Communications of the CCISA
Vol. 27 No. 1 Feb. 2021

全系統廣告阻擋軟體的隱憂與挑戰

張禾金[1]、黃俊穎[2*]
[12]國立交通大學網路工程研究所、國立陽明交通大學網路工程研究所
[1]eft823177@gmail.com、[2]chuang@cs.nctu.edu.tw

摘要

隨著數位廣告的興起，大量投放廣告在網站及各個行動應用程式中已經成為日常生活中的常態，已獲取利益或使用者個資。為與之抗衡，市面上興起各式各樣的廣告阻擋軟體。本論文提出一個全系統廣告阻擋的概念，探討目前全系統廣告阻擋軟體在隱私及安全上的議題，並指出其可能面臨的三個挑戰：網域名稱加密、基於網域名稱的廣告阻擋效益，以及阻擋效率。

我們首先探討新興的 DNS-over-HTTPS (DoH) 機制。DoH 針對網域名稱進行加密，可能大幅壓抑廣告阻檔的功能。我們提供一個概念實驗以與 DoH 抗衡。我們接下來分析基於 URL 或網域名稱的黑名單廣告阻擋，並比較其阻擋成效。而我們發現兩者差異尚可接受 (網域名稱比 URL 只少擋了 9.8% ~ 17%) 。因此在行動裝置上，即使無法檢查應用層的 URL 路徑，僅查看 DNS 請求應已足夠阻擋廣告流量。我們最後實做一個基於 Android VPN 服務的全系統廣告阻擋機制，能同時兼顧安全性 (不需中間人解密 HTTPS 流量) 與阻擋成效 (只檢查域名就能阻擋 8 成以上的廣告) 。

關鍵詞：廣告阻擋、隱私、安全、DoH、行動裝

* 通訊作者 (Corresponding author.)

Regular Paper

Communications of the CCISA
Vol. 27 No. 1 Feb. 2021

The Risks and Challenges for System-Wide Ad-Block Services

He-Jin Zhang[1*], Chun-Ying Huang[2]
[1 2]Department of Network Engineering College of Computer Science
National Chiao Tung University
National Yang Ming Chiao Tung University
[1] eft823177@gmail.com, [2]chuang@cs.nctu.edu.tw

Abstract

With the rise of digital advertising, it is common to place advertisements on websites and mobile applications to gain profit or obtain user information. To counteract this, a variety of ad-blocking software has emerged on the market. In this paper, we discuss the risks and challenges of implementing a system-wide ad-blocking mechanism on Android. We discuss system-wide ad-blocking solutions' privacy and security issues and point out three significant challenges: domain name encryption, domain-name only blocking effectiveness, and blocking efficiency.

We first discuss the emerging development of DNS over HTTPS (DoH), which encrypts domain names and could significantly impair ad-blocking functions. We then provide a proof-of-concept to counteract this. Second, we analyze the effectiveness of ad-blocking using URL-based or Domain-based blacklist. Our evaluation results show that the difference between the two is not significant (domain-based is less 9.8 ~ 17% lower than URL-based). Therefore, even if it not feasible to check the URL paths at the application layer, it would be sufficient to perform ad-blocking based only on DNS requests. Finally, we implement our proposed approach as a VPN service in Android, which balances the security (no content decryption in the middle) and blocking effectiveness (domain-based blocking with more than 80% accuracy) for system-wide ad-blocking.

Keywords: Ad-Blocking, Privacy, Security, DoH, Mobile Device

壹、前言

現今全球數位廣告蓬勃發展，根據 statista 統計[16]，2019 年全球行動廣告支出總計 1900 億美元，預計到 2022 年將超過 2800 億美元，意味著行動應用程式普遍內建廣告。由於手機螢幕有限，廣告的投放將嚴重影響用戶體驗。資安公司趨勢科技 Trend Micro [17]在 2019 年 1 月總共發現 85 個 Android 應用程式投放大量具有詐欺性的廣告 (全屏廣告、隱藏廣告以及在後台運行的廣告) ，總計至少 900 萬受影響的用戶。某些廣告還具有惡意傾向，包含隱藏代碼使用戶不經意執行惡意程式或是利用廣告蒐集用戶的敏感資訊[1]。因此數位廣告對使用者的影響層面廣泛，包刮使用者體驗、網路效能降低、隱私及安全問題。

為了與之抗衡，市面上興起各式各樣的廣告阻擋軟體。在 2019 年底，超過 7.63 億的用戶正在使用廣告阻擋軟體。在桌機上，只需針對網頁進行廣告過濾。例如 Adblock Plus[18]，在瀏覽器安裝此擴充套件，所有從瀏覽器出去的 HTTP/S 請求都會被 Adblock Plus 攔截。而它不只可以根據請求中的 URL 進行過濾，還能更細粒度的過濾並非來自第三方提供或是 URL 規則沒匹配到的廣告：在網頁加載的同時比對 HTML 元素，並更新 CSS 樣式表讓瀏覽器做渲染 (render) ，以"隱藏"廣告相關的圖像，稱為"元素隱藏" (Element hiding) ，因此並非真的阻擋廣告。

在行動裝置上可透過 VPN 攔截網路流量，若要阻擋所有應用程式中內嵌的廣告，只能檢查 HTTP/S 請求或是 DNS 請求。但根據[2]實驗指出，應用程式內的廣告請求大部分為加密的 HTTPS 流量。此時需利用"中間人"手段，用戶須安裝軟體開發者提供的憑證，讓開發者解開加密的流量，從中獲取 URL 欄位以過濾廣告，例如 AdGuard [19]。但是信任第三方證書將有安全疑慮，有心人士可從中監聽甚至修改流量內容，而使用者卻不自知。另一方面，DNS 請求只針對域名進行廣告過濾，設計上較為簡單也安全，例如 DNS66 [20]、Adaway [21]。

在 2018 年時，IETF RFC8484 規範文件發布 DNS over HTTPS (DoH) [22]，將域名解析安全化的技術。把 DNS 解析請求透過 HTTPS 協定加密連線傳輸，來增強隱私或降低域名劫持等攻擊。Mozilla 基金會[23]在 Firefox 62 版開始支援 DoH 功能，並與 DNS 服務業者 Cloudflare 合作，以 1.1.1.1 實作以支援 DoH 的可信遞迴解析器 (Trusted Recursive Resolver) 。在 2020 年 2 月，Firefox 率先針對美國地區的使用者預設開啟 DoH 服務。另外在 2018 年 10 月到 2019 年年底，TWNIC、Google 以及 Quad9 等知名的 DNS 服務商也逐步支援 DoH。

