Journal of e-Business
第十七卷　第三期　2015 年 9 月（pp.375～392）

以汙染傳遞為基礎之行動軟體威脅行為偵測

陳嘉玫 [*,a]　林哲銘 [a]　歐雅惠 [a]　賴谷鑫 [b]

[a] 國立中山大學資訊管理所

[b] 中國文化大學資訊管理學系

摘要

隨著科技的進步，各企業組織提供客戶與員工無所不在的運算，線上服務也增加行動版，以提升競爭力與效率。為了方便使用與隨時連線，個人資料也因此儲存於行動裝置中，造成隱私資料洩漏之風險。動態分析需要隔離環境做分析，且分析時間較久，分析速度可能無法趕上惡意程式成長速度。此外，在分析過程中是否能成功觸發惡意行為，一直是動態分析的難題。本研究以靜態分析方式，以汙染傳播法追蹤程式碼資料流，利用惡意程式家族中歸納出威脅模式，再將追蹤之資料流與威脅模式進行比對，並回報符合之資料傳遞行為。實驗資料乃採用 19 個行動惡意程式家族進行測試。實驗結果證明本研究可以有效的偵測 Android APP 的惡意程式，正確率高達 91.6%。

關鍵詞：靜態分析、逆向工程、汙染傳播法、惡意軟體偵測

Detecting Mobile Application Malicious Behavior Based on Taint Propagation

Chia-Mei Chen[a]　Je-Ming Lin [a]　Ya-Hui Ou[a]　Gu-Hsin Lai[b]

[a]Department of Information Management, National Sun Yat-sen University

[b]Department of Information Management, Chinese Culture University

Abstract

Businesses provide mobile applications for ubiquitous computing. Personal information often is stored in mobile devices for convenience, which implies a potential information leakage risk for users as well. Dynamic analysis requires a controlled environment to observe the execution behaviors and it is time-consuming and computational intensive work. Some malicious behaviors are triggered in certain

[*] 通訊作者　電子郵件：cchen@mail.nsysu.edu.tw

DOI: 10.6188/JEB.2015.17(3).04

conditions or input sequences, which makes the detection more challenging. In this study, static analysis based detection method is proposed and defines threat patterns based on the literature review and malware families. The proposed taint propagation algorithm tracks the sensitive data flows and the detection system verifies if the sensitive information is released by the target software. The experiment adopted 19 mobile malware families and the results indicated that the proposed detection method can detect malicious behaviors efficiently with the true positive rate of 91/6%.

Keywords: Static analysis, reverse engineering, taint propagation, malware detection.

1. 緒論

現有的行動裝置的功能隨著科技的進步日益多元，網路、儲存等功能也讓現在的行動裝置較以往傳統手機有價值，也因此讓個人隱私像是個人文件、相片甚至是網路購物的憑證等，成為可能被盜取濫用的資訊。隨著運算能力的提升與行動通訊技術的發達，行動裝置變得更普及化、上網人口的比率也逐年增加。因此，行動裝置也出現以往個人電腦所面臨的威脅，像是木馬程式的攻擊、資料的盜取與阻斷式攻擊等，這使得行動裝置的安全問題變成資安領域的新課題（Schmidt et al., 2009; McAfee, 2013），惡意程式發展的速度已經嚴重影響網路的安全。

根據 Hot for Security 網站每月的分析（Arsene, 2012），現行的攻擊手法大多是利用偽造的應用程式（App）夾帶惡意程式，並且發布於第三方應用程式商店，第三方應用程式商店通常具有免費下載使用的誘因，因此手機使用者容易因為下載而觸發惡意程式執行（Nauman et al., 2010）。根據 F-Secure 2012 年第四季統計，目前大多數出現的行動惡意程式都是為了達到經濟上的利益（F-Secure, 2012）。像是取得受害者手機的簡訊收發權限，再發送簡訊到犯罪組織於電信業者登記的付費簡訊號碼來收取暴利，這種透過簡訊取得金錢利益的惡意程式是目前最常見的（F-Secure, 2012）。此外，有些行動惡意程式則為了進行情報蒐集會竊取受害人資訊，雖然較簡訊惡意程式少，但其為數也相當驚人（Spreitzenbarth, 2012）。

相較於傳統個人電腦，行動裝置上的安全防護必須考慮更嚴苛的資源利用，像是手機本身的電池限制與運算能力較弱的缺陷（Kim et al., 2008），因此，在手機上提供惡意程式偵測必須要在不過度影響使用者的使用為前提。多數靜態分析的偵測方式主要是以特徵碼為基礎的辨別與檢測（Dai, 2010），其缺點就是只能針對已知的惡意

程式進行偵測，對於變種的惡意程式或是修改程式結構的變形便無法正確偵測。動態分析的研究，主要是透過實際安裝與執行該應用程式，記錄應用程式運行中的各項行為，觀察該應用程式是否有進行惡意活動。由於會進行安裝與執行惡意程式的動作，本機難免會有機會受到感染並造成感染擴散等問題。加上需要執行應用程式一段時間才能取得分析結果，比較耗費時間。

以 Android 作為系統平台的移動裝置，通常具有開發的開放性與自由性，因此更受惡意程式開發者的青睞。也因為市場較其他作業系統廣大，使用者眾多，其影響程度更令人擔憂。Android 系統是以 Linux 的作業系統為基礎，其應用程式是使用 Java 語言撰寫，將其應用程式轉換為 Dalvik 格式（.dex）後在 Android 平台上的 Dalvik 虛擬器上執行，相較於其他語言，較容易進行逆向工程。相較於個人電腦、其他移動裝置平台等，其逆向工程後所獲得資訊較完整（Kang et al., 2007），利用 Android 平台的應用程式易於逆向工程的特性以進行靜態程式碼分析是較適當的，因此，本研究的系統環境便以 Android 系統為主要目標平台。根據 AndroLib（2014）統計，Android Market App 的數量就已達到 60 萬筆，尖峰時段曾在一個月內增加了四萬多筆 App，還不包含第三方市場的 App 數量，其中惡意程式的比例也同比例的大幅度增加。

