測。我們的系統在接收到這個訊息後，會去尋找哪一個應用程式被使用者安裝了。並且把此應用程式的 apk 檔拿來檢測，若有需要的話會進行靜態分析。

第四章

權限過濾器與靜態分析

4.1 權限過濾器

跟 Kirin[11]一樣，在我們系統第一層權限過濾器的階段訂了安全規則。但是 Kirin 的安全規則有很大的缺點，他的安全規則只是根據那些有可能做惡意行為的權限組合而訂的。像 Kirin 中的規則六，應用程式不可以有 RECIVE_SMS 和 WRITE_SMS 權限，作者覺得攻擊者可以在遠端經由寄簡訊的方式對使用者的手機下指令。而規則七，應用程式不可以有 SEND_SMS 和 WRITE_SMS 權限，是防止惡意軟體透過這兩個權限使手機變成殭屍一樣隨意的寄簡訊給其他使用者。然而一般的程式開發者有可能想要為使用者提供簡訊管理的服務，可以透過這個應用程式編輯簡訊、寄簡訊和未讀簡訊提醒等，就會被判斷程惡意軟體。惡意軟體可以使用這些權限來達成惡意行為，相反的正常的應用程式也可能需要使用這些權限的組合，如果這樣判斷的話就會造成較多的誤判。

所以我們將權限過濾器的結果分為三類:

1）普通應用程式:

判斷結果為普通應用程式代表此應用程式對系統或使用者沒有威脅。

2）可疑應用程式:

若應用程式被判斷為可疑的，代表我們無法直接從這個應用程式所宣告的權限來判斷此應用程式是否為惡意軟體，需要再經由靜態分析來判斷。

3）惡意應用程式:

此種應用程式宣告了一般開發者無法使用的權限，對系統或使用者有很大的威脅。

Android 系統中定義了很多系統權限，這些系統權限都可以讓第三方的應用程式取得系統的資源。但是在這些權限中，有的可能只是讀取手機電池電量的統計資料且對系統無害的權限(BATTERY_STATS)，但是有的權限(INSTALL_PACKAGE)卻可以讓一個應用程式執行靜默安裝，在使用者無察覺下安裝另外的應用程式。所以 Google 將開發者可能需要用到的權限分在 Normal 或 Dangerous 等級，這兩種等級的權限一般開發者可以去使用。然而將對系統更具威脅、為其他廠商開發的客製化權限和一般開發者不需要使用到的權限歸在 Signature 或 SignatureOrSystem 等級。這兩種等級的權限是一般開發者無法使用的，根據系統的定義，要有相同的憑證或是存在系統資料夾底下的應用程式才可以使用這兩種等級的權限。若一般使用者要使用 Signature 或 SignatureOrSystem 等級的權限的話就必須使手機取得管理員權限(root phone)或者是其他的惡意行為。所以我們認為應用程式如果要使用到 Signature 或 SignatureOrSystem 等級的權限，就必須要先對系統做些惡意行為，才可以獲得權限。所以第一條規則:只要應用程式有使用 Signature 或 SignatureOrSystem 等級的權限，會被判斷為惡意應用程式。

根據我們在 1.2.3 節中提到的，現在 Android 系統中的惡意軟體主要傾向於竊取使用者的隱私資料或是造成使用者的額外花費藉此賺錢。在造成使用者的話費部分我們考慮兩個權限:SEND_SMS 和 CALL_PHONE。這兩個權限都被歸類在 COST_MONEY 這個權限群(permission group)中。由於惡意程式只要有 SEND_SMS 或是 CALL_PHONE 就可以對使用者造成傷害，所以第二條規則:只要應用程式有宣告 SEND_SMS 權限就會被判斷為可疑應用程式，第三條規則:只要應用程式有宣告 CALL_PHONE 權限就會被判斷為可疑應用程式。在竊取使用者的隱私資料方面，攻擊者必須透過手機的網路或者是簡訊才能取回手機的隱私資料。由於第二條規則已經有 SEND_SMS 權限了，只要有寄簡訊的權限就會被

判斷為可疑應用程式，所以在這邊我們不需要考慮攻擊者使用簡訊的方式取回隱私資料。另一個方式就是攻擊者透過網路的方式將使用者的隱私資料傳到遠端的伺服器，所以第四條規則:只要應用程式的清單檔案中有宣告 INTERNET 權限並且有其他跟使用者或系統隱私資料相關的權限就會被判斷為可疑應用程式。可疑應用程式會自動執行靜態分析。

若應用程式經過權限過濾器的判斷，沒有被判斷成惡意應用程式且沒有被判斷為可疑應用程式的話，權限過濾器會將此應用程式歸類為普通應用程式。

權限過濾器的安全規則定義如下:

1. 任何應用程式若有使用 Signature 或 SignatureOrSystem 等級的系統權限會被權限過濾器歸類為惡意應用程式。

2. 任何應用程式有使用 SEND_SMS 權限會被權限過濾器歸類為可疑應用程式。

3. 任何應用程式有使用 CALL_PHONE 權限會被權限過濾器歸類為可疑應用程式。

4. 任何應用程式有使用 INTERNET 權限並且有使用到下列任一個權限
   READ_SMS || READ_CALENDAR || ACCESS_FINE_LOCATION ||
   RECORD_AUDIO || PROCESS_OUTGOING_CALLS ||
   READ_CONTACTS ||READ_PHONE_STATE || READ_LOGS
   會被權限過濾器歸類為可疑應用程式。

5. 任何應用程式沒有違反上述的安全規則會被歸類為普通應用程式。

4.2 靜態分析

4.2.1 使用Q-Gram的缺點

在靜態分析中，我們使用 Androguard[14]當作程式源碼分析的工具。Androguard 是一個由 Python 程式語言所開法的軟體，開放原始碼且免費，可以用來分析 Android 的應用程式。

Androguard 可以將 apk 檔中的 classes.dex 進行分析。classes.dex 是 java 源碼編譯後產生的 java 字節碼文件，其中可包括多個 classes 檔案。簡單的說 classes.dex 就是這個應用程式的源碼，只是經過編譯為字節碼而已。Androguard 可以分析程式的源碼，將程式分成一個個的基本區塊(basic block)，如圖 4.1 的 BB，再根據圖 4.2 的文法將程式轉換成字串。基本區塊是指在這個區塊內的程式碼只有一個進入點和一個離開點，進入點就是區塊內的第一行，代表說在基本區塊內的程式碼並不會有其他跳躍指令的目的地，離開點就是區塊內的最後一行的指令，只有最後一行的指令可以跳躍到其他的基本區塊。

圖 4.1 Androguard 將程式轉換成字串

```
Procedure ::= StatementList
StatementList ::= Statement | Statement StatementList
Statement ::= BasicBlock | Return | Goto | If | Field | Package | String | Exception
Return ::= 'R'
Goto ::= 'G'
If ::= 'I'
BasicBlock ::= 'B'
Field ::= 'F'0 | 'F'1
Package ::= 'P' PackageNew | 'P' PackageCall
PackageNew ::= 'C'
PackageCall ::= 'M'
PackageName ::= Epsilon | Id
String ::= 'S' Number | 'S' Id
Exception ::= Id
Number ::= \d+
Id ::= [a-zA-Z]\w+
```

圖 4.2 Androguard 使用的字串結構的文法

表 4-1 使用 Q-gram 計算 Pjapps 中變種的相似度

	a1	a2	a3	a4	a5
a1	1	-6.26	-1.11	-1.42	-2.57
a2	0.11	1	0.29	0.32	0.35
a3	0.28	-0.98	1	0.21	-0.07
a4	0.27	-0.69	0.29	1	0.28
a5	0.20	-0.19	0.30	0.47	1

根據[10]的方法，我們使用 400 個惡意軟體和 400 個正常的應用程式來選出 500 組基底。根據[10]中定義的相似度公式，兩個特性向量的相似度值為 1 代表兩個程式完全一樣，兩個特性向量的相似度越低代表越不相像。PJAPPS 是 Android 系統的惡意軟體，

我們從 PJAPPS 群中隨機挑選 5 個惡意軟體來檢查他們的相似度。相似度的結果如表 4-1，我們可以發現不同程式之間的相似度都很低，代表說以[10]的方法來判斷的話會認為這些惡意程式很不像。但是防毒公司將這些惡意軟體分在同一群一定有他的原因。會造成這樣的主要原因是在這些惡意軟體的程式碼中只有小部分的程式碼是執行惡意行為，而大部分的程式碼是執行正常的行為。如圖 4.3 所示，應用程式一是個棒球遊戲，但是使用者在玩的同時他會寄簡訊到高話費的號碼，而應用程式二是一個瀏覽器，在使用者使用的同時也會執行同樣的惡意行為，若以整個應用程式的角度觀察的話會覺得這兩個應用程式很不像。但是以惡意行為來觀察的話會發現他們是一樣的，所以會被分類在同一種惡意軟體。像這種重新包裝應用程式的方法很常見，攻擊者只需要開發好惡意行為的程式碼，再去 Android Market 上面下載熱門的應用程式，經由反編譯將惡意程式加入應用程式中。最後把修改過的應用程式上架，讓使用者下載使用。所以造成表 4.1 中，雖然是同一群的惡意軟體但是相似度卻很低。