從 Android 9 Pie 開始[24]，將內建 DNS over TLS (DoT) [25]，與 DoH 目的一致，都是對 DNS 請求進行加密。然而比起 DoT，DoH 隱密性更高。不管未來 Android 在系統端是否改為支援 DoH，或任何應用程式本身提供 DoH 服務，廣告阻擋 APP 將無法依據以明文傳輸的 DNS 請求實現過濾機制。外界反對聲浪也不斷質疑 DoH 的可行性，第一，仰賴過濾機制來進行安全監控的組織或企業都將無法實施；第二，原本 DNS 傳輸

散布在世界各地的伺服器間，受所在地相關法律管轄，且使用者可自由選擇 DNS 伺服器。一旦 DoH 廣泛使用，集中化管理網域名稱解析的任務將使得少數第三方掌握用戶的所有活動，反而帶來安全與隱私的問題[3]。綜合以上所述，我們在接下來的章節裡，會針對行動裝置上的限制以及 DoH/DoT 對廣告阻擋軟體的影響進行研究與分析，並提出合適的做法。

貳、文獻探討

2.1 廣告阻擋 APP 的安全與隱私議題

應用程式內的廣告本身可能使廣告網路 (Ad Networks) 與廣告主 (Advertiser) 之間洩漏用戶的敏感訊息。根據作者 S. Son、D. Kim 和 V. Shmatikov [4]的研究顯示，廣告主可利用 MoPub AdSDK 蒐集用戶的地理位置。MoPub 是當今熱門的行動廣告發布平台之一，它提供 SDK (軟體開發套件) 使得 APP 開發者可以有效地整合到應用程式內。這種專門用於移動設備的託管廣告服務使得三方 (MoPub 伺服器、廣告主、應用程式開發者) 皆能從中獲利。

首先，各個廣告主會將自己的廣告內容 (HTML 格式) 與追蹤 URL (Tracking URL) 一起上傳到 MoPub 伺服器。追蹤 URL 的網域為廣告主的伺服器 (advertiser.com) ，後方填入 MoPub 提供的任何參數，例如 lon/lat (用戶所在經/緯度) 、id (用戶識別碼) 、yob (用戶生日) 與 gender (用戶性別) 等等。有了追蹤 URL，日後一旦廣告被競標成功，並投放到用戶手機的螢幕後，廣告主便可取得當初所想要的用戶資訊。

當用戶手機上的 MoPub AdSDK 執行時，它會依據擁有的存取權限從裝置上蒐集用戶資訊，因此若用戶在不了解的情況下允許 APP 要求的權限，例如 ACCESS_FINE_LOCATION (存取位址) 、GET_ACCOUNTS (手機上儲存的通訊錄與帳號) 、READ_LOGS (讀取日誌) ，將造成隱私外洩。MoPub AdSDK 接著便能發送廣告請求 (夾帶關於用戶資訊的參數) 至 MoPub 伺服器。MoPub 伺服器會把接收到的實際數據填入廣告主的追蹤 URL 中的參數，再將此 URL 作為廣告內容的一部分發送回手機裝置。因此，螢幕上便會展示此廣告。經由收到的廣告，觸發 AdSDK 發送 HTTP (S) 請求 (包含參數值 lat 與 id) 至廣告主，廣告主便順利取得用戶資訊。

然而，部分廣告阻擋 APP 卻也有以上隱私問題，即使它們達到阻擋廣告的功能。根據作者 M. Ikram 及 M. A. Kaafar[5]統計，他們從 Google Play 商店中約 150 萬個應用程式裡，發現有 97 個為廣告阻擋 APP。作者從這些 APP 的原始碼做靜態分析發現 68%的廣告阻擋 APP 本身含有第三方函數庫 (Tracking/Ads Libraries) ，可能會將使用者資訊洩露給第三方；24%的廣告阻擋 APP 本身還展示廣告；89%請求使用者的敏感權限 (用戶的聯絡方式、帳號、簡訊和瀏覽網站的歷史記錄) ；13%在其原始碼中夾帶惡意程式，

例如用間諜軟體監視用戶的行為。

NoMoAds[6]、AdGuard[19]及 Adblock Mobile[26]的阻擋廣告方法為透過本地端 VPN 攔截所有流量。然而，他們為了提高阻擋廣告的準確度，誘使用戶安裝應用程式開發者提供的憑證，以允許解密 HTTPS 流量來過濾廣告相關的 HTTP 請求，但是信任第三方憑證有 MITM (Man In The Middle，中間人攻擊) 的安全疑慮，因為使用者所傳遞的隱私訊息都將掌握在開發者手中，他不只能竊聽訊息，還能修改進行封包偽造。

personalDNSfilter[27]和 DNS66[20]亦與上述三個應用程式有相同的阻擋方法，但這兩者只針對域名進行廣告過濾。然而 personalDNSfilter 還允許使用者開啟 DoH 服務，提取出的域名經過檢查如果不是廣告相關，此網域名稱會被重新建構成 DoH 封包，接著轉發到 APP 開發者預設的 DoH 伺服器，由此伺服器執行 DNS 解析任務。若網域名稱跟廣告相關，則直接丟棄此 DNS 請求封包。

2.2 廣告阻擋 APP 的效能考量

廣告阻擋 APP 因其架構為 VPN 攔截加上規則過濾，會使得速度下降，但須控制在一定的用戶體驗容忍度。根據作者 Chien-Yuan Lan [7]的論文，他實作出一套兼顧彈性及效率的封包過濾器 APP "androidPF"，試圖優化本地端 VPN 處理封包所帶來的負荷。與現有的其他應用程式相比，如 LocalVPN、AntMonitor 以及 NetGuard，androidPF 在不同封包大小和會話數量 (number of sessions) 的頻寬測試，比他們三個 APP 還要快 1.5 到 2 倍。而與沒開 VPN 的情形下相比，雖然在單一封包越小的時候，速度落差越顯著，但單一封包越大時速度將明顯回升；在會話數量的測試上，20 個 session 同時連線時，速度平均只下降了約 20%。3.4 章節將會說明我們把廣告阻擋功能整合至 androidPF 上。

