系統化量測Android軟體檢測覆蓋率之方法設計與實作

林仲儀[1] 邱敬翔[2] 黃俊穎[3]
[1 3] 國立交通大學 資訊工程學系
[2] 國立臺灣海洋大學 資訊工程學系
[1] daniel810736@gmail.com [2] pkriachu@gmail.com [3] chuang@cs.nctu.edu.tw

摘要

在這個研究裡，我們提出一個以系統化的方式量測Android軟體檢測覆蓋率的方法。測量軟體檢測覆蓋率是用來理解一個動態分析工具能力的基礎。過去研究人員通常使用動態插入指令來測量軟體檢測覆蓋率。然而，這種方法只有在可以本機端完全受控的環境中才適用，同時常常也需要搭配原始碼進行檢測。我們主要透過產生靜態Android軟體修補的方法來進行檢測覆蓋率的量測。經過我們所修補的軟體可以同時用來測量本機和遠端的軟體檢測覆蓋率工具。我們的測量結果顯示目前市面上現有的軟體檢測覆蓋率大多介於10%到50%，表示這些工具仍有很大的空間可以改善。這些結果可以提供詳細的資訊給研究人員和開發人員，使他們對相關技術的能力有更徹底的了解以並提供可能的改善方向。

關鍵字:Android，軟體檢測覆蓋率，動態分析，軟體測試

Abstract

In this study, we propose a systematic approach that measures the code coverage of Android dynamic analysis tools. Code coverage measurement is a fundamental step in understanding the capabilities of such tools. Previously, researchers often measured code coverage by using dynamic instrumentations. However, dynamic it is effective only in certain controlled environments and, therefore, are only applicable to local applications having source codes. Our approach resolves the aforementioned problems by generating statically patched Android packages for profiling dynamic analysis tools. The generated packages can be used to measure the code coverage rate of both local and remote tools. The measurement results reveal that the code coverage rate for the evaluated tools is between 10%–50%, indicating that these tools still require improvement. The results provide detailed information for researchers and developers to thoroughly understand and improve dynamic analysis techniques.

Keywords: Android, code coverage measurement, dynamic analysis, software testing

壹、 前言

手機可能是和現代生活最依賴的電子設備。現今手機市場所使用的作業統主要為 Android 和 iOS。根據來自 IDC[19]的報告，在 2015 年第二季中，Android 系統和 iOS 系統分別占了手機市場裡 82.8%和 13.9%的比例。和 iOS 相比，Android 提供了一個相對更開放的環境給使用者和開發者。開發者可以在大多數主流的作業系統上開發 Android 的應用程式，再透過 Google Play market 甚至是其他不被信任的第三方市場或是儲存空間發佈自己開發的應用程式。而使用者可以自行判斷、安裝並設定自己習慣的軟體、介面和操作方式。雖然這高彈性的設計吸引了許多使用者，但同時它開放式設計的特性也增加了 Android 使用者遭到惡意攻擊的機會。因此，即使是從官方市場安裝的應用程式，相較於 iOS，使用者仍然認為 Android 是危險許多的。安裝應用程式的風險包含過多的的廣告、系統效能退化、占用系統資源、破壞系統程式、超出特權的許可要求、個資外洩、甚至是導致網路攻擊。

許多方法都可以防止 Android 系統受到危險應用程式的干擾。其中，最基本的方法便是將 Android 作業系統預設為拒絕所有來源為不可信任的程式[15]的安裝請求。其他替代方案如防毒軟體，也可防止使用者遭到惡意程式的入侵。像 McAfee 和 Kapersky 這些防毒軟體公司便在 Android 使用者進行檔案執行時提供即時的保護。由於效能考量以及 API 的限制，大多數的防毒軟體的檢測技術是基於樣式比對。除了在手機內檢測，也有在手機外進行檢測的方法。而大部分的方法則都是線上工具。例如，VirusTotal[37]便是一種利用多種防毒掃描器來掃描上傳軟體的的線上檢測工具。同樣的，AndroTotal[24]則是另一種專門為 Android 應用程式所設計的線上檢測工具。這些工具掃描由使用者上傳的檔案並回傳經由一個或多個掃描器檢測所產生的掃描報告。使用者便可得知一個應用程式是否是惡意的。

然而，僅僅得知一個應用程式是否為惡意程式是不足的。為了更加的了解一個應用程式的行為和特性，需要更有效的分析工具。Androguard[9]是一款可以對 Android 應用程式進行逆向工程的靜態分析軟體。它同時還提供了一套工具以便管理及使用控制流程圖的結構來偵測惡意 Android 應用程式。Andrubis[20](之後整合為 Anubis)是一款使用動態分析技術來仔細的偵測以及回報 Android 應用程式行為的線上工具。或者，使用者也可以使用如 Tracedroid[36]和 DroidBox[21]來取得行為分析報告。這些行為分析工具皆可以靜態或動態分析技術實作。靜態分析工具並不需要執行應用程式；而是透過深入分析程式碼(使用低階或高階語言皆可)和其他資源檔案來產生報告。相反的，動態分析工具則須執行應用程式並分析應用程式行為已取得運行時的訊息，包括系統呼叫追蹤、檔案訪問、網路活動、和系統日誌來產生報告。雖然動態分析技術需要較多資源，但它們可以用來

防禦程式碼混淆並且更適合 Android 採用的混合執行環境。因此，許多研究皆採用來自動態分析工具的報告以便能更深入的偵測可疑的行動。

雖然動態分析技術已經被廣泛採用以搜集應用程式的行為，然而，關於這類工具的能力仍沒有一個明確的衡量方式，且測量動態分析技術能力的方法也還未有標準。使用者或是研究人員對於動態分析技術的另一個疑慮是它們可能沒有完整的檢查應用程式。因此，一個最基本衡量動態分析工具的方法便是測量它的程式碼覆蓋率。程式碼覆蓋率是一支程式在動態分析時被執行的程式碼比例。程式碼覆蓋率在軟體檢測和評估研究中是一個常見的度量[2, 7, 27]。程式碼覆蓋率可以在不同的精細度被測量，包括指令層級、行層級、區塊層級、方法層級、或是類別層級精細度。然而，測量程式碼覆蓋率並不是一件直觀的事情，特別是在當此動態分析工具只可在網路上執行時。簡而言之，現今的程式碼覆蓋率量測方法只能應用在有限的環境上，且無法進行對動態分析工具進行系統性和大規模的評估。

在這篇論文中，我們提出一個系統性的架構，能以產生修補 Android 軟體的方式，建立用於量測 Android 動態分析工具的程式碼覆蓋率。就我們所知，目前並沒有相關的工具可以取得使用。我們的方法可以產生量測線上或是離線動態分析工具的程式碼覆蓋率軟體。基於我們以前所做的研究[17](細節將於 3.3.3 和 3.3.4 中討論)，我們提出了一個改進整體量測效能的方法[3.3.5]。此外，我們也利用這個工具，對許多著名的動態分析工具進行了大規模的測量，包括在線工具(Anubis, Tracedroid 和 NVISO ApkScan)和離線工具(Android emulator, DroidBox 和 DroidScope)。我們呈現的方法反映了一個動態分析工具的能力以及為它未來的改進提供了全面性的資訊。

本論文章節安排規畫如下。我們在第二節介紹了相關的研究。第三節介紹以及討論了我們所提出的方法與設計，第四節呈現我們對於一些線上和離線工具的實驗結果。最後，在第五節總結。

貳、文獻探討

許多研究單位都有自行研發測試 Android 應用程式行為的工具。Enck 等人[10]提出 TaintDroid 來監測 Android 設備裡的資訊流。TaintDroid 既不是動態分析工具也不是沙盒，而是一個定製的框架，追蹤著應用程式是如何面對可疑的資訊。因此，將 TaintDroid 實作為一個獨立的應用程式是不可能的。使用者必須完整的更換他們的系統韌體以便和 TaintDroid 合力運行。Lantz 等人[21]提出 DroidBox，擴展了 TaintDroid 的函數並提供額外的靜態預先檢驗以及 API 監控功能。DroidBox 靠著汙染敏感資料以及在整個 API 中放置汙點以偵測資料外洩。再者，靠著紀錄相關 API 函式參數和回傳值，DroidBox 可以發掘潛在的惡意軟體以便