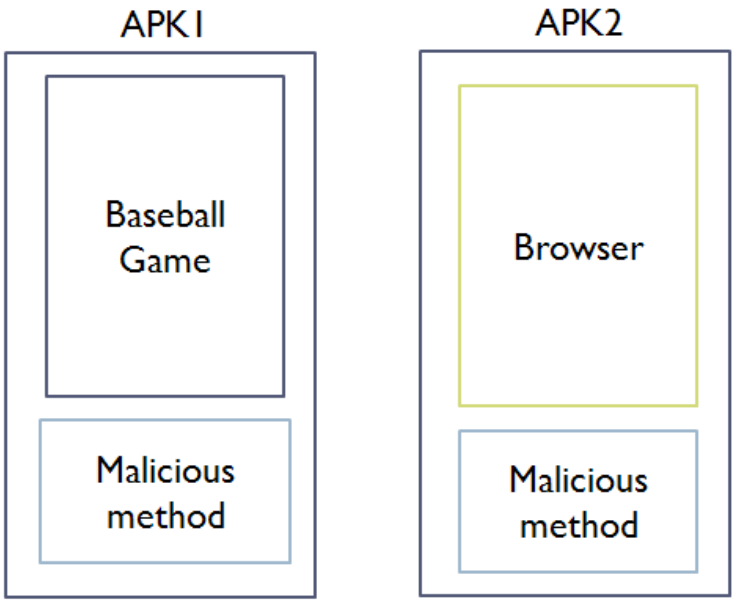


圖 4.3 變種惡意軟體的例子

4.2.2 方法相似度—Method Similarity Index

由於上述的攻擊者重新包裝惡意軟體，兩個同樣攻擊方式的惡意軟體整體而言可能很不像,但是惡意程式的部分是相同的。所以我們提出 MethSI(method similarity Index)來偵測惡意軟體。

方法(method)是一段程式碼，在 Java 程式語言中的方法就跟 C 程式語言中的函數(function)一樣。方法可以在程式的任何地方被呼叫，當呼叫方法後會去執行方法本身的程式碼，執行完方法以後再回到呼叫方法的下一行程式碼繼續執行，如圖 4.4。我們可以把方法想像為一個子程式，通常會輸入一組資料然後回傳一個值或完成某件任務。

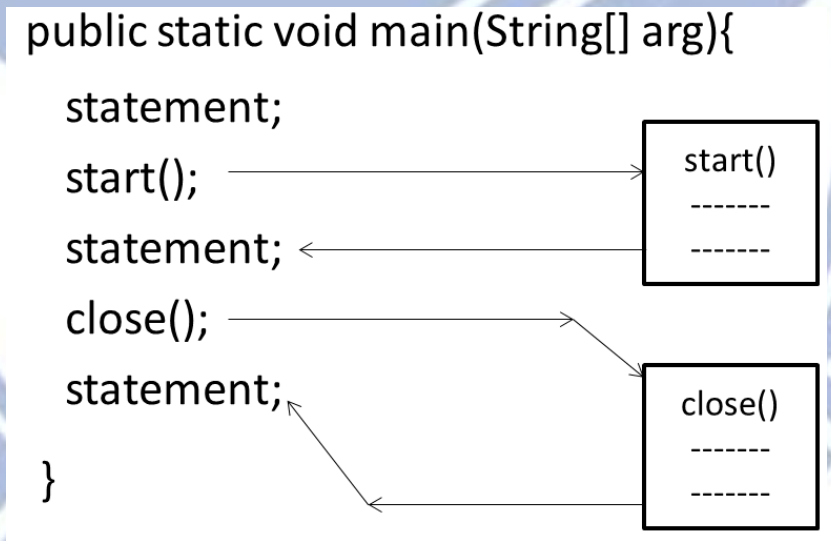


圖 4.4 方法的例子

Android 的應用程式是用 Java 程式語言來開發的,而 Java 是物件導向的程式語言。在物件導向的程式中所有的東西都是物件，若要使物件達成某件事情就必須撰寫方法，使這個方法可以達成想要做的事情。例如:為了要使應用程式寄簡訊到高費率的號碼，

那攻擊者就要開發一個方法，讓這個方法可以使手機寄簡訊到指定的號碼。所以一個惡意程式要有一個或多個方法來執行惡意行為，像是一個方法用來讀取使用者的隱私資料，另一個方法將資料上傳到遠端的伺服器。MethSI 使用 Androguard 以一個方法為單位產生出方法的特徵字串，在資料庫中存放的是惡意行為的方法的特徵字串。以惡意軟體 KMIN 為例，ChargeUtil 類別中的 getUrl 方法是執行惡意行為的地方，那我們就把 getUrl 的特徵字串存入資料庫中，當作判斷惡意軟體 KMIN 的依據。如圖 4.5 所示，一個應用程式有很多個方法，若詢問的應用程式其中有方法和資料庫裡的方法相似度大於臨界值，就會被判斷為惡意行為。例如:若在一個應用程式中，若其中有一個方法和 KMIN 的惡意方法 getUrl 相似度大於臨界值，那這個應用程式就會被判斷為 KMIN。

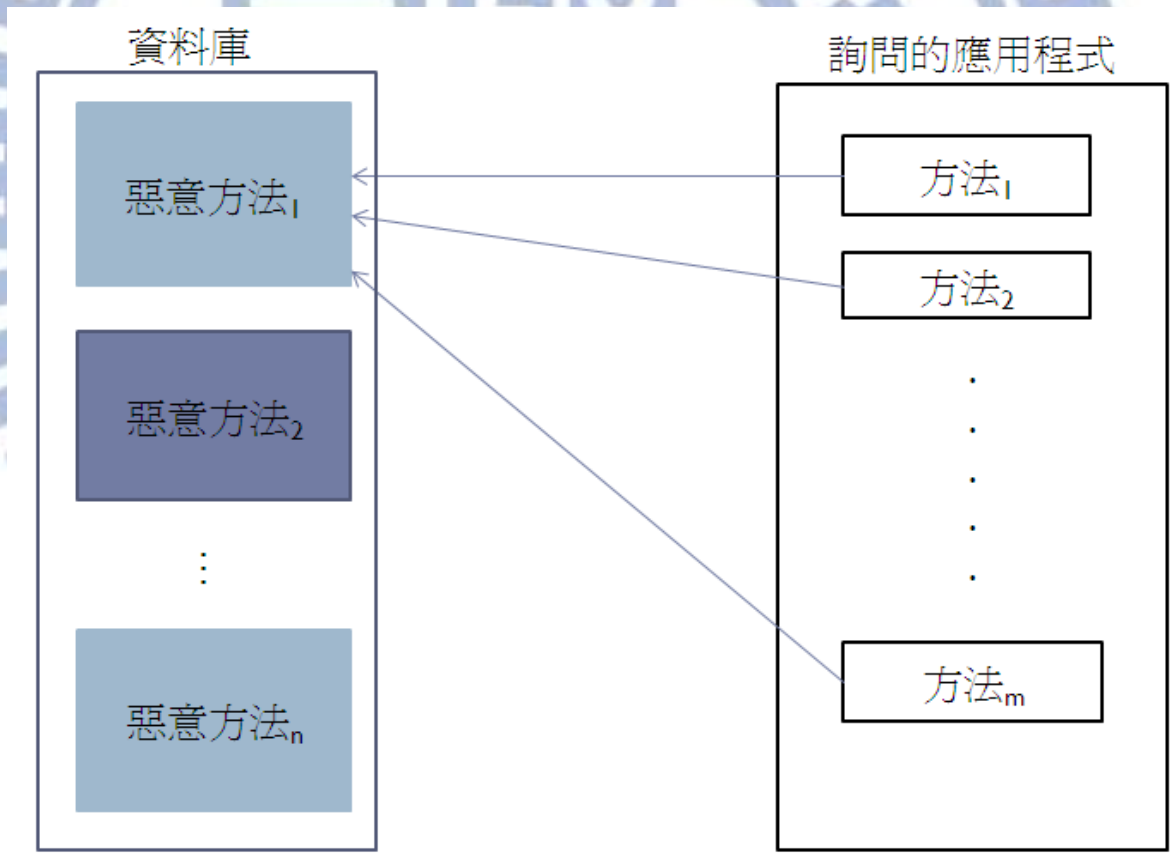


圖 4.5 MethSI 示意圖

我們定義兩個字串的相似度如下:

$$sim(p,q_i) = 1 - \frac{ed(p,q_i)}{|q_i|}$$

在相似度的公式中，p 為資料庫中的某一個特徵字串，q_i 為要詢問的應用程式中第 i 個方法的字串，$|q_i|$ 是指 q_i 字串的長度，而 $ed(p,q_i)$ 是指 p 字串和 q_i 字串的編輯距離(edit distance)。編輯距離是計算從一個字串變到另一個字串所需的最少步數，在編輯距離演算法當中允許三種方式來操作字串:替代、刪除和插入。例如要把字串 kitten 變成 sitting，首先第一步先把 k 換成 s，第二步把 e 換成 i，第三步在字串的最後面插入 g，就完成了，沒有其他方法會比這樣操作還要少步數。所以 kitten 和 sitting 之間的編輯距離為 3。若兩個字串很不像，那編輯距離就會很大，算出來的相似度就會小。相反的，若兩個字串完全一樣，那編輯距離就會是 0，相似度發生極大值 1。

