Communications of the CCISA
Vol. 21 No. 2 Apr. 2015

雲端應用程式資訊流之動態污點傳播分析

王平[1] 林文暉 趙文傑
崑山科技大學資訊管理系
[1]pingwang@mail.ksu.edu.tw

摘要

現有大多數的惡意程式威脅分析涉及主機安全漏洞分析與發現網路攻擊有關的安全漏洞。近期雲端服務供應商(Cloud service provider，CSP)採用汙點檢驗（Taint checking，TC）作為應用服務上架前之檢測方法，以確保用戶之資訊安全。針對此一議題，本研究提出一以動態污點追踪方法，假設在開放式的行動網路下，隨意選擇惡意程式(汙染源)與系統漏洞(受害端)假設下，以檢測資訊汙染源與特定的手機應用程式及雲端服務漏洞之間的資訊流的污點傳播的行為與範圍。本文針對雲端應用程式提出一個資訊流之動態污點傳播分析模型，其整合加權生成樹(Weighted Spanning Tree)與污染標記(Taint marking)法以解決資安漏洞與動態污點追踪的問題;在測試案例中，以委託的三方開發之 Android 的應用程式執行汙點檢驗情境，分析手機應用程式、後台雲端服務內部模組與網路汙染源連線之資訊交換與風險。透過概率風險指標客觀評估每一可能的汙染路徑風險，協助防衛者評估惡意程式威脅的汙染範圍和估計造成的損失。經案例之污點路徑分析，所提的動態污點傳播分析模型可協助管理者提升網路安全的威脅分析之正確性。

關鍵字:：污點檢驗、行動安全、動態汙點傳播、加權生成樹

壹、前言

雲端運算利用網際網路提供資訊服務，它涉及的大規模資訊平台的部署，因此，在雲端資訊平台之商業資訊可能會成為網路攻擊的目標。檢測開放的網路入侵者的攻擊威脅分析(Attack threat analysis)，並提出有效的安全防禦機制對於雲端服務供應商(Cloud service provider，CSP)至關重要。防禦者已透過汙點檢驗方法（Taint checking，TC）作為應用服務(Application, APP)上架前之檢測方法，檢測程序漏洞檢測和分析經不可信任連線所帶來的被污染的威脅與資安隱憂，強化自行開發或委託第三方 APP 的風險評估。但現有風險分析集中於主機之資訊資產為目標，但 2014 年 9 月發生之 iCloud 明星私密照外洩事件，故雲端服務需要一個能分析主機內部可疑的模組間(intra-module) 及主機與網路外部主機的程序間(inter-app)之資訊交換的風險，例如 intra- module 之非法存取記憶體或緩衝區溢攻擊，以及 inter-app 之僵屍網路的惡意程式下載及分散式網路攻擊。

* 本研究接受國科會編號：103-N-358-NSC-R-040 經費補助。

污點分析不同於傳統的威脅分析技術，其重點是在靜態污點檢測程式之資安漏洞(Security hole)，動態污點技術則強調檢測 APP 執行時與網際網路連線的汙染行為，判斷資料流的數據是從用戶輸入或由外界不當輸入造成汙染(contaminated) [1]。一般而言，TC 常應用於評估的第三方開發軟體的特定的資安隱患，如 SQL injection 程式漏洞或緩衝區溢出等攻擊。

現有雲端服務程式之威脅分析常假設通過防火牆可過濾駭客的惡意連線，但駭客透過使用 Hypertext Transfer Protocol logging, kernel hacks, and library hack 等技術可輕易繞過防火牆為基礎的檢測。此外，現有威脅分析並未針對資訊連線的數據汙染做進一步分析，當發生資訊外洩時，無法做正確的損害管制(damage control) ，意即雲端服務供應商必須於短時間內回答(1)多少網路節點受到入侵，(2)各網路節點內多少資訊遭受汙染。通常，污點分析運用兩種基本方法以協助管理者決定雲端服務之數據是否受到汙染: (1)靜態污點分析（STA）：STA 是透過檢查程式原始碼，其中陳述(statements)視為執行的節點，流程的轉換視為存在一個邊(edge)可透過分析控制流程圖(control flow graph)的多條執行路徑，找出可能被污染的模組。(2)動態污點分析（DTA）：透過執行 APP 以調查程序運行的各項指令。特點是一次檢查單運行的單一路徑，並確定確切的污點值(taint values)。因為 DTA 技術增加細部的資訊流之活動的標示作業，一般認定 DTA 較 Sandbox 分析技術更能準確地監控運行的應用程序之資訊流細節，追踪發送敏感數據的污點是否從預設的敏感工作區，以利汙染源的反追蹤作業。

著名 DTA 分析包括 Newsome and Song[8]開發的 TC 自動化分析工具－TaintCheck，期能協助管理者檢測惡意軟體與分析程式漏洞，包括跨資訊系統間數據流的多種類型攻擊 [6]。Song [14]研發一個程式錯誤檢查工具，Panorama 用以檢測和分析惡意軟體，通過捕捉及分析未知惡意軟體樣本的行為，提供了程式碼分析和惡意軟體研究人員的寶貴協助。Kim [5]開發了 DTA 的工具來跟踪中分佈在企業間作業的程式的混合數據流。

實務上，DTA 的方法存在三個主要缺點：一、針對汙染源產生正確汙染源的特徵(signature of exploits, SOEs)，將有利轉化為 TC 自動化工具之偵測法則；但網路攻擊程序及程式執行路徑的多樣化，容易造成不易歸納汙染源特徵的問題。二、研究中沒有運用風險評價指標分析遭受污點源對防禦目標的可能性及風險。本研究是研究對象是不受信任的網路管道之資訊流進行標記與跟踪汙染源(Taint source)之傳播分析，針對問題一及問題二提出解決的方法，解決傳統動態汙染分析法對於行動資訊安全中汙染源特徵不易決定及判斷汙染風險(Taint risk)高低的問題。

有鑑於此，本研究參考圖學理論之加權連通圖進行污染路徑為基礎之動態污點傳播分析模式（Dynamic Taint Propagation Analysis, DTPA）的建立，以入侵受害端主機之攻擊封包作為分析數據，以加權生成樹演算法（Weighted Spanning Tree Algorithm, WSTA）與污染標記(Taint marking)方法，搭配污點檢測工具 Valgrind [17]、Comdroid[1] 與 Androguard[4]以推估不同的汙染源傳播途徑(taint paths)以判定特定資料是否受到汙染，並以數學分析模擬找出具有相對最小資源之隱私洩漏檢測模式以降低的誤報率