回報和做更深入的分析。Anubis 是一款線上動態分析工具，最初的設計是為了檢察在個人電腦上運行的惡意軟體。它的核心組成是由 Bayer 等人[3]所研發。在 2012，Anubis 增加了一個沙盒環境來檢查 Android 應用程式(名為 Andrubis)。除了動態分析，Andrubis 也提供了靜態分析、產生應用程式執行時的資訊、服務、需要的外部資料庫、以及實際所需的權限。更多有關 Andrubis 的介紹可在[22]中查看。Yan 和 Yin [39]提出 DroidScope 來分析 Android 行為。DroidScope 是在 Quick Emulator(QUEM)上建立，且可從外部重新建立 OS 和 Java 層級的語意觀。並且，許多工具，包括 API 追蹤器、本地指令追蹤器、Dalvik 指令追蹤器、以及汙點追蹤都被實作出來以便更深入的分析。Reina 等人[28]提出 CopperDroid，另一個基於 QEMU 所建立的動態分析工具，有著和 DroidScope 相似的設計。CopperDroid 監視低階系統呼叫，從而可以監視惡意軟體行為、無論這種行為是從 Java、JNI 或是本地程式碼執行。

除了研發工具，也有許多基於動態行為偵測惡意應用程式的研究。Xie 等人[38]提出 pBMDS，一個以隱藏馬可夫模型來確定應用程式和用戶行為，以便觀察手機應用程式以及使用者在輸入和輸出限制設備間的獨特行為。根據他們的資料，pBMDS 可以區分出使用者和惡意軟件的行為差異。Blasing 等人[4]，提出 AASandbox，能在 Android 程式上執行靜態分析以及動態分析來自動偵測可疑的應用程式。靜態分析掃描在源程式碼層描軟體中可疑的惡意型樣；而動態分析在沙盒中執行應用程式來監察以及紀錄對系統的低層級訪問以作進一步的分析。然而，作者並沒有報告提到此工具分析惡意軟體的效能。Burguera 等人[6]也提出靠動態行為來偵測惡意軟體。他們研發一個名為 Crowdroid 的用戶來監督 Linux 核心的系統呼叫，並回報給一個集中式的伺服器。根據所收集的資料集，研究人員透過分割聚類演算法來對每個資料集做分群，因此區分出良性和惡性應用程式。Zhou 等人[40]提出使用 DroidRanger 來偵測惡意的應用程式，基於靜態以及動態的特性。他們首先開發了一個基於許可的行為腳印方案，來用於檢測已知的 Android 中的新樣本惡意軟體家族。他們接著應用了啟發式過濾方案來區別未知惡意軟體家族的固有行為。結果顯示出他們的系統可以檢測已知的惡意軟體和一些零時差惡意程式。

儘管許多動態分析工具已經被研發，仍未有太多的研究調查出它們的軟體檢測覆蓋率以及量測軟體檢測覆蓋率的方法。Veen 和 Rossow[35, 36]宣稱平均的軟體檢測覆蓋率為 33%。然而，它們藉由特製的方法來測量覆蓋率，因此是無法被應用在其他平台的。雖然動態分析工具的軟體檢測覆蓋率並沒有被測量，仍有一些文獻提出一些方法改進測量軟體檢測覆蓋率的方法。最簡單的方法便是利用 Android Monkey [1]來隨機產生輸入值。Mahmood 等人[25]提出一個系統性的方法來生成許多測資用於模糊應用，同時也用產生的測資來生成試驗台，並在許多模擬於雲端的 Android 系統平行執行。這些情況是通過對一個應用程式做逆向工

程並產生使用者介面和程式碼之間的關係。雖然產生的測資相較於隨機產生的測資有較佳的軟體檢測覆蓋率，如何測量軟體檢測覆蓋率仍不明。Machiry 等人[23]提出 DynoDrioid，一個產生輸入來改進測試 Android 應用程式軟體檢測覆蓋率的系統。DynoDroid 使用 Emma[29]來測量軟體檢測覆蓋率。然而，在 Android 上使用 Emma 遇到了些許的限制，將在本節末提及。

Horvth 等人[16]提出一個測量 Android 設備軟體檢測覆蓋率的方法，可能是和本篇論文要提及的方法最相近的方式。然而，他們的方法只能適用於鄰近的設備，因為他們的方法需要執行中的設備額外安裝一個監控服務來蒐集測試中的程式的結果。一個遠程動態分析工具不能用於安裝這樣的附加服務，並且數據必須能夠由遠程蒐集。Android 應用程式通常是由 Java 語言所編寫。由 Java 來研發應用程式並不是甚麼新奇的事情，也有許多測量 Java 程式碼軟體檢測覆蓋率的工具。許多動態指令相關的工具都為了 Java 而被研發[8, 26, 31, 32]。其中一個最有名的工具便是 Emma，一個最初為測量 Java 軟體檢測覆蓋率所設計的工具，隨後併入 Android 做為 SDK 的一部分。兩個使用 Emma 的主要好處如下所述。第一，大部分有程式碼的 Android 應用程式都可被測量。第二，Emma 產生為開發人員產生一份詳細全面的報告，而軟體檢測覆蓋率在不同精細度，如類別層級、方法層級、區塊層級、以及行層級都有被提及。雖然 Emma 有許多特性，試著使用 Emma 的開發人員遇到了諸多限制:

1. 動態產生的模式不被支援；因此，無法測量預先編譯好的軟體。
2. 程式碼必須被修改以便載入 Emma。
3. Emma 僅支援 Java 機器碼，而不是 Dalvik 機器碼[11]。
4. 必需透過命令提示使用 am 腳本來啟動檢查應用程式。或是搭配使用 Eclipse 或 Android Studio IDE 內的插件。

因為這些限制，Emma 和其他相似的工具只能執行並量測擁有完整程式碼的本機應用程式。顯然的，以運行時檢查和大規模測量的觀點來說，Emma 並不適合用來測量動態分析工具的軟體檢測覆蓋率。4.2 節會詳細說明 Emma 是如何在 Android 上使用的。

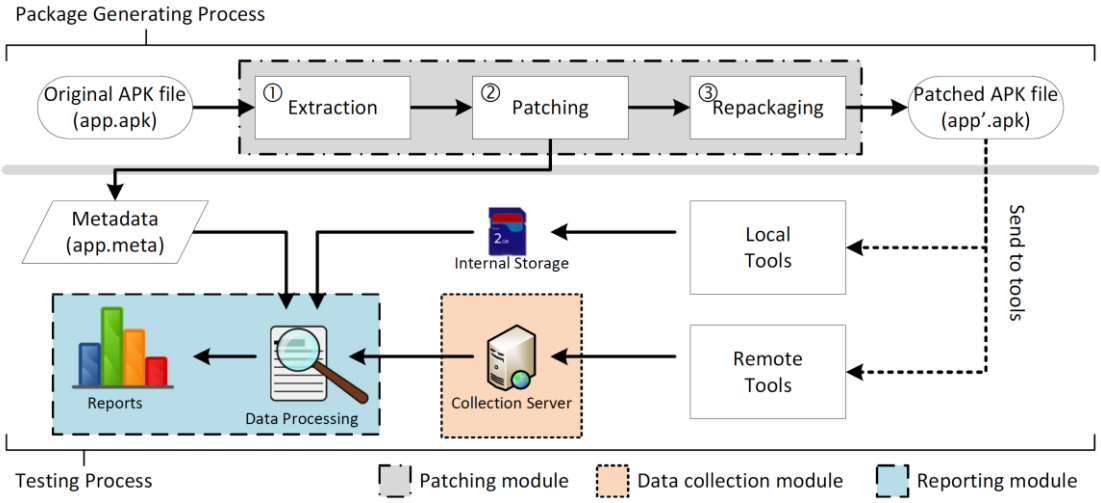

圖 1: 我們所提出的方法流程。

參、方法

3.1 設計目標

我們提供一個系統性的架構給研究人員去測量動態分析工具的效能。這個方法的設計目標包含下列幾項:

1) 無相依性。我們在二進制層級中測量軟體檢測覆蓋率，並不需要原始碼。此外，經過修補的應用程式可以單獨地在標準的 Android 環境執行運作。
2) 彈性。我們的方法可以測量離線和線上工具。使用者可以選擇將測量結果存在設備的儲存空間或是傳送到專門蒐集資料的伺服器。
3) 高精細度。我們提供和 Emma 類似的程式碼覆蓋率測量精細度。目前，精細度包括類別層級、方法層級、區塊層級、以及行數層級。

3.2 概述

圖 1 描繪了我們方法的工作流程圖。圖中有三個主要的組成元素，分別為修補模組(點線矩形)，數據蒐集模組(點矩形)，和報告模組(虛線矩形)。修補模組會處理一個給定的應用程式套件檔案(apk)並生成修補過的 apk 檔案。修補模組可以被配置為產生用於測量本機(離線)工具或遠程(線上)工具。除了產生套件，修補模組產生一個中繼資料，將儲存所需的資料以便將修補套件產生的測量結果解碼。

一旦修補套件產生後，可以用在本機進行量測，或是傳送給遠端工具以測量軟體檢測的覆蓋率。一個測量本機軟體檢測覆蓋率的修補套件會將結果以一個單獨檔案的形式存於運行設備的內部儲存中。使用者可以使用報告模組來取得檔案

裡的軟體檢測覆蓋率報告。而用於遠端測量的修補套件則會在量測過程中傳送分段的測量結果到數據蒐集模組中。報告模組會讀取分段以及對應的中繼資料檔案，接著產生測量報告。接下來的子節會詳細介紹每個模組。

3.3 修補模組

修補模組涉及三個步驟:解壓縮、修補(patching)、以及重新包裝。我們只簡略的介紹解壓縮以及重新包裝，因為這兩個步驟可以靠著現有的工具執行。修補將會被詳細介紹，因為它是我們方法中最重要的部分。

3.3.1 解壓縮

第一步，我們將檔案裡的 Android apk 檔案解壓縮並進行反編譯。雖然 apk 檔案已經是 zip 存檔，仍須使用 apktool [34]進行解壓縮因為它可以將如 AndroidManifest.xml 這種檔案轉為人類易讀的格式。此外，apktool 呼叫 baksmali 工具[14]來反編譯 apk 檔案。baksmali 讀取 classes.dex 檔案裡的 apk 檔案並將機器碼轉為組合語言。同一個類別裡的字節碼反編譯後被存在一個.smali 檔案。組合語言語法是基於 Jasmin 的 dedexer 的語法並支持 dex 格式的完整功能，包含註釋、除錯訊息、以及行數訊息。

3.3.2 修補: 概述

第二步，我們主要針對在第一步取得的反編譯檔案進行修補。和 Emma 相似，我們需要在 AndroidManifest.xml 檔案中要求寫入測量結果的許可。為了處理反編譯原始碼，我們實作一個編譯器來解析 smali 程式碼並插入定製的指令碼。而為了達成不同層級的細緻度，編譯器必須指出合適的位置來插入客製的指令碼。達到類別層級的細緻度是很直觀的因為 smali 使用一個單獨的檔案去存取一個類別。每個類別中的建構函數可以被確定並且指令碼可以插入在每個建構函數的進入點。達到方法層級的細緻度也很直觀，因為 smali 使用.method 這個關鍵字來識別出一個函式的開頭。相同的，指令碼可以插入在每個方法的進入點。JNI 方法則是一個例外。既然沒有 Java 程式碼涉及在 JNI 方法中，指令碼便在每次 JNI 方法被呼叫前被插入。至於要達成區塊層級的細緻度則比較複雜，因為編譯器必須能識別出每個區塊的進入點。我們使用下列的規則來識別區塊:

1) 一個標籤(label)標示了一個區塊的開頭。然而，如果一個標籤是用來定義 switch-case，陣列數據、以及 try-catch，則歸類為例外。為了避免產生大小為零的區塊，背靠背的標籤將被忽略，只有最後一個標籤會被考慮。
2) if 指令家族。這個指令指出一個區塊的結尾以及一個子區塊的開頭。指令包括 if-eq, if-ne, if-lt, if-ge, if-gt, if-le, if-eqz, if-nez, if-ltz, if-gez, if-gtz, if-lez。

3) goto 指令家族以及 return 指令家族。這兩個家族表示一個區塊的結尾。雖然在 goto 或 return 後的程式碼可能是其他區塊，後續的程式碼並不被視為是一個區塊，因為在 goto 和 return 後的可達程式碼至少需要有一個標籤，而這些區塊可以靠規則 1 去識別。
4) 呼叫 JNI 方法的 invoke 指令家族。這些區塊只是用來確定 JNI 方法是否需要被呼叫。

最後，對於行數層級的細緻度，我們計算識別區塊有效的指令數(既非空以及非標籤的指令)，將有效指令的個數存在每個識別區塊的中繼資料。當執行區塊開頭的檢測程式碼時，所有在區塊裡的程式碼都被認為是已執行的。雖然在行數層級細緻度所取得的數字可能和 Emma 分析 Java 原始碼的結果不同，這仍然反應出最精細層級的軟體檢測覆蓋率。根據此邏輯，我們標示區塊進入點並在每個區塊插入指令碼。一旦區塊的進入點被識出，我們便開始透過注入指令碼到原始碼來修補 smali 檔案。我們使用三種方法去實現程式碼修補:簡單/直接呼叫策略，簡單/間接呼叫策略，以及基於編碼的策略。

```
public class Coverage {
   static Hashtable<String, Long> map =
        new Hashtable<String, Long>();
   public static void doCount(String id) {
     if(map.containsKey(id)) {
       map.put(id, map.get(id)+1);
     } else {
       map.put(id, 1);
     }
   }
}
```

圖 2: 簡單的 doCount 函數實作-用於簡單的修補機制

3.3.3 修補: 簡單/直接呼叫策略

一開始我們用一個簡單的機制來修補程式碼。我們指派一個唯一的 ClassId 給每一個識別出來的的類別。同樣的，MethodId 和 BlockId 也分別被指派到每一個被識別出來的方法和區塊。經由這些識別代碼的組合，我們可以為每個指令點生成一組唯一的字串。這些識別字串以 ClassId#MethodId#BlockId 的格式呈現。輸入至 smali 程式碼裡的指令碼是一個簡單的函式呼叫，doCount(ClassId#MethodId#BlockId)，doCount 是一個由高階類別匯出的靜態函式。圖 2 呈現一個 doCount 函式的範例，其中 Hashtable 是用來計算每個字串出現的

次數(被呼叫的次數)。

雖然在標示的位置插入檢測程式碼看起來很簡單，然而要由 Dalvik 虛擬機器的函式呼叫協定來實作是很複雜的。Dalvik 虛擬機使用暫存器來傳遞函式呼叫參數而不是使用堆疊。因此，當檢測程式碼嘗試進行函數呼叫，並使用額外的暫存器用於傳送字串時，可能會影響其他原本應用程式所使用的暫存器，甚至使應用程式當掉。為了防止修補程式當掉，我們使用兩種不用的方法，直接呼叫和間接呼叫，在不使修補應用程式當掉的情況下插入檢測程式碼。

```
const-string v1, "32#0#0"
invoke-static {v1}, Lorg/codroid/util/Coverage;->doCount(Ljava/lang/String;)V

const-string v17, "32#23#0"
invoke-static/range {v17 .. v17}, Lorg/codroid/util/Coverage;->doCount(Ljava/lang/String;)V
```

圖 3: 使用簡單/直接呼叫方式插入程式碼

```
.method public static count_32_23_0()V
  .locals 1
  const-string v0, "32#23#0"
  invoke-static {v0}, Lorg/codroid/util/Coverage;->doCount(Ljava/lang/String;)V
  return-void
.end method
```

圖 4: 使用簡單/間接呼叫方式插入程式碼

表 1: Dalvik 函數呼叫的暫存器安排: 假設有二個區域變數和三個函數參數。

Register	Alias	Purpose
v0		Store the first local variable.
v1		Store the second local variable.
v2	p0	Store the first parameter passed to the method.
v3	p1	Store the second parameter passed to the method.
v4	p2	Store the third parameter passed to the method.