另一方面，阻擋規則的數量亦會影響速度，作者 P. Snyder、A. Vastel 等人[8]針對目前最知名、基於 URL 的黑名單列表"EasyList"[28]，其規則數量在近十年的歷史中，從數百條規則發展至快七萬條。作者將 EasyList 應用於 10,000 個網站上 (Alexa 上排名前五千的網站和從排名第 5001 到 1 百萬名的網站裡隨機挑選五千個) ，反覆測試多次後實驗結果為大約有 90.16%的規則都沒有被使用到；4.45%的規則用過 1 到 100 次；3.56%的規則用過 100 到 1000 次；1.83%的規則用超過 1000 次。意味著，EasyList 列表所有規則七萬筆裡，只有大約 10% (七千筆) 實際使用到。對於桌機版的廣告阻擋軟體，其效能影響不會差距太多，但在行動裝置平台上，受限於有限的硬體資源，規則越多越複雜將使阻擋效率嚴重下降[9]。為此許多論文[6][9][10][11]利用機器學習或自動化修補過濾列表來增進阻擋效率，然而如 2.1 小節所述，基於安全性，不建議在行動裝置上安裝第三方憑證以解密 HTTPS，而是只針對域名進行規則匹配，如此能大幅簡化規則的複雜度，並且將於第 4 章節證明只檢查域名不會比基於 URL 列表阻擋的結果差太多 (平均只差 9.8 ~ 17%) 。

2.3 探討 DoH 可能的缺失

2018 年出現 DoH 技術，將 DNS 請求隱藏在 HTTPS 協議裡，而客戶端應用程式本身需實作 DoH 功能。因其使用 443 埠號，因此與一般的 HTTPS 流量沒有區別，目的就是為了增強隱私，有更安全的用戶瀏覽體驗。許多相關研究者開始著手探討它的效能以及各方面的影響。在論文[12][13]以及 Firefox 團隊的研究[29]發現，其實驗數據表明了在 DNS 請求查詢和網頁加載的速度均沒有顯著的下降，在用戶體驗上影響不大。但隨著網路環境品質下降，速度差距會擴大，與傳統未加密的 DNS 相比，DoH 的性能會變差。

經由 2.1 節的討論，目前廣告阻擋軟體仍有一些缺失，以及可能面臨的挑戰 DoH。我們並不建議廣告阻擋軟體，例如 personalDNSfilter [27]或 Pi-hole [30]開始支援 DoH。根據[3][14][31]，可從四個面向探討 DoH 的缺失：

1. DoH 並不能完全阻止 ISP 業者追蹤用戶的網路瀏覽。因為 DNS 不是瀏覽網頁中涉及的唯一協議。例如用戶造訪 HTTP 加載的網站，則 DoH 沒有意義；若是造訪 HTTPS 的網站而且其 SNI 欄位或 OCSP 連線沒有加密，就能從這兩個資訊得知用戶造訪的網站；儘管所有相關流量都經過加密，ISP 或其他第三方有 95%準確度識別 IP 位址所對應到的網站[15]。
2. DoH 會影響企業政策，允許員工存取被公司禁止的網站。
3. DoH 削弱網路安全。國際網路安全培訓組織 SANS 和荷蘭國家網路安全中心 NCSC.NL 都公告攻擊者將可繞過安全監控。在 2019 年 7 月，出現第一個利用 DoH 來隱藏 DNS 流量的後門程式，Linux DDoS 機器人"Godlua" [32]，攻擊者用 Confluence 軟體的漏洞 (CVE-2019-3396) 感染 Linux 伺服器，再利用這些受害伺服器發動分散式阻斷服務 (DDoS) 攻擊。此外，阻止用戶存取關於政治的敏感性內容、恐怖主義內容、成人網站或侵入式廣告都讓政府或組織無法用黑名單過濾。
4. DoH 對整個 DNS 伺服器的分散式生態系統產生影響。目前 DoH 傾向於集中式管理，少數幾家 DoH 解析器掌握用戶的瀏覽紀錄，產生新的隱私風險。並且，可能導致攻擊者將重心放在少數伺服器上，進行單點攻擊，一旦大型的 DNS 服務平台失去運作，將產生大規模的網路中斷。

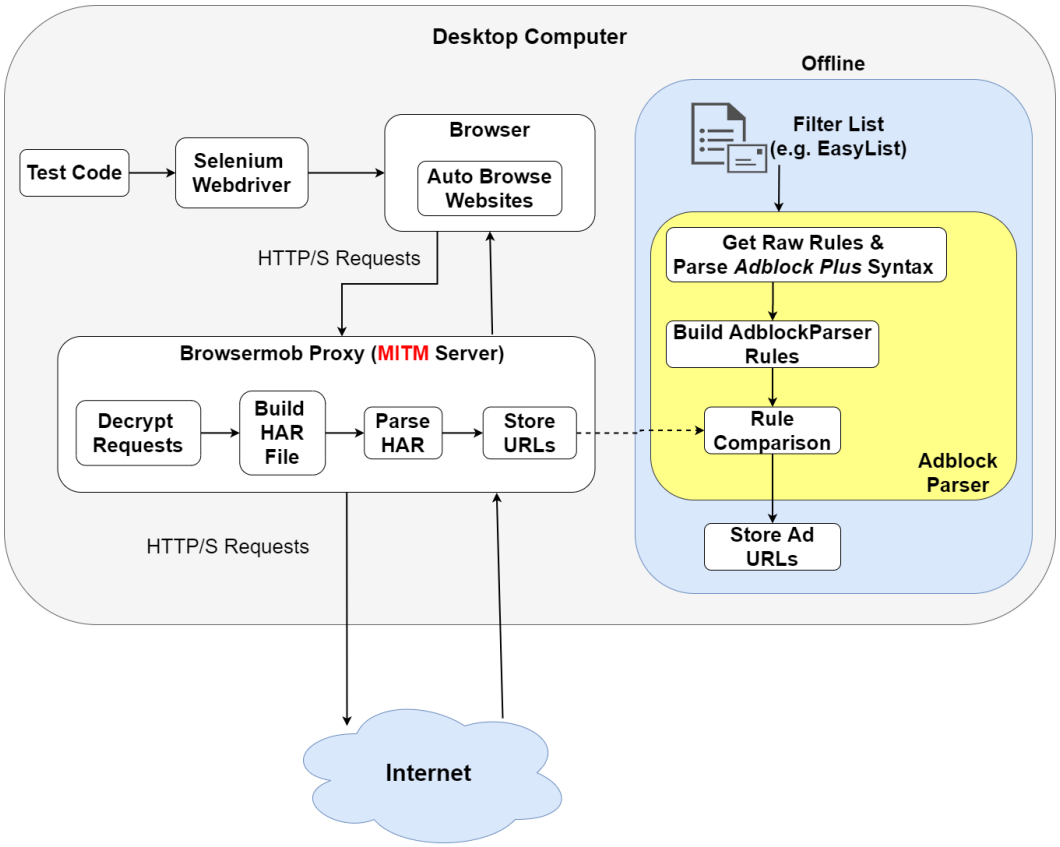
參、方法

數位廣告的投放大都來自三個元素組成：圖像、HTML 和 Javascript 程式碼，這三者都須用 HTTP 請求以獲取資源，因此 HTTP 請求裡的 URL 資訊對於廣告阻擋軟體是相當重要的依據。然而廣告流量的特徵不斷在變化，過濾廣告的黑名單也必須定期更新