(False alarm rate)，回答不同攻擊情境下資訊汙染範圍與風險的關係，協助防衛者作出適當的防護決策判斷。

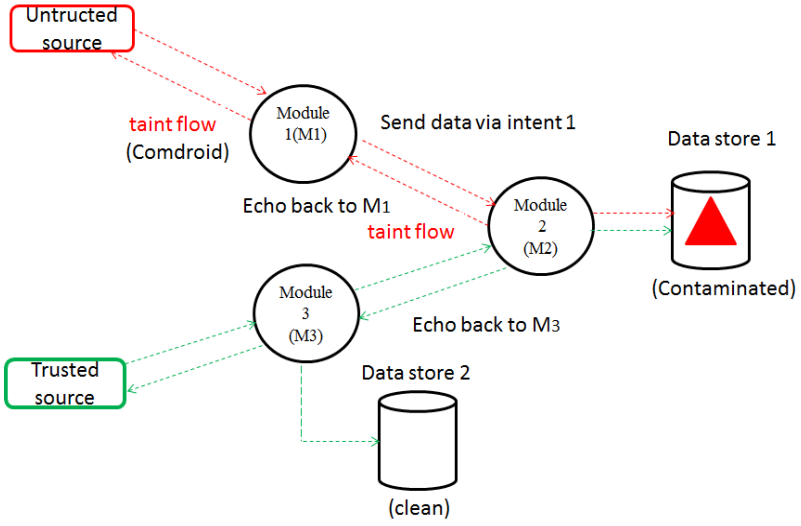
本分析模式可應用於自行開發或第三方開發的 APP 程式與可疑的網路外部主機間 inter-app 之資訊交換及 intra-module 的連線影響，以深入分析黑名單中不同汙染源的威脅與風險，探討污點分析的應用限制。

貳、以污點資料分析之數據流模式的建立

建立數據污點分析之數學分析模式主要目的是要回答下列問題:(1)汙染源的真正位置；(2)不同攻擊情境下之資訊汙染傳播途徑及傳播準則；(3)可模擬及分析各汙染源傳播途徑的汙染範圍，以協助防衛者作出適當的防護的決策。與先前研究的最大不同點是採用貝氏機率的信效度分析，以決策理論協助防衛者決定防護的資訊是否遭受汙染。

2.1 基本的想法

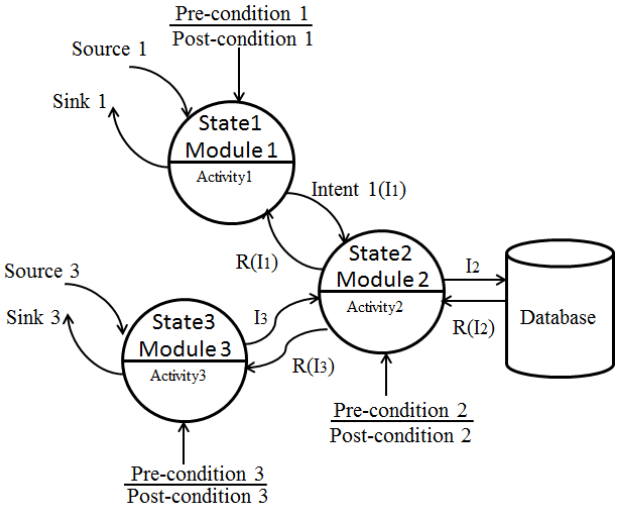
目前資安研究學者已投入研發程式污點分析技術[9][14][8][13]與分析機置的開發，例如 Panorama [14]，Valgrind [17]，TEMU[16]，以及 TAJ [11]，共同的作法是透過標記黑名單中污染輸入源端做為追踪資料流的模組所輸出的數據(intents)的作為分析的依據。數據污點分析基本想法是假設是汙染源及受害端存在一個通訊網路，運用連通圖形之加權生成樹演算法(Weighted Spanning Tree Algorithm, WSTA)及切割集(cutting set)理論，初始標記所有程序為未受污染，若汙染源的資料來自黑名單的未受信任管道(untrusted channel)所鏈結之程序及產出的汙染數據(Intents)將被標記為受污染(相連兩程序間的邊)，反之標示為未受污染(去除兩程序間相連的邊)如圖一。



圖一：污點分析輸入數據來源及資料流中模組呼叫以決定是否造成汙染

值得注意的是，在圖一中，現有的 TC 的方法並沒有考慮 APP 模塊中包括系統狀態轉換的先前條件(pre-condition)和後置條件(post-condition)。雖然接收模組有接收汙染資訊，因為防衛者採用的安全保護，如修復作業系統及編譯程式的漏洞及網路系統安裝防火牆的，將阻止污染資訊傳播的效果。反之，如果沒有修補安全漏洞將會引起污染資訊

的傳播。因此，本研究加入有限狀態機（Finite State Machine, FSM），在網路流量分析技術基礎上以適切地表示動態污點傳播分析的實況，如示於圖二所示。



圖二：加入有限狀態機於動態污點傳播分析

實務上，界定所有的受汙染的範圍牽涉及大量數據的數學運算和邏輯比對，當眾多污點源將造成產生大量分析負擔與成本，生成樹演算法可提供乙太網路產生無迴路(loop)鏈結之網路路徑分析，其已應用於分析 IEEE 802.1D 網路協定之網路鏈結管理協議，主要功能是確保兩網路節點之連線只存在單一條連通路徑。意即，數據污點就是找出受污染連通聯通網路之子生成樹，以界定受汙染的範圍及評估可能的風險。一般而言，生成樹 T 適用於圖論的數學領域，它一個連接的無向圖 G，包括無向圖 G 所有連通的頂點和邊，建構成一棵樹，但它不包含任何迴圈。對於 n 維超立方體關係圖，依據 KIRCHHOFF'S THEOREM 定理，生成樹 T 子樹的數量Q_n為 [18]

$$t(G) = 2^{2^n - n - 1} \prod_{k=2}^{n} k^{\binom{n}{k}} \qquad (1)$$

一般而言，對於任何圖形 G 中，T（G）使用基爾霍夫矩陣定理(Kirchhoff's matrix theorem)可在多項式時間內完成計算子樹的數量；亦即，對於一個給定的連接圖 G 有 N 個標記的頂點，讓 Λ_1，Λ_2，...，Λ_{N-1} 是它的拉普拉斯算子矩陣(Laplacian matrix)的非零特徵值， G 的生成樹的數目是

$$t(G) = \frac{1}{n} \lambda_1 \lambda_2 ... \lambda_{n-1}, \qquad (2)$$

亦即，對於一個給定的連接圖 G 有 n 個標記的頂點，$\lambda_1,\lambda_2,...,\lambda_{n-1}$ 是它的拉普拉斯算子矩陣(Laplacian matrix)的非零特徵值。此定理亦證明生成樹的數目等於連通圖 G 的拉普拉斯矩陣的共因數(cofactor)。