直接呼叫的方法需要在指令點直接的呼叫 doCount 函式。圖 3 闡述涉及直接呼叫的例子。因為一個識別字串必須被傳送給計數函數，修補程式碼最簡單的方式便是增加一個額外的程式碼來儲存識別字串。然而，這個方法因為 Dalvik 暫存器的索引限制而不是永遠有效。表 1 呈現一個擁有兩個區域變數以及三個參數的暫存器安排範例。Dalvik 虛擬機使用低索引數的暫存器來儲存區域變數，高索引數的暫存器用來儲存函式參數。別名是為了函式參數所造。在這個例子中，p0，p1，p2 分別是暫存器 v2，v3，v4 的別名。當一個額外的暫存器因為要存取區域變數的原因被加入(如新增識別字串)，增加的暫存器會被叫做 v2，而所有的參數暫存器索引值都要加一。然而，別名仍然對應到正確的暫存器(即 p0，p1，p2 現在分別為 v3，v4，v5 的別名)。增加一個區域變數暫存器有可能會導致執行上的問題。因為不是所有指令都適用於所有編號的暫存器。比如說，有些指令如 move 和 invoke 僅使用一個 4-bit 的值來索引暫存器。這些指令就只能存取暫存器 v0-v15。如果一個參數的暫存器從 v15 移動到 v16，那麼當初用來執行 v15 的指令不一定適用於 v16。下列是改動暫存器可能遇到的狀況。假設我們最初有 l 個區域變數暫存器以及 p 個參數暫存器。

a) **$1 + p \leq 15$:**這種情況是安全的。當一個額外的區域變數暫存器加入，暫存器總數仍會維持在 16 以下。
b) **$1 \geq 16$:**這個狀況也是安全的，因為所有參數暫存器的索引都超過 15。這些暫存器必定使用相容的指令進行存取。
c) **$1 < 16$ 和 $1 + p \geq 16$:** 這個情況是不安全的。這個狀況裡，一個參數暫存器可能會從 v15 移動到 v16。不相容的指令存取 v16 暫存器將觸發異常狀況並導致程式中斷。

當狀況(c)被察覺時，我們嘗試在不增加新的區域變數暫存器的情況下執行指令。這個狀況中，我們嘗試找出未經使用的暫存器用來傳遞參數。然而，如果沒有暫存器在入處可用時，我們就無法完成程式碼的修補。

3.3.4 修補:簡單/間接呼叫策略

直接呼叫策略並非永遠有效。因此,我們需要一個替代的方法來修補程式碼。不像直接呼叫策略,間接呼叫策略將呼叫 doCount 的行為包裝進一個新產生的函式並在插入點呼叫它。圖 4 呈現了函式包裝的範例。在範例中,呼叫 doCount("32#23#0")被實作在 count_32_23_0 這個函式裡。對應的插入點可以簡單的呼叫 count_32_23_0 函式,而不需任何參數;因此,不須新增任何額外的暫存器。雖然實作簡單呼叫策略比較簡潔,但因為處理暫存器動態的複雜性,間接呼叫策略是優於直接呼叫策略的。其中一個間接呼叫策略的限制是在一個單獨的.dex 檔案中,只能存取最多 65,536 個函式。為了避免函式的數量超過限制,一個單獨的.dex 檔案可能得被分成多個.dex 檔案。然而,而這仍未被 apktool 支援。包裝函式的數量通常與在應用程式中被標示的程式碼區塊的數量成正比。因此,間接呼叫策略可能無法適用於已經有許多函式的大型應用程式。因此,我們提出第三個基於編碼的策略來改善修補成功的比率。

當程式指令已經在簡單/直接策略和簡單/間接策略下完成修補後,會生成一個中繼數據以儲存分別從 ClassId 到類別名的映射以及 MethodId 到方法名的映射。

3.3.5 修補:基於編碼策略

上述兩種簡單的策略都有其限制。直接呼叫策略不易管理暫存器動態,間接呼叫策略受到函式數量的限制。因此,我們提出基於編碼的策略來克服這些限制。與其使用字串來辨識一個類別、方法、區塊、基於編碼策略很簡單的使用一個很長的整數(64-bit)做為區塊的識別碼來代表程式碼的插入點。因此,基於編碼的策略中的中繼數據檔案將下列額外的資訊存在區塊識別裡。

1. 屬於該區快的類別名稱以及方法名稱。
2. 區塊裡非空和無標籤行數的數量。
3. 旗標(flag),用來指出一個區塊是否為該類別的建構函數中的第一個區塊,第一個方法,或是 JNI 呼叫。

關於基於編碼策略,我們使用一個 4-bit 單位來對區塊標示符進行編碼。例如,假設一個區塊有一個識別整數值為 9beef6(16 進位),指令碼必須依序執行下列的函式呼叫,init(),update9(),updateB(),updateE(),updateE(),updateF(),update6(),以及 commit()。圖 5 顯示如何執行這些函式。為了減少呼叫靜態函式的數量,我們使用圖 6 呈現的演算法進行函數數量的縮減。這個演算法和 Huffman 編碼[18]有相近的概念。給定一個 apk 檔案 P,有著 u 個類別,$P = \{C_i | 1 \leq i \leq u\}$。首先,為 P 建造一個 call graph [30],接著計算逆向函式呼叫數(即,一個函式被其他函式呼叫的次數)。假設類別C_1由 v 個方法組成,類別C_2由 w 個方法組成;因此$C_1 =$

$\{M_{1,j}|1 \le j \le v\}$以及$C_2 = \{M_{2,k}|1 \le k \le w\}$。若$M_{1,8}$呼叫$M_{2,5}$，$M_{2,5}$的全域計數器增加一；即 counter[$M_{2,5}$]=counter[$M_{2,5}$]+1。在所有反編譯的組合語言被執行後，所有確定的方法都被存在一個串列 L 中，再根據逆向函式呼叫數遞減排序。最後，方法 m 反覆地從 L 中取得，區塊 id 也按照順序的分配給方法 m 中的區塊。

基於編碼策略有兩個重要的特性:固定數量的函式用於測量軟體檢測覆蓋率，以及沒有函式呼叫參數被傳送到計數函式。因此，修補成功率相較前兩種方法大幅提升。這個方法便是我們目前修補模組使用的方式。

3.3.6 重新包裝

最後一步便是將組合語言編譯成.dex 檔案再將所有檔案重新包裝 Android 套件。編譯(重新包裝)可以由 smali 工具(apktool)達成。當一個修補套件準備好時，它可以被傳送到線上或是離線的動態分析工具來產生結果。除了將結果存於本機端的空間，修補程式碼可以被配置成定期將測量結果傳送到伺服器的數據蒐集模組。為了創造大量不同的應用程式套件來評估動態分析工具的能力，修補過程可以全自動的不經人為干涉執行。

```
public class Coverage {
   static Hashtable<Long, Long> map =
        new Hashtable<Long, Long>();
   static Long one = new Long(1);
   static Long key = new Long(0);
   static Lock mutex = new ReentrantLock();
   public static void init() {
     mutex.lock();
     key = 0;
   }
   // update functions for 0--5, 7, 8, A,
   // C, and D are omitted for saving spaces.
   public static void update6() {
     key <<= 4;
     key |= 6;
   }
   public static void update9() {
     key <<= 4;
     key |= 9;
   }
   public static void updateB() {
     key <<= 4;
     key |= 0x0b;
   }
   public static void updateE() {
     key <<= 4;
     key |= 0x0e;
   }
   public static void updateF() {
     key <<= 4;
     key |= 0x0f;
   }
   //
   public static void commit() {
     Long i = null;
     if((i = map.get(key)) != null) {
       map.put(key, i + 1);
     } else {
       map.put(key, one);
     }
     mutex.unlock();
   }
}
```

圖 5: 基於編碼策略的程式碼插入函數(部分實作)。

Require: A *counter* implemented as an associative array.
Require: A package P containing u classes,
$P = \{C_i | 1 \leq i \leq u\}$.
1: **for** each class c in P **do**
2: **for** each method m in c **do**
3: **if** m calls method m' **then**
4: $counter[m'] = counter[m'] + 1$.
5: **end if**
6: **end for**
7: **end for**
8: Initialize L to be an empty list.
9: Add all keys in *counter* to L.
10: Sort L based on *counter* value in a descending order.
11: Let $seq = 0$.
12: **for** each method m in L **do**
13: **for** each block b in m **do**
14: Assign seq to b as its block identifier.
15: $seq = seq + 1$.
16: **end for**
17: **end for**