本研究認為惡意程式從出現到被發現、擷取與完成特徵碼散播之間的空窗期是必須縮短的，行動裝置的安全性問題具有迫切性。所以，利用 Android 系統應用程式容易逆向工程之特性，以靜態的程式碼分析，配合汙染傳播法（Taint Propagation）定義出包含攻擊屬性與攻擊程序的威脅模式，來達到手機應用程式的潛在惡意行為偵測。以汙染傳播法為基礎的偵測方式，其優點便是可以增加監視模式，利用輸入的資料流確定攻擊者能控制的變數以檢測漏洞。相較於程式結構為基礎的偵測方式，必須先擷取共同結構的特徵，再比對各種程式程序特徵以計算其相似程度，程序繁複且存在較多的誤報。本研究所提出的方法可以克服動態分析可能觸發惡意程式以致影響本機的問題，同時將其他學者提出的靜態分析研究成果整合為威脅模式，成功分析出行動裝置應用程式是否具有威脅行為。

2.文獻探討

2.1　動態分析

在手機動態分析上，Shabtai et al.（2010）提出使用知識基礎時間摘要（Knowledge-based Temporal Abstraction）的方法偵測行動裝置上的惡意程式，該方法透過分析已發現的惡意程式行為，為每個種類的惡意程式制定出時間相關的實體參數、事件、狀態、虛擬參數與模式。實體參數就像是 CPU 使用率、傳送封包量等可

實際取得的訊息；事件就是由系統或使用者觸發的動作，如碰觸螢幕或安裝軟體等；藉由預先定義好的模式，如在沒有 App 擁有照相權限的狀態下，軟體觸發照相事件、使用網路等，可能該軟體就是屬於間諜程式；又或者沒有 App 擁有簡訊發送權限也沒有使用者互動的情況下，卻有簡訊發送的事件，則屬於簡訊濫用程式的模式。該研究提及的模式都基於使用者是具有資訊安全背景知識的，不會輕易賦予軟體系統權限，但在真實世界中，只要簡單的社交工程技巧，惡意軟體就能取得相關權限，如濫發簡訊的惡意程式將自己偽裝成簡訊管理程式，就可順利規避掉上述的防禦方式。

Shabtai et al.（2012）提出了 Host Based 的 Android 惡意程式偵測框架，運用了 Android 平台開放的特性，可以取得 Android 系統許多底層的資訊，如硬體、電源、網路、記憶體等使用資訊。並使用 k-Means、邏輯回歸、Histograms 等機器學習的方式，分析當行動裝置執行已知為惡意程式 App 與執行正常程式的軟硬體資訊差別，透過特徵選擇的方法選出較有相關的軟硬體資訊種類，作為監控的閥值。這種基於軟硬體資訊的偵測方式有其瑕疵，因為同樣是發送簡訊，可能就有發送給正常聯絡人與犯罪組織收費號碼的差別，單單基於軟硬體資訊我們無法做出上述的判斷，同樣的，傳送網路資訊除了正常使用網路外，也有可能是發送隱私資訊給攻擊者，從網路使用狀態流量、頻率，我們一樣無法做出上述的比較。

2.2 靜態分析

Apvrille and Strazzere（2012）利用 Android 容易進行逆向工程的特性，將爬蟲所取得的 App 進行逆向工程後，取得 APK 檔案原始碼，再從原始碼的 API 呼叫、Permission 請求、使用 Unix 作業系統指令、是否存有可執行檔與 zip 檔案、APP 產生國家以及是否帶有特殊網址等資料，利用 Heuristic 的方式進行分析，對每個情況給定一個權重，最終將所有權重加總後由分數高低來判斷是否為惡意程式，雖然最後分析的結果沒有很精準，但是該研究對 Android API 的整理完整，可作為靜態分析研究的參考。

Grace et al.（2012）使用兩個階段的方式來偵測大量的 App 是否含有惡意程式，其重點是找出是否含有針對 Android OS 系統漏洞的攻擊指令，若有則回報為高風險的 App，另外偵測 Android 中所有可能造成花費權限（Permission-group. COST_MONEY）的 API，觀察是否存在使用者互動的事件處理函式，即如果未經使用者互動而呼叫的 API 應歸納為中度風險的 App。

Wu et al.（2012）所提出的方式是將 APK 拆解為 manifest 與原始碼，先由 manifest 分析其基本行為，並從原始碼中取出 API 呼叫、Intent 與其他特徵，再利用 K-means 與 EM 演算法進行偵測模型的訓練。由於其方法是依照程式之特徵，進行機器學習的分群，僅能將分類一個應用程式是善意或惡意，但無法確實了解其被分類為

善意或惡意的原因，並且無法實際由程式之資料傳遞行為去判斷是否具有潛在威脅。

由於上述靜態分析多以 API 出現與否等方法進行偵測，未從其原始碼之資料流進行判斷，沒有利用到 Android 容易逆向工程的特點進行應用程式行為偵測，因此本研究基於上述靜態分析所用之特徵加以延伸，藉由汙染傳播法可以檢驗程式資料流行為的特色，提出一套新的行動應用程式行為偵測方法。

2.3　汙染傳播法（Taint Propagation）

汙染傳播法原本是用來檢測程式中攻擊者可用的潛在漏洞，將使用者視為攻擊者，藉由資料流的方式來分析攻擊者所能控制的變數是否會影響到具敏感性的函式，進而發動攻擊。汙染傳播法需要知道髒污資料進入點與資料在程式中的流程，取得程式原始碼後較容易進行，因此這方式主要應用在系統開發時，進行系統弱點偵測。

