Regular Paper

Communications of the CCISA
Vol. 28 No. 2 May 2022

基於 Quark 引擎之 Android 應用程式惡意行為偵測規則自動化生成系統

鄧宇翔[1,*]、陳坤裕[2]、呂昇峰[3]
[1,2]財團法人電信技術中心資通安全組、[3]國立中山大學資訊工程學系
[1,2]{shaun.dang, kunyu.chen}@ttc.org.tw、[3]m103140005@nsysu.edu.tw

摘要

隨著智慧型行動裝置時代來臨，Android 已成為高市佔率之行動裝置作業系統，因此駭客也逐漸以 Android 裝置作為其犯罪標的。Quark 引擎為一開源之 Android 惡意程式分析工具，惡意程式分析師可透過 Quark 引擎產出之分析情資，快速定位樣本之惡意行為。Quark 引擎亦為 rule-based 偵測引擎，亦即偵測規則數量與其實用性成正比。然而，若以人工撰寫規則，所需勞務成本過高，且規則品質取決於惡意程式分析師資安經驗多寡。因此，Quark 團隊過往已研發出兩代自動化規則生成系統，為改進前兩代規則產出效率低，以及人工篩選規則勞務成本過高之瓶頸，本研究提出第三代系統，並以兩種真實惡意程式樣本進行實驗，比較三個版本生成規則之命中率 (有效規則數與生成規則數之比例) 與規則生成率 (生成規則數與計算量之比例)。實驗結果顯示，第三代自動化規則生成系統之命中率平均為 49.6%，規則生成率平均為 26.1%，皆遠高於前兩代系統。研究結果表明，第三代規則生成系統為最高效益之版本。然而，該系統仍有規則篩選流程優化，以及目標函式挑選流程優化之改進空間。本研究依照上述 2 點提出未來研究方向與建議。

關鍵詞：Android、惡意程式、偵測規則、分析情資、自動化生成規則

* 通訊作者 (Corresponding author.)

Regular Paper

Communications of the CCISA
Vol. 28 No. 2 May 2022

Automatic Android Malware Detection Rule Generation based on Quark Engine

Yu-Shiang Dang[1*], Kun-Yu Chen[2], Sheng-Feng Lu[3]
[1,2]InfoCom Security Division, Telecom Technology Center,
[3]Department of Computer Science and Engineering, National Sun Yat-sen University
[1,2]{shaun.dang, kunyu.chen}@ttc.org.tw, [3]m103140005@nsysu.edu.tw

Abstract

With the advent of the smartphone era, the Android operating system has the highest market share worldwide on mobile devices. Consequently, the Android platform has become the biggest target for threat actors. Quark Engine is an open-source Android malware analysis system that provides threat intelligence of Android malware. It helps threat researchers quickly detect the behavior in malware samples. Since Quark Engine is a rule-based system, the number of Quark rules is proportional to its practicality. However, manually creating rules is time and effort consuming. In addition, the quality of Quark rules depends on the experience of the threat researcher. Therefore, The Quark team has developed two versions of Quark rule generation system. To improve the drawbacks of the systems, this study presents the third version of the rule generation system. We experiment with two different families of Android malware samples. The experiment compares accuracy (the ratio of effective rules number to output rules number) and productivity (the ratio of output rules number to the amount of computation). The result shows that the third version of the rule generation system has 41.5% accuracy and 26.1% productivity in average, which are both much higher than other versions of rule generation system. Thus, the result shows that the third version is the most efficient system. Last, we proposed several suggestions for improvements, including the process of rule selection and the target function selection.

Keywords: Android, Malware, Detection rule, Threat intelligence, Rule generate

壹、前言

隨著智慧型行動裝置時代來臨，駭客鎖定之作業系統亦隨之轉移。根據 Statista 統計，截至 2022 年 1 月，在所有行動裝置中，Android 作業系統市占率高達 70% [1]。因此，Android 使用者也成為駭客之犯罪標的。不僅如此，為能從源頭讓使用者裝置感染惡意程式，駭客亦將攻擊範圍延伸至 Google Play 商店，其利用使用者對官方下載平台的信任，透過多種手法繞過 Google 官方設立之檢測機制，並將惡意程式上傳至該平台，讓使用者在不知情下安裝惡意程式。此外，駭客亦常使用釣魚手法，透過簡訊或社交平台，傳送不法連結，誘騙使用者點擊，藉以在使用者裝置上安裝惡意程式，或直接進行個資竊取。為此，隨著攻擊手法不斷變化，資安分析人員在分析惡意程式上所花費時間亦大幅成長。也因此，資安分析人員亦須不斷發展新型態分析工具，應對與日俱增之資安威脅複雜度。

開放源始碼專案 (Open Source Project)，是指任何人，將其所撰寫之專案程式碼分享於網路上，透過公開討論與協同合作，任何對該專案有興趣之使用者，皆可以任何形式做出貢獻，例如：編輯說明文件、新增功能以及修復程式錯誤。知名開放專案如 Linux，截至 2021 年，已累積 12,352 位貢獻者，該專案未來發展之能量，亦隨著貢獻者數量增加而不斷提升。透過公開討論與協作，開源專案得以吸收不同觀點並擇優落實。Quark 引擎，便是基於此想法所誕生之開源專案，希望透過開源的力量，協同對抗變化多端之資安威脅。

Quark 引擎 [2] 為一開源之 Android 惡意程式分析工具。其目的是將 Quark 引擎偵測出之行為，以語意方式呈現，使資安分析人員快速掌握分析標的之潛在惡意行為。透過開源協作，Quark 引擎亦獲得不同分析觀點，以及來自各方之貢獻，例如：資安廠商 Fortinet 首席研究員 [3] 、知名開源專案 Intelowl、APKLab 與 MobSF 作者。

相較於其他提供 Android 惡意程式概覽資訊的分析工具，Quark 引擎以語意方式，提供描述惡意程式具體行為之概覽分析報告，是相對少見的。Quark 引擎亦是 rule-based 偵測引擎，亦即偵測規則數量越高，其實用性越高，概覽分析報告之內容亦越加豐富。然而，若以人工撰寫偵測規則，其勞務成本與惡意程式複雜度成正比，並與分析人員之資安經驗豐富度成反比。為克服偵測規則撰寫之問題，過往 Quark 團隊已發展出 2 個自動化規則生成演算法。然而，這些演算法在效能與規則實用度上，仍有改進空間。綜上所述，本研究具體目的，為提出一新規則自動生成演算法，提升偵測規則生成之效率與實用度。

貳、文獻探討

2.1 開源 Android 應用程式分析工具探討

開源軟體中，許多工具提供 Android 惡意程式分析功能。其中，Apktool [4] 為知名 Android 應用程式靜態分析工具，其核心功能是將 Android 應用程式轉譯為 smali 語法，分析人員即可透過閱讀 smali 程式，進行惡意程式分析。Jadx [5] 為知名 Android 應用程式反組譯靜態分析工具，其核心功能是將 Android 應用程式反編譯為 Java 原始碼。相較於 smali 語法，Java 原始碼更易於閱讀。此外，Jadx 也開發許多功能，例如：xref 查找與關鍵字搜尋。因此，以 Jadx 進行靜態分析更有效率。

然而，靜態分析仍有其限制與瓶頸，例如：駭客為避免程式碼被分析，因而對程式碼進行加密、加殼或混淆。這些躲避分析的手法，使應用程式難以被反編譯，或反編譯後難以閱讀。動態分析則是在特定分析環境中，實際運行應用程式，並在此情境下分析記憶體資訊以找出線索，如此，便無需克服部分加密、加殼或混淆之問題。