圖 6: 用於指派區域識別碼的演算法。

3.4 數據蒐集模組

只有遠端測量才需數據蒐集模組,即傳送一個修補 apk 到線上動態分析工具。當執行本機測量時,結果存於本地的儲存空間;每次測量都存於一個單獨的檔案。然而,測量結果並沒有同時被寫入檔案。我們讓一個量測套件去固定寫入測量結果,例如每十秒一次寫入,因為套件的執行時間是不可預測的。這個有規律性的寫入策略可以在不降低整體應用程式效能的情況下盡可能的寫入結果。一但一個測量任務結束,使用者可以輕易地從設備中取得檔案。量測結果將儲存於用 measure.[UNIX-timestamp].out 的檔案名稱。時間戳記便是應用程式起始的執行時間。我們使用 Java 的 Hashtable 物件來將結果存於內部。Hashtable 存取執行的區塊標識以及每個區塊被執行的次數。因此,每個紀錄被表示成兩個長整數(共 16bytes),並以 big-endian 排序。一個整數用來記錄區塊標識,另一個用來記錄計數器。關於此細節也可於網站上查看。

當遠端測量執行,結果被傳至數據蒐集模組,網頁服務接收從修補套件發送的 POST 請求。和本機端測量相近,結果會以一定的頻率傳送至數據蒐集模組。在此網頁服務是使用 PHP 腳本語言實現。為了簡單起見,數據蒐集模組將每次 POST 的結果存於單一個檔案。因此,每個測量任務都有大量的檔案。為了將測量結果傳送回數據蒐集模組,修補套件會將雜湊表結構轉為 JSON 物件,將 JSON 物件轉為字串,使用 zlib 資料庫將字串壓縮,將壓縮後的二進制字串編碼為 base-64 格式,最後以 POST 請求傳送資料。除了資料,套件標識、測試標識、時間戳記等,也會與每個 POST 請求一起傳送。在收到 POST 資料時,資料蒐集伺服器

使用套件標識，測試標識，以及時間戳記來命名檔案。接著解碼並將資料解壓縮以將 JSON 字串存進檔案裡。

3.5 報告模組

實作報告模組相當簡單。對於本機測量，報告模組只載入一個單獨的測量結果檔案，並且從標題讀取測試標識和套件標識。對於遠端測量，報告模組載入同一個測量任務的所有檔案，由套件標識和測試標識所得知。報告模組也載入對應的中繼數據檔案。中繼數據檔案恢復類別名稱，方法名稱，以及行數總數來生成軟體檢測覆蓋率報告。

肆、測試

在這個小節，我們從三個方面來評估我們提出的方法。我們先評估可以成功被逆向工程、修補、重新包裝的套件。我們接著比較我們測量的軟體檢測覆蓋率以及使用 Emma 測量的軟體檢測覆蓋率。最後，我們呈現選取的動態分析工具的結果。

4.1 重新包裝成功率

我們從 Google Play 市場上提供的 24 個類別下載了 112 個應用程式套件，測試一個套件是否可以正確地被我們進行修補和執行。我們透過下列的規則去選擇套件:

1. 被選中的套件必須要得以在模擬器中執行，因為大多數動態分析工具都是透過模擬器實作。
2. 被選中的套件必須得以獨立運作。根據這個規則，我們沒有下載依賴 Google 服務的插件、資料庫、或應用程式，這些可能無法在模擬器中使用。
3. 每個類別必須至少有一個套件比 8 megabytes 還要小，因為 Anubis 對於套件有著 8megabytes 的大小限制。
4. 我們沒有從通訊或是社群類別選擇應用程式，因為這類的應用程式多半被登入畫面所阻絕，服務條款訊息，或是許可協議訊息。

表 2、修補成功率的測試結果。欄位說明: Size (套件大小/MB)、Emu (模擬器中可執行)、AP(可重新封裝)、AR(封裝後可執行)、CP(可修補)、CR(修補後可執行)

Category	ID	Size	Emu	AP	AR	CP	CR
Books & Reference	com.chaozh.iReader	10.8	O	O	X		
	project.SimpleBibleTAndroid	3.0	O	O	O	O	O
	pdf.reader	5.9	O	O	O	O	O
	com.merriamwebster	29.8	O	O	O	O	O
	com.hotdog.guidefortempleruntwo2	7.5	O	O	O	O	X
Business	com.intsig.BCRLite	21.1	O	O	O	O	O
	com.fuho.vacronviewer	9.7	O	O	O	O	X
	com.indeed.android.jobsearch	1.7	O	O	O	O	X
	com.enlightment.appslocker	2.2	O	O	O	O	O
	com.obicat.vroom	2.6	O	O	O	O	O
Comics	jp.linebd.lbdmanga	7.5	O	O	O	O	O
	com.naver.linewebtoon	12.8	O	O	O	O	O
	com.dena.mj	6.8	O	O	O	O	O
	com.notabasement.mangarock.android.titan	7.3	O	O	O	O	O
	com.chraman.anime.girl.live.Wallpaper	4.9	O	O	O	O	O
Education	com.ted.android	8.4	O	O	O	O	O
	com.acobot.en	4.4	O	O	O	X	
	com.korean_vocab	3.0	O	O	O	O	O
	cplusmoment.ds.ced	34.1	O	O	O	O	O
	com.gv.gvdict	20.8	O	O	O	O	O
Entertainment	com.rantaz.sgfunnyquotes	2.4	O	O	O	O	O
	com.outfit7.superstarfree	23.3	X				
	com.sorasu.armiku	23.5	O	O	X		
	tw.com.off.taiwanradio	11.4	O	O	O	O	O
Family	com.oki.phonenew	9.8	O	O	O	O	O
	com.pescapps.kidspaint	3.3	O	O	O	O	O
	air.com.tutotoons.app.sweetbabygirlbeautysalon	39.2	O	O	O	O	X
	com.zeptolab.ctrexperiments.ads	29.3	O	O	O	O	X
	com.artelplus.howtodraw	17.1	O	O	O	O	O
Finance	com.kpmoney.android	9.5	O	O	O	O	O
	jp.united.app.kanahei.money	9.4	O	O	O	O	O
	com.lib.cwmoney	5.6	O	O	O	O	O
	com.ficc.ahorro	27.5	O	O	O	O	O
	com.citibank.mobile.tw	24.4	X				
Health & Fitness	jp.nc.hardyinfinity.bluelightfilter.free	1.8	O	O	O	O	O
	com.popularapp.periodcalendar	9.9	O	O	O	O	O
	com.meihillman.eyeprotection	1.4	O	O	O	O	O
	com.popularapp.sevenmins	6.9	O	O	O	X	
	com.geekslab.eyeprotection	1.3	O	O	O	O	O
Libraries & Demo	com.tplink.tether	3.4	O	O	O	O	O
	com.bettertomorrowapps.cameralockfree	2.05	O	O	O	X	
	com.startialab.actibook	7.5	O	O	O	O	O
	com.liferecipes.healthyrecipes	3	O	O	O	O	O
	com.mujerhairdesign.hijabfashionandtutorial	3.7	O	O	O	O	O
Lifestyle	com.hantor.CozyMag	1.2	O	O	O	O	O
	com.ikea.catalogue.android	26.4	X				
	jp.gmomedia.coordisnap	10.9	O	O	O	X	
	com.endless.koreanrecipes	4.4	O	O	O	O	O
	com.soloseal.awesomealarmclock	7.6	O	O	O	O	O
Live Wallpaper	com.cleanmaster.security	8.8	O	O	O	O	O
	oms.mmc.fortunetelling.gmpay.almanac2	14.4	O	O	X		
	com.ogqcorp.bgh	7.2	O	O	O	O	O
	com.devuni.flashlight	1.4	O	O	O	O	X
	com.poker.clubs.wallpapers.cute.anime.girl	2.2	O	O	O	O	O
Media & Video	com.herman.ringtone	2.9	O	O	O	O	O
	com.machipopo.media17	16.3	O	O	O	O	O
	com.youku.phone	24.5	O	O	O	O	X
	com.android2014.tubeclient	8.6	O	O	O	O	O
	com.perfect.player	11.7	O	O	O	O	O
Medical	com.h2sync.android.h2syncapp	10.4	O	O	O	X	
	com.hotloud.braincrush	4.6	X				
	com.despdev.weight_loss_calculator	4.0	O	O	O	O	O
	com.smsrobot.period	5.2	O	O	O	O	O
	com.developica.contracker	1.3	O	O	O	O	O
Music & Audio	com.sqgy.hongkongradio	8.4	O	O	O	O	X
	music.bassbooster.equalizer	2.8	O	X			
	com.ezhoop.music	3.4	O	O	O	O	O
	media.audioplayer.musicplayer	3.6	O	O	O	O	O
News & Magazines	com.ideashower.readitlater.pro	14.4	O	O	O	O	X
	com.hiiir.yahoo.friday	2.8	O	O	O	O	O
	com.nextmedia	12.1	O	O	O	O	O
	bbc.mobile.news.ww	11.5	O	O	O	O	X
	com.tam.gionews	7.8	O	O	O	O	O
Personalization	com.dotools.flashlockscreen	3.2	O	O	O	O	O
	com.apalon.wallpapers	14.4	O	O	O	O	X
	com.sharpregion.tapet	3.4	O	O	O	X	
	com.rayg.sirenringtones	10.9	O	O	O	O	O
Photography	com.instagram.layout	1.9	O	O	O	O	O
	com.pipcamera.activity	21.3	O	O	O	O	O
	us.pinguo.selfie	20.7	O	O	O	O	O
	com.lyrebirdstudio.montagenscolagem	18.5	O	O	O	O	O
	com.alensw.PicFolder	2.6	O	O	O	O	O
Productivity	com.shere.easytouch	3.15	O	O	O	O	O
	com.surpax.ledflashlight.panel	4.84	O	O	O	O	O
Shopping	com.thecarousell.Carousell	6.9	O	O	O	O	O
	com.taobao.htao.android	12.0	O	O	O	X	
	com.pinkoi	11.4	O	O	O	X	
	com.zalora.android	5.6	O	O	O	O	X
	com.google.zxing.client.android	1.0	O	O	O	O	O
Sports	com.nbaimd.gametime.nba2011	33.9	O	O	O	X	
	com.yinzcam.nba.warriors	14.6	O	O	O	X	
	com.protrade.sportacular	1.27	O	O	O	X	
	com.pacomob.basketballstrategyboard	3.6	O	O	O	O	O
	topigeon.com.tw.smsreport	8.1	O	O	O	X	
Tools	com.easyx.coolermaster	2.4	O	O	X		
	com.rvappstudios.flashlight	7.0	O	O	O	O	O
	com.mcafee.batteryoptimizer	6.5	O	O	O	O	X
	apps.ignisamerica.cleaner	8.2	O	O	O	X	
	com.thanh.informationdevice	1.0	O	O	O	O	O
Transportation	nexti.android.bustaipei	6.8	O	O	O	O	O
	com.booking	23.2	O	O	O	O	X
	com.ubercab	16.1	X				
	com.joelapenna.foursquared	16.4	O	O	O	X	
Travel & Local	com.hotelscombined.mobile	3.6	O	O	O	O	X
	com.axis.mobile.chapters.trans	2.4	O	O	O	O	O
	com.hcom.android	23.2	O	O	O	O	X
	com.wego.android	12.4	O	O	O	O	X
	com.tigerairways.android	5.0	O	O	O	O	O
Weather	com.yahoo.mobile.client.android.weather	19.8	O	O	O	O	O
	com.exovoid.weather.app	15.9	X				
	com.monirapps.thermometer	6.2	O	O	O	O	O
	com.accuweather.android	13.4	O	O	O	O	O
Total		112	106	105	101	87	70