Chess and West（2007）提出進行汙染傳播分析時，需要用到的傳播規則要素如下：

(1) Source 髒污資料進入點：在傳統汙染傳播法中，髒污資料進入點所指的是外部資料進入系統的函式，如 read()、gets()、getenv()、getpass() 等函式都能算是 C 語言中的髒污資料進入點。

(2) Sink 敏感性函式：敏感性函式指的是不應與受汙染資料進行互動的函式，如 Java 中執行 SQL 查詢式的 Statement.executeQuery()，若直接與攻擊者所能操縱之受汙染資料進行互動，則可能有遭到 SQL injection 的風險；而以 C 語言為例，使用 strcpy() 進行字串的處理則有可能遭到 buffer overflow 的攻擊。

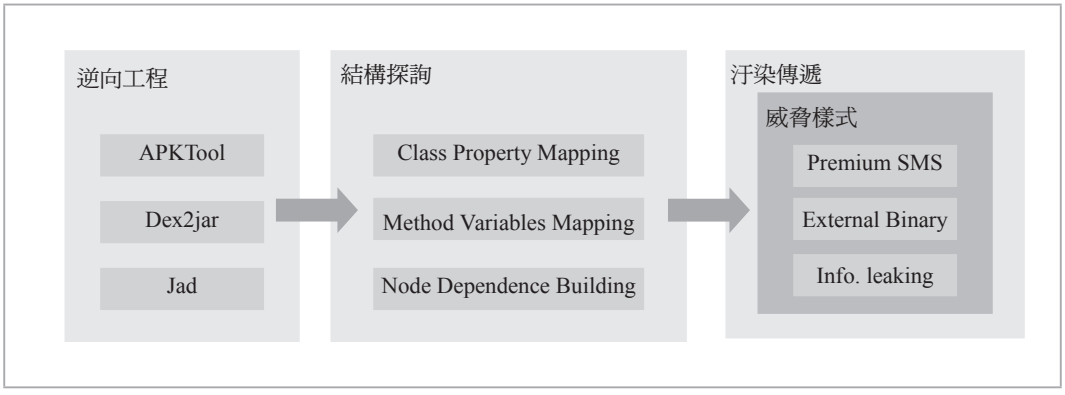
(3) Pass-through 汙染：汙染準則如常見的字串運算：字串串接、字串複製等，如果運算的參數是受汙染的資料，則可以定義運算後的字串也是受汙染的。如 Python 中的 + 號字串運算，假設 Python 的變數 a 為受汙染的資料，則變數 c 進行 c=a+b 的字串串接運算後，c 也被視為受汙染資料。

(4) Cleansing 淨化：淨化準則則是定義受汙染的資料在什麼時機下受汙染的狀態會消失，就 SQL injection 為例，最簡單的防禦方式是將所有使用者輸入的文字使用標準表示式來做驗證，濾除特殊字元等，而這個濾除的動作可以定義為淨化準則。

3. 研究方法

本研究的架構包含逆向工程、結構探詢與汙染傳播三個模組，如圖 1，首先利用

逆向工程模組將受檢測之 APK 檔案拆解並產生原始碼檔案，彙整為一份索引表轉交由結構探詢模組進行分析。接著，結構探詢模組將遍歷所有原始碼，同時依照原始碼之架構建立資料節點，再依原始碼之資料相依性建立節點鏈結。最後，結構探詢所產生之資料模型（Data Model）交付給汙染傳遞模組進行資料流追蹤分析。



▲ 圖 1　系統架構

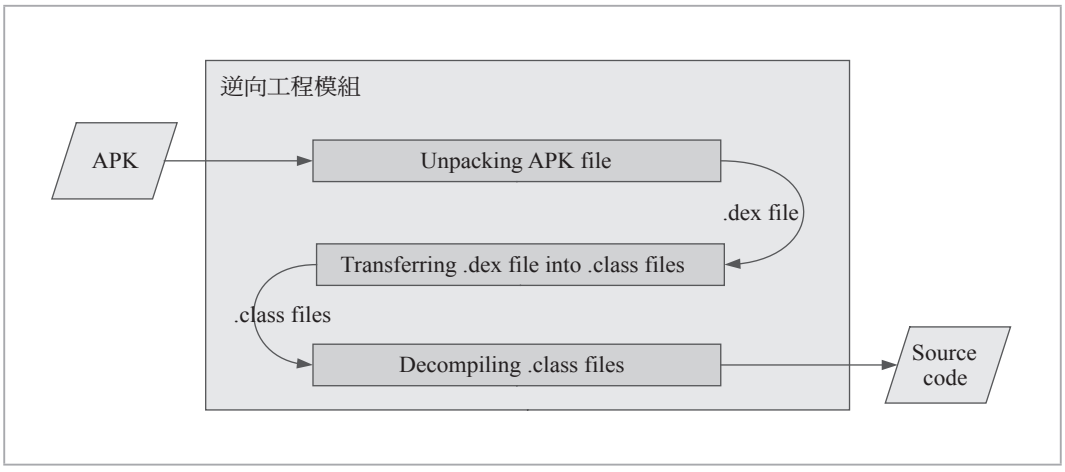
3.1　逆向工程模組

逆向工程模組的主要功能是將 Android 應用程式的 APK 檔還原為 Java 程式碼，如圖 2。本模組又細分為三個步驟，首先是 unpacking APK file 階段，利用 APKTool 工具進行反組譯，主要目的是將 APK 檔案反組譯成 .dex 文件。接著是 Transferring .dex file into .class files 階段，利用 Dex2jar 工具將 .dex 檔轉譯為 .jar 檔，並將 .jar 檔 unzip 後可以得到該程式中所有的類別文件（.class file），其目的是將其中附檔名為 .dex 的文件轉譯回 .class 文件。最後是 Decompiling .class files 階段，使用遞迴遍歷的方式，將所有類別文件反轉為 .jad 文件，即為原始碼，目的是將 .class 文件反編譯成 Java 原始碼，以利進行分析。