2.2 動態汙點傳播數學分析模式

本研究將整合現有應用程式的動態汙染檢測工具，針對不受信任的網路管道之資訊流傳播進行標記以跟踪汙染傳播途徑，協助管理者標識出受汙染範圍，以利管控新型網路威脅之風險。

汙點傳播分析可透過生成樹的建立的三個基本關鍵步驟：(1)建立與污染源連通之網路拓撲，(2)建立優化的生成樹以分析汙染傳播途徑，(3)決定個別服務網路之污染範圍。從資訊流分析的觀點，WSTA 可確保兩點連線間存在單一條連通路徑；數據污點就是找出受污染連通路徑之子生成樹，污染傳播造成的威脅程度的邊(路徑)將賦予較高的權重值，透過污染路徑的連結，解析各子生成樹找出數據流之汙染傳播範圍(spread of taint propagation) ，以界定受汙染的損失(loss)及評估可能發生汙染傳播的風險。理論上應優先處理高權重值的子生成樹，其代表污染傳播造成的威脅程度較高。考慮到隱私資訊洩露的損失，污染路徑的加權值由式(3)加以計算

$$w_j(t) = \frac{l_l(t)}{\sum_{j=1}^{n} l_j(t)}, \qquad (3)$$

實務上，污點傳播路徑（以下簡稱為污點路徑）的權重需要考量污點資料透過污點路徑的不斷蔓延，將可能會增加企業的損失。因此，污點路徑上的權重在新增的污點的路徑下，由式(3)須更新為

$$w_j(t+1) = \frac{l_l(t)}{\sum_{j=1}^{n} l_j(t)} + \Delta l_l(t+1), \qquad (4)$$

另一觀點是找出受污染連通路徑之子生成樹，列出各污染路徑之投入防禦資源；當專案時間限制或防禦經費有限時，投入防禦資源之最小生成樹(Minimum spanning trees, MST)可應用於達成最小資源的最大隱私洩漏防禦方案的擬定。

進一步說明，WSTA 應用於行動汙染之動態追蹤概念是透過資訊流之汙染傳播作為以連通路徑來形成生成樹，當生成圖上有部份模組未受汙染(不連通)時，則會產生有多棵「生成子樹」，當分析生成樹可考量汙染傳播造成的損失，於生成樹的每一個邊加入權重，生成樹的權重為樹上每一邊的權重總和，此時可計算出權重最大的生成樹必須優先防護。

WSTA 演算法實施是透過蒐集網路流記錄，將相同汙染源視為同一路徑序列，污染傳播造成的威脅程度的邊(路徑)將賦予較高的權重值，並統計回傳污染源連線深度及連線次數的多寡作為連通路徑之權重給予的考量依據，建立汙染源的特徵，提升網路威脅分析的精確度。此外，考量有限資安資源時，最小生成樹可用於選擇汙染範圍大(汙染傳播範圍大的生成子樹)但卻較低防護成本的路徑，優先編列預算執行資安修補以改善防禦。

步驟 1.污染源分析(Taint source analyses)

針對特定用戶網址進行連線或主動探測 Google 公告黑名單網站分析及比對汙染源連線的特定指令，搭配微軟研發之 Process Explorer 工具記錄感染過程與結果，並將可疑連線紀錄存檔，判定汙染源是善意或惡意。再使用工具 Dedexer 工具反組譯 DEX 檔

案，再運用 ComDroid [1]分析從 Dedexer 反組譯輸出檔案，並記錄潛在威脅的模組程式(components)和傳送資訊(intents)。

步驟 2. 動態汙點傳播分析(Dynamic Taint Propagation Analysis)

步驟 2.1 汙點傳播規則之關聯分析(Correlation Analysis for Propagation Rule)

針對行動客戶端及伺服端虛擬主機之應用程式，運用跟蹤污染資訊流之行為，定義特定汙染源之汙點傳播規則(PR)及汙染特徵(SOE)。實務上，汙點傳播規則可透過統計並關聯先前已發生汙染源連線中的資料封包、安全日誌、執行檔指令 PE(program executables)及 API calls 輸出間之關聯 [1]，選擇情節法則分析特定頻繁情節之比對以確認惡意指令集(Malicious Instruction Sets, MISs)；本研究透過交叉比對正常 App 之調用指令集與惡意程式調用指令集順序為基礎，我們首先定義汙點傳播規則如表一。

表一：汙點傳播規則案例

惡意程式	汙染源	模組名稱	惡意指令集的調用程序
lottery.apk	SubscriberId service url	com.lotsynergy	[Source:SubscriberId,Sink$_1$:Landroid/telephony/TelephonyManager/getSubscriberId] [source:service,Sink$_1$:Landroid/content/Intent] [source:url,Sink$_1$:Ljava/net/URL/openConnection

但執行汙點傳播規則過程中，很難正確定義表一中惡意指令集的調用程序，理論上可透過採用關聯法則(Association rules)、頻繁情節法則(frequent episode rule)及資料挖掘(data mining) 歸納出特定頻繁情節 (frequent episode)。特定頻繁情節 α 在事件序列 s 中出現的機率高代表此一威脅的風險程度較高，計算公式如下:

頻繁情節 α 在資安事件序列 s 中出現的機率：

於時間視窗 $w= (w,T_s, T_e)$透過特定時序性的項目集合進行比對，支持度 sup 是事件序列的所有行為事件中，特定情節(α)發生機率為一情節 α 在每個視窗中出現的次數除以寬度為 *win* 視窗總數：[6]

$$\sup(\alpha) = p(\alpha, s, win) = \frac{|\{\alpha \ \ occurs \ \ in \ \ w\}|}{|\{W(s, win)\}|} \qquad (5)$$

完成汙點傳播規則後，可歸納出特定汙染源的特徵；實務上針對每個污點源，可由惡意程式的行為分析中收集過個可能的汙點傳播規則作為 SOE 攻擊特徵的內容。

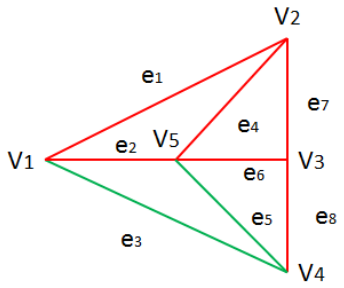
步驟 2.2 以加權生成樹分析汙染傳播途徑(Taint Propagation Paths Analyzed by the Weighted Spanning Tree Algorithm)

完成傳播規則定義，透過工具可自動關聯以追蹤資訊流之傳播流向，運用加權生成樹演算法以執行動態汙點追蹤分析，並標識受汙染範圍。WSTA 演算法由任一可能的汙

染源開始，執行連通圖的加權生成樹深度優先搜尋法(DFS Weighted Spanning Tree)，以深度優先搜尋，透過路徑蒐尋逐步檢查與頂點有連結的模組連成一個邊，標記經過的每一相鄰的邊，原則上取權重值最大的邊(代表污染傳播造成的威脅程度高)先連結，在逐一連結其他的頂點，但新增的邊不能形成迴路，否則不取。如此逐步檢查，每一次加入一個邊，當拜訪結束後，將標誌歷程的邊形成一棵樹，為一 DFS 加權生成樹。