表 2 呈現最終測試的結果。在總有的套件中，有 106 個是可以成功的被標準的 Android 模擬器執行(執行 Android4.3.1，API 級別 18)。在這 106 個套件中，105 個可以使用 apktool 重新包裝，這 105 個中有 101 個可以成功地用模擬器執行。我們對這些應用軟體進行修補並再測試一次後；在 101 個套件中，有 70 個可以成功的被執行。基於編碼策略的修補成功率是 69.3%，是我們之前使用簡單策略的兩倍[17]。一個套件沒辦法成功的在模擬器被執行是有幾個原因的。根據我們的檢驗，如果我們的套件偵測出運行時環境不是真實設備會拒絕執行。此外，一個套件可能會需要額外的硬體資源如硬體加速圖型，因此，這樣的應用程式變無法在模擬器中執行。因此，大多數遊戲類中的應用程式無法在模擬器中進行評估，我們便忽略這類的應用程式。對於重新包裝和修補套件，失敗可能會在反編譯，編譯，套件簽章時，套件驗證，以及運行時等情況發生。

EMMA Coverage Report (generated Wed Nov 19 09:37:34 CST 2014)

[all classes]

OVERALL COVERAGE SUMMARY

name	class, %	method, %	block, %	line, %
all classes	58% (13/26)	42% (77/185)	36% (2069/5718)	37% (428.3/1147)

OVERALL STATS SUMMARY

```
total packages:          8
total executable files:  11
total classes:           26
total methods:           185
total executable lines:  1147
```

COVERAGE BREAKDOWN BY PACKAGE

name	class, %	method, %	block, %	line, %
com.monead.games.android.sequence.ui.shape	0% (0/2)	0% (0/4)	0% (0/84)	0% (0/17)
com.monead.games.android.sequence.reporting	50% (1/2)	9% (1/11)	2% (13/522)	4% (4/105)
com.monead.games.android.sequence.util	50% (1/2)	20% (1/5)	12% (14/121)	17% (7/41)
com.monead.games.android.sequence	55% (6/15)	41% (31/75)	25% (555/2342)	28% (128/451)
com.monead.games.android.sequence.model	100% (1/1)	57% (16/28)	55% (556/1027)	33% (69.5/210)
com.instrumentation	100% (2/2)	75% (9/12)	46% (113/245)	50% (30.9/62)
com.monead.games.android.sequence.ui	100% (1/1)	57% (12/21)	70% (909/1298)	68% (164.6/242)
com.monead.games.android.sequence.sound	100% (1/1)	100% (7/7)	94% (73/78)	95% (18/19)

[all classes]

EMMA 2.0.5312 (C) Vladimir Roubtsov

圖 7: Emma 量測產生的 HTML 報表

4.2 和 Emma 比較

我們接著比較我們分析的軟體檢測覆蓋率結果以及Emma的結果。雖然Emma有一些限制，它仍是一個有價值的工具。我們簡單的介紹 Emma 是如何在 Android 上運作。讀者可以遵從下列的步驟以便在 Android 使用 Emma 檢測自己的 Android 專案。

a) 通過添加一個<instrumentation>標籤並指定要測量的應用程式的套件標識來修改 AndroidManifest.xml 檔案。
b) 請求執行內部儲存空間的許可。這是為了使 Emma 得以存取測量的結果。
c) 創造一個新的類別來取代原有的 MainActivity。這是應用程式中新的進入點，且為 Emma 執行所有的初始化任務。

使用者可以在執行完上述步驟後編譯並產生修補的應用程式套件。一個新

的.em 資料(例如，coverage.em)被生成來存取中繼資料需要的指令以及紀錄解析。使用者便可以靠以下指令(經由 adb)在指令模式中執行應用程式:

```
$ adb shell am instrument -e coverage true  -w package_name/runner_class
```

其中 runner_class 是繼承自 MainActivity 的新起始點類別。測量結果被存於一個.ec 檔案(如 converge.ec)，以便在後來可以與.em 檔案一同載入，解釋和呈現測量結果。圖 7 呈現一組 Emma 產生的 HTML 報告。

因為 Emma 需要 Java 原始碼，我們從 F-droid 網站[12]下載了五個應用程式，一個開源 Android 應用程式的線上儲存庫。每個專案都透過 Android 官方的模擬器編譯以及測試五次。表 3 呈現比較的結果，反應出使用我們的方法和使用 Emma 所得的到結果類似。然而，Emma 在 Java 程式的語言層級計算行數，我們則在 smali(組合)程式語言層級計算行數。smali 程式碼每行只有一個單獨的指令。