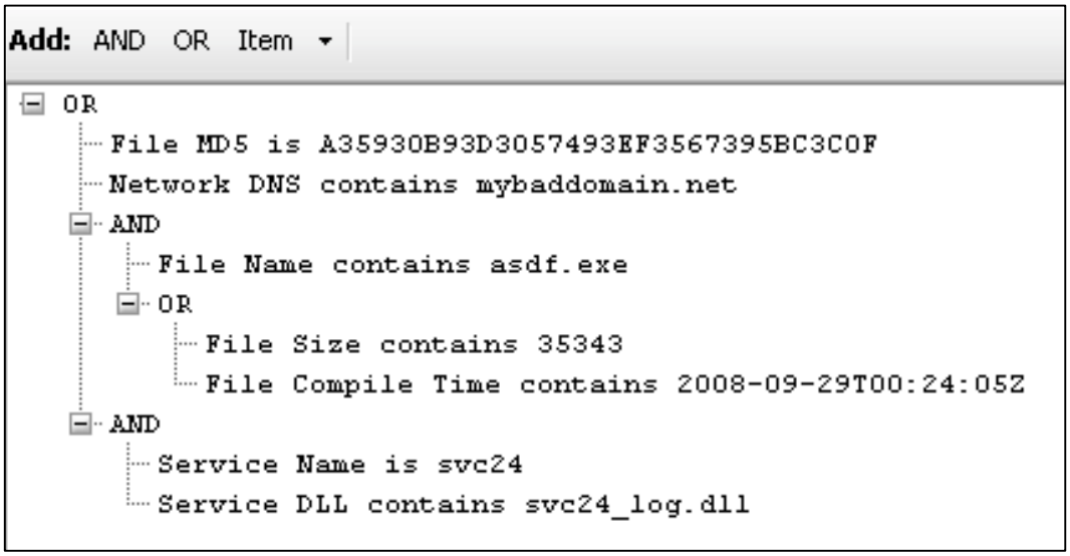
其中，frida [6] 便是一套廣受歡迎之動態分析軟體，該軟體之核心功能是將 Javascript 程式碼片段 (frida 內建或使用者自定義) 注入於應用程式中，藉此觀察與追蹤應用程式內之目標應用程式介面(API, Application programming interface)。在 frida 基礎上，亦發展出許多動態分析工具如 Objection [7] 與 Dexcalibur [8]。其中，Objection 藉由 frida API 之組合與封裝，開發出許多簡單而實用之功能，使分析人員得以在文字指令介面 (CLI, Command Line Interface) 下，進行互動式分析，取得實質進展。Dexcalibur 則提供分析人員圖形化介面 (GUI, Graphic User Interface)，使分析人員得以在視覺化體驗下，展開分析工作。

分析惡意程式時，不同分析工具所提供之分析線索互有長短且各有互補。因此，多種工具相互搭配使用，藉以拼湊出惡意程式完整樣貌，為分析任務之常態。例如：當惡意程式之反分析手法較複雜時，分析人員若單純以 APKtool 或 Jadx 調查，將不易取得惡意程式真正的 payload (具有惡意行為之載體)，在此情況下須搭配動態分析，從記憶體中鎖定並輸出 payload，才能進行後續分析。然而，即使取得 payload，分析人員仍須以人工方式進行初步檢驗，此舉亦耗費大量勞務成本。為此，Quark 引擎提供行為分析功能，並以語意方式自動化產出報告，大幅縮短分析人員調查時間，進一步讓惡意程式分析鏈更加完整。

2.2 開源工具之情資偵測規則探討

為便利惡意程式特徵擷取、保存與交流，入侵威脅指標 OpenIOC (Open Indicator Of Compromise) [9] 概念被提出。分析人員以標準格式排版與欄位定義，紀錄惡意程式特徵，例如：hash 值、檔案名稱、domain name 等。並以 AND 及 OR 將上述資訊進行

簡單邏輯判斷，加速對惡意程式之辨識。也因 IOC 是以標準格式撰寫，使威脅情資共享之便利度大幅提升。OpenIOC 規則範例請參閱圖一。



圖一：偵測 Stuxnet 惡意程式之 OpenIOC 規則範例

YARA [10] 亦是廣受歡迎之情資偵測規則。與 OpenIOC 差別在於 YARA 之欄位定義更具有彈性，使用者可自由定義特徵欄位，並以字串形式填入欄位值。此外，與 OpenIOC 相比，YARA 規則亦將相關資訊分成三類，meta 資料、strings 字串資料與 condition 判斷邏輯。在 meta 資料區塊中，使用者可撰寫規則相關背景資訊。在 strings 字串區塊中，使用者可撰寫惡意程式入侵威脅指標。而在 condition 中，使用者則可利用 strings 區塊中已定義之入侵威脅指標進行基本邏輯運算，藉以判定某惡意程式，是否符合該規則中描述之指標。相較於 OpenIOC 規則運用排版結構進行基本邏輯運算，YARA 則是偏向以語意描述方式進行基本邏輯運算。YARA 規則範例請參閱圖二。

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true
    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
    condition:
        $a or $b or $c
}
```

圖二： Yara 官方提供之規則範例

capa [11] 為一套開源二進制檔分析工具，透過 capa 規則，使用者可從二進制檔中探詢是否符合特定入侵威脅指標。capa 規則之設計，融合 YARA 與 OpenIOC 兩者之特色。首先，capa 採用 YARA 將相關資訊分類之特點，並從三類簡化成兩類，將 strings 字串資料合併入 meta 資料區塊中，並保留 features 判斷邏輯區。這兩區塊資料之描述方式則採用 OpenIOC 規則中排版結構來處理。capa 規則範例請參閱圖三。

```yaml
rule:
  meta:
    name: hash data with CRC32
    namespace: data-manipulation/checksum/crc32
    author: moritz.raabe@mandiant.com
    scope: function
    examples:
      - 2D3EDC218A90F03089CC01715A9F047F:0x403CBD
      - 7D28CB106CB54876B2A5C111724A07CD:0x402350  # RtlComputeCrc32
  features:
    - or:
      - and:
        - mnemonic: shr
        - number: 0xEDB88320
        - number: 8
        - characteristic: nzxor
      - api: RtlComputeCrc32
```

圖三：capa 官方提供之規則範例

與上述規則相比，Quark 偵測規則之撰寫與交流希望能更便利，故採用 JSON 格式撰寫分析規則，Quark 偵測規則範例請參閱圖四。另外，Quark 引擎也更專注在 Android 二進制檔內行為之偵測，因此，我們將焦點放在 Android 原生 API 之呼叫過程與參數傳遞。使用者可透過撰寫兩個 Android 原生 API 到規則中。Quark 引擎即可協助使用者判定下列五項資訊：

1. Android 原生 API 所須之權限是否被請求？
2. 關鍵 API 是否被呼叫？
3. 特定 API 組合是否被呼叫？
4. 特定 API 是否依照正確順序被呼叫？
5. 兩個 API 間是否進行參數傳遞？