4.2.3 字串壓縮

在編輯距離的演算法中，通常是以動態規劃(dynamic programming)的方式實現。此種方式會建立一個 n 乘 m 的矩陣，其中 n 和 m 分別是兩個字串的長度，再計算矩陣中每一格的值，把全部值計算完以後才會得到結果。所以我們可以知道計算兩個字串編輯距離的時間和這兩個字串的長度相關，因此只要能夠減少字串的長度，就能夠減少計算的時間。為了要減少字串的長度，我們修改了 Androguard 中字串的文法，如圖 4.6。

```
Procedure ::= StatementList
StatementList ::= Statement | Statement StatementList
Statement ::= Basic Block | Return | Goto | If | Filed | Package | String |
Exception
Return ::= 'R'
Goto ::= 'G'
If ::= 'I'
Basic Block ::= ']'
Field ::= 'E' | 'F'
Package ::= PackageNew | PackageCall
PackageNew ::= 'P'
PackageCall ::= 'Q'
PackageName ::= Epsilon | Id
String ::= 'S' Number | 'S' Id
Exception ::= Id
Number ::= \d+
Id ::= [a-zA-Z]\w+
```

圖 4.6 修改後字串結構的文法

根據上面的文法我們可以把字串的長度減小，但是不會喪失任何資訊。我們將 F0 換成 E、F1 換成 F、P0 換成 P、P1 換成 Q 和 B[]換成]。例如:原本的字串是 B[P0ISF0] B[SSP1F1R]用新的文法產生出來的字串是 PISE]SSQFR]。基本區塊只是將不同區塊的程式碼分開，所以將原本的 B[]改成]並不會造成資訊的損失。

4.2.4 長度區間

一個應用程式可能有 5000 個方法，若一個一個比對會耗費大量時間，在這一節我們將介紹如何只針對有可能相似的字串做比對。首先我們先看的範例，現在有兩個字串，aa 是資料庫中的字串而 aaaaaaaaaa 是詢問要比對的字串。接著根據公式計算相似度，這兩個字串的相似度為 0.2，相似度並不大。此結果並不讓人意外，因為這兩個字串的長度相差 8，編輯距離就至少是 8 了，所以相似度不可能會高。由上述的例子我們可以發現相似度和兩個字串長度的差距有關係，我們想要找出兩個字串長度的差距和臨界值

之間的關係—是不是兩個字串長度相差超過某個值時，這兩個字串的相似度不可能大於臨界值。若資料庫中的字串長度和詢問的字串長度相差太多，彼此的相似度不可能大於臨界值的話就不需要去計算了，因為即使計算了也不會把此詢問的字串判斷為惡意行為字串。為了要推倒此結果，首先將相似度的公式移項:

$$sim(p,q) = 1 - \frac{ed(p,q)}{|q|} > t$$

$$\Rightarrow ed(p,q) < (1-t)|q|$$

其中 p 是資料庫中的一個字串，q 是詢問的字串，t 是臨界值，若兩個字串的相似度大於臨界值，這兩個字串會被判斷為相似。接下來我們要推導出詢問的字串長度界在甚麼區間的時候跟資料庫的字串相似度才有可能大於臨界值。

$$If\ |p| > |q|$$

$$a + |p| - |q| < (1-t)|q|$$

在這條式子，我們將 ed(p,q)拆成 a+|p|-|q|。由於|p|>|q|，|p|-|q|代表字串 p 換成字串 q 要經過幾次刪除，而 a 代表字串 p 經過刪除後還需要經過幾次的替換才能將 p 字串轉換成 q 字串，很明顯的 a 大於等於 0。

$$\Rightarrow |q| > \frac{|p|}{(2-t)} + \frac{a}{(2-t)}$$

若要讓|q|的範圍最大，則後面的值要最小，a 的值為大於等於 0，當 a 等於 0 時|q|有最大的範圍。我們可以得到當 p 字串的長度比 q 字串的長度大，且 q 字串的長度滿足下式的話，p、q 兩字串的相似度有可能大於臨界值。

$$|q| > \frac{|p|}{(2-t)}$$

根據上面的推倒，我們可以以同樣的想法推導出另一個情況— p 字串的長度小於 q 字串的長度。

$$If\ |p| < |q|$$

$$a + |q| - |p| < (1-t)|q|$$

$$\Rightarrow |q| < \frac{|p|}{t} - \frac{a}{t}$$

$$\Rightarrow |q| < \frac{|p|}{t}$$

由以上兩個情況的推倒我們可以知道當一個詢問的字串 q，其字串長度為|q|，和資料庫的字串 p，其字串長度為|p|，兩者的相似度要大於臨界值 t 的話 q 字串的長度必須滿足下式才有可能發生，若不滿足下式則相似度不可能大於臨界值。

$$\frac{|p|}{(2-t)} < |q| < \frac{|p|}{t}$$

MethSI 偵測惡意軟體的方式如圖 4.7，在資料庫中每一個惡意行為的字串都會先根據臨界值和字串本身的長度計算出每一個字串的長度區間。當一個可疑的應用程式進入到 MethSI，會先根據結構文法將方法轉換成字串，接著算出每一個方法字串的長度。在比對的時候需要比對資料庫中每一個病毒的惡意方法，但是不需要檢查每一個詢問的應用程式的方法。例如詢問的應用程式總共有 m 個方法，在比對第一個惡意方法的時候，只需要去比對 m 個方法字串中有界在 a_1 到 b_1 之間的方法字串。因為不在這個範圍內的字串和第一個惡意方法字串之間的相似度不可能會大於臨界值，所以先判斷有沒有在長度區間內而不用去計算編輯距離，將可以節省大量的比對時間。

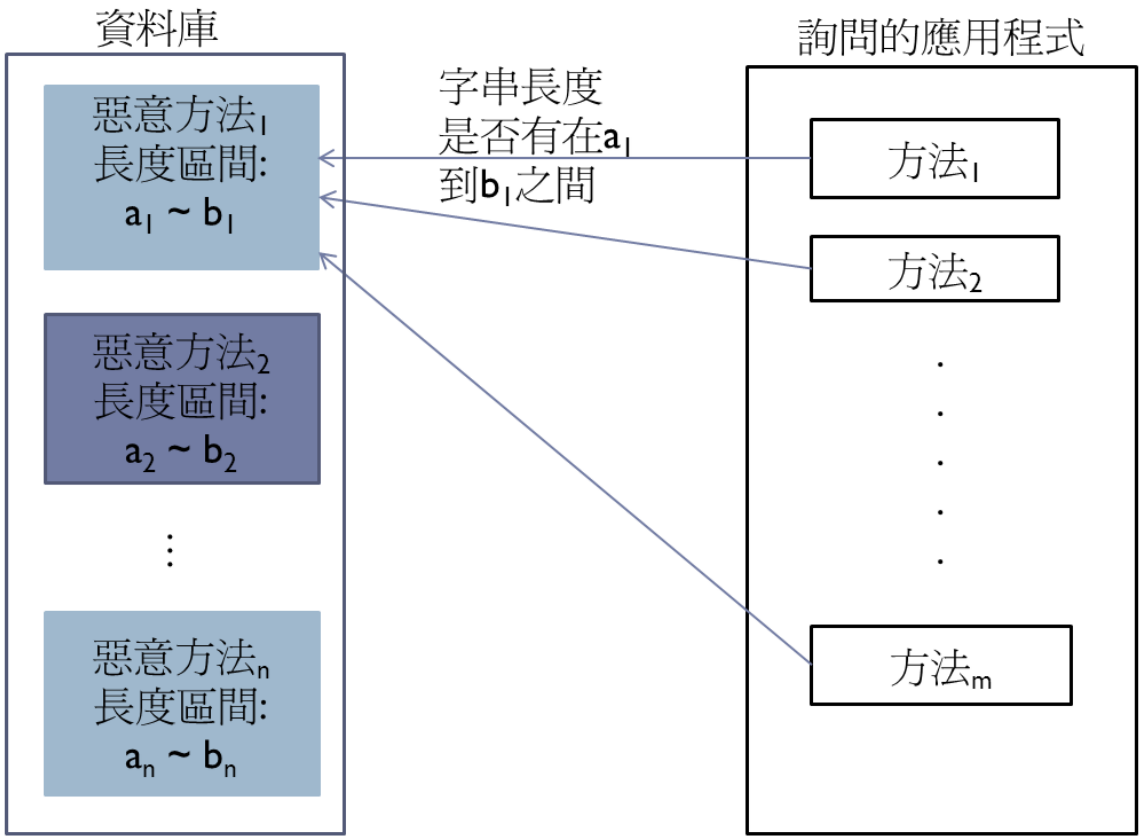


圖 4.7 MethSI 使用長度區間的示意圖

第五章

實驗模擬與結果

5.1 權限過濾器與Kirin的比較

在這一節中我們將比較權限過濾器和 Kirin 的安全規則。Kirin 的安全規則跟權限過濾器不一樣，沒有將應用程式分類為惡意、可疑和正常，但他認為應用程式不能有某些權限的組合，如 2.1.1 節中介紹。所以若應用程式沒通過 Kirin 的安全規則我們將結果歸類為惡意程式，但是通過 Kirin 安全規則的應用程式並不代表是正常的應用程式，因為 Kirin 的原始用意是減少手機安裝到惡意軟體的機會，所以若應用程式通過 Kirin 安全規則，我們將其分類為可疑應用程式。我們從 contagio[15]下載了 400 個 Android 的惡意程式，再根據權限過濾器和 Kirin 的安全規則，分類結果如圖 5.1。