規則。URL 因為有完整路徑，所以比起只看 Domain，更能細粒度的阻擋廣告並適時調整規則。例如聯合新聞網網站 https://**udn.com**/news/breaknews/1，透過 Chrome 的開發人員工具，從網路流量發現其中一筆與廣告相關的 HTTP 請求網址 https://udn.com/static/js/**google-dfp-**native.js?...，它被知名的瀏覽器擴充套件 Adblock Plus 判定為廣告，因它與阻擋規則**/google-dfp-**匹配。其中 dfp 全名為 DoubleClick for Publishers，是 Google 提供的託管廣告服務平台。此廣告請求網址必須透過 URL 的完整路徑才能過濾，無法倚靠 Domain，否則整個網站都無法存取。因此，我們接著會在桌機上與 Android 平台分別擷取瀏覽網頁的 HTTP 流量，並使用基於 URL 和基於 Domain 的黑名單列表來統計各個黑名單所阻擋的內容與數量，以便後續進行分析、比較其過濾結果相差多少。

3.1 在桌機上的流程架構

圖一為在桌機上自動化阻擋廣告的流程，透過 Selenium[33]進行網頁自動化測試，模擬用戶操作瀏覽器，並為瀏覽器設置代理伺服器 Browsermob Proxy[34]，讓它連上大約兩萬個網站。Browsermob Proxy，簡稱 BMP，是一個 HTTP 代理服務，與 Selenium 有著完美的搭配。BMP 要查看加密的 HTTPS 流量必須在"創建 Proxy"的函數內設置參數 "trustAllServers"為 true，讓 BMP 做為中間人，以攔截所有 HTTP 請求與回應，接著匯出成 HAR 檔案。HAR (HTTP Archive format) 是 JSON 格式的檔案，多用於記錄網頁瀏覽器與網站的互動資訊，例如網頁加載時間、所有 HTTP 請求與回應、HTTP 版本等等。在本次實驗中，我們只需要 HTTP 請求裡的 URL，因此每爬取一個網站，就把它的 HAR 裡所有 URL 都取出，一一傳給 Adblock Parser[35]判斷是否為廣告請求網址。

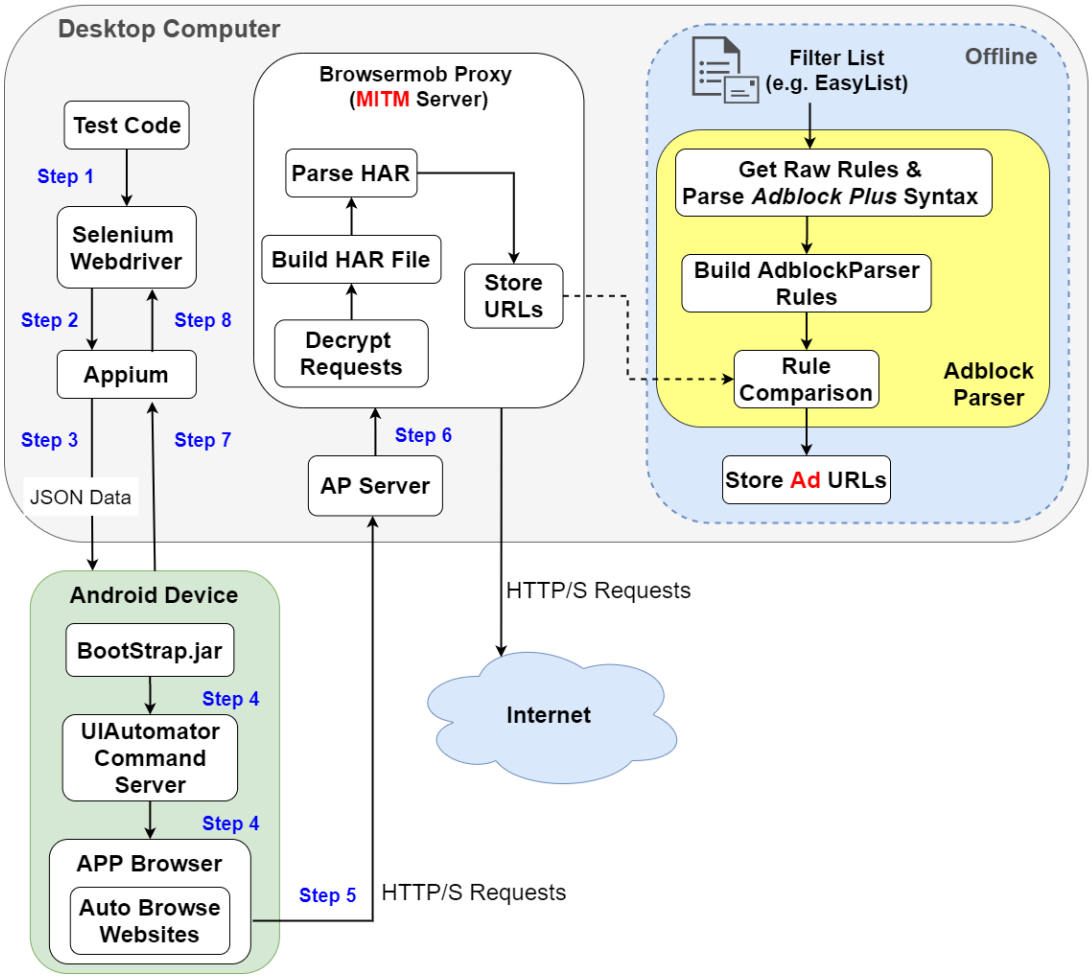


圖一：在桌機上自動化瀏覽網頁並過濾廣告請求

Adblock Parser 是一個能解析 Adblock Plus 語法並進行規則比對的第三方 Python 函數庫，Adblock Plus 是先前提到的瀏覽器擴充套件，此廣告阻擋軟體使用的黑名單列表為 EasyList[28]，其規則例如：**&ad_sub=** 或 **/banners/ads-**（URL 中的參數）、**||kickass.best^$script**（帶有條件的格式，表示要阻擋某 HTTP 請求須符合兩個條件，一是此網域為 kickass.best，二是將存取的資源為 JavaScript 檔案）。許多知名廣告阻擋軟體都參考此語法加以修改、擴充。我們從 23,294 個網站蒐集所有 HTTP 請求（共 2,907,173 個），並將它保存下來，分別餵給不同來源的黑名單列表，透過 Adblock Parser 輸出各個阻擋結果。第 4 章節將說明所使用的黑名單列表來源以及分析其過濾結果來評估廣告阻擋的成效。