```
{
    "crime": "Initialize the recorder and start recording",
    "permission": [],
    "api": [
        {
            "class": "Landroid/media/MediaRecorder;",
            "method": "prepare",
            "descriptor": "()V"
        },
        {
            "class": "Landroid/media/MediaRecorder;",
            "method": "start",
            "descriptor": "()V"
        }
    ],
    "score": 1,
    "label": ["record"]
}
```

圖四：偵測錄音行為之 Quark 規則

2.3 開源資安工具偵測規則自動化生成機制探討

隨著資訊科技之進展，駭客攻擊手法亦不斷推陳出新。相關資安偵測工具之偵測規則亦須隨之增加。然而，若只以人工撰寫偵測規則，分析人員會遇到許多瓶頸。例如：規則品質優劣取決於分析人員資安經驗豐富與否、撰寫規則勞務成本過高，導致產生規則速度不及駭客攻擊手法演變速度。

為解決此問題，偵測規則自動化生成機制逐漸受到重視與討論。以 OpenIOC 規則為例 [9]，其研發出一套以自然語言處理 (NLP, Natural Language Processing) 方式分析資安報告，並從中萃取出有價值之入侵威脅指標。而針對 YARA 規則之自動化生成機制之探討亦甚多，例如：使用 Biclustering 技術自動化生產規則之學術研究 [12]，開源專案 yarGen [13]、yaraGenerator [14] 與 yabin [15] 等實作出規則自動生成系統。而相較於 OpenIOC 與 YARA，capa 在偵測規則自動化生成機制上，則是由 capa 團隊自行發展 [11]，而非仰賴學術研究或開源社群之研析。

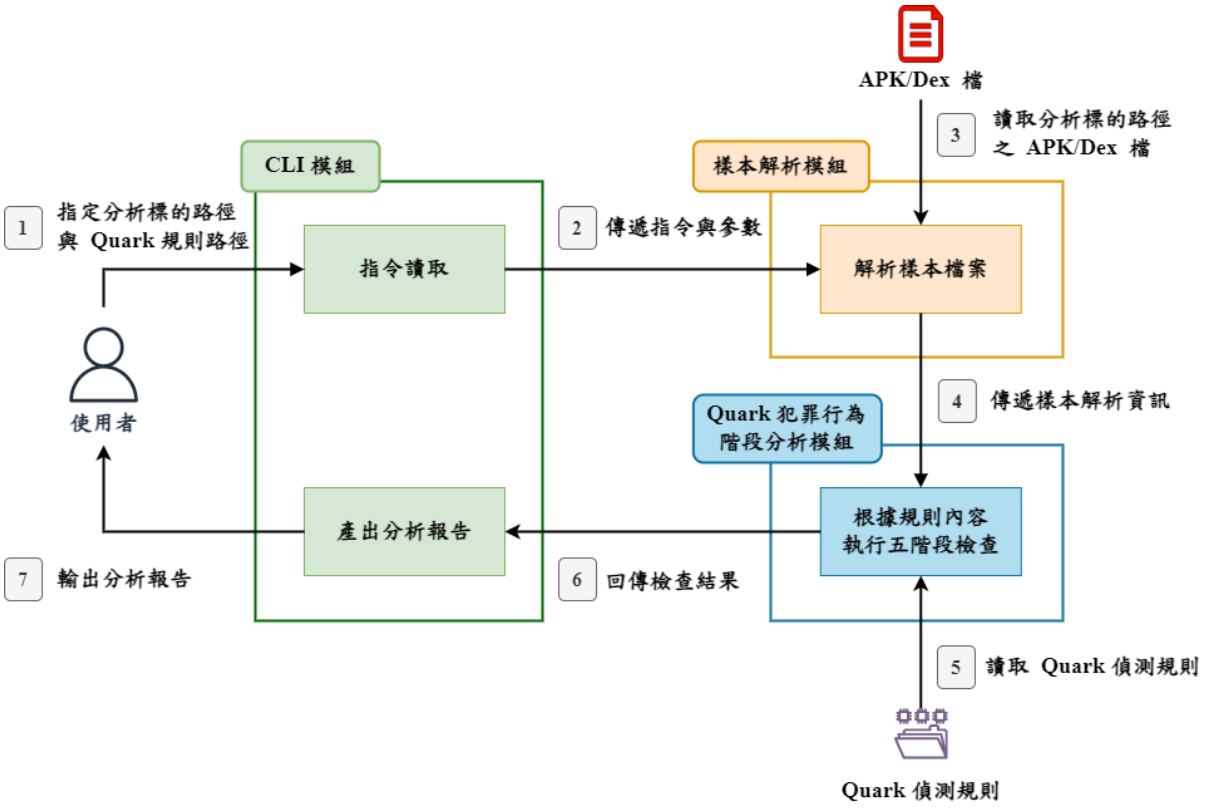
Quark 引擎團隊亦自主研發自動化規則生成機制，從規則生成之速度、高價值規則篩選及使用者操作體驗上，均投入巨量心力進行改善。此前，Quark 規則自動化生成機制已研發兩代系統 [16]，產出並篩選出 181 條高質量偵測規則。然而，我們認為前兩代系統在規則生成速度與高價值規則選取上，仍有許多改善空間。因此，本研究提出第三代偵測規則自動化生成機制，以大幅改善規則生成之效率。

參、研發方法與設計原理

本章節首先介紹 Quark 引擎，說明偵測 Android 應用程式潛在惡意行為之方法。接著，介紹 Quark 偵測規則之組成與撰寫方法。最後，介紹過往研發之兩個自動化規則生成系統，說明其設計原理，點出兩版本系統之限制，並針對這些限制提出解決方案，研發出第三代自動化規則生成系統。

3.1 Quark 引擎設計解說

Quark 引擎之系統架構包含三大功能模組。文字指令介面 (CLI, Command Line Interface) 模組，其核心功能為提供使用者文字互動介面，以及處理 Quark 引擎之輸入 (例如：指令讀取) 與輸出 (例如：產出分析報告)。樣本解析模組，其核心功能為解析分析標的之 Android 應用程式檔案 (APK 或 Dex 檔)，並提供 Quark 引擎分析標的之解析資訊 (例如：請求之權限、使用之原生 API 與函式等資訊)。Quark 犯罪行為階段分析模組，其核心功能為依照輸入之 Quark 偵測規則，針對分析標的進行五階段檢查，並計算檢查結果。Quark 引擎之系統架構與執行流程請參閱圖五。



圖五：Quark 引擎系統架構與執行流程圖

根據圖五之流程，Quark 引擎之運作可分為七個步驟。第一步，使用者輸入之指令與參數至 CLI 模組，輸入指令可包含：欲分析之 Android 應用程式檔案 (以下簡稱樣

本) 路徑、欲分析之規則路徑、輸出分析報告之類型，以及選擇其他 Quark 引擎所提供之功能等。第二步，將輸入指令與參數傳遞至樣本解析模組。第三步，依據使用者輸入之路徑，讀取分析樣本 APK/Dex 檔，並解析出樣本資訊，包含：樣本之 Android 權限 (permissions) 、Android 原生 API 與自定義函式資訊。第四步，將樣本資訊傳至 Quark 犯罪行為階段檢查模組。該模組為 Quark 團隊藉由解構法律設計之犯罪行為階段論，研發出針對 Android 惡意行為之五階段檢查，其核心原則在於，偵測之階段越多，越能確定該行為被確實執行。第五步，讀取 Quark 偵測規則，並依照規則內容，以 Quark 犯罪行為階段檢查模組，針對樣本進行五階段之檢查，檢查項目如下：

1. 檢查 Android 原生 API 所須之權限是否被請求？
   該階段檢查樣本中是否請求規則定義之權限。部分高風險之原生 API ，需事先向 Android 系統請求相應之權限，否則無法執行。因此，權限請求可視為潛在惡意行為之前置作業，故首先須確認該樣本之權限請求。然而，並非所有高風險 API 皆須請求權限，在此情況下該項檢查可被忽略。
2. 檢查關鍵 API 是否被呼叫？
   該階段檢查樣本中是否呼叫規則定義之任一 API。Quark 規則所定義之惡意行為，以兩個關鍵 API 組成，該階段之目的為確任關鍵 API 是否被樣本所呼叫。
3. 檢查特定 API 組合是否被呼叫？
   該階段檢查樣本中是否呼叫規則定義之兩個 API。目的為近一步確認，樣本內是否存在該行為之 API 組合。
4. 檢查特定 API 是否依照正確順序被呼叫？