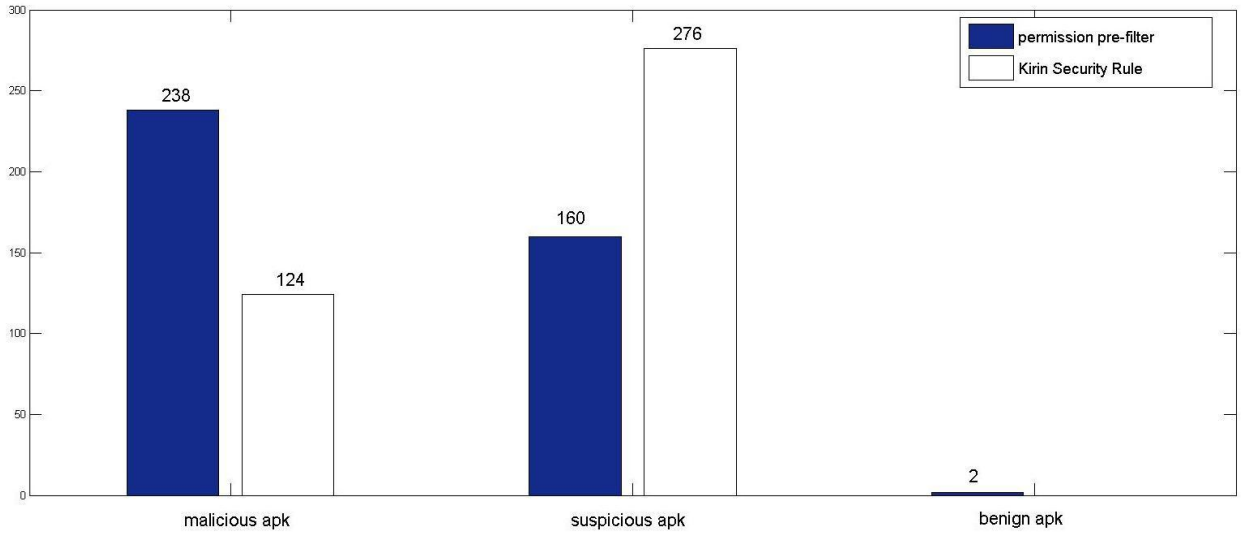


圖 5.1 使用權限過濾器和 Kirin 安全規則分析 400 個惡意程式

我們期望權限過濾器可以將惡意程式判斷為惡意的數量越多越好，但是不能有惡意程式被判斷為正常。因為惡意程式被判斷為正常的話，此惡意程式將不會經過第二層的靜態分析，會造成漏報(false negative)。漏報是指應用程式是惡意軟體，但是系統判斷為正常的應用程式。根據圖 5.1，我們發現有兩個惡意程式被權限過濾器判斷為正常的應用程式。所以我們去檢查這兩個惡意軟體，發現這兩個都是 z4Root 惡意軟體，只是不同的版本。z4Root 可以不宣告任何的權限而去取得手機的管理員權限，這個應用程式是透過 Linux 系統的漏洞，rage against the cage，來取得手機的管理員權限，所以被認為是惡意程式。但是這個 Linux 的漏洞在 Android 2.2 版後已經被修復了，惡意程式沒辦法使用這個系統的漏洞來取得管理員的權限。而 Android 系統的使用者幾乎都已經 2.3 版以上了，所以這個應用程式就算會通過惡意軟體偵測但是也沒辦法執行惡意行為。

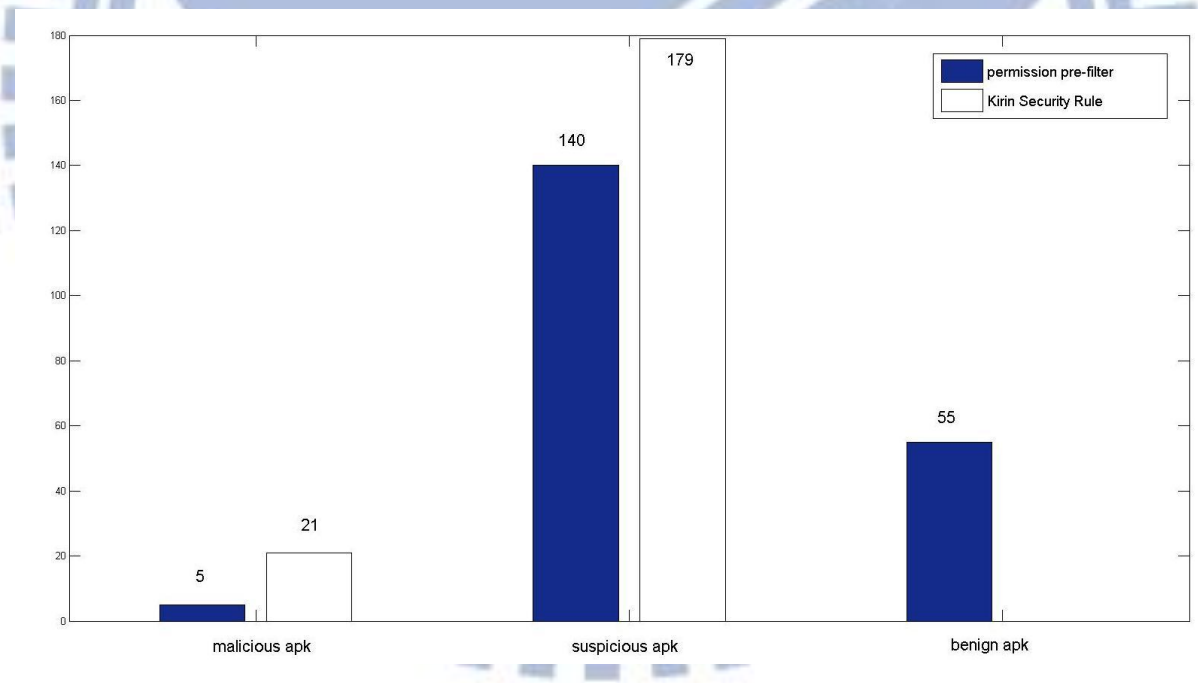


圖 5.2 使用權限過濾器和 Kirin 安全規則分析 200 個正常程式

接下來我們分析下載回來的200個應用程式,而這些應用程式經由VirousTotal[16]判斷為正常的應用程式。VirousTotal 是一個網站，可以將上傳的應用程式分析，並且顯示 42 個防毒軟體判斷的結果。根據圖 5.2 可以發現權限過濾器有 5 個誤報(false positive)，誤報的意思是應用程式是正常的，但是系統將其判斷為惡意程式。會發生這5個誤報的原因是開發者錯誤的在清單檔案中宣告權限。舉例來說，應用程式需要讓手機螢幕一直亮著、防止手機進入休眠狀態，為了要達到此目的只需要宣告 WAKE_LOCK 權限就好，其保護等級為 Normal，但是有些網路上的教學卻說還需要用到 Signature 等級的 DEVICE_POWER 權限，所以造成一些開發者的誤會和錯誤的宣告。而 Kirin 有 21 個誤報，其主要原因是像之前提到的，好的應用程式和惡意程式都有可能去使用那些權限組合。

5.2 靜態分析

5.2.1 臨界值的選擇

在這一小節中將介紹我們如何選擇 MethSI 的臨界值。我們拿 Kmin、Pjapps、YZHC 這三類惡意軟體和 200 個正常的應用程式來比較相似度。在正常的 200 個應用程式中，總共有 515461 個方法。首先用 Kmin 的惡意方法字串跟全部正常的方法做相似度的比較如圖 5.3。其中縱軸單位為個，代表有幾個方法落在這個區間，而橫軸代表相似度值的區間。