3.2　結構探詢模組

結構探詢模組的任務是依照原始碼將每一個 Class、Attribute、Method、Variable 建立與其對應的節點。

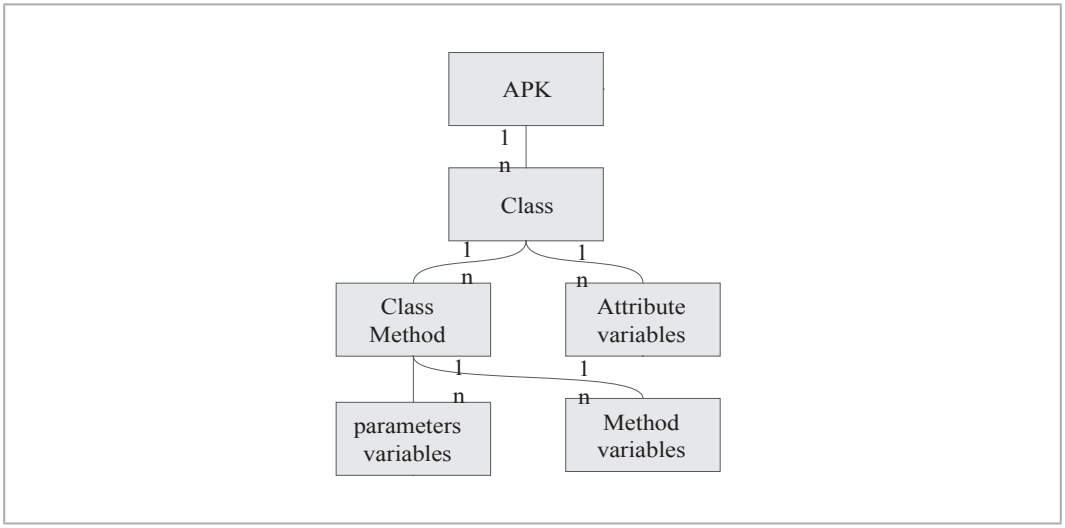
結構探詢模組首先會依據逆向工程模組所產生的 indexes 逐一對原始碼進行分析，其分析步驟可細分為 Class Property Mapping（CPM）、Method Variable Mapping（MVM）與 Node Dependence Building（NDB）三個步驟。



▲ 圖 2　逆向工程模組：逆向工程模組將 APK 文件反組譯後，將其 .dex 文件轉譯為 .class 文件，最後再將 .class 反編譯為原始碼

3.2.1 Class Property Mapping（CPM）

在 CPM 中系統使用正規表示式進行原始碼語意的判斷，當確定文件為類別檔，即建立與其對應之類別節點（Class Method）與屬性節點（Attribute variables），並將該類別檔中的方法抽出，建立與其相對應之方法節點，如圖 3 所示。在 CPM 步驟中，依據 .jad 文件為其建立節點，但並非每一個 .jad 都需要建立，interface 這種不具有實作方法的 .jad 文件就不需要建立節點。



▲ 圖 3　CPM 步驟將應用程式中的所有類別與類別屬性、類別方法建立相對應的節點

本研究再將 CPM 細分 3 個步驟：

(1) 分析 .jad 文件，找出次類別以分析原類別：由於所分析的 .jad 文件是經過逆向工程與反組譯所產生的原始碼，相較原始碼更為整齊一致，內部類別（Inner Class）與區域類別（Local Class）會被分割到獨立的 .jad 文件中形成次類別。由於 android API 大多使用匿名物件的方式撰寫，因此反編譯後，許多類別會延伸出次類別，這些次類別具有下列特徵：(a) 與原類別名字不同，(b) 包含原類別文件的部分片段，(c) 從匿名類別與區域 / 內部類別進行比較，因為匿名類別無法直接判斷何時為進入點，且匿名類別通常位於原類別中一般程式區塊中，不具有建構式之呼叫，也不具有變數類型特徵；而區域 / 內部類別則可以藉由宣告類型與建構式判定程式進入點。系統建立類別節點進行分析時，只需要對匿名類別分析其原類別即可，而區域 / 內部類別則需要分析所產生之次類別 .jad 文件。

(2) 建立類別節點與屬性節點：由於 interface 不具有實作方法，所以在進行逐行審視 jad 文件分析時，使用“interface” 作為關鍵字進行搜尋。一旦發現，隨即跳過此份文件不進行分析；若沒有，進行搜尋動作找出類別名稱。因為類別名稱必定跟在保留字“class”之後，當找到類別名稱後，對其區塊下的變數建立與其對應之類別節點，並記錄名稱、類別與所使用到之 Android 原生 API（Application Programming Interface）。

(3) 建立方法節點：將類別區塊下的方法宣告建立方法節點，由於匿名類別的宣告方式類似一般程式碼，同時該匿名類別只被該方法所使用，因此在方法區塊下所宣告的匿名類別一律併入該方法節點。

建立方法節點時，由方法宣告的該行程式碼可知其參數名與參數種類，分別為其參數建立變數節點並放入方法節點的 parametersList 屬性；同時所有屬於該方法的程式區塊全置入該節點的 rawCode 屬性中，以便 MVM 步驟進行分析。