表 3: 我們的方法與 Emma 的量測結果比較。

Package: net.sourceforge.opencamera				
	Class	Method	Block	Line
Emma	50.30%	46.36%	44.19%	44.29%
Ours	49.09%	43.77%	34.97%	43.20%
Package: com.uberspot.a2048				
	Class	Method	Block	Line
Emma	100.00%	64.71%	70.48%	68.04%
Ours	100.00%	65.88%	64.24%	75.26%
Package: com.monead.games.android.sequence				
	Class	Method	Block	Line
Emma	50.83%	35.03%	32.40%	32.96%
Ours	50.00%	35.49%	26.20%	38.88%
Package: com.notriddle.budget				
	Class	Method	Block	Line
Emma	40.00%	25.28%	16.54%	18.15%
Ours	40.63%	29.40%	21.42%	23.36%
Package: home.jmstudios.calc				
	Class	Method	Block	Line
Emma	94.55%	45.73%	44.99%	38.77
Ours	95.91%	45.33%	30.56%	37.34

表 4: 隨機挑選 10 個測試套件的結果：覆蓋率統計。

Name	ID	Classes	Methods	Blocks	Lines
Super-Bright LED Flashlight	com.surpax.ledflashlight.panel	5,017	31,058	97,846	478,778
QuickPic Gallery	com.alensw.PicFolder	2,609	17,743	62,629	298,440
Bluelight Filter for Eye Care	jp.ne.hardyinfinity.bluelightfilter.free	106	517	1,348	7,398
App Lock	com.enlightment.appslocker	288	1,715	4,652	25,440
Flashlight - Torch LED Light	com.rvappstudios.flashlight	2,459	17,087	51,079	344,641
Kids Paint Easy	com.pescapps.kidspaint	35	177	334	2,440
Top Music Player	com.ezhoop.music	3,645	28,772	73,591	426,532
Awesome Alarm Clock	com.soloseal.awesomealarmclock	35	211	503	6,433
Bus Tracker Taipei	nexti.android.bustaipei	2,086	15,489	46,654	312,357
Voice Translator	com.axis.mobile.chapters.trans	575	2,803	8,027	40,047

表 5: 隨機挑選 10 個測試套件的結果：執行時間。

Name	Emulator	DroidScope	DroidBox	Anubis	NVISO ApkScan	TraceDroid
Super-Bright LED Flashlight	95	367	-	91	-	71
QuickPic Gallery	72	-	93	589	82	23
Bluelight Filter for Eye Care	132	141	42	94	-	24
App Lock	40	143	31	2724	85	167
Flashlight - Torch LED Light	89	-	42	67	71	67
Kids Paint Easy	84	181	31	352	58	88
Top Music Player	82	143	42	393	84	5
Awesome Alarm Clock	103	190	-	53	84	84
Bus Tracker Taipei	96	166	85	514	-	29
Voice Translator	114	101	146	354	83	61

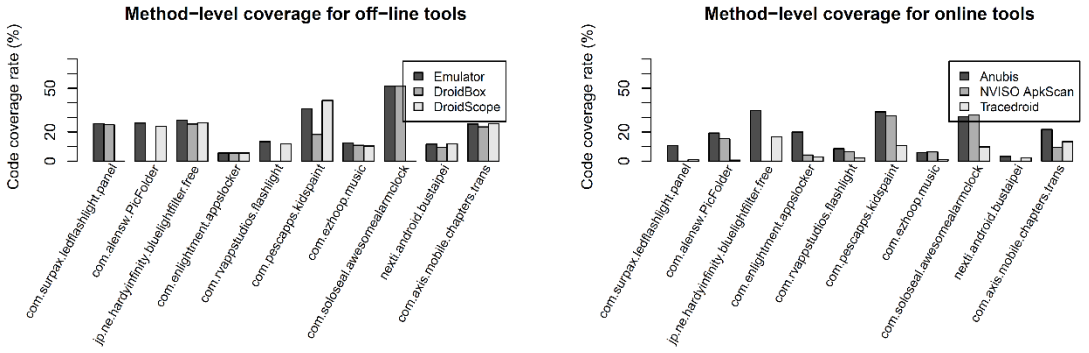

圖 8: 覆蓋率量測結果：函數層級精細度。(左)離線工具；(右)線上工具。

4.3 測量 Android 動態分析工具

我們使用本研究所提出的方法來分析 Android 動態分析工具的軟體檢測覆蓋率，選取線上(Anubis,NVISO,ApkScan,以及 Tracedroid)以及離線(Android 官方的模擬器，DroidBox,以及 DroidScope)兩種工具。我們沒有測量 CopperDroid 線上工具，因為當我們在寫這篇論文時 CopperDroid 正在進行重整[33]。每個連線工具的模擬器配置如下所述。官方模擬器執行 Android 4.3.1(API 層級 18)，DroidBox 制行 Android 4.1.2(API 層級 16)，以及 DroidScope 執行 Android 2.3.5(API 層級 10)。用來產生基準測試的套件是從 4.1 節中抽樣。我們選取 10 個應用程式套件並使用所有的評估工具測試五次。圖 8 和圖 9 分別反應出方法層級的結果以及區塊層級的結果。

表 4 呈現 10 個隨機選取套件量測數據。表 5 則顯示每個動態分析工具的測量時間。對於本機測試，我們使用 MonkeyRunner 來模擬 500 個隨機事件。本機

和遠端的平均執行時間明顯不同。NVISO ApkScan 以及 TraceDroid 的運行時間限制並沒有對外公布。至於 Anubis，檔案中[22]宣稱每個應用程式都被測試了 240 秒。我們的測量結果顯示出有時一個應用程式的執行時間比預期的短，可能是因為我們的修補應用程式無預警的終止了。結果也顯示出有時一個應用程式執行的時間比預期的長，可能是因為此應用程式沒有被評測工具完整的終止。雖然執行時間彼此迥異，結果顯示出運行時間和覆蓋率是互相獨立的；即比較短的執行時間也有可能導致比長執行時間還要高的軟體檢測覆蓋率。

根據測量結果，我們的發現總結如下。第一，線上工具和離線工具的效能相近，可能因為許多線上工具都是建構於有名的離線工具之上。第二，一些測量結果顯示接近 0 的軟體檢測覆蓋率。根據我們使用的離線分析工具，這個現象可能是因為以下兩個主要原因。第一，考慮到有限的 API 級別是在動態分析工具中實現的，有些零覆蓋率可歸因於不相容的 API 級別。在這種情況下，應用程式沒有被該工具啟動。相反的，一些零覆蓋率是由於應用程式快速的崩潰或終止。然而，大多數工具仍然回報執行成功。最後，我們發現大多數測量的工具軟體檢測覆蓋率不足，大多介於 10%到 50%之間。一個應用程式套件可能會有更低的軟體檢測覆蓋率，因為它有可能被 UI 操作阻擋，如登入情況，認證同意，升級公告，以及彈出的廣告。這些發現反應出研究者仍可以改善他們的動態分析工具。

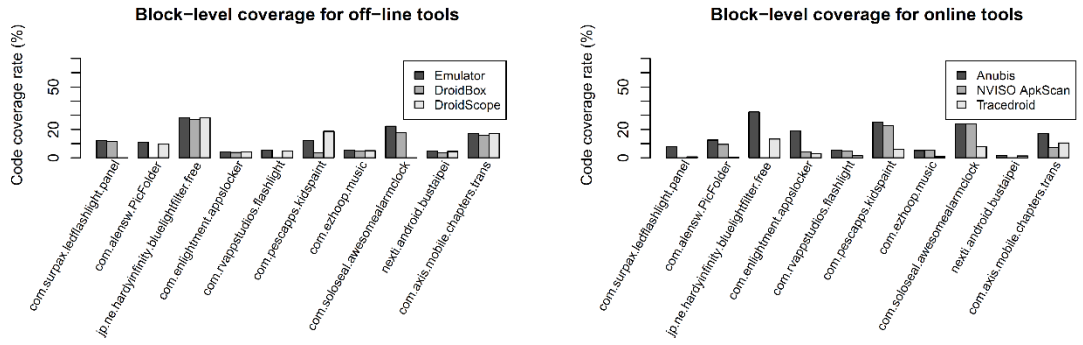

圖 9: 覆蓋率量測結果：區塊層級精細度。(左)離線工具；(右)線上工具。

伍、結論

現在的 Android 運行時行為取證工具經常涉及使用動態分析技術來確定應用程式運行時的行為。然而，不同動態分析工具的能力對於使用者來說是無從得知的。在這篇論文裡，我們提出一個系統性測量 Android 動態分析工具軟體檢測覆蓋率的新方法，並且透過測試線上和離線工具來顯示它的有效性。評估的工具包括 Anubis, Tracedroid, NVISO ApkScan, DroidBox, DroidScope,以及 Android 官方的模擬器。測量結果顯示:1)線上工具和離線工具有著相近的表現；2)一個有用的