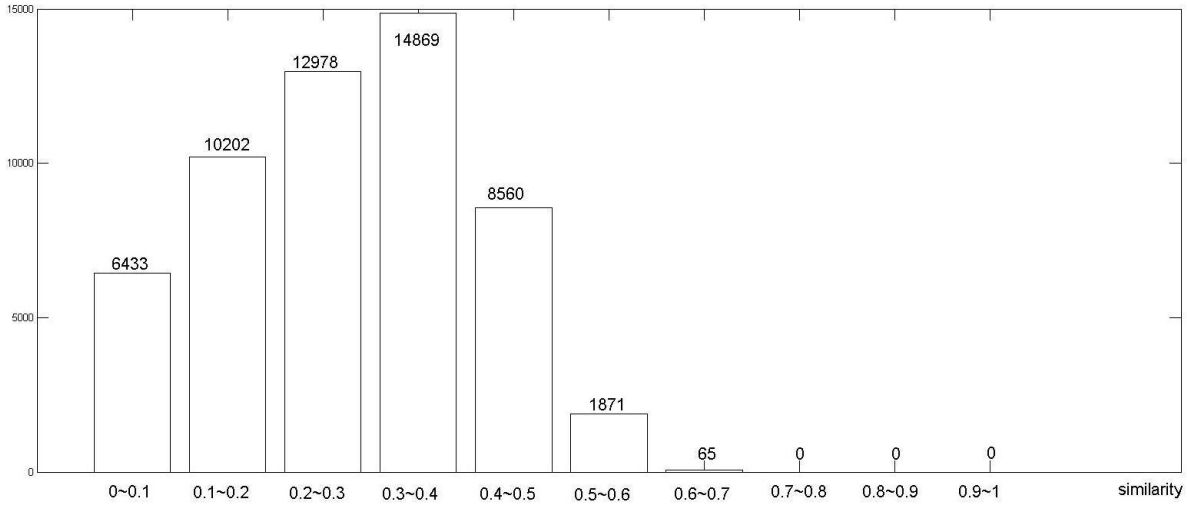


圖 5.3 正常應用程式的方法和 Kmin 惡意方法之比較

接著我們在同一類惡意程式中，使用其惡意方法字串去計算變種惡意程式的相似度。在一個應用程式中，只要有一個方法字串和資料庫中惡意字串的相似度超過門檻值，那我們就會認為這個方法執行惡意行為，所以把這個程式判斷為惡意軟體。在表 5-1 中，表示 Kmin 的惡意方法字串和 Kmin 惡意軟體中的所有方法比較，選出相似度最高的方法的值。圖 5.4、5.5，表 5-2、5-3 分別顯示 Pjapps 和 Yzhc 跟正常的應用程式方法之間的關係。

表 5-1 使用 MethSI 計算 Kmin 變種的相似度

	k_1	k_2	k_3	k_4	k_5	k_6
相似度	1.0	1.0	0.86	0.86	0.86	1.0

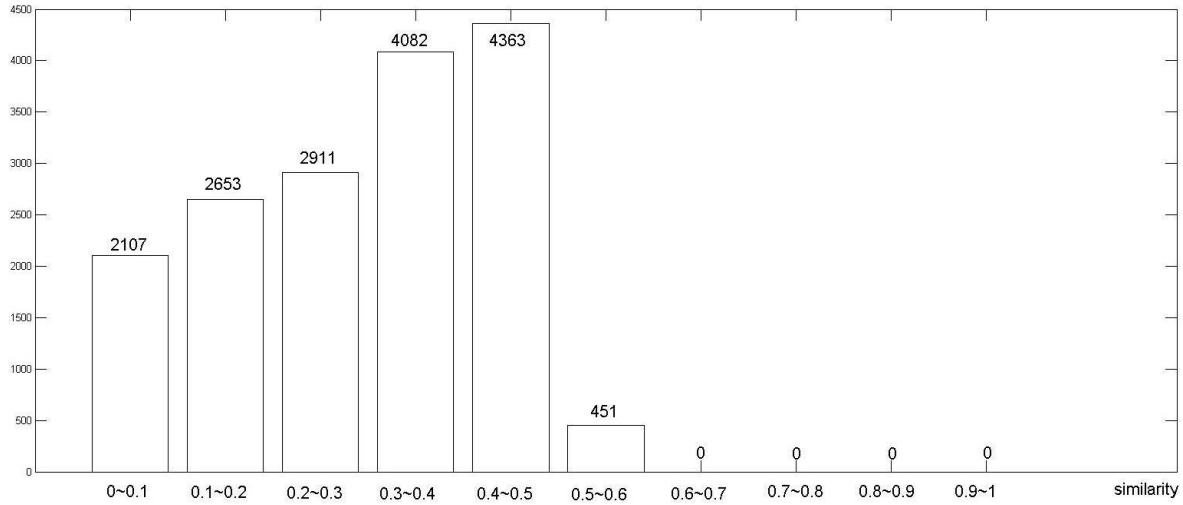


圖 5.4 正常應用程式的方法和 Pjapps 惡意方法之比較

表 5-2 使用 MethSI 計算 Pjapps 變種的相似度

	P_1	P_2	P_3	P_4	P_5	P_6
相似度	1.0	1.0	0.73	0.73	0.53	1.0

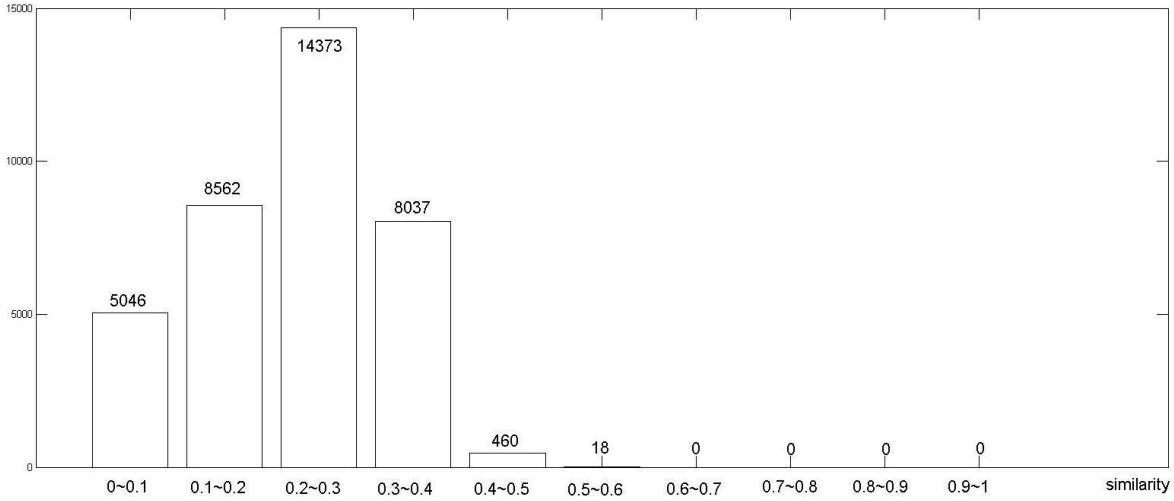


圖 5.5 正常應用程式的方法和 Yzhc 惡意方法之比較

表 5-3 使用 MethSI 計算 Yzhc 變種的相似度

	y_1	y_2
相似度	1.0	1.0

臨界值的選擇理想上是要讓誤報(false positive)率和漏報率越低越好。誤報是指這個應用程式是正常的，但是系統判斷他為惡意程式而提出警告。在表 5-2 中可以發現，最低的相似度是 0.52，但是若我們選擇使用 0.5 當作臨界值將會造成很大的誤報率。所以根據上面三類惡意方法字串的分析與比較，我們選擇 0.7 做為系統的臨界值。在查詢應用程式的所有方法中，只要有任何一個方法字串和資料庫中的字串相似度超過 0.7，我們會將這個應用程式判斷為惡意程式。