   該階段檢查兩 API 是否出現於相同函式中，且呼叫順序是否與規則所定義之順序一致。API 呼叫之順序為判定惡意行為關鍵之一，例如：藉由簡訊將使用者帳戶資料傳出，該行為須優先執行 getAccount() 以取得帳戶資訊，並接續執行 sendTextMessage() 以寄送簡訊，該行為才得以成立。
5. 檢查兩個 API 間是否進行參數傳遞？
   該階段檢查兩 API 之呼叫是否共用相同暫存器。此階段可證明，兩 API 具有傳遞參數之關係，例如：若 getAccount() 與 sendTextMessage() 共用暫存器，即可證明，該樣本確實將帳戶資訊傳遞至簡訊寄送之 API。至此，可確認該規則所定義之潛在惡意行為被確實執行。

此外，五階段檢查以信心指數 (confidence) 量化分析結果。若規則符合一個階段，其信心指數增加 20%，符合兩個階段則為 40%，依此類推 (五階段共 100%)。當規則具有 100% 之信心指數，則可確認該規則所描述之行為於樣本中被確實執行。第六步，將五階段檢查之結果，包含規則內容與信心指數傳至 CLI 模組。第七步，將分析結果，以簡潔易懂之概覽報告呈現於指令介面，分析人員透過該報告可快速理解樣本內之潛在惡意行為。

Quark 引擎之核心設計，專注於解析 Android 原始碼，找出惡意程式 payload 存在之惡意行為特徵。Quark 偵測規則為惡意行為定義出關鍵 API 資訊，作為 Quark 引擎搜尋樣本內惡意行為之重要依據。因此，偵測規則之設計亦為 Quark 引擎之核心。以下將說明 Quark 偵測規則之設計原理與資料組成。

3.2 Quark 偵測規則設計解說

在 Quark 偵測規則中，我們以 Android 權限 (permission) 以及兩個 Android 原生 API，定義出一個潛在惡意行為，並以語意方式提供該惡意行為之描述。Quark 規則為 12 項資訊組成之 JSON 檔案，一個惡意行為即一個規則，亦即一個 JSON 檔。Quark 偵測規則之資組成如表一所示。

JSON (JavaScript Object Notation) 為常見之資料呈現格式，Quark 偵測規則以 JSON 格式呈現，其目的為利用 JSON 之通用性，使撰寫規則之門檻降低，分析人員無須花費額外學習成本即可撰寫 Quark 偵測規則。

其中 API1 與 API2 為一 JSON Array 所組成，其定義出兩個 API 之組合與呼叫順序 (API1 優先於 API2 被呼叫)。API1 與 API2 之 Class, Method 以及 Descriptor 皆以 smali 呈現，該語法為 Android 運行於 Dalvik 虛擬機器(DVM, Dalvik Virtual Machine) [17] 之組合語言 (Assembly language)。Android 應用程式逆向分析之方法之一，即是將應用程式轉譯為 smali 語法。因此，Quark 規則皆以 smali 語法表示 API 之資訊。其中 Score 為評斷規則定義之惡意行為風險分數，作為 Quark 引擎評分系統之重要依據。然而，目前規則之數量尚不足以使評分系統產生效用，因此 Score 皆預設為 1。

表一：Quark 規則之組成資訊

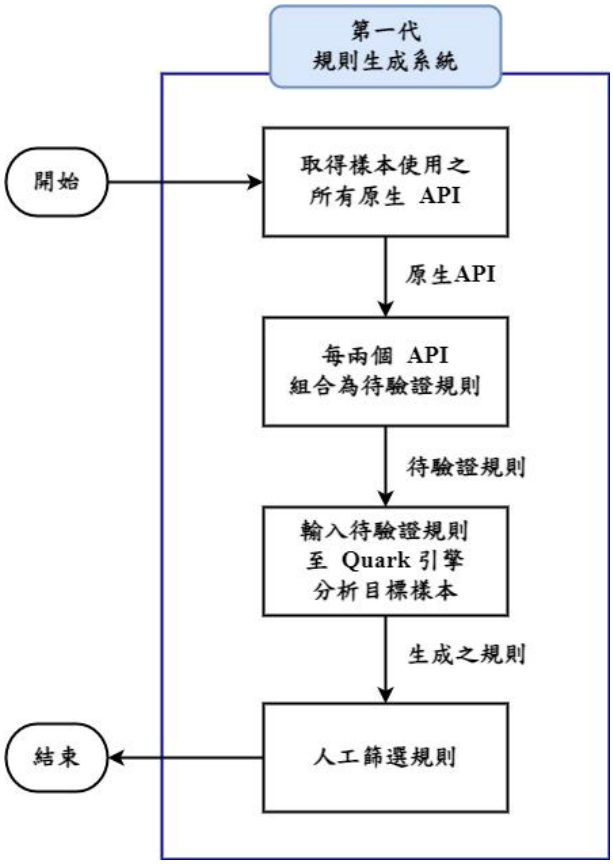
元素		**內容描述**
Crime		該規則所定義之潛在惡意行為描述。
Permissions		該惡意行為所需存取之 Android 權限。
API1	Class	第一個 API 之類別。
	Method	第一個 API 之函式名稱。
	Descriptor	第一個 API 之輸入參數與回傳值型態。
	Match keywords	第一個 API 輸入參數之關鍵字。
	Class	第二個 API 之類別。

API2	Method	第二個 API 之函式名稱。
	Descriptor	第二個 API 之輸入參數與回傳值型態。
	Match keywords	第二個 API 輸入參數之關鍵字。
Score		該規則定義之惡意行為風險分數 (預設為 1)。
Label		該規則之分類關鍵字。

Quark 引擎作為 rule-based 偵測引擎，規則數量越多，其分析結果越可靠。然而，人工撰寫規則，需耗費大量勞務成本。再者，其規則品質仰賴撰寫者之資安經驗多寡。因此，為克服上述兩項瓶頸，Quark 團隊研發自動化規則生成系統，快速且大量生成 Quark 偵測規則。包含本研究所研發之第三代自動化規則生成系統，目前一共有三個版本。以下將針對各版本逐一介紹並說明其原理。

3.3 Quark 第一代自動化規則生成系統

根據以上針對人工撰寫限制之論述，Quark 引擎首先面臨的是缺乏規則之問題，若偵測規則數量少，Quark 引擎則無法提供有效之分析報告。因此，為提升偵測規則之數量，Quark 團隊研發出第一代自動化規則生成系統。其生成規則流程如圖六所示。



圖六：第一代自動化規則生成系統流程圖

本研究依據該流程設計一演算法，如 Algorithm 1 所示。首先，輸入欲分析之樣本，並取出樣本使用之所有 Android 原生 API。接著，每兩個 API 組合為待驗證規則 (忽略兩 API 相同之情況)，該方法等同不重複排列 (Permutaion)，若所有 API 數為 n 則待驗證規則之數量為 P_2^n。每個待驗證規則一旦生成，便將其輸入至 Quark 引擎以分析目標樣本，檢查該待驗證規則是否符合 100% 信心指數 (confidence)。若是，則將其輸出成 JSON 檔，該規則即是系統之生成規則。最後，以人工方式從生成規則中篩選出有效規則 (可偵測實際惡意行為之規則)。

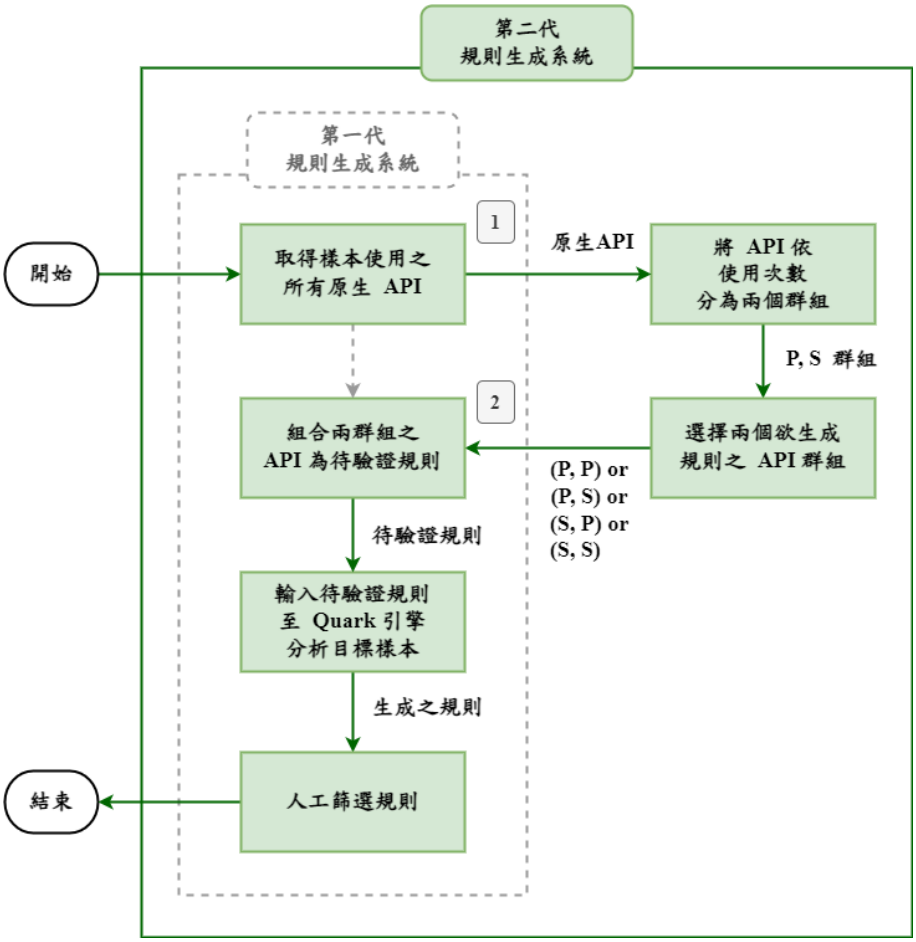
Algorithm 1: First version of Quark rule generation algorithm