3.2 在 Android 上的流程架構

圖二為在 Android 裝置上進行自動化瀏覽網頁並過濾流量的實作流程圖。首先，要能自動控制手機的任何行為（如同使用者手動開啟 APP 並執行一系列的操作），可透過 Appium[36]，它是一個開源的自動化測試工具，對原生 APP 的測試不需要包含其它 SDK 套件或重新編譯 APP，就能透過指令對 APP 進行操作。而它實際上是一個基於 node.js 的 web 伺服器，與前面章節提到的 Selenium WebDriver 進行整合。



圖二：在 Android 上自動化瀏覽網頁並過濾廣告請求

如圖所示，步驟為：

1. 在主機上編寫測試腳本 (支援多種語言，如 Java、JavaScript、Python…等等) 。
2. 執行腳本後，WebDriver 發送 HTTP 請求到 Appium，請求內容在本實驗為：開啟 Chrome 瀏覽器 APP 並輸入網址進行上網。
3. Appium 解析此請求內容，並調用對應的框架 (此處為 Android 平台) ，將解析後的 JSON 格式數據透過 USB 線傳到手機。
4. 透過 Appium 進行初始化時，預先安裝在手機裡的 BootStrap.jar 來接收此數據，並翻譯成 Android 平台能讀懂的語法以執行手機內建的 UI 指令。
5. 由於手機所連的 WiFi 是我們主機上的 AP 伺服器，因此所有手機端的網路流量都會傳到我們的主機。
6. 如同 3.1 章節在桌機平台的實驗一樣，把自動瀏覽網頁所產生的 HTTP/S 請求都導到 Browsermob Proxy，以便進行中間人竊聽，查看其明文內容，取出請求

裡的 URL。但要能解密 HTTPS 必須在手機上安裝 Browsermob Proxy 的 CA 證書。

7. 執行的結果由 BootStrap.jar 返回給 Appium，Appium 也會把結果返回到 WebDriver，告知指令是否執行成功，若成功才會繼續執行下一個指令 (瀏覽下一個網頁) 。

我們所瀏覽的網站共 23,294 個與桌機平台上的測試相同，而在 Android 平台上共蒐集到 2,297,715 個 HTTP 請求。4.1 章節會針對這兩個平台所收集的流量並經過過濾後，進行初步分析、比較是否有無不同之處。

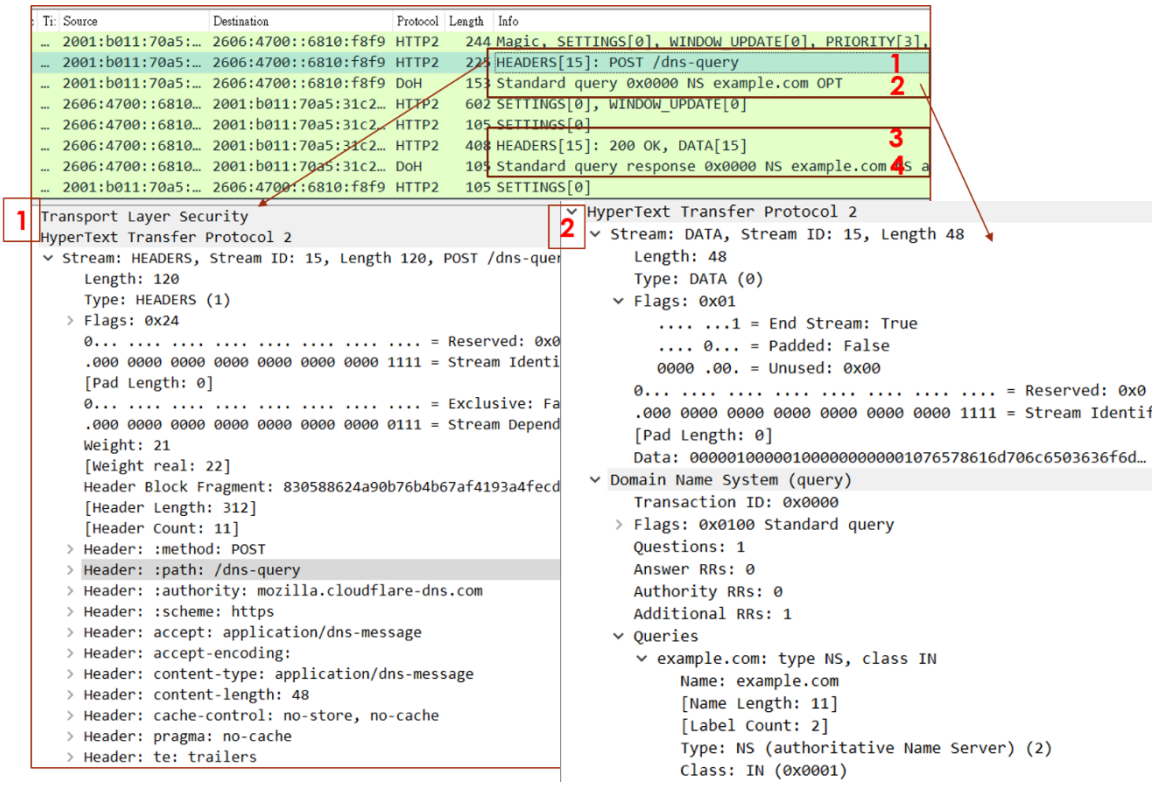
3.3 分析 DoH 封包與 Fallback 機制

我們選擇 Firefox，因為它是最早投入並支援 DoH 的應用程式。從瀏覽器上的選單列表內點開"選項"，接著找到"網路設定"，勾選開啟 DNS over HTTPS，可使用預設或自行選擇 DoH 供應商。在此實驗中我們選擇預設的 DoH 供應商"Cloudflare"，它是 Firefox 的第一個合作夥伴，於 2018 年七月便開始進行 DoH 的性能測試。