工具必須提供多樣的 API 層級，使其盡可能檢查多種套件；3)軟體檢測覆蓋率不足，大多介於 10%到 50%之間。這些發現告訴我們動態分析工具的效能仍是可以被改善的。一個有效的動態分析工具是建立良好的鑒識分析以及軟體品質評估工具的基礎。透過我們所設計的方法，研究者以及開發者可以更深入的理解動態分析工具的效能並進一步改進這些工具。

[致謝]

本研究感謝科技部計畫補助（編號：104-2221-E-009-200-MY3）。本論文中所提及之立場及論述僅代表作者本人，並不代表研究補助單位。

參考資料

[1] Android, "UI/Application exerciser Monkey." [Online]. Available: http://developer.android.com/tools/help/monkey.html

[2] N. Antunes and M. Vieira, "Assessing and comparing vulnerability detection tools for web services: Benchmarking approach and examples," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 269–283, March–April 2015.

[3] U. Bayer, C. Kruegel, and E. Kirda, "TTAnalyze: A tool for analyzing malware," in *Proceedings of the 15th European Institute for Computer Antivirus Research Annual Conference*, ser. EICAR, 2006.

[4] T. Blsing, L. Batyuk, A.-D. Schmidt, S. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE)*, Oct 2010, pp. 55–62.

[5] D. Bruschi, L. Martignoni, and M. Monga, "Detecting selfmutating malware using control-flow graph matching," in *Proceedings of the 3rd International Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, July 2006, pp. 129–143.

[6] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM, 2011, pp. 15–26.

[7] M.H. Chen, M. R. Lyu, and W. E. Wong, "Effect of code coverage on software reliability measurement," *IEEE Transactions on Reliability*, vol. 50, no. 2, pp. 165–170, June 2001.

[8] S. Chiba, "Javassist: Java bytecode engineering toolkit since 1999." [Online]. Available: http://jboss-javassist.github.io/javassist/

[9] A. Desnos and G. Gueguen, "Reverse engineering, malware and goodware analysis of android applications ... and more (ninja !)." [Online]. Available: https://github.com/androguard/androguard

[10] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. Mc-Daniel, and A. N. Sheth, "TaintDroid: An information flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[11] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proceedings of the 20th USENIX Conference on Security*, 2011, pp. 21–21.

[12] F-Droid.org, "F-Droid: Free and open source app repository." [Online]. Available: https://f-droid.org/

[13] C. Florian, "Most vulnerable operating systems and applications in 2014," February 2015. [Online]. Available: http://www.gfi.com/blog/most-vulnerable-operating-systems-and-applications-in-2014/

[14] J. Freke, "smali: An assembler/disassembler for Android's dex format." [Online]. Available: https://github.com/JesusFreke/smali

[15] Google, Inc., "Google settings (android): Protect against harmful apps." [Online]. Available: https://support.google.com/accounts/answer/2812853

[16] F. Horvth, S. Bognr, T. Gergely, R. Rcz, . Beszdes, and V. Marinkovic, "Code coverage measurement framework for android devices," *Acta Cybernetica*, vol. 21, pp. 439–458, 2014.

[17] C.-Y. Huang, C.-H. Chiu, C.-H. Lin, and H.-W. Tzeng, "Code coverage measurement for android dynamic analysis tools," in *Proceedings of IEEE International Conference on Mobile Services*, July 2015, pp. 209–216.

[18] D. A. Huffman, "A method for the construction of minimum-redundancy codes," in *Proceedings of the I.R.E*, no. 40, pp. 1098–1101, 1952.

[19] International Data Corporation, "Smartphone OS market share, Q2 2015." [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[20] International Secure Systems Lab (iSecLab), "Anubis: Malware analysis for unknown binaries." [Online]. Available: http://anubis.iseclab.org/

[21] P. Lantz and A. Desnos, "DroidBox: An android application sandbox for dynamic

analysis." [Online]. Available: https://code.google.com/p/droidbox/

[22] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, "Andrubis - 1,000,000 Apps Later: A View on Current Android Malware Behaviors," in *Proceedings of the the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, 2014.

[23] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for android apps," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 224–234.

[24] F. Maggi, A. Valdi, and S. Zanero, "Andrototal: A flexible, scalable toolbox and service for testing mobile malware detectors," in *Proceedings of the 3rd Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*. ACM, November 2013.

[25] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou, "A whitebox approach for automated security testing of android applications on the cloud," in *Proceedings of the 7th International Workshop on Automation of Software Test*, ser. AST '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 22–28.

[26] L. Marek, Y. Zheng, D. Ansaloni, A. Sarimbekov, W. Binder, P. Tma, and Z. Qi, "Java bytecode instrumentation made easy: The DiSL framework for dynamic program analysis," in *Proceedings of the 10th Asian Symposium on Programming Languages and Systems*, December 2012, pp. 256–263.

[27] L. Mei, Y. Cai, C. Jia, B. Jiang, W. K. Chan, Z. Zhang, and T. H. Tse, "A subsumption hierarchy of test case prioritization for composite services," *IEEE Transactions on Services Computing*, vol. 8, no. 5, pp. 658–673, September–October 2015.

[28] A. Reina, A. Fattori, and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," in *Proceedings of the 6th European Workshop on System Security (EUROSEC)*, Prague, Czech Republic, April 2013.

[29] V. Roubtsov, "EMMA: a free Java code coverage tool." [Online]. Available: http://emma.sourceforge.net/

[30] B. G. Ryder, "Constructing the call graph of a program," *IEEE Transactions on Software Engineering*, vol. SE-5, no. 3, pp. 216–226, May 1979.

[31] A. Seesing and A. Orso, "InsECTJ: a generic instrumentation framework for collecting dynamic information within Eclipse," in *Proceedings of the 2005*

OOPSLA workshop on Eclipse technology eXchange, 2005, pp. 45–49.

[32] D. Slife and M. Chesney, "jCello." [Online]. Available: http://jcello.sourceforge.net/

[33] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "CopperDroid: Automatic reconstruction of Android malware behaviors," in *Proceedings of the Network and Distributed System Security Symposium*. The Internet Society, 2015.

[34] C. Tumbleson, "A tool for reverse engineering Android apk files." [Online]. Available: http://ibotpeaches.github.io/Apktool/

[35] V. van der Veen, "Dynamic analysis of android malware," Master's thesis, VU University Amsterdam, August, 2013.

[36] V. van der Veen and C. Rossow, "Tracedroid – dynamic Android app analysis." [Online]. Available: http://tracedroid.few.vu.nl/

[37] VirusTotal, "VirusTotal - free online virus, malware and URL scanner." [Online]. Available: https://www.virustotal.com/

[38] L. Xie, X. Zhang, J.-P. Seifert, and S. Zhu, "pBMDS: A behavior-based malware detection system for cellphone devices," in *Proceedings of the Third ACM Conference on Wireless Network Security*, ser. WiSec '10. New York, NY, USA: ACM, 2010, pp. 37–48.

[39] L. K. Yan and H. Yin, "DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic android malware analysis," in *Proceedings of the 21st USENIX Security Symposium*, ser. Security'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 29–29.

[40] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets." in *Proceedings of the Network and Distributed System Security Symposium*. The Internet Society, 2012.