步驟 2.3 決定污染範圍

去除與汙染源無連通的子圖即為污染範圍。例如圖三數據來源及資料流中去除與污染源無連通的子圖{e_3, e_5}，污染範圍之 DFS 加權生成樹為{e_1, e_2,e_4, e_6, e_7, e_8}。



圖三：污點分析決定污染範圍

2.3 受污染模組的風險評估

假設雲端服務存在一組安全漏洞，面臨威脅 i 風險評估項目 $j(j=1,...,n)$，防衛者執行雲端服務動態污點傳播分析以評估系統風險 r_i，本研究汙染風險參考資訊系統弱點資料庫(NVD)之通用弱點評估系統 (Common Vulnerability Scoring System，CVSS)之評估程序並考量兩個重要因素：(1)汙染傳播路徑之權重配分 (w_j): (2) 汙染傳播路徑之損失評分(s_{ij}):考量遭受特定威脅 i 之汙染傳播路徑 j 所造成的損失進行估算。評估分數顯示該弱點的風險程度，分數愈高表示風險程度越高。

$$r_i = \sum_{j=1}^{n} w_j \times s_{ij} \qquad (5)$$

假設威脅風險表（Threat Risk Matrix, TRM）是描述 APP 存在一組安全漏洞面對資安威脅 $_i$ (i=1,…,m)，污點源透過汙染傳播路徑 $_j$(j=1,…,n)所造成的損失評分為：

$$S = [s_{ij}]_{mxn} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1j} & \cdots & s_{1n} \\ s_{21} & s_{22} & \vdots & \vdots & \vdots & \vdots \\ \vdots & \cdots & s_{33} & \vdots & \vdots & \vdots \\ & \cdots & \cdots & s_{ij} & \vdots & \\ \vdots & \cdots & \cdots & \cdots & \ldots & \vdots \\ s_{m1} & s_{m2} & \cdots & s_{mj} & \cdots & s_{mn} \end{bmatrix}, \qquad (6)$$

其中 X_i 是 APP 存在的威脅所對應發生的資安事件，$p_h(X_i)$ 表示防衛者於資安檢測估計安全漏洞找到資安問題的機率。假設防衛者執行污點分析並記錄測試結果，確定一組安全漏洞所產生汙染傳播路徑 $j, j=1,...,n$，然後檢查威脅事件 $X_i, i=1,...,m$ 和汙染傳播路徑之間的聯連為 $P_h(X_i \mid S_j)$，其代表一個裝置發生的各種威脅事件後驗概率。因此，未知威脅的資安事件所對應檢測出安全漏洞影響汙染傳播路徑目之評分 $P_h(S_j \mid X_i)$ 可透過貝式決策規則來估計如下:

$$P_h(S_j \mid X_i) = \frac{P_h(X_i \mid S_j) * P_h(S_j)}{P_h(X_i)}, \qquad (7)$$

顯然 $P_h(S_j \mid X_i)$ 與 $p_h(X_i)$ 和 $P_h(S_j)$ 成正比。

完整污點資料分析模式的虛擬代碼整理如圖四：

Algorithm Dynamic Taint Propagation Analysis

1:begin--Initial phase;
2: Input a directed graph G = (V, E); V = {u, v_1, v_2,...,v_m}, E = {e_1, e_2,...,e_n}
3: m = |V|; n = |E|; taint source: u; v: taint sink;
4: if (m>1) then (taint source =TRUE);
5: initialize an m×n matrix with all '0' elements; $[]_{ij}$= 1 (boolean value indicating that taint source passes the $intent_{ij}$)
6: --Construct a set of taint graph G'.
7: loop {
8: if (taint source) then {
9: Taint_graph(v) {
10:─perform a DFS starting with a taint source, u
11: for (each unvisited neighbor u of v) {
12: if two neighboring modules have passed taint data then contaminated=TRUE;
13: if (contaminated) then
14: mark the edge from u to v;
15: assign a weight to the edge using Eq.(3) by considering the path of taint data causes the losses of information leaks
16: add the weights for each taint path for spanning sub trees;
17: list the taint paths for spanning sub trees;
18: assign a defense investment value to the edge;
19: find the Minimum Spanning Trees (MST) to decide the priority to guard the taint path;
20: write a list into propagation rule PR [u,v_j=1,…,n,]
21: write a list into propagation rule SOE [u, PR_j=1,…, n,]
22: else
23: traverse another taint source u to v in G;
24: call Taint_graph(u);
25: }--endif
26: }--end for
27: }--endif
28: }--end loop
29: output the MST and the taint routes of spanning sub trees for graph G
30: return (u, PR, SOE);
31: end

圖四：Algorithm TA for cloud APPs

參、系統測試與驗證

本研究將運用攻擊路徑分析模式以建立雲端應用程式之資訊流污點傳播與威脅分析，以下兩個實驗說明分析過程:

案例一:汙染源惡意行為之判斷

實驗前準備:分析工具 Comdroid 安裝

首先，至 ComDroid 工具載點下載(https://github. com/daniel666/comdroid/ blob/ master/Comdroid/ Comdroid-1.04/)以檢測 Android 應用程式的潛在漏洞。應用包括可於 Android 平台 Dalvik 虛擬機上運行的 DEX (Dalvik executable)檔案。

步驟 1. 污染源分析

先將 APP 安裝檔案 APK (Android Package)至 test 資料夾，再執行污點自動化分析工具 Comdroid，由 details.log 檔可檢視 APP 程式中之控制活動的紀錄及模組間傳輸之 intents，以推估特定模組及資料庫是否受到汙染。圖五說明分析可疑的連線運行，查詢 Comdroid 之 details.log 檔可檢視 APPs 間 activity、service、broadcast sink 的數量或搭配工具 logcat 進一步作詳細分析。點選 allActions，可檢視從 activity intent 中所執行的 actions，如圖六。其他工具如 FlowDroid, Epicc 與 TaintCheck 分析可獲得類似汙染源的特徵資訊。

```
Activity Sinks: 112
Service Sinks: 3
Broadcast Sinks: 15
        sendBroadcast Sinks: 15
        sendOrderedBroadcast Sinks: 0
        sendStickyBroadcast Sinks: 0
        sendStickyOrderedBroadcast Sinks: 0

Print exposure summary
*******************************************************
Exposure to Components
Malicious Activity Start: 0 of 13
Malicious Service Start: 0 of 2
Malicious Data Injection: 5 of 9
Protected Activity Vuln: 0
Protected Service Vuln: 0
Protected Receiver Vuln: 4

Exposure to Intents
Activity Hijacking: 70 of 112
Service Hijacking: 0 of 3
Broadcast Exposure: 15 of 15
        Intent Sniffing: 15
        Intent Theft: 15
        Result Modification: 0
```