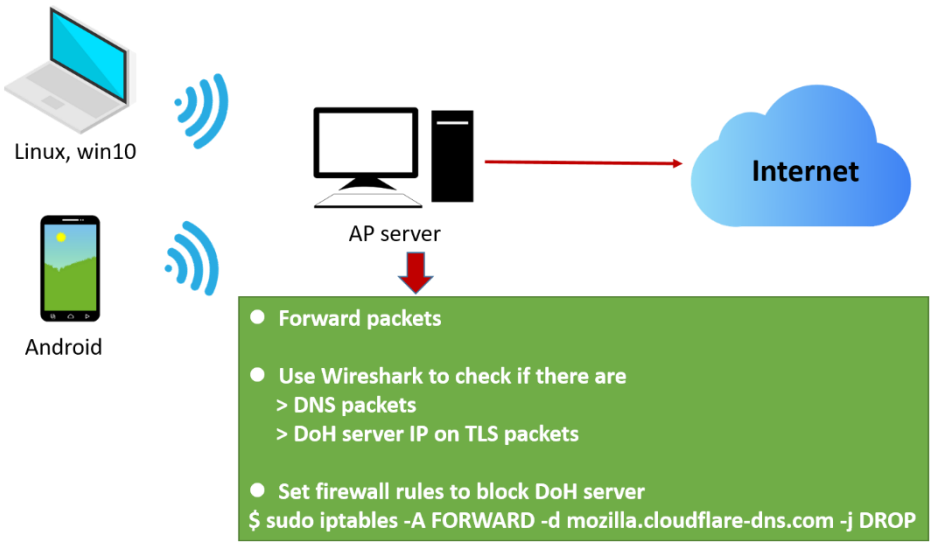
為了查看 DoH 封包的明文內容，必須解密 HTTPS 流量。很幸運的，Wireshark 在 3.1 版本之後完整地支援解密 DoH。圖三為 TLS 封包被解開後，可看到有 HTTP2 協議和 DoH 協議的封包。根據 RFC8484 文件[22]，HTTP2 是與 DoH 一起使用的 HTTP 協議最低推薦版本。從圖中用紅色方框框起來的部分明顯可見，一個為 DNS 請求，另一個為對應的 DNS 回應。

圖中的標號 1 " HEADERS[15]: POST /dns-query "是 HTTP2 請求的 HEADERS frame，圖的左下方展開其詳細資訊。因為 HTTP2 用 HPACK 壓縮 Header，因此解壓縮後可看到多個 Header 欄位，從中可以發現目的端為 DoH 解析器 Cloudflare (mozilla.cloudflare-dns.com) ，其 IP 為 104.16.248.249。並且 content-type 為 application/dns-message，表示要用 DNS 格式。而每個 frame 都有 Stream ID (此例為 15) ，同一組的請求與回應封包，其 Stream ID 相同，以在同一條 TCP 連線中區別每一組 HTTP2 請求/回應；標號 2 是 DoH 請求的實際內容，圖的右下方展開其詳細資訊，裡面有 DATA frame 以及標準的 DNS 請求格式。含有 DATA frame 主要目的是告知其 frame 型別、Stream ID 以及 End Stream flag (是否為此次請求的最後一個封包) ；標號 3 與 4 是回應封包，與請求封包有類似的格式，在此不再詳細解釋。

在使用 Wireshark 分析 DoH 封包時，我們還發現 Firefox 在進行 DoH 請求之前，會先用傳統 DNS 請求查詢用戶選定的 DoH 解析器的 IP，此作法在 RFC8484 文件中亦有說明。此 DNS 請求封包的待查詢域名是 mozilla.cloudflare-dns.com，回應的 IP 為 104.16.248.249 與 104.16.249.249。我們用防火牆規則阻擋這兩個 IP，並持續瀏覽其他網站，從 Wireshark 發現再也沒有 DoH 封包，全部 Fallback 回傳統 DNS 封包。



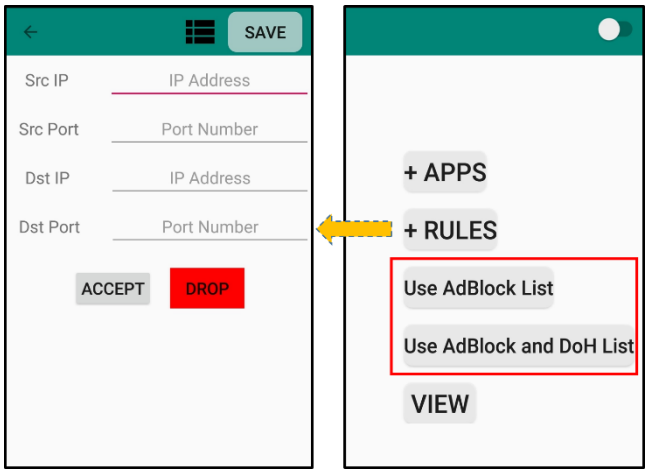
圖三：HTTP2 以及 DoH 封包的詳細資訊



圖四：DoH Fallback 測試環境

圖四為 Fallback 測試環境，架一台 AP 伺服器讓任何平台 (Win10、Linux、Android) 的用戶端皆能透過此伺服器連網，並在伺服器上開啟 Wireshark 和防火牆功能。Wireshark 用來觀察是否有 DoH Fallback 行為；防火牆則是透過 iptables 指令來丟棄目的端為 DoH 解析器的封包。因此，我們只需盡可能蒐集 DoH 解析器的域名，把它新增到基於 Domain

的黑名單列表，提供給廣告阻擋軟體來繞過 DoH 機制。如果有應用程式提供用戶直接設定 DoH 解析器的 IP，而非設定其網域名稱，那麼就必須改以阻擋此 IP。在 4.4 章節將會針對不同平台 (Win10、Linux、Android) 以及目前已知且有支援 DoH 功能的軟體來驗證是否都有 Fallback 機制。



圖五：androidPF 介面：IP/Port 規則設定以及增加廣告阻擋功能

3.4 整合廣告阻擋功能至 androidPF APP

在 2.2 小節提到 androidPF 是一個封包過濾器 APP，但它只解析 IP 以及 Port，並沒有往上層 (應用層) 去解析 DNS 封包，因此我們針對它的原始碼進行修改，以便阻擋廣告以及 DoH 相關的封包，架構為透過 local VPN 在本地端 (即裝置上) 過濾封包。圖五的左方是作者提供 IP 與 Port 的規則設定，而右方框線中的兩個選項是我們新增的功能，其差別為是否要阻擋 DoH (其亦可阻擋 DoT) ，此設計為方便後續的測試。點選其中之一皆會從我們架設的 Web 伺服器上下載基於 Domain 的阻擋列表。當攔截到埠號 53 的封包時，進行解析並提取請求裡的 Domain，若於列表中匹配成功，其目的端 IP 會被我們重導向至 127.0.0.1，以阻擋此 DNS 封包。

與現有基於 Domain 的廣告阻擋軟體相比，如先前提到的 DNS66、personalDNSfilter 或 Adaway，它們一樣是 local VPN 的機制，但只看 DNS 封包，其餘封包皆不會路由至 VPN，雖然此種作法效率較佳，但在某些情況下將無法阻擋廣告，例如：含有 DoH 功能的應用程式 (如 Firefox) 能直接指定 DoH 解析器的 IP。因已知要使用的 DNS IP 為何，基於 Domain 的廣告阻擋軟體無法阻擋 DoH/DoT 解析器，因此也無法阻擋廣告。反觀 androidPF，所有封包皆路由至 VPN，雖然要處理的流量較大，但作者在查找與儲存每個連線資料的實作上，提出了有效率的方法，得以兼顧效能。透過 androidPF，我們能阻擋 DoH IP+Port 443 或是阻擋 DoT Port 853。