```
Input  : sample
Output: generatedRuleList
1  sample← Android malware APK/Dex file
2  apiSet ← All native APIs used in sample
3  generatedRuleList ← [ ]
4  for api_1 in apiSet do
5      for api_2 in apiSet do
6          combination ← CombineAPI(api_1, api_2)
7          rules ← Quark(sample, combination) // Quark analysis
8          generatedRuleList.append(rules)
9      end for
10 end for
11 return generatedRuleList
```

然而，該演算法存在兩技術瓶頸。其一，計算量極高，若 $apiSet$ 數量為 n，且如 Algorithm 1 第 4 至 10 行所示，演算法兩層 for loop 執行 n^2 次規則組合，扣除重複情況則為 $n^2 - n$ 次，以漸進表示法取最高項次，其時間複雜度為 $O(n^2)$ 。其二，篩選規則之勞務成本高，生成規則未經整理歸納，其龐大數量造成人工篩選之勞務成本過高，篩選效率低。因此，為克服計算量以及規則篩選工作量龐大之問題，本研究針對該系統進行改進，研發出第二代自動化規則生成系統。

3.4 Quark 第二代自動化規則生成系統

第二代自動化規則生成系統以第一代為基礎進行改良，目的是以較低計算量生成高價值規則。因此，Quark 團隊改良第一代系統，於步驟 1 (取得樣本使用之所有原生 API) 與步驟 2 (每兩個 API 組合為待驗證規則) 之間，新增兩個步驟。其一，將原生 API 依照使用次數分成兩個不同群組。其二，依照使用情況選擇兩個 API 群組生成規則。如圖七所示。



圖七：第二代自動化規則生成系統流程圖

為產出少量但高價值之規則，透過 80/20 法則 [18]，將 API 依照在樣本中被使用次數分為兩個群組，P (Primary) API 使用次數最少之 20%；S (Secondary) 使用次數最多之 80%。根據我們過往惡意程式分析經驗，使用次數少之 API 具有較高價值，舉例來說，相較於使用次數少之 API sendTextMessage()，API toString() 使用次數多，其功能可廣泛應用在各種情境，因此分析師難以判定其使用情境，故該 API 價值較低。

Algorithm 2: Second version of Quark rule generation algorithm