3.2.2 Method Variables Mapping（MVM）

在 MVM 步驟中，系統會分析在偵測資料模組中每一個方法節點的 rawCode 屬性，並為所有於 rawCode 中出現的變數宣告建立變數節點，置入該方法節點的 variablesList 中。在方法區塊中，將所有接在 "return" 文字後的程式碼字串，置入該方法節點的 resultsList 中，但此時並非所有的方法皆完成節點建立。因此在 parametersList 中使用變數節點，而在 resultsList 中使用字串作為存值格式，以便在 NDB 步驟中做參照。

3.2.3　Node Dependence Building（NDB）

NDB 步驟會將所有變數節點依照資料相依性建立關聯。變數節點可能是類別下之屬性變數、方法下之參數與方法下之區域變數等。在 NDB 步驟中，再次逐行的審視每一方法節點的 rawCode 屬性，由於此時所有屬性、變數與方法節點皆已建立，系統可以用正規表示式過濾出每一行之行為，檢視經過編譯與反編譯的動作後，程式碼是否存在規律或錯序的特徵。

在 NDB 的步驟中，系統需要分辨單行程式碼屬於何種模式以決定如何建立節點關係，根據實際正常樣本與惡意樣本所反譯過的程式碼中，本研究歸納出函式呼叫模式、變數指派模式與流程控制模式以符合建立節點關連之需求。

在此階段中需要分割程式碼，並分別查找其所代表之節點，由於可能有同名屬性或變數的情況發生，尤以常見的字典檔名稱混淆技術，也就是將程式碼所有名稱皆以固定字詞替換。因此在查詢節點時，依照 java 的參照原則，需要以該段程式碼所存在位置，依照區域變數、類別屬性等順序查起。

3.2.4　建立關聯

欲將節點 A 與片段 B 建立關聯時，首先系統會先查詢片段 B 所代表之節點 B，如果該節點 B 存在，則須將 A、B 兩節點依照資料相依性建立關聯，將被相依的 B 節點放入 A 節點之 ancestorList 屬性，反之 B 節點放入 A 節點之 descendantLt 屬性。如果片段 B 之節點不存在，則將片段 B 放入節點 A 的 groundApiList 中，作為潛在的髒污資料進入點，以便在污染傳遞模組時做分析。

3.3　汙染傳遞模組

與一般使用汙染傳播（Taint Propagation）作弱點偵測的系統的最大差異在於本研究將汙染傳播用以偵測 Android 上惡意的 APP，將威脅來源定位為軟體本身，因此在套用 Taint Propagation 演算法時，針對髒污資料進入點（Source）的定義需做修改，同時需為不同的攻擊方式定義威脅模式，即特定的污染標誌（Taint Tag）只對特定的敏感性函式（Sink）有效。

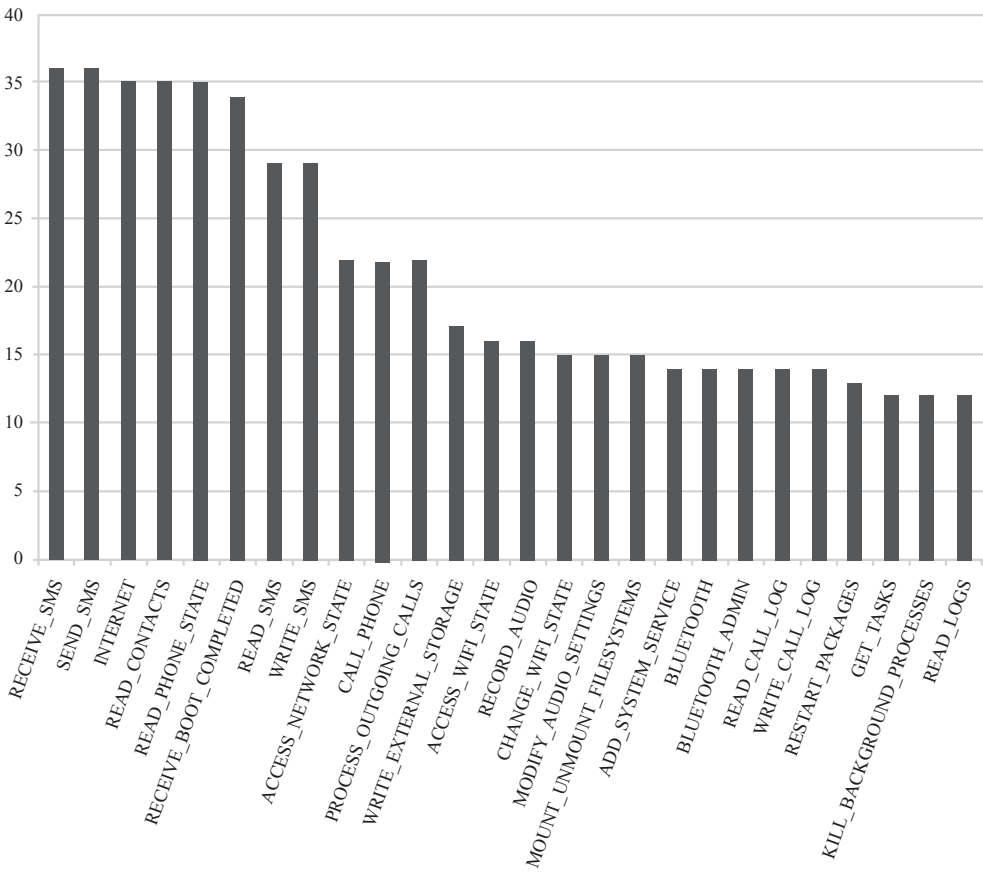
3.3.1　威脅模式

汙染傳播法具有髒污資料進入點、敏感性函式、汙染準則與淨化等四項主要傳播規則，本研究以資料相依性作為汙染準則。因為不同攻擊行為所使用之資料類型不同，使用之敏感性函式也不同，因此髒污資料進入點與敏感性函式的匹配組合則需針對特定威脅進行定義，本研究稱此組合為威脅模式（Threat Patterns）。