圖五：Comdroid 檢視雲端服務主機與網路主機間之資訊交換風險

圖六：分析主機間之 activity intent 中所執行的 action

步驟 2. 動態汙點傳播分析

步驟 2.1 汙點傳播規則之關聯分析

APP 安裝包文件 APK 到測試文件夾中，然後執行 TC 工具，ComDroid 和 API_Monitor 分析收集給追查汙染來源的傳遞信息，檢查 API calls 輸出，啟動的服務(Service)，活動(Activity)和接收器和廣播(receiver and broadcast)污點模組間通信的路徑。一個污點源的分析案例結果整理如表二。

表二：污點源的惡意行為分析

Taint source	Module	Operation	Intent receiving communication			
			Activity sink	Service sink	Broadcast sink	Pending intent sink
DDream Light com.Beauty.Leg-1.apk	com/Beauty/Leg/lightdd/c	handle-Message	1		1	
	com/Beauty/Leg/lightdd/Receiver	onRe-ceive		1		
	com/Beauty/Leg/lightdd/CoreService					
	com/Beauty/Leg/lightdd/a/	onStart			1	
	com/Beauty/Leg/SexyImages/a	Run		1	1	
	com/Beauty/Leg/SexyImages		1			

透過 N 元模型(N-gram modeling)方法搭配 Comdroid 及 API_Monitor 以統計 30 個 APP 之汙染資訊之 MIS 惡意指令集出現的相對頻率，詳如表三；透過交叉比對正常 App 之調用指令集與惡意程式調用指令集順序做基礎，定義汙點傳播規則如表四。

表三：汙染源之調用惡意指令集清單

API	Total	activity sink	Service sink	broadcast sink	pending intent sink
Ljava/lang/Void	1				1
Ljava/util/List	1		1		
Ljava/lang/Class	1		1		
Landroid/net/Uri	1	1			
Landroid/view/View	1	1			
Landroid/view/MenuItem	1	1			
Landroid/graphics/Bitmap	1			1	
Landroid/content/ServiceConnection	1		1		
Landroid/os/Message	3	3			
Landroid/webkit/WebView	3	3			
Landroid/content/DialogInterface	3	3			
Landroid/app/Activity	5	5			1
Landroid/os/Bundle	11	8	2		5
Ljava/lang/String	14	8	1	1	3
Landroid/content/Intent	17	6	7	1	8
Landroid/content/Context	20	4	7	1	1

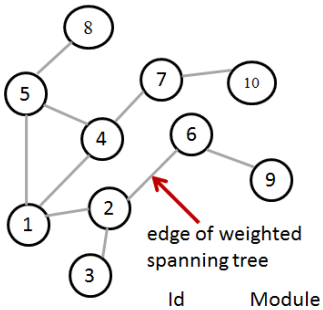
表四：汙點傳播規則案例

惡意程式	汙染源	模組名稱	惡意指令集的調用程序
SMSsender. apk	sms_sent SubscriberId	com.c101421042723	[Source:SubscriberId, Sink1:Landroid/telephony/TelephonyManager/getSubscriberId
vksafe.apk	File io, sms	com.vksafe	[source:file io, Sink1:java/io/BufferedWriter/write, Sink2:com/vksafe/WatcherService/openFileInput,Sink3:Ljava/io/BufferedReader/readLine] [source:sms,Sink1:Landroid/telephony/SmsManager/sendTextMessage

步驟2.2以加權生成樹分析汙染傳播途徑

以生成樹演算法確認汙染範圍:由任一可能的汙染源端開始,透過路徑蒐尋逐步檢查與頂點有連結的模組連成一個邊，原則上取權值最大的邊(代表回傳污染源連線次數較高)先連結，在逐一連結其他的頂點，但新增的邊不能形成迴路，否則不取。如此逐步檢查，每一次加入一個邊，形成受污染連通聯通網路之子生成樹，以利分析受汙染的範圍及評估風險。以 APK:Geinimi-Banking Trojan 入侵雲端服務為例（資訊取自 www.ipay.com.cn），分析 Android 應用程序間的通信；先至下列網址 http:// contagiominidump.blogspot.tw/2011/10/ geinimi-banking- trojan-wwwipaycomcn.html 下載樣本，以生成樹分析雲端服務資訊流汙染之活動圖。實驗流程根據 comdroid detail.log 檔將 sink 模組進行連線，結果如圖七，假設組織的資安預算有限，依據可能投入防禦資

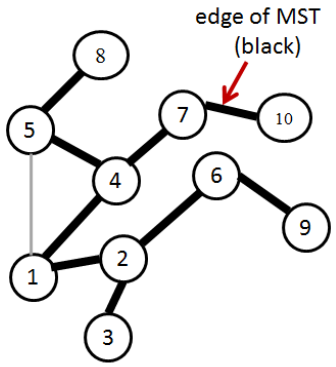
源訂定汙染連線權重，進行 Kruskal 最小生成樹的產生，去除與汙染源源無連通資訊流的子圖即為最小資源防禦的污染範圍，包括與汙染源聯繫 activities 及 intents，受汙染的範圍如圖八。



Source	Sink	Weighting
1	2	5.0
1	4	3.0
1	5	4.0
2	3	1.0
2	6	10.0
4	5	3.0
4	7	6.0
6	9	10.0
7	10	3.0
8	5	4.0

Id	Module name
1	com.geinimi.AdActivity
2	com.lotsynergy.Main
3	com.lotsynergy.Contacts
4	com.geinimi.AdService
5	com.geinimi.custom.GoogleKeyboard
6	com.lotsynergy.Protocol
7	com.geinimi.AdServiceThread
8	com.geinimi.AdServiceReceiver
9	com.HttpConnect
10	com.geinimi.custom. Ad001_01

圖七：以生成樹分析雲端服務資訊流汙染之活動圖



Source	Sink	Weighting
1	2	5.0
1	4	3.0
1	5	4.0
2	3	1.0
2	6	10.0
4	5	3.0
4	7	6.0
6	9	10.0
7	10	3.0
8	5	4.0

圖八：以Kruskal最小生成樹分析最小資源能防禦的污染範圍

步驟2.3決定污染範圍

運用工具 Comdroid 列出特定 intents 所有連線之的模組名稱（圖九），管理者再透過反組譯以檢視每個 activity，以確認是否被汙染，可與 androidmanifest. xml 配置檔與所有 activity 做比較加以確認。