```
Input  : sample
Output: generatedRuleList
1  sample← Android malware APK/Dex file
2  apiSet ← All native APIs used in sample
3  generatedRuleList ← [ ]
4  P, S ← Sorting(apiSet) // Sort and split apiSet by used count
5  // Choose (P, P) or (P, S) or (S, P) or (S, S) to generate rules
6  // Below is the case of (P, P)
7  for api_1 in P do
8      for api_2 in P do
9          combination ← CombineAPI(api_1, api_2)
10         rules ← Quark(sample, combination) // Quark analysis
11         generatedRuleList.append(rules)
12     end for
13 end for
14 return generatedRuleList
```

第二代系統之演算法如 Algorithm 2 所示。其中第 4 行為依照 API 使用次數分為 P 與 S 兩群組，可排列為 (P,P)、(P,S)、(S,P) 與 (S,S) 四組合，分析師可依照需求，選擇任一組合生成規則，如第 5 至 13 行所示。相較於第一代系統，第二代將規則分為四種組合，其生成規則之價值與計算量預估之比較，請見表二所示。時間複雜度之計算參考第一代系統，若 apiSet 數為 n 則 P 為 $0.2n$，S 為 $0.8n$，而，P 與 S 之組合其計算次數如表二所示。結果顯示，(P,P) 之時間複雜度至少比第一代之 $O(n^2)$ 減少 96% 之計算量。

表二：四組合之規則價值與計算量預估

組合	規則價值	計算量	時間複雜度
(P,P)	高	低	$O(0.04n^2)$
(P,S)	中	中	$O(0.16n^2)$
(S,P)	中	中	$O(0.16n^2)$
(S,S)	低	高	$O(0.64n^2)$

然而，即使第二代演算法之 (P,P) 組合已大幅將低生成規則數量以及人工篩選成本，該演算法則犧牲產出規則之偵測命中率 (部分 API 被 80/20 法則過濾而無法生成規則)。因此，第三代演算法改進之重點，在於針對特定目標函式之 API 生成規則，以提升其命中率。

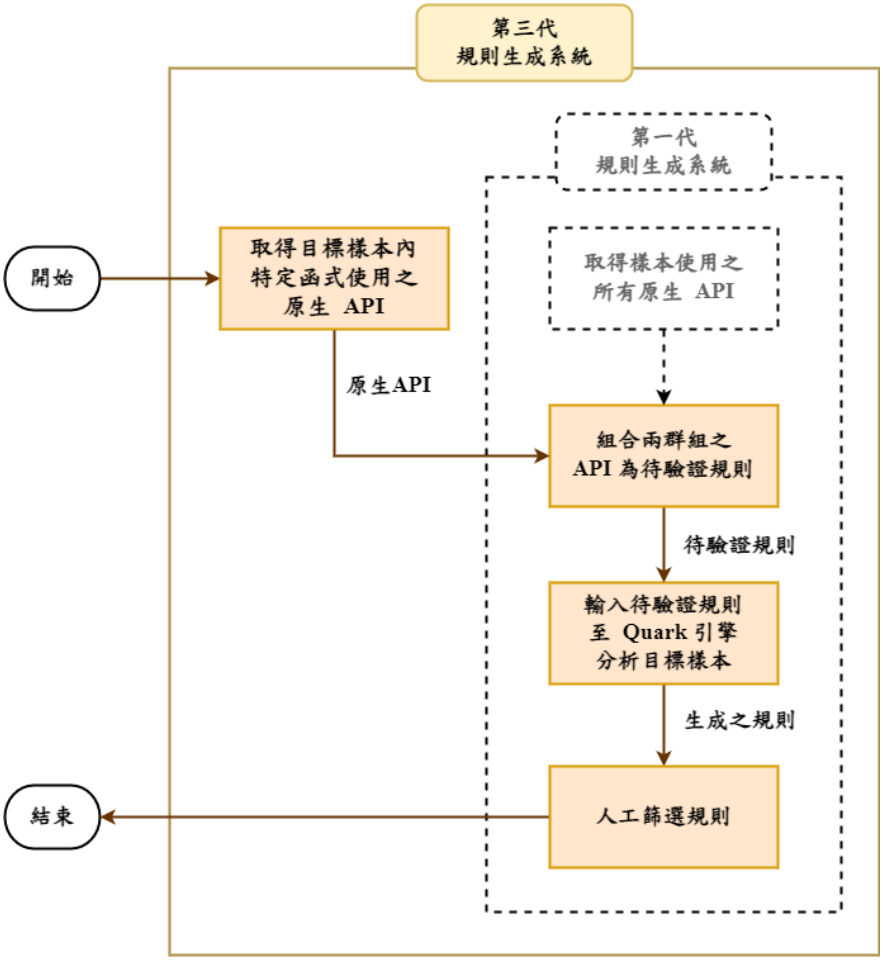
3.5 Quark 第三代自動化規則生成系統

第二代自動化規則生成演算法，優先挑選高價值 API 以生成規則。其大幅降低運算量同時也能產出高價值規則。然而，該演算法犧牲規則命中率，導致惡意程式分析師

可能遺漏關鍵線索。再者，即使部分生成規則，可偵測出關鍵惡意行為，仍會因部分規則缺失，而無法完整解釋惡意行為之脈絡。

第一代與第二代系統之演算法反映出，規則數量與規則命中率之取捨問題。為找出兩者之平衡，我們研發出第三代系統。該系統之演算法，將生成規則之 API 範圍，從惡意程式樣本使用之所有 API，限縮為該樣本中目標函式所使用之 API。此方法降低 API 組合之基數，進而大幅減少計算量。同時，該演算法無需過濾 API，故生成規則也可完整揭露目標函式之行為，使分析師有充足資訊可理解其脈絡。

第三代規則生成系統流程，以第一代為基礎進行改進，其流程如圖八所示。第一代系統之 API 範圍包含目標樣本中所有使用之原生 API，而第三代系統則聚焦於，目標樣本中特定函式所使用之 API，其餘步驟皆與第一代相同。



圖八：第三代自動化規則生成流程圖

其演算法新增一個輸入 targetFunction，表示為樣本中欲生成規則之目標函示。而 apiSet 之 API 來源也改為 targetFunction 中使用之 API，如 Algorithm 3 第 2 行所示。與第一代系統比較，若樣本內之使用 API 數量為 a，目標函式使用之 API 為 b，一般情況下，$a > b$ 故 $O(a^2) > O(b^2)$，其時間複雜度為第三代較小於第一代，亦即第三代之計算量少於第一代。

Algorithm 3: Third version of Quark rule generation algorithm