本研究除了使用傳統汙染傳播法外，還改良汙染傳播法的污染標誌為一組組合數

列，節點須具備威脅模式中所定義的所有髒污資料進入點之汙染標誌，並進入敏感性函式，才會回報該威脅。

依據 Spreitzenbarth（2012）所整理之惡意程式家族與行為特徵，本研究整理具有共同行為之家族原始碼，並根據中華電信資安監控中心於台灣駭客年會 HITCON 2014 的報告中指出，36 隻詐騙惡意程式最常使用的權限中，排名中以存取簡訊的權限問題最為嚴重，像是：允許程式監控即將收到的簡訊、允許程式發送簡訊、允許程式讀取簡訊、允許程式撰寫簡訊…等（如圖 4 所示）（劉清雲、施汎勳，2014）。再結合 Apvrille and Strazzere（2012）之研究成果，針對目前最為氾濫的 Android 惡意程式威脅像是付費簡訊攻擊、訊息竊取攻擊與嵌入外部執行檔，本研究定義 8 種威脅模式如表 1，並說明如下。



資料來源：劉清雲、施汎勳（2014）

▲ 圖 4　詐騙惡意程式最常使用權限的順序

▼ 表 1　威脅樣式總表

威脅模式類型		
付費簡訊攻擊 -String	付費簡訊攻擊 -int	付費簡訊攻擊 -url
訊息竊取攻擊 - 軟硬體資訊	訊息竊取攻擊 - 通話紀錄	訊息竊取攻擊 - 聯絡人資訊
訊息竊取攻擊 - 簡訊資訊	外部惡意執行檔攻擊	

(1) 付費簡訊攻擊：藉由分析付費簡訊攻擊的惡意程式，本研究發現大部分付費簡訊攻擊的惡意程式所發送的號碼是存在固定規則的，相較正常的簡訊應用程式所發送簡訊之目標號碼，因為是使用者輸入，比較不規律，兩者之間有顯著不同。本研究定義其基本模式為字串常數的付費簡訊攻擊模式，其行為模式為將網路資料轉為字串作為簡訊發送到目標號碼。

(2) 訊息竊取攻擊模式：其嚴重性僅次於付費簡訊惡意程式，其判斷方式是只要經由 TelephonyManager 類別的手機資訊函式所取得的資訊成為 HttpPost 類別 execute 函式的參數，則視為訊息竊取的攻擊，訊息流出的途徑可能不只 HttpClient。

(3) 嵌入外部惡意執行檔攻擊模式：許多惡意程式將惡意執行檔（binary file）偽裝為圖片檔、音訊檔等，存放於 APK 內的 assets 資料夾中，將其複寫至 Android 作業系統資料夾中，並使用 Android 中對作業系統直接下指令的方式運行惡意執行檔，此種惡意行為常見於取得手機 root 權限以進行越權操作。針對這種嵌入外部執行檔的攻擊手法，本研究定義了以下威脅模式（表 2）。

▼ 表 2　外部進入資料作為簡訊發送目標的付費簡訊攻擊模式

髒污資料進入點	AssetManager 類別的 getAssets 函式
	AssetManager 類別的 open 函式
	FileOutputStream 類別的 write 函式
敏感性函式	經由 getRuntime 所取得的 runtime 類別其 exec 函式

此一威脅模式用以偵測夾帶執行檔於 APK 文件中是否使用 InputStream 與 FileOutputStream 將該檔案從 APK 文件中抽出，最後再經由 getRuntime 所取得的 runtime 物件執行 exec 函式。

3.3.2 檢驗資料模型

得到結構探詢模組所產生的資料模型（Data Model）後，本研究接下來要進行分析的主要步驟有：掃描髒污資料進入點、傳遞汙染標誌與掃描敏感性函式。當變數節點的 groundApiList 被檢驗出具備某髒污資料進入點特徵，則在該節點之 taintTagList 屬性增加代表該特徵之整數。

經過掃描後，每一節點可能具備多種特徵標誌，再利用 NDB 步驟中將節點的相依關係，將其汙染標誌擴散傳遞予與其有相依性之節點。本系統使用 BFS 的方式進行汙染擴散演算法，排除重複擴散以增進效率。最後，依據所定義的威脅模式逐行檢驗每一方法節點下的 rawCode 屬性，對所有應用程式的敏感性函式進行檢驗。

4. 系統評估

本研究將針對所提出的行動惡意軟體偵測系統之效能做評估，分兩個層面：偵測率和時間效能評估。在偵測率效能評估方面，本研究根據 Zhou and Jiang（2012）等之分類研究做為評估系統偵測率的準則，並對照動態分析的結果，評估靜態分析過濾的時間效能。在時間效能評估方面，以惡意樣本共 1260 個做時間效能評估。

4.1 偵測率評估

本研究分析以付費簡訊攻擊、訊息竊取與嵌入外部執行檔三種類型為主的惡意樣本，其樣本主要是根據 Zhou and Jiang（2012）的分類結果、Symantec（2012）與 Spreitzenbarth（2012）所分類之行為，取出其中具有本研究所述攻擊行為的樣本家族共有 19 個家族，總計 252 個惡意程式。

本研究所使用的方法是利用應用程式的資料流對其行為進行判斷，一旦其原始碼中包含符合威脅模式的資料傳遞行為，加上多數是以字典檔改名作為混淆機制，故更能夠被本系統成功識別。以 Asroot 家族為例，該家族中的惡意程式會先利用 AssetManager 取得手機資源，藉由 InputStream 將二進位檔存入 byte 陣列中，並使用 FileOutputStream 將 byte 陣列中資料存入 file2 中，最後將 file2 路徑存入 as 字串中，並放入 runtime 中執行。本系統可以成功偵測其解壓縮包在 APK 中的外部執行檔並執行之行為，如圖 5 所示。