圖九：列出特定intents所有連線之的模組名稱

步驟3. 受污染模組的風險評估

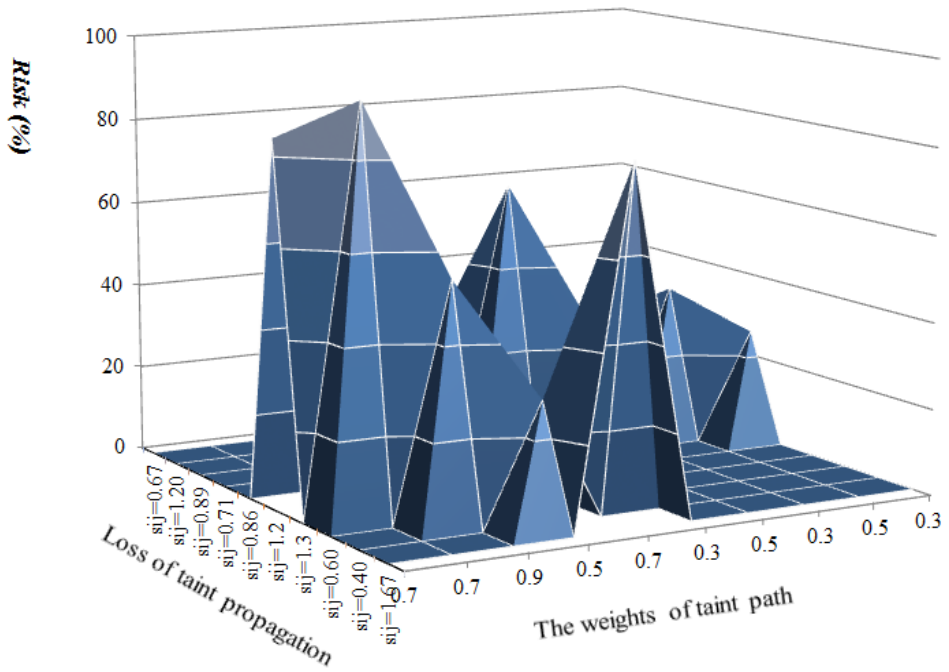
為評估在不同污點源的污點傳播路徑和風險，選擇不同位置的污點源、污點傳播路徑與防禦資源進行分析。依據公式(3)、公式(5)及公式(6)中s_{ij}定義可知，風險值會隨著污點源之污染路徑的加權(w_i)及汙染傳播路徑長度(PR_{ij})兩者數量增加而上升，若降低的防禦資源則污點傳播的損失會上升，如式（8）所示；此外，傳播路徑長度越長對於系統危害性也變為越大，亦即在相同汙染源之情況下，顯示汙染傳播路徑的長短對於資訊系統的危害有所影響。

$$s_{ij} = \forall_j \frac{PR_{ij}}{d_j}, \tag{8}$$

實驗污點源樣本包括從 Contagio 部落格所取得的惡意程式樣本，不同的污點源在執行的實驗結果如表五和圖十所示。表五及圖十顯示污點源 i 透過汙染傳播路徑 $j(j=1,\cdots,n)$ 所造成的損失(s_{ij})與傳播路徑長度(PR_{ij})數值成正比，但與單一污點源威脅投入防禦資源(d_i)成反比。

表五：不同污染源的傳播路徑和風險

	w_i (0-1)	PR_{ij} (0-10)	d_j (0-10)	s_{ij} (0-10)	Taint risk (%)
TS_1	0.3	4	6	0.67	20
TS_2	0.5	6	5	1.2	60
TS_3	0.3	8	9	0.89	26.7
TS_4	0.5	5	7	0.71	35.7
TS_5	0.3	6	7	0.86	25.7
TS_6	0.7	6	5	1.2	84
TS_7	0.7	4	3	1.33	93.3
TS_8	0.9	3	5	0.6	54
TS_9	0.7	2	5	0.4	28
TS_{10}	0.7	7	6	1.17	81.7



圖十：不同污染源的傳播路徑所產生的風險

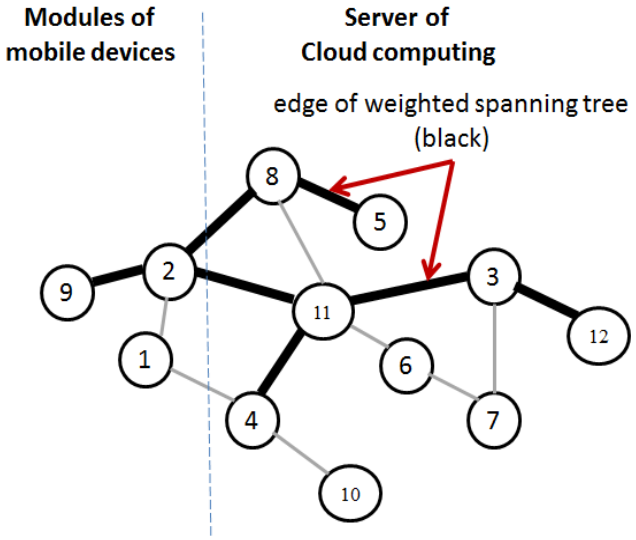
案例二:汙染傳播行為分析與風險評估

－加入有限狀態機(FTM)為基礎之污點傳播分析

案例一未考量 APP 之各模組的系統狀態，包括先前狀態及事後狀態，雖然模組接收遭汙染的內容，但若有系統防護或即時作業系統的安全漏洞修補，將會產生不同的污染傳播結果；反之管理者若未修補安全漏洞會則將造成威染的傳播。故案例二將針對案例一加入有限狀態機(FSM)以動態模擬污點傳播，協助資訊流汙染分析；值得一題，運用有限狀態機分析污染傳播將在不同的防護情況下會產生不同的污染傳播結果。假設資訊系統有安裝網路防火牆與入侵偵測系統，管理者並未定期落實漏洞修補，當發生網路攻擊事件時，分析汙染傳播行為如圖十一；圖十一指出分配給用於移動設備的路徑的權重（相對損失的嚴重程度）小於分配至雲端運算的服務器路徑的權重。

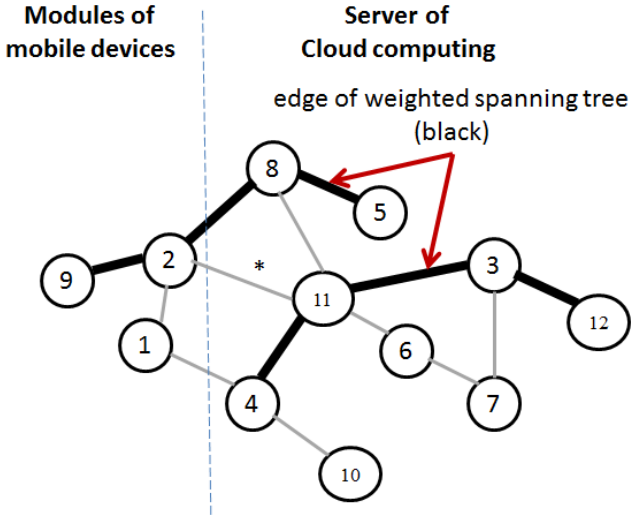
假設管理者發現汙點傳播路徑，進行修補模組 2 至模組 11 間資安漏洞，並成功阻止了汙點傳播，則汙染傳播行為修正如圖十二所示。完成確認系統漏洞和感染路徑後，跟踪惡意網站之間的污點路徑可找出汙染源、系統漏洞與應用程序的對應關聯為 SOE。SOE 格式為定義為（ID，TS_i，PR_K，$PR_{K+1,}$ PR_{K+2}，...），SOE 案例如表六。