```
Input  : sample, targetFunction
Output: generatedRuleList
sample ← Android malware APK/Dex file
apiSet ← All native APIs used in targetFunction
generatedRuleList ← [ ]
for api_1 in apiSet do
    for api_2 in apiSet do
        combinations ← CombineAPI(api_1, api_2)
        rules ← Quark(sample, combination) // Quark analysis
        generatedRuleList.append(rules)
    end for
end for
return generatedRuleList
```

本研究之自動化規則生成系統，其時間複雜度大小取決於 API 組合數量多寡 (每個組合之有效性須以 Quark 引擎驗證)，故時間複雜度亦可視為該系統計算量之預估。若第一代系統 API 數量為 a，第二代系統之 (P,P) 群組 API 數量為 b，第三代系統 API 數量為 c。根據本章節各代系統之論述，b 為 $0.2a$ (經 80/20 法則篩選) 故 $a > b$。一般來說，第三代系統輸入函式使用之 API 數應小於應用程式使用 API 總數之 20%，故 $b > c$。各代系統時間複雜度之比較，$O(a^2) > O(b^2) > O(c^2)$。由此可知，第二代與第三代之計算量皆遠小於第一代 (第二代計算量預估為第一代之 0.04 倍)，且第三代之計算量亦小於第二代，故第三代系統所須之時間成本、運算成本皆比前兩代低。

根據以上論述，自動化規則生成系統每次迭代更新，新演算法效能皆更優於上代系統，而本研究提出之第三代系統之效能最佳。然而，每代系統皆有其優劣之處。例如，第一代系統之有效規則數最多，但計算量也最高。而新系統之解決方案，皆需要犧牲部份優勢。例如，第二代系統大幅降低計算量，但生成規則無法完整解釋惡意行為脈絡。第三代系統在有明確分析標的之情況 (已鎖定惡意程式樣本之特定函式)，是最有效率之解決方案。因此，下一章將提供真實惡意程式之實驗數據，觀察各代系統之數據比較是否與本章所述一致，以驗證第三代系統之演算法為真實有效之方案。

肆、生成規則實驗實驗

本研究以兩種真實惡意程式樣本作為實驗標的，比較第三代自動化規則生成系統，與前兩代系統產出規則之命中率與規則生成率。根據第三章所述，自動化規則生成系統主要目的為減少人工撰寫規則之勞務成本。因此，為盡可能降低人為介入，首先需保證生成規則之有效性，亦即確保生成規則可確實偵測出惡意行為。其次，提升系統之運作

效能，提高規則生成率，優化規則生成之效率。因此，為確保第三代規則生成系統之效益，以本章之實驗進行驗證。

4.1 實驗標的

根據第三章之描述，Quark 引擎之定位是專注於分析惡意程式 payload，並快速偵測該 payload 之行為，因此本實驗所挑選之實驗標的，為真實 Android 惡意程式 payload。為驗證自動化規則生成系統，在分析不同惡意程式特徵下皆能發揮效用。實驗標的所挑選之 Android 惡意程式 payload (以下簡稱樣本)，為兩不同家族之應用程式 (同家族之惡意程式具有相似原始碼)，分別為 Romaing Mantis 與 Ahmyth Rat。兩樣本資訊如表三所示。

表三：目標樣本資訊

樣本家族	packagename	md5
Roaming Mantis	ertt.fgh.nfg	7aa46b4d67c3ab07caa53e8d8df3005c
Ahmyth Rat	ahmyth.mine.king.ahmyth	893e05aabd8754236ea70d3da8363d52

Roaming Mantis [19] 為 2018 年卡巴斯基 (Kaspersky) 防毒軟體公司所發現之惡意程式，該惡意程式以 DNS 劫持 (DNS hijacking)，誘使受害者點擊經偽裝之連結，使受害者下載並安裝惡意程式 payload，其中針對 Android 設備之 payload 為木馬程式，可控制受害者設備，進一步竊取敏感資訊，例如：控制設備錄音、取得帳戶資訊等。其後續之變種則可透過簡訊釣魚 (SMS phishing) 方式傳播 payload，本實驗所使用之樣本即為文獻 [19] 中提供 payload 之一。Ahmyth Rat 為一開源惡意程式樣本 [20]，該樣本為針對 Android 之間諜應用程式，其惡意行為包括控制手機拍照、控制手機錄音、竊取 SMS 資訊，並將資料傳至攻擊伺服器之行為。

4.2 實驗設計

為比較各版本自動化規則生成系統之有效規則比例，以及生成規則之效率，本實驗以命中率 (1) 與規則生成率 (2) 進行比較，其計算方式如下所示。

$$\text{命中率} = \text{有效規則總數} / \text{生成規則總數} \times 100\% \quad (1)$$

$$\text{規則生成率} = \text{生成規則總數} / \text{API 組合總數} \times 100\% \quad (2)$$

命中率為有效規則與生成規則 (100% confidence 之規則) 總數之比例。有效規則為經人工篩選後，確認可偵測出惡意行為之規則，因此生成規則之數量，亦可視為人工篩選規則所需勞務成本之量化數據。規則生成率為平均 1 個 API 組合可生成之規則數比

例，其中 API 組合數可視為系統生成規則之計算量。

在本實驗中，我們參考 Roaming Mantis 分析報告 [19]，以及 Ahmyth Rat 官方文獻 [20]，從中各選擇 3 個惡意行為，作為篩選有效規則之標準。其中 Roaming Mantis 挑選錄音 (Audio Recording)、控制手機發送簡訊 (Send SMS) 與連接外部網站三個惡意行為。而 Ahmyth Rat 則挑選錄音 (Audio Recording)、控制手機拍照與竊取 SMS 資訊。兩樣本皆選擇錄音行為，目的為驗證同行為不同特徵下，生成規則演算法依然有效。

系統生成規則後，依照上述惡意行為之 API 關鍵字搜尋，舉例來說，兩樣本之錄音行為使用之 API，其 Class 名稱皆有 MediaRecorder 關鍵字，因此以包含 MediaRecorder 關鍵字之規則，作為錄音行為之有效規則。接著，以我們過往分析經驗篩選較有價值之規則，但為求客觀數據，因此實驗數據僅以關鍵字過濾後之數量進行比較。

需注意的是，第三代自動化規則生成系統，需額外選定目標函式，以生成相對應之規則。因此，上述惡意行為皆各需挑選一目標函式，作為第三代系統之輸入。該函式之挑選方法，依照上述惡意行為之 API 關鍵字，搜尋樣本原始碼內出現關鍵字之自訂義函式 (忽略第三方函式庫之函式) 作為目標函式。兩樣本之目標函式請參閱表四。

表四：第三代系統之目標函式

Roaming Mantis		**Ahmyth Rat**	
惡意行為	**目標函式**	**惡意行為**	**目標函式**
控制 手機錄音	Lcom/j;->a()V	控制 手機錄音	Lahmyth/mine/king/ahmyth/MicManager;->startRecording(I)V
控制手機 發送簡訊	Lcom/Loader$k$1;->a(Ljava/lang/Object;)V	控制 手機拍照	Lahmyth/mine/king/ahmyth/CameraManager;->startUp()V
連接 外部網站	La/b;->a(Ljava/lang/String;)[B	竊取 SMS 資訊	Lahmyth/mine/king/ahmyth/SMSManager;->getSMSList()Lorg/json/JSONObject;

本實驗依照上述方法，以各版本自動化規則生成系統，對兩樣本進行規則生成，並紀錄有效規則數、生成規則數與 API 組合數，接著依照方程式 (1) (2)，計算命中率與規則生成率，最後針對各版本系統之數據進行比較。以下說明實驗結果。

4.3 實驗結果

兩樣本之有效規則數、生成規則數與 API 組合數，如表五所示。其中，每代系統之有效規則數，為依照該樣本於實驗設計中挑選之 3 個惡意行為，取各行為有效規則數之總和。舉例來說，第一代系統生成 Roaming Mantis 之規則中，可偵測錄音、控制手機發送簡訊與連接外部網站，三個惡意行為之有效規則數總和為 818。

實驗數據顯示，第一代系統之有效規則數、生成規則數與 API 組合數，皆遠高於其他版本之系統，亦即第一代系統即使有最多有效規則，其篩選規則之勞務成本 (生成規則數) ，以及計算量 (API 組合數) 亦最高。第二代系統之有效規則數，雖不及第一代系統，但 (P,P)、(P,S) 與 (S,P) 之生成規則數與 API 組合數，則遠低於第一代，其中 (P,P) 之差距尤為明顯，亦即第二代系統可明顯改善，第一代之龐大計算量與人工篩選規則之勞務成本。然而，第二代系統因部分 API 被 80/20 法則過濾無法生成規則，故無法完整解釋惡意行為之脈絡。第三代系統之有效規則數，雖不及第一代之多，但其生成規則數與 API 組合數則遠低於其他系統，且有效規則數與第二代之 (P,P) 群組相比無明顯差距，亦即第三代系統能以更少之計算量與人工篩選成本，達到第二代系統之效益。

表五：樣本生成規則之數據

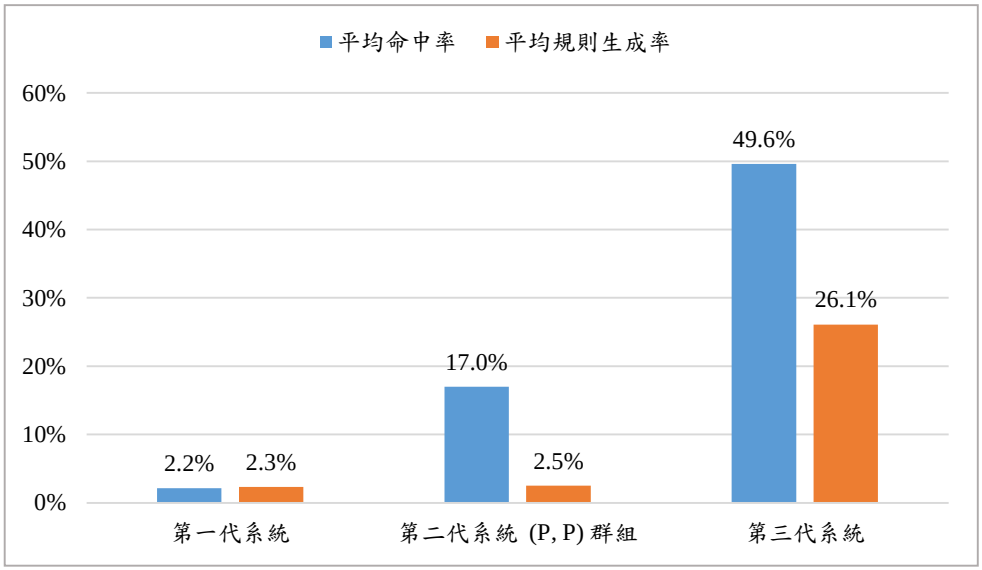
系統版本		Roaming Mantis			Ahmyth Rat		
		有效規則數	生成規則數	API 組合數	有效規則數	生成規則數	API 組合數