再以 HippoSMS 為例，如圖 6，該家族皆先定義一 sendsms 函式，該函式會利用 Android 所提供的 sendTextMessage 函式進行簡訊發送動作，將簡訊發送到 s 字串的號碼，而發送內容為 s1，並於另一函式 onStart 進行呼叫，當系統啟動該程式時便進行發送簡訊到特定號碼。因其符合本研究定義之付費簡訊的威脅模式，故本系統可以成功偵測出。

```
AssetManager assetmanager;
assetmanager = getAssets();
. . .
InputStream inputstream1 = assetmanager open(s);// 取得手機資源
byte abyte1[];
abyte1 = new byte[inputstream1.available()];// 存入 byte 陣列
. . .
File file2 = File.createTempFile("extract", "tmp", file.getParentFile());
FileOutputStream fileoutputstream = new FileOutputStream(file2);// 於硬碟建立檔案
file2
. . .
fileoutputstream.write(abyte1);// 寫入 file2
. . .
String as[] = new String[3];
. . .
as[2] = file2.getCanonicalPath();
process = Runtime.getRuntime().exec(as)'// 執行 file2
```

▲ 圖 5　Asroot 家族的資料傳遞行為

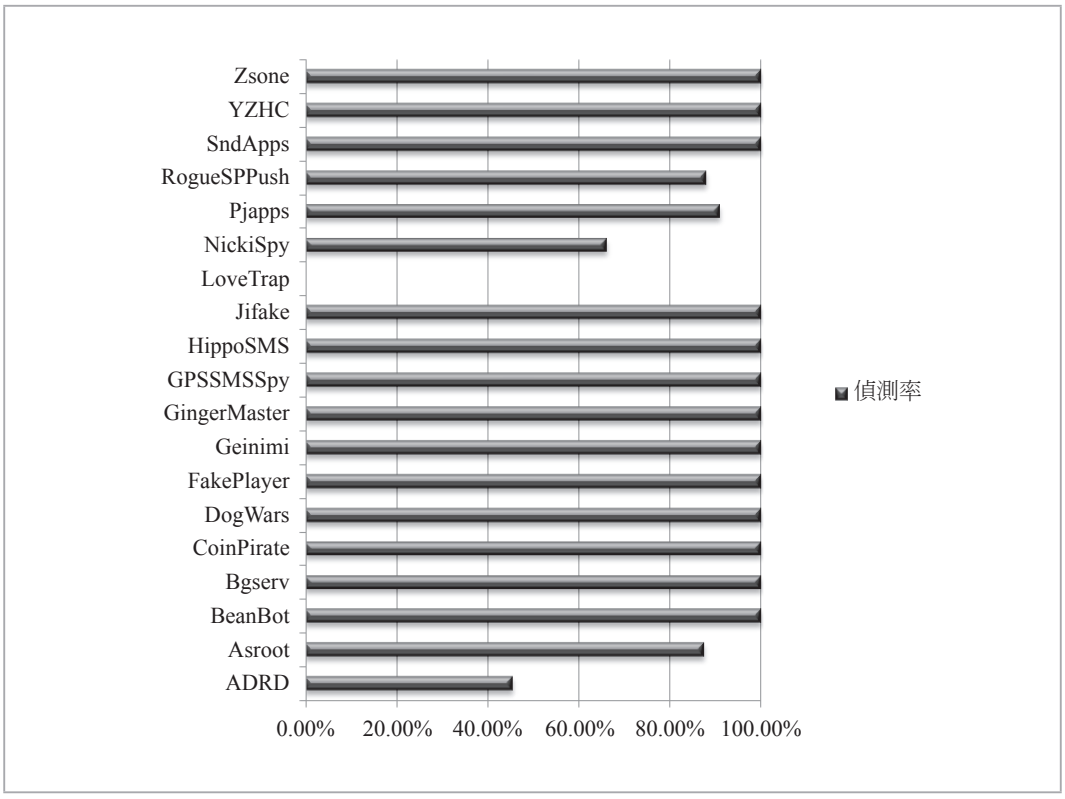
```
//function sendsms(String s, String s1, String s2, Context context)
. . .
SmsManagcr.gctDcfault().scndTcxtMcssagc(s, null, s1,
PendingIntent.getBroadcast(context, 0, new Intent("SMS_SENT"), 0),
PendingIntent.getDroadcast(context, 0, new Intent("SMS_DELIVERED"), 0))// 發送簡訊
```

▲ 圖 6　HippoSMS 家族的資料傳遞行為

系統分析的結果中，多數的惡意家族都能被系統成功偵測並達到 91.6% 的正確判斷率，其中一個家族未能被偵測出來的原因是因為該家族樣本數少，本研究只發現一筆資料，所以未能成功偵測該筆資料（如圖 7）。

4.2 時間效能評估

本研究將以動態分析為基礎之惡意行動軟體偵測系統（陳嘉玫等人，2014），作為對照組，做時間效能評估。該系統利用動態分析進行監控、紀錄應用程式的行為，透過行為分析的方式追蹤應用程式執行過程與存取敏感性資料的行為，作為判斷是否為惡意程式之基準。