肆、評估與討論

4.1 在桌機與 Android 上的廣告阻擋結果

首先，要能過濾出廣告相關的 URL/Domain，需有一定程度可信的來源列表，因此，我們先收集知名的廣告阻擋軟體，從中查看它們所使用的列表為何，最後整理出的黑名單列表為 3 個基於 URL 和 6 個基於 Domain。表一是每個阻擋列表分別在桌機與 Android 平台上過濾廣告的結果。表格最上方是我們從 Alexa 上取得 2020 年排名前 5,000 名的熱門網站 (Domain) ，當作進入點，分別針對每個網站進行網頁爬取，總共取得 23,294 個內部網址。在 3.1 與 3.2 章節已說明如何自動化瀏覽這 23,294 個網頁，並保存所有產生的 HTTP 請求裡的 URL。表格第三列為 URL 總數 (例如在桌機上共有 2,907,173 個) 和去除重複後的 URL 數量 (1,436,784 個) 。接著把這些"不重複"的 URL (桌機上共蒐集 1,436,784 個；Android 上共蒐集 1,430,960 個) 當作輸入，分別餵給 11 個阻擋列表。Domain_Combine_All 和 URL_Combine_All 分別是把所有基於 Domain/URL 阻擋列表合併起來。由表中數據顯示，在所有 HTTP 請求裡大約有 25%～45%被視為廣告相關的請求。另一方面，某些基於 Domain 的黑名單列表比基於 URL 的黑名單列表阻擋的數量超出許多，如 Domain_StevenBlack、Domain_Combine_All。在接下來的章節裡會說明它們阻擋了哪些內容並檢查是否有誤擋的情形。

表一: 每個阻擋列表分別在桌機與 Android 平台上過濾廣告的結果

Entry points (domains) ：5,000		
已瀏覽的網站總數：23,294		
HTTP 請求/不重複的 URL 總數:	2,907,173 / 1,436,784	2,297,715 / 1,430,960
過濾列表	**(Desktop) 阻擋數量**	**(Android) 阻擋數量**
Domain_AdGuard_DNS_Filter	486822 domains (33.9%)	503818 domains (35.2%)
Domain_StevenBlack	567095 domains (39.5%)	552717 domains (38.6%)
Domain_AdAway	471393 domains (32.8%)	448161 domains (31.3%)
Domain_Someonewhocares	331224 domains (23.1%)	295443 domains (20.6%)
Domain_Peter_Lowe_List	445566 domains (31.0%)	431378 domains (30.1%)
Domain_EasylistChina+Easylist	389533 domains (27.1%)	365546 domains (25.5%)

Domain_Combine_All	655572 (45.6%) domains	643614 (45.0%) domains
EasylistChina+Easylist	365489 (25.4%) urls	357810 (25.0%) urls
AdGuard_Base_Filter_Opt.	356570 (24.8%) urls	347990 (24.3%) urls
Easylist	338145 (23.5%) urls	331239 (23.1%) urls
URL_Combine_All	384624 (26.8%) urls	375644 (26.3%) urls

我們在桌機與 Android 上分別查看數量排名前 25 的網域 (從 Domain_Combine_All 列表阻擋的結果來進行分析) 。針對這些網域我們人工檢查以確認其來源為何，而無論如何分類皆能把它們歸在兩大項目：廣告相關與追蹤相關。接著我們在兩個不同平台間進行交叉比對，得到只出現在桌機的網域共有 586 種和只出現在 Android 平台的共有 696 種。然而，經過檢查發現絕大多數的網域都不特定屬於某一平台，不能保證是否只服務於桌機上或只服務於行動裝置上。有可能只是跑實驗的過程中，某網站在當時剛好有些服務沒順利取得或是投放的廣告不同。

經由上述討論，由瀏覽器發出的廣告請求在這兩個平台並無太大區別，並且觸發到的廣告網路可能不夠全面。於是，我們轉往針對行動裝置上 APP 裡的內嵌廣告，有別於網頁上的廣告是把廣告版面格式、廣告請求的程式碼 (HTML 與 CSS) 安插在網頁裡，APP 內嵌廣告則是透過 Ads SDK，整合至應用程式裡，以呼叫 Java 函數的方式請求並加載廣告。

我們從 AppBrain 網站[37]蒐集前 20 名、提供 Ads SDK 的行動廣告網路 (Mobile Ad Network) ，例如 Google 的 AdMob，APP 開發者若把它的 Ads SDK 整合於程式裡，在使用者操作 APP 時，將會在特定介面觸發廣告請求事件，而此 HTTP 請求的目的端即為行動廣告網路*.admob.com，因此，只要阻擋這些網域就能阻擋廣告。經檢查，大多數的行動廣告網域有出現在我們蒐集的基於 Domain 阻擋列表裡，並且也檢查出大部分的行動廣告網路 (70%) 並"沒有"或"無法"於瀏覽網頁時觸發到。間接證明這些阻擋列表能有效阻擋所有應用程式內嵌的廣告。

4.2 URL 阻擋列表過濾的結果餵給 Domain 阻擋列表

表二是把在瀏覽網頁上，基於 URL 阻擋列表過濾的結果餵給基於 Domain 阻擋列表，以評估其差距。表格為 Android 平台上的數據，因桌機上的結果非常近似於此，故不再列出桌機版的結果。第一列顯示 3 個基於 URL 阻擋列表所過濾出的廣告網址數量，而此數據來自於表一，將這些網址作為輸入餵給以下 8 個基於 Domain 的阻擋列表。可看出 Domain_StevenBlack、Domain_EasylistChina+Easylist 和其他合併列表 (廣告阻擋 APP 其預設的過濾列表) 有 8 成到 9 成的比率接近 URL 阻擋列表的結果。證明基於 Domain 過濾列表雖然無法像基於 URL 過濾列表細粒度地檢查網址完整的參數，但仍有

良好的阻擋成效。

接下來，我們查看 Domain 阻擋列表遺漏的廣告網域為何。在 URL_Combine_All 列表阻擋的結果裡 (375,644 個) ，共有 35,735 個網址是 Domain_Combine_All 列表所遺漏的。取出其網域則有 891 種。此數據為 Android 平台上的結果，與桌機上的非常近似，故不再列出桌機版的結果。前 2 名遺漏的網域為 www.google.com 和 www.google.com.tw，廣告網址範例為：https://www.google.com.tw/**ads**/ga-audiences?v=1...，無法以網域直接阻擋。雖然基於 Domain 阻擋列表無法擋這兩者，但它們佔整體數量 (375,644) 不高 (分別為 3.87%與 3.68%) ，並且其它網域 (第 3～10 名) 經查證後都與廣告相關，加入黑名單內不會有問題。因人工查看，只檢查前 10 個網域來源，但第 10 名以後的出現率非常低 (0.03%以下) ，故影響不大。