第一代		818	26,868	1,178,310	146	11,059	474,032
第二代	(P,P)	147	1,654	47,306	71	282	18,906
	(P,S)	85	2,141	189,224	34	859	76,038
	(S,P)	143	2,109	189,224	41	782	76,038
	(S,S)	443	19,737	752,556	0	9,136	303,050
第三代		265	534	1778	62	125	561

根據表五之數據，我們計算出各代系統之命中率與規則生成率，如表六所示。實驗結果顯示，第三代系統之規則命中率平均為 49.6%，而規則生成率平均為 26.1%，皆為各代系統中最高。其次，第二代系統 (P,P) 群組之命中率平均為 17 %，然而其規則生成率平均為 2.5% 與第一代之 2.3%相比無顯著提升。而第一代系統雖然可生成最多數

量之有效規則，但其命中率平均為 1.8%，生成率平均 2.3%，在各代系統中最低。

表六：樣本之命中率與生成率

		Roaming Mantis		Ahmyth Rat		兩樣本平均數據	
系統版本		命中率	規則生成率	命中率	規則生成率	命中率	規則生成率
第一代		3%	2.3%	1.3%	2.3%	2.15%	2.3%
第二代	(P,P)	8.8%	3.5%	25.1%	1.5%	17%	2.5%
	(P,S)	3.9%	1.1%	4%	1.1%	3.95%	1.1%
	(S,P)	6.7%	1.1%	5.2%	1%	5.95%	1.05%
	(S,S)	2.2%	2.6%	0%	3%	1.1%	2.6%
第三代		49.6%	30%	49.6%	22.2%	49.6%	26.1%



圖九：三代系統之平均命中率與平均規則生成率

實驗結果表明，相較於第一代系統與第二代系統 (P,P) 群組，第三代系統之效率最高，其命中率與規則生成率，皆遠高於其他版本之系統，如圖九所示。且其生成規則數最少，亦即篩選規則之勞務成本也最低。因此，在確認樣本之惡意行為，且已鎖定欲生成規則函式之情況，該系統無疑是最有效率之解決方案。然而，若在毫無線索之情況(分析未知來源之樣本)，第二代系統之 (P,P) 群組，其平均命中率仍高於第一代，且計算量則遠小於第一代，篩選規則之勞務成本也相對少，可作為快速概覽樣本行為之折衷方案。

伍、結論

此前，Quark 團隊已研發出兩代自動化規則生成系統，該系統主要解決以人工撰寫 Quark 規則，其品質不穩定 (規則品質優劣取決於撰寫人員資安經驗豐富與否)，以及勞務成本過高 (規則產出速度無法跟上惡意程式變化) 之問題。本研究提出第三代系統以改進前兩代之瓶頸，並以兩個真實惡意程式進行生成規則實驗。以下則根據實驗結果加以討論，並提出效益、限制以及未來研究方向。

5.1 研究結果

研究結果顯示，自動化規則生成系統能大幅提升偵測規則產出速度，同時降低撰寫規則之勞務成本。此外，該系統提取目標樣本所使用之原生 API ，並自動化組合 API 以產生規則，因此可解決人工撰寫規則品質不穩定之問題。包括本研究提出之第三代，三個版本之自動化規則生成系統皆有其優劣之處。第一代系統生成之有效規則數最多，但其計算量與篩選規則之勞務成本最高。第二代系統之命中率與規則生成率皆優於第一代，但會因過濾 API 使用次數而導致規則缺失。第三代系統各項數據皆為最優，根據本研究之實驗數據，其平均命中率高達 49.6%，亦即近半數生成規則能夠偵測該樣本之惡意行為。除此之外，其有效規則數遠低於第一代與第二代，有效規則數少，亦即人工篩選規則之勞務成本低。總體而言，第三代規則生成引擎之效益最佳。然而，第三代系統需事先選定欲生成規則之函式，因此在毫無線索之情況下，分析師較難選定目標函式。

5.2 研究效益

本研究之第三代自動化規則生成系統，其大幅讀優化前兩代之效能，也因此提高惡意程式分析師之工作效率，除自動化生成可有效偵測惡意行為之規則，分析師透過解析有效規則，能清楚理解 Android 惡意程式之行為脈絡。例如，生成針對操控相機行為之規則，可發現該行為被拆成設定相機參數、設定輸出檔案路徑、控制相機拍照、圖片壓縮與將壓縮圖片傳至 JSON 等五個步驟，分析師即可透過這些資訊，快速理解該樣本之行為。此外，該系統也為 Quark 引擎新增 204 個通用且可偵測出惡意行為之規則，為惡意程式分析師提供快速概覽樣本之方法，避免漫無目的從龐大原始碼中搜尋惡意行為線索。

5.3 研究限制與未來研究方向

本研究在過程中礙於 Quark 引擎底層函式庫，記憶體使用效率問題，使自動化規則生成系統在分析 Android 樣本時，記憶體佔用量過於龐大，部份樣本在分析過程因此

中斷。因問題源頭在於底層函式庫之設計，故難以從根本上解決，建議能使用其他相同功能之底層函式庫進行研究。

未來研究方向建議針對以下兩點進行自動化規則生成系統之優化：

1. 規則篩選流程之優化

篩選之規則為確保能正確分析出惡意行為，通常需具備資安相關經驗之分析師進行篩選，故完全消除人為介入相當困難。因此，未來研究應著重於提升篩選規則之效率。例如，將生成規則進行自動化過濾，去除第三方套件之 API 組合，或自動以關鍵字篩選相關 API。若篩選規則之效率提高，也能提升自動化規則生成系統之實用性。

2. 目標函式挑選流程之優化

在某些情況下，分析師無從得知樣本之惡意行為，更難以找出執行惡意行為之函式。因此，當分析師使用第三代系統生成規則時，無法定義出欲輸入之函式。未來研究方向建議可結合第二代系統之 (P, P) 群組，快速生成高價值之規則，同時列出這些規則之函式，接著再以第三代系統針對每個函式進行分析。如此一來，便能解決分析師無從搜尋目標函式之情況。

[誌謝]

特此感謝國立中山大學資訊工程學系資訊安全實驗室，提供本研究演算法與實驗設計建議，以及協助自動化規則生成系統之計算量評估。同時，感謝諸多 Quark 貢獻者如 Fortinet 首席研究員 Axelle Apvrille、MobSF 專案發起人 Ajin Abraham 與 IntelOwl 專案發起人 Matteo Lodi 等人，協助 Quark 程式碼維護、偵錯以及效能改善。

參考文獻

[1] F. Laricchia, "Mobile operating systems' market share worldwide from January 2012 to January 2022," 2 2021. [Online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

[2] Quark Engine, https://github.com/quark-engine/quark-engine.

[3] A. Apvrille, "Reverse Android malware like a Jedi Master", https://vblocalhost.com/presentations/reverse-android-malware-like-a-jedi-master/.

[4] Apktool, https://ibotpeaches.github.io/Apktool/.

[5] Jadx, https://github.com/skylot/jadx.

[6] Frida, https://frida.re.

[7] Objection, https://github.com/sensepost/objection.
[8] Dexcalibur, https://github.com/FrenchYeti/dexcalibur.
[9] D. K. WILL GIBB, "OpenIOC: Back to the Basics," https://www.mandiant.com/resources/openioc-basics.
[10] Yara, https://github.com/VirusTotal/yara.
[11] Capa, https://github.com/mandiant/capa.
[12] Biclustering, https://scikit-learn.org/stable/modules/biclustering.html.
[13] yarGen, https://github.com/Neo23x0/yarGen.
[14] yaraGenerator, https://github.com/Xen0ph0n/YaraGenerator.
[15] yabin, https://github.com/AlienVault-OTX/yabin.
[16] Quark Rule Generate, https://github.com/quark-engine/quark-rule-generate.
[17] Android Runtime（ART）and Dalvik, https://source.android.com/devices/tech/dalvik.
[18] C. TARDI, "80-20 Rule", https://www.investopedia.com/terms/1/80-20-rule.asp.
[19] S. ISHIMARU, "Roaming Mantis dabbles in mining and phishing multilingually", https://securelist.com/roaming-mantis-dabbles-in-mining-and-phishing-multilingually/85607/.
[20] Ahmyth Rat, https://github.com/AhMyth/AhMyth-Android-RAT.