5.2.2 長度區間的效率

在這一節中，我們將比較使用長度區間過濾掉相似度不可能大於臨界值的方法字串和沒有使用長度區間的差別。

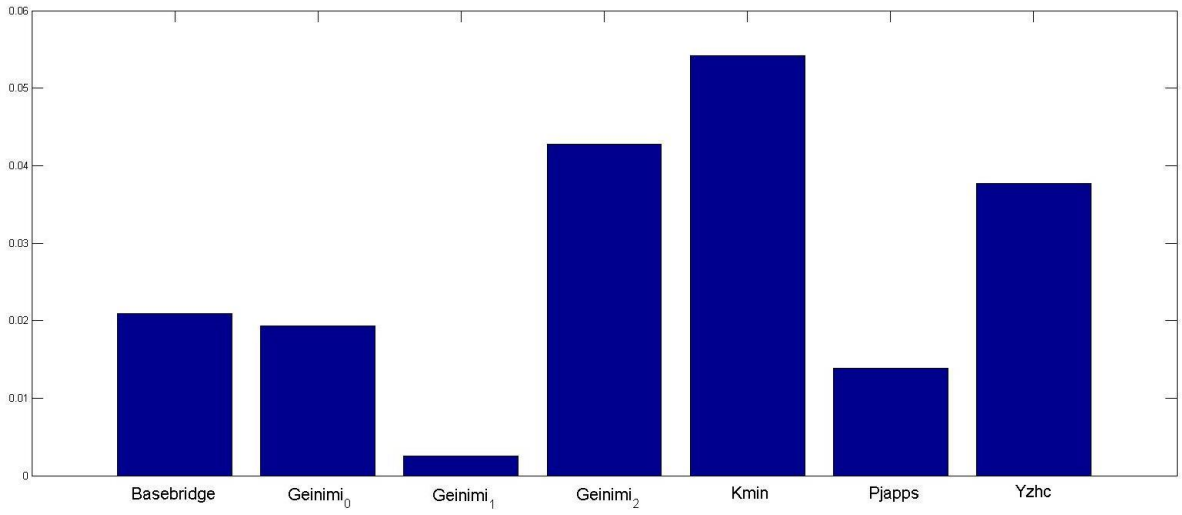


圖 5.6 使用長度區間的效率

我們的資料庫裡面有 Basebridge、Geinimi、Kmin、Pjapps 和 Yzhc 這 5 個惡意軟體的惡意方法字串，其中 Geinimi 有三個惡意方法，只要應用程式和任何一個相似度超過臨界值就會被判斷為 Geinimi。以 200 個正常的應用程式當作要詢問的對象，這 200 個應用程式總共有 515461 個方法，我們想要知道使用長度區間，只需要去計算多少數量的方法。所以用 200 個應用程式，看資料庫裡面每一個字串需要計算多少個方法。圖 5.6 顯示每一個惡意字串只需要計算一個應用程式多少比率的方法。以 Basebridge 為例，一個應用程式以 2500 個方法來計算，使用長度區間的話 Basebridge 字串大概只要偵測 50 個方法。

我們的模擬環境是使用 Java 來編譯，2.0GHz Dual CPU，3GB 的記憶體，作業系統是 Windows XP SP3。現在知道了只需要對少部分的方法做比對就好，我們還需要知道真正運作的時間。如圖 5.7，顯示了使用長度區間和未用長度區間在字串比對時間上的比較，其中橫軸代表 200 個應用程式，縱軸代表比對的時間單位是秒。我們可以發現一個應用程式有越多方法的時候，使用長度區間減少的效率會越好。但是應用程式的方法很

多，就算使用長度區間，比對時間還是會比方法少的應用程式還要久，因為計算編輯距離佔了大部分的時間，方法越多比對的時間也越久。

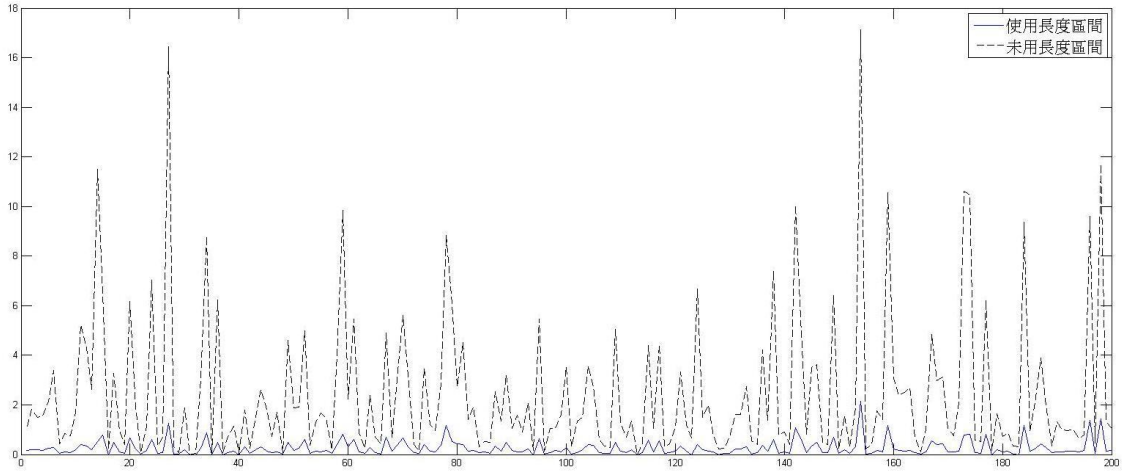


圖 5.7 使用和未用長度區間的時間比較

5.2.3 MethSI 和 Q-gram 的比較

在這一節，我們要比較 MethSI 和 Q-gram 在時間上和偵測惡意軟體的能力的優缺點。在資料庫中有 Basebridge、Geinimi、Kmin、Pjapps 和 Yzhc 這 5 個惡意軟體，MethSI 是存相關的惡意方法字串，Q-gram 是存特性向量。如我們之前提的在 Q-gram 方法中只需要先把應用程式的字串轉成特性向量，在跟資料庫中的向量做相似度的計算，向量的相似度計算花費的計算時間很小。而 MethSI 需要跟資料庫中的每個字串比較經過長度區間過濾所剩下來的方法字串。

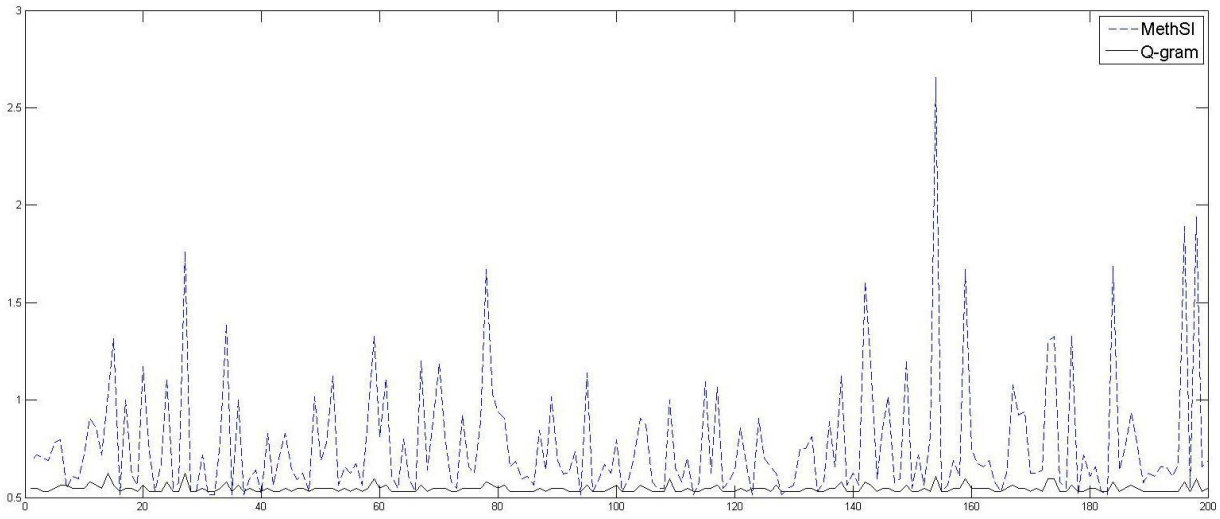


圖 5.8 MethSI 和 Q-gram 檢測的時間比較

圖 5.8 顯示兩個方法的計算時間，橫軸代表有 200 個應用程式，縱軸代表運算的時間單位是秒，環境控制在同樣的狀況下，都只有開 Java 的編輯平台 Eclipse。由圖 5.8 可以很明顯的發現在某些情況下 MethSI 的偵測時間很大，其主要原因是應用程式的方法很多，在字串比對上面需要花較多的時間。使用 Q-gram 的平均時間是 0.5455 秒，而使用 MethSI 的平均時間是 0.7717 秒。

接下來我們要比較兩種方法在偵測 Basebridge、Geinimi、Kmin、Pjapps 和 Yzhc 這五類變種惡意軟體的能力。在 Q-Gram 方法的資料庫中存放每一類惡意軟體中的一個惡意程式的特性向量，而 MethSI 方法的資料庫中存放一樣的惡意程式中的惡意方法字串。我們想知道只存放一個惡意軟體的資訊，兩種方法偵測變種惡意程式或混淆的惡意程式的能力，如表 5-4、5-5、5-6、5-7 和 5-8。以表 5-4 為例，MethSI 方法中我們以其中一個 Basebridge 的惡意方法字串去比較同一類惡意軟體的相似度，取所有方法中最大的相似度來代表。而 Q-gram 方法中我們用同一個 Basebridge 的特性向量跟同一類惡意軟體的特性向量做比較，統計他們的相似度分布的情形。

表 5-4 兩種方法統計 Basebridge 相似度的結果

相似度 / 方法	<0	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1
MethSI	0	0	0	0	0	1	3	0	0	0	0
Q-gram	3	0	0	0	0	1	0	0	0	0	0

表 5-5 兩種方法統計 Geinimi 相似度的結果

相似度 / 方法	<0	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1
MethSI	0	0	0	0	0	1	1	0	23	0	3
Q-gram	0	1	4	1	5	5	6	4	2	0	0

表 5-6 兩種方法統計 Kmin 相似度的結果

相似度 / 方法	<0	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1
MethSI	0	0	0	0	0	0	0	0	0	7	32
Q-gram	5	0	0	0	0	0	0	0	0	0	34

表 5-7 兩種方法統計 Pjassp 相似度的結果

相似度 / 方法	<0	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1
MethSI	0	0	0	0	0	0	2	0	4	0	9
Q-gram	8	0	1	2	1	3	0	0	0	0	0