Source	Sink	Weighting	state transition
9	**2**	**2.5**	**9->2 Yes**
2	1	2.0	2->1 No
2	**8**	**3.0**	**2->8 Yes**
8	11	2.5	8->11 No
2	**11**	**6.5**	**2->11Yes**
1	4	3.5	1->4 No
8	**5**	**5.8**	**8->5 Yes**
11	6	3.9	11->6 No
11	**3**	**6.5**	**11->3 Yes**
11	**4**	**7.5**	**11->4 Yes**
4	10	4.5	4->10 No
6	7	4.0	6->7 No
3	7	4.5	3->7 No
3	**12**	**8.5**	**3->12 Yes**



圖十一：有限狀態機為基礎之資訊流汙染分析
(安裝防火牆、入侵偵測系統，未修補資安漏洞)

Source	Sink	Weighting	state transition
9	**2**	**2.5**	**9->2 Yes**
2	1	2.0	2->1 No
2	**8**	**3.0**	**2->8 Yes**
8	11	2.5	8->11 No
***2**	**11**	**6.5**	**2->11** No
1	4	3.5	1->4 No
8	**5**	**5.8**	**8->5 Yes**
11	6	3.9	11->6 No
11	**3**	**6.5**	**11->3 Yes**
11	**4**	**7.5**	**11->4 Yes**
4	10	4.5	4->10 No
6	7	4.0	6->7 No
3	7	4.5	3->7 No
3	**12**	**8.5**	**3->12 Yes**

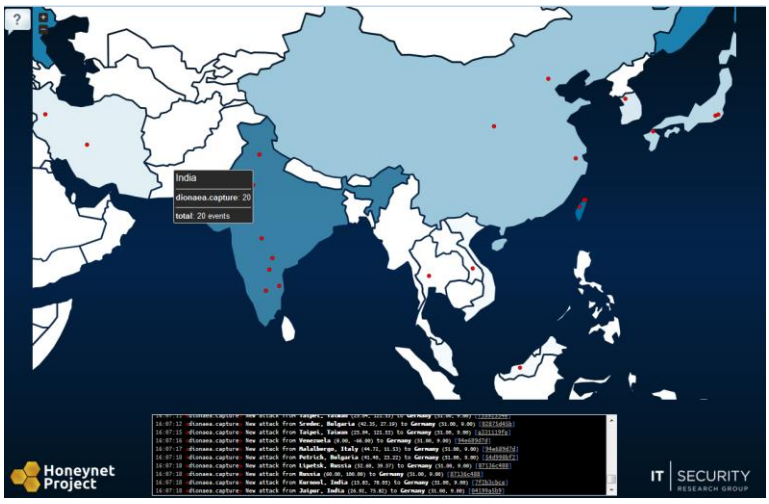


圖十二：有限狀態機為基礎之資訊流汙染分析
(安裝防火牆、入侵偵測系統，已修補資安漏洞)

表六：汙染源的特徵(SOE)

ID	Taint source	Package name	Propagation rule
1	com.c101421042723	SMSsender.apk	[Source:SubscriberId, Sink1: Landroid/telephony/TelephonyManager/getSubscriberId
2	com.lotsynergy	lottery.apk	[Source:SubscriberId,Sink1:Landroid/telephony/TelephonyManager/getSubscriberId] [source:service,Sink1:Landroid/content/Intent] [source:url,Sink1:Ljava/net/URL/openConnection
3	com.vksafe	vksafe.apk	[source:file io,Sink1:java/io/BufferedWriter/write, Sink2:com/vksafe/WatcherService/openFileInput Sink3:Ljava/io/BufferedReader/readLine] [source:sms,Sink1:Landroid/telephony/SmsManager/sendTextMessage

再透過動態汙點傳播分析工具追蹤可疑遠端汙染源連線，以攻擊圖(attack graph)製作汙染源以關聯汙染源相關的連結紀錄。此外，追蹤可疑汙染源網址的攻擊路徑，以 Honeynet Map 描繪汙染源及受感染虛擬主機位置的對應，定期監控汙染源及擬定防護策略，支援管理者主動式預防安全管理，如圖十三。

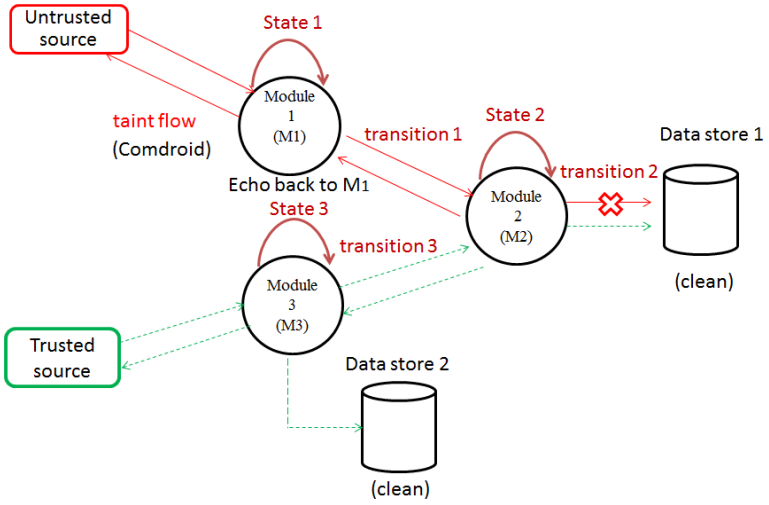


圖十三：以 Honeynet Map 描繪可疑汙染源之連線

肆、討論

本節運用有限狀態機探討(1)考量系統防護的狀態(Stateful)及(2)未考量系統資安防護的現狀(non- Stateful)之汙點傳播規則比較分析。首先須設定須先調查各模組發生事件及可能改變的系統狀態表以了解面對資安事件之危害：執行 Stateful 與 non- Stateful 之汙點傳播分析比較如下：