▲ 圖 7　系統偵測惡意家族的分析結果

本研究對照其動態分析結果做為偵測時間效能的比較，發現動態分析在惡意行為的偵測效率上，對於大部份的惡意程式家族而言，能夠在平均 60 秒內有效的偵測出其惡意行為。但是，動態分析有其缺點，像是部份家族如 BaseBridge、DroidKungFu4 等家族，偵測的效率相對比較差，主要原因是這類家族是採取感染良性程式的方式散播，透過重新植入、改寫等方式感染良性程式並透過第三方平台來散播，惡意的程式本身還保有良性程式功能，所以在觸發到惡意行為之前時間較為不穩定。例如：在遊戲結束提交分數時才會觸發惡意行為，導致偵測時間被拉長。本系統以靜態分析為基礎之偵測方式，實驗顯示，本系統將所有惡意程式家族總計 1260 個樣本進行偵測的時間只需 124 分鐘，每一個代測軟體平均不到 6 秒時間即可完成靜態分析與偵測。靜態分析在時間效能上比動態的要好。

5. 結論

針對 Android 平台，本研究利用 Java 程式語言易於執行逆向工程之特性，提出

一套利用逆向工程後之原始碼資料流為依據的分析方式，不只能由原始碼辨識應用程式之行為，還能克服動態分析無法成功觸發惡意行為之瓶頸。

本研究整合之前其他學者在 Android 靜態分析上的成果，將 API 函式呼叫出現與否的特徵加入資料流辨識，以汙染傳播法追蹤程式碼資料流，由已發現之惡意程式家族中歸納出威脅模式，再將追蹤之資料流與威脅模式進行比對，並回報符合之資料傳遞行為。本研究以 19 個家族 252 個惡意樣本進行分析，實驗結果證明本研究之方法能夠成功識別具有惡意行為之應用程式，正確判斷率達 91.6%。

本研究所提出的靜態分析方式利用程式原始碼之相依性偵測出其行為，可以避免類似程式結構分析的繁複過程，透過定義完整的威脅模式驗證輸入的資料流，快速篩選出惡意程式的攻擊屬性與惡意行為。除了較以往靜態分析方式單純以特徵作為權重更精準外，還能提供動態分析需要花費更多時間方可得知的行為特徵。在系統驗證的分析中，也能成功辨識大多數應用程式之行為。

由於良性樣本的特徵較難歸納出模式，而且良性應用程式樣本多數會存在將機碼外洩的行為，會影響判斷。因此，本研究針對惡意程式的程式碼作為資料流辨識的主軸進行分析，也因此無法進行良性樣本誤判率的評估。未來研究可以更進一步對良性樣本的程式碼進行分類與分析，作為偵測的準則以增進系統的偵測效果。

參考文獻

陳嘉玫、江玟璟、歐雅惠（2014）。開放資料應用於行動惡意程式分析研究。*電子商務研究，12*（3），319-335。

劉清雲、施汎勳（2014）。行動應用程式檢測與鑑識。*台灣駭客年會（HITCON 2014）*，台灣，台北市。

AndroLib. (2014). *Number of new applications in Android market by month*. Retrieved May 15, 2014, from http://www.androlib.com/appstats.aspx

Apvrille, A., & Strazzere, T. (2012). Reducing the window of opportunity for Android malware Gotta catch'em all. *Journal in Computer Virology*, *8*(1-2), 61-71.

Arsene, L. (2012). *Android mobile malware report - 2012*. Retrieved October 15, 2013, from http://www.hotforsecurity.com/blog/android-mobile-malware-report-august-2012-3459.html

Chess, B., & West, J. (2007). *Secure programming with static analysis* (e-book). MA: Addison-Wesley Professional.

Dai, S. (2010). Behavior-based malware detection on mobile phone. *Proceedings of the*

6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM), Chengdu, China.

F-Secure. (2012). *Mobile threat report - Q4 2012*. Retrieved October 15, 2013, from https://www.f-secure.com/static/doc/labs_global/Research/Mobile%20Threat%20Report%20Q4%202012.pdf

Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. (2012). RiskRanker: Scalable and accurate zero-day android malware detection. *Proceedings of the 10th international conference on Mobile systems, applications, and services*, Ambleside, UK.

Kang, Y., Park, C., & Wu, C. (2007). Reverse-engineering 1-n associations from Java bytecode using alias analysis. *Journal of Information and Software Technology*, *49*(2), 81-98.

Kim, H., Smith, J., & Shin, K. G. (2008). Detecting energy-greedy anomalies and mobile malware variants. *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, Colorado, USA.

McAfee. (2013). *McAfee threats report: First quarter 2013*. Retrieved October 15, 2013, from http://www.mcafee.com/tw/resources/reports/rp-quarterly-threat-q1-2013.pdf

Nauman, M., Khan, S., & Zhang, X. (2010). Apex: Extending Android permission model and enforcement with user-defined runtime constraints. *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, Beijing, China.

Schmidt, A. D., Schmidt, H. G., Batyuk, L., Clausen, J. H., Camtepe, S. A., Albayrak, S., & Yildizli, C. (2009). Smartphone malware evolution revisited: Android next target? *Proceedings of the 4th International Conference on Malicious and Unwanted Software (MALWARE 2009)*, Montreal, Canada.

Shabtai, A., Kanonov, U., & Elovici, Y. (2010). Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method. *Journal of Systems and Software*, *83*(8), 1524-1537.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., & Weiss, Y. (2012). "Andromaly": A behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, *38*(1), 161-190.

Spreitzenbarth, M. (2012). *Forensic blog: Mobile phone forensics and mobile malware*. Retrieved October 15, 2013, from http://forensics.spreitzenbarth.de/2012/02/12/detailed-analysis-of-android-bmaster/

Symantec. (2012). *Security response*. Retrieved October 15, 2013, from http://www.symantec.com/security_response/

Wu, D. J., Mao, C. H., Wei, T. E., Lee, H. M., & Wu, K. P. (2012). DroidMat: Android malware detection through manifest and API calls tracing. *Proceedings of the 7th Asia Joint Conference on Information Security*, Tokyo, Japan.

Zhou, Y., & Jiang, X. (2012). Dissecting Android malware: Characterization and evolution. *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, California, USA.