表 5-8 兩種方法統計 Yzhc 相似度的結果

相似度 / 方法	<0	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1
MethSI	0	0	0	0	0	0	0	0	0	0	1
Q-gram	0	0	0	0	0	0	0	0	0	0	1

圖 5.9 是使用 Q-gram 的方法比較 5 個惡意軟體跟 200 個正常的應用程式計算相似度統計的結果，在 Q-gram 方法中的臨界值我們選定為 0.7。MethSI 和 Q-gram 在偵測 200 個正常軟體時都不會判斷為惡意軟體，誤報率皆為 0。而根據表 5-4 到 5-8 我們統計了偵測惡意軟體的能力，如表 5-9。MethSI 和 Q-gram 的資料庫中都是存放同樣的惡意軟體，但是我們可以發現 MethSI 在偵測變種的能力比 Q-gram 好很多。

表 5.9 偵測能力

方法	偵測能力
MethSI	79/87
Q-gram	37/87

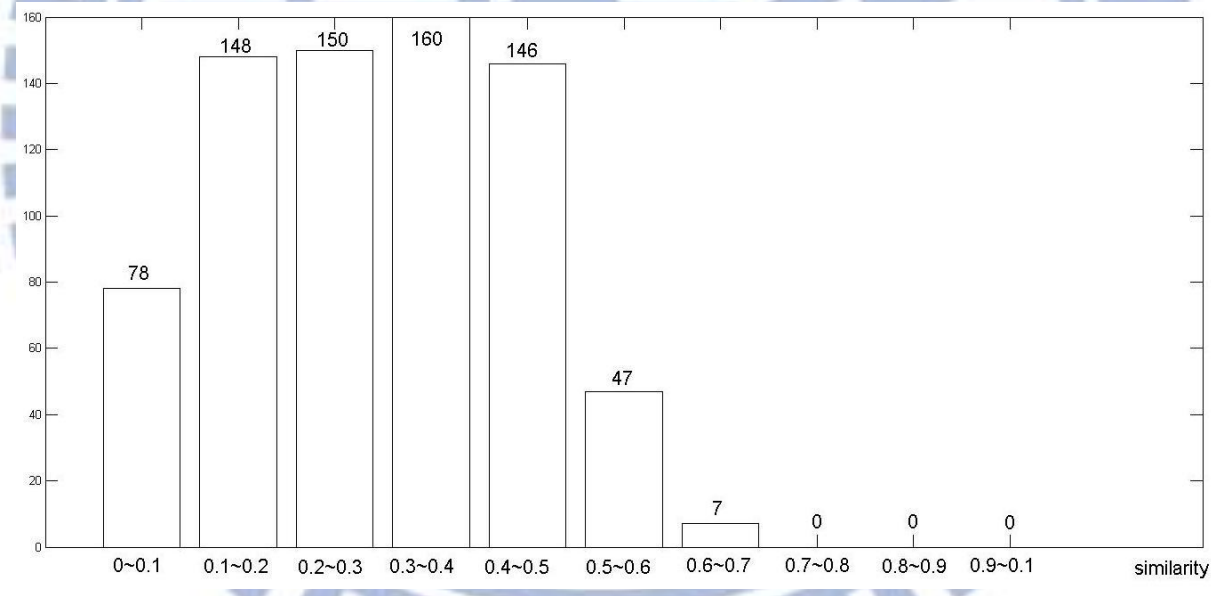


圖 5.9 使用 Q-gram 統計正常程式和惡意軟體的相似度

第六章

結論

在這篇論文中，我們提出了使用兩層的系統架構去偵測 Android 的惡意軟體。第一層先檢查應用程式使用的權限，其結果分為三類:正常、可疑和惡意。只有權限過濾器判斷為可疑的應用程式才需要經過第二層的靜態分析。在靜態分析中我們提出只使用方法來偵測惡意軟體，因為惡意軟體中一定會有某個或某幾個方法來實現惡意行為。但是一個應用程式有可能會有非常多個方法，我們推導出一個字串的長度區間，要詢問的字串的長度落在區間內才需要去計算相似度，若沒有落在長度區間內則不需要去計算相似度，因為相似度不可能超過臨界值。模擬的結果顯示，只使用惡意的方法來偵測同一類的惡意軟體會比使用整個應用程式的偵測能力還要好，但是因為一個程式會有很多的方法，所以比對起來也會比較慢。目前花費最多計算時間在計算編輯距離的部分，或許還有其他更好的字串比對演算法來改進計算時間。

參考文獻

[1] Gartner, Android Market Share Doubles, iOS Drops In Q3 http://articles.businessinsider.com/2011-11-15/tech/30400455_1_ios-iphone-smartphone-market

[2] Wikipedia, ”Symbian”, http://en.wikipedia.org/wiki/Symbian

[3] Wikipedia, ”Android”, http://en.wikipedia.org/wiki/Android

[4] Wikipedia, ”HTC_Dream”, http://en.wikipedia.org/wiki/HTC_Dream

[5] Lookout, “mobile threat report”, https://www.mylookout.com/mobile-threat-report

[6] Wikipedia, ”Android_(operating_system)” , http://en.wikipedia.org/wiki/Android_(operating_system)

[7] Wikipedia, ”Dalvik_(software)”, http://en.wikipedia.org/wiki/Dalvik_(software)

[8] Google, “Developer”, http://developer.android.com/index.html

[9] Eric Chien, “Motivations of Recent Android Malware,” Symantec Security Response

[10] Cesare, Silvio, Yang Xiang, “Malware Varant Detection Using Similarity Search over Sets of Control Flow Graphs,” in IEEE International Conference on TrustCom, Nov. 2011

[11] William Enck, Machigar Ongtang, Patrick McDaniel, “On Lightweight Mobile Phone Application Certification,” in Proc. ACM CSS’09 , 2009

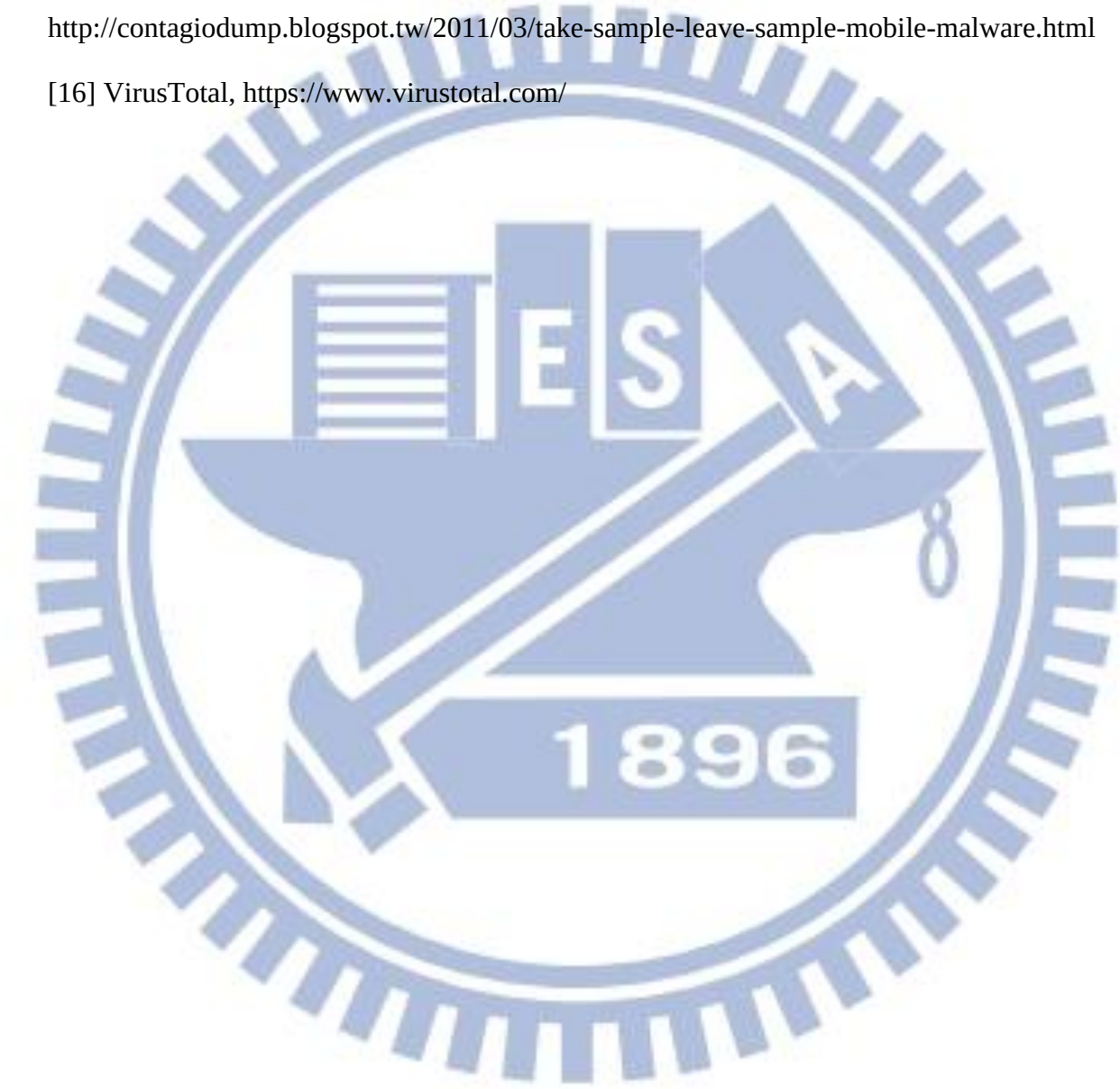
[12] Wei Tang, Guang Jin, Jiaming He, Xianliang Jiang, “Extending Android Security Enforcement with A Security Distance Model,” in IEEE International Conference on iTAP, Aug. 2011.