圖一加入有限狀態機，在網路流量分析技術基礎上以適切地表示動態污點傳播分析的實況。值得注意的是，運用基於的污點傳播分析滿足不同的資安配置的現況，並將會產生不同的污點資訊傳播結果，圖十四說明資訊系統可能在良好的資安防護下，圖一中模組 2 污染的擴散被有效阻止。從圖一和圖十四比較可知，污點分析加入有限狀態機於模組狀態的轉換是更適合於 TC 分析。



圖十四：有限狀態機為基礎之資訊流汙染分析
(完成資安漏洞修補並安裝防火牆、入侵偵測系統)

由圖十四可知汙點傳播準則會因為加入有限狀態機，而產生以下不同的惡意指令傳播的順序變化：

If ([pre-condition]) then PR_1= [MIS_1→…→MIS_n]
If ([pre-condition] and [post-condition]) then PR_2= [MIS_1→…→MIS_n]
If ([pre-condition] and not [post-condition]) then PR_3= [MIS_1→…→MIS_n]
If (not [pre-condition] and [post-condition]) then PR_4= [MIS_1→…→MIS_n] (9)
If (not [pre-condition] and [post-condition]) then PR_5= [MIS_1→…→MIS_n]
If (not [pre-condition] and not [post-condition]) then PR_6= [MIS_1→…→MIS_n]
If ([pre-condition] or not [post-condition]) then PR_7= [MIS_1→…→MIS_n]

進一步比較兩者方法的特色、優點及限制整理如表七。

表七：Stateful 與 non- Stateful 之汙點傳播分析比較

	Stateful 分析	non- Stateful 分析
特色	考量雲端服務之各模組的系統狀態,包括先前狀態及事後狀態	未考量雲端服務之各模組的系統先前狀態及事後狀態
優點	較嚴謹 •能分析在不同的防護情況下產生不同的污染傳播結果	簡單及快速
限制	•分析較複雜及運算速度慢 •須先調查各模組發生事件及可能改變的系統狀態	可能判斷錯誤

伍、結論

本研究以加權生成樹演算法為基礎進行網路應用服務之汙染動態傳播分析之研究，整合運用現有汙點傳播分析工具污點檢測工具 Valgrind、Androguard 與 Comdroid 以推估不同的汙染源傳播路徑以判定特定模組及資料庫是否受到汙染，並考量系統資安防護的狀態，以貼近網路攻防的真實情境，並針對不受信任的網路管道資訊流之汙染資訊進行標記與跟踪汙染源之傳播與風險評估，完整記錄被汙染模組，解決傳統動態汙點傳播分析法對於汙染源特徵容易誤判的缺點。

參考文獻

[1] G. Balakrishnan, T. Reps, WYSINWYX: What you see is not what you eXecute, ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 32, no.6, 2010, pp.1-84.

[2] W. A. Chad, M. Bamshad, B. Robin, Defending recommender systems: detection of profile injection attacks, Service Oriented Computing and Applications, vol. 1, no.3, 2007, pp.157-170.

[3] E. Chin, A. P. Felt, K. Greenwood, D, Wagner, Analyzing Inter-Application Communication in Android, in Proceedings of the 9th international conference on Mobile systems (MobiSys2011), 2011.

[4] A. Desnos, Androguard, avaliable at http://code.google.com/p/androguard/ wiki/Usage. (2014/10/7)

[5] H.C. ,Kim, A.D. Keromytis, M. Covington, R.Sahita, Capturing Information Flow with Concatenated Dynamic Taint Analysis, iN Proceedings: International Conference on Availability, Reliability and Security, 2009, pp.355-362.

[6] H. Mannila, H. Toivonen, and I.A. Verkamo Discovery of frequent episodes in event sequences, Data Mining and Knowledge Discovery, vol. 1, no.3, 1997, pp.259-289

[7] J. McClurg, J. Friedman, and W. Ng, Android Privacy Leak Detection via Dynamic Taint Analysis, EECS 450 (2013).

[8] J. Newsome and D. Song, Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software, Carnegie Mellon University, Research Showcase @CMU, 2005.

[9] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. SIGOPS Oper. Syst. Rev., vol.40, no.4, 2006.

[10] V. Rastogi, Y. Chen, W. Enck, AppsPlayground: automatic security analysis of smartphone application, in Proceedings of the third ACM conference on Data and application security and privacy (CODASPY13), 2013, pp. 209-220.

[11] O. Tripp, M. Pistoia, S. J. Fink, M. Sridharan, O. Weisman, TAJ: effective taint analysis of web applications, In Conference on Programming language design and implementation, 2009, pp. 87-97.

[12] W. T. Tsai, X. Wei, Y. Chen, Paul R. , J.Y. Chung, D. Zhang, Data provenance in SOA: security, reliability, and integrity, Service Oriented Computing and Applications, vol.1, no.4, 2007, pp.223-247.

[13] P. Wang, K.M. Chao, C.C. Lo, W.J. Chao, Using Taint Propagation Checking for Threat Analysis of Cloud Services, IEEE International Conference on e-Business Engineering, 2014, pp.185-190.

[14] H.Yin, D.Song, M. Egele, C. Kruegel, and E.Kirda, Panorama: capturing system wide information flow for malware detection and analysis. In Proceedings of the 14th ACM Conference on Computer and Communications Security, 2007, pp. 116-127.

[15] Honeynet project (2012), DroidBox. available at http://www.honeynet.org/gsoc/ slot11. (2014/10/15)

[16] TEMU, avaliable at http://bitblaze.cs.berkeley. edu/temu.html

[17] Valgrind, available at http://valgrind.org/docs/manual/dist.news.html (2014/11/25)

[18] Wikipedia, Spanning tree, available at: http://en.wikipedia.org/wiki/Spanning_tree, avaible at: http://en. wikipedia.org/ wiki/Taint_checking. (2014/12/12)

[19] Wikipedia, taint checking, available at: http://en.wikipedia.org/wiki/Taint_checking. (2014/12/12)

[20] 王平,林文暉,周宇軒,呂仁貴,雲端運算虛擬主機之攻擊模擬與分析,資訊安全通訊, 第19卷, 第2期, 2012年, 頁1~16。

[21] 王鐵磊，Research on Binary-Executable-Oriented Software，北京大學信息科技技術學院，博士論文，2011年。

[22] 任霏霏，莊洪林，吳理發，潘璠，跨主機動態污點跟踪技術研究，計算機工程，第39卷, 第3期，2013年3月。

[23] 梁光成, 動態污點分析淺述，2012年12月，available at: http://blog.sina.com.cn/s/blog_a3e16b1101017hf8.html

[24] 陳嘉玫,以汙染傳遞為基礎之行動軟體威脅行為偵測,第六屆台灣區Botnet 偵測與防治技術研討會(BoT2014)，2014年12月26日。

[25] 項國富,金海,鄒德清,陳學廣,基於虛擬化的安全監控, 軟體學報(Journal of Software), 第23卷, 第8期，2012年，頁2173−2187。