[13] WIRED, “Android Market 下載次數突破百億，付費 App 大降價”, http://wired.tw/2011/12/14/android-market-downloads-apps/index.html

[14] Anthony Desnos, “Androguard,” http://code.google.com/p/androguard/

[15] contagio- malware dump, http://contagiodump.blogspot.tw/2011/03/take-sample-leave-sample-mobile-malware.html

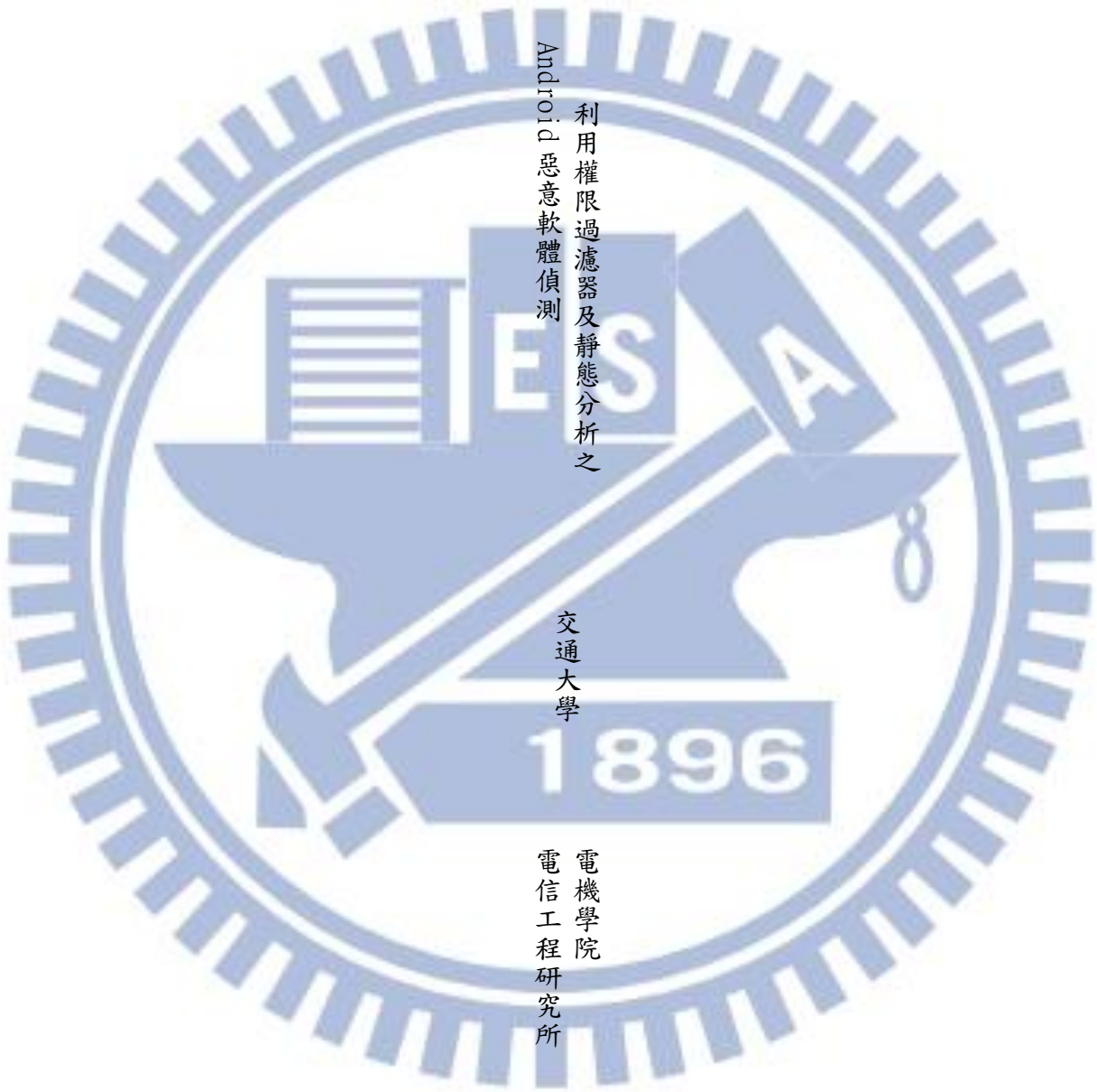
[16] VirusTotal, https://www.virustotal.com/



101

碩士論文

利用權限過濾器及靜態分析之Android惡意軟體偵測

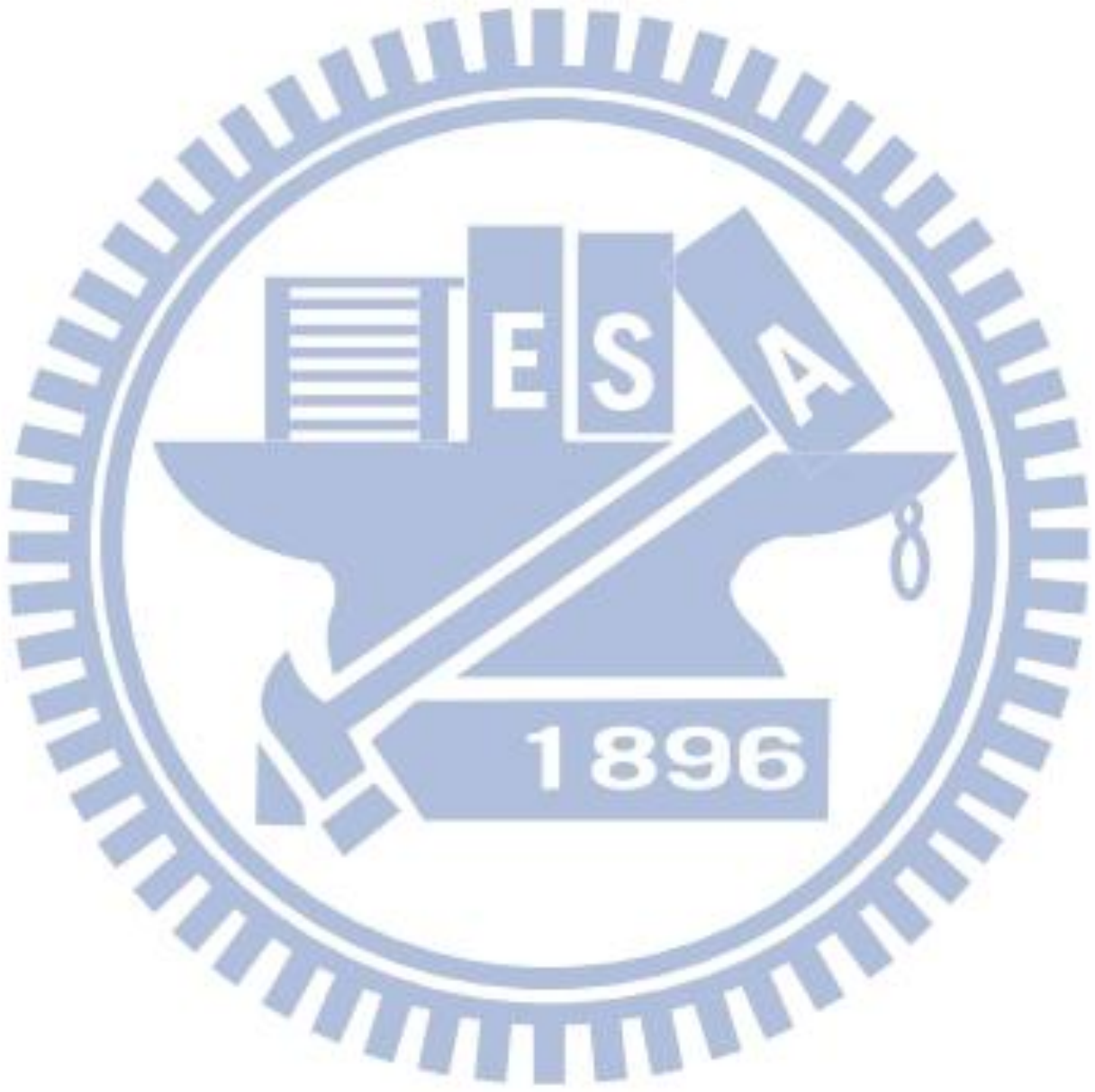


交通大學

電機學院

電信工程研究所

姜家安


E
S
A
1896