表二: 基於 URL 阻擋列表過濾的結果餵給基於 Domain 阻擋列表

URL 阻擋列表過濾的廣告網址總數：		357,810	347,990	331,239
廣告阻擋 APP	**Domain/URL 阻擋列表**	**URL_EasylistChina+Easylist**	**URL_AdGuard_Base…**	**URL_Easylist**
AdGuard, AdClear	(Domain) EasylistChina + Easylist + AdGuard DNS filter (#40962)	308489 (**86.2%**)	312853 (**89.9%**)	296304 (**89.5%**)
personalDNSfilter	AdAway + Peter Lowe (#15284)	296205 (**82.8%**)	296046 (**85.1%**)	289061 (**87.3%**)
AdAway	AdAway + Peter Lowe + Steven Blacks (#57498)	305931 (**85.5%**)	306285 (**88.0%**)	298725 (**90.2%**)
PiHole, DNS66	Steven Blacks (#57463)	305917 (**85.5%**)	306271 (**88.0%**)	298711 (**90.2%**)
Blokada	AdGuard DNS filter (#36646)	275717 (77.1%)	283576 (81.5%)	266357 (80.4%)
-	(Domain) EasylistChina + Easylist (#24595)	301180 (**84.2%**)	298677 (**85.8%**)	289086 (**87.3%**)
-	AdAway (#12066)	263635 (73.7%)	260999 (75.0%)	256503 (77.4%)
-	Peter Lowe (#3489)	265589 (74.2%)	271649 (78.1%)	264824 (79.9%)

4.3 Domain 阻擋列表過濾的結果餵給 URL 阻擋列表

與 4.2 節的實驗相反，我們改為將基於 Domain 阻擋列表過濾的結果餵給基於 URL

阻擋列表，一樣只列出 Android 平台上的數據 (與桌機版的結果很相近) ，如表三。第一列顯示 7 個基於 Domain 阻擋列表所過濾出的廣告網域數量，而此數據來自於表一，將這些網域原本帶有的完整網址作為輸入餵給以下 4 個基於 URL 的阻擋列表。我們發現大多數的比率都很低 (除了 Domain_EasylistChina+Easylist 那一欄有 80%以上) ，因為此列表與基於 URL 阻擋列表有著相同的來源 (Eyeo GmbH 公司) 。URL_AdGuard_Base_Filter_Opt.雖然來源為 AdGuard 公司本身，但它是拿 URL_Easylist 加以修改、擴充。

表三: 基於 Domain 阻擋列表過濾的結果餵給基於 URL 阻擋列表

Domain 阻擋列表過濾的網域總數	503,818	552,717	448,161	295,443	431,378	365,546	643,614
URL/Domain 阻擋列表	**AdGuard**	**StevenBlack**	**AdAway**	**Someone whocares**	**Peter Lowe**	**EasylistChina + Easylist**	**Combine_ All**
URL_Easylist China+Easylist	275392 (54.7%)	305629 (55.3%)	258272 (57.6%)	209082 (70.8%)	265430 (61.5%)	300575 (**82.2%**)	321674 (50.0%)
URL_AdGuard_ Base_Filter_Opt.	283216 (56.2%)	305977 (55.4%)	255637 (57.0%)	210892 (71.4%)	271483 (62.9%)	293981 (**80.4%**)	326117 (50.7%)
URL_Easylist	265970 (52.8%)	298424 (54.0%)	251141 (56.0%)	204480 (69.2%)	264665 (61.4%)	288500 (78.9%)	309499 (48.1%)
URL_Combine_All	292837 (58.1%)	314249 (56.9%)	263313 (58.8%)	215692 (73.0%)	272847 (63.3%)	306884 (**84.0%**)	339251 (52.7%)

其餘比率較低的原因有兩個：基於 Domain 阻擋列表必須一蓋把黑名單網域全部擋掉，無論它是否為廣告請求，但基於 URL 阻擋列表會先從網址參數檢查以決定是否要阻擋;另一原因為基於URL阻擋列表只聚焦在阻擋跟廣告相關的請求,然而基於Domain阻擋列表不只如此，根據它們列表上的說明，除了過濾廣告相關，還會過濾追蹤 (用戶瀏覽紀錄、個人資訊) 、惡意網域、社交媒體按鈕...等等。

接下來，我們查看 URL 阻擋列表遺漏的廣告網域為何。在 Domain_Combine_All 列表阻擋的結果裡 (643,614 個) ，共有 304,363 個網址是 URL_Combine_All 列表所遺漏的。檢查前 10 名網域發現這些網域幾乎都與追蹤相關，而追蹤類別可分成兩大項：一為追蹤廣告曝光率、點擊率和廣告競價資訊；二為追蹤網站流量、停留時間、跳出率和用戶瀏覽行為等等。因基於 URL 阻擋列表只擋第一類與廣告較相關的網址，所以大多

數網址被它們放行，少部分的則阻擋。

4.4 DoH Fallback 實驗結果

目前 DoH 功能主要實作在瀏覽器裡，但微軟宣布未來將在 Win10 上支援 DoH 協議，以在系統端為所有應用程式都透過 HTTPS 加密傳輸 DNS 流量[38]。另外在 Android 平台，目前 Android 9 Pie 版本以上已在系統端內建 DoT 功能，尚無支援 DoH[24]。使用者可在網路進階選項裡開啟"Private DNS"模式並設定 DoT 解析器的網域名稱，但 Private DNS 模式不具有 Fallback 機制，需改成"Automatic"模式，然而此模式無法自行選擇解析器，它只會根據當前電信供應商 ISP 所指定的 DNS 伺服器是否支援 DoT，若無，就會使用傳統 DNS 明文傳輸。

用 3.3 章節與圖四提到的 Fallback 測試流程，實驗結果如表四所示，我們測了在桌機和 Android 版本的瀏覽器以及 curl 指令。第二到第四欄用斜線區隔桌機與 Android 上的應用程式版本和 Fallback 結果，"yes"表示 Fallback 成功；"no"表示無提供 Fallback 機制；"NA"表示無法被評估，因為無提供 DoH 服務。被標示為"NA"的原因是雖在應用程式上有開啟 DoH 功能的按鈕，但卻因 Android 的限制無法設定 DoH 伺服器的網域或 IP，需靠第三方工具"DNS Changer"，但用戶不僅要額外裝一個第三方應用程式還會有 DNS 劫持的風險，因此不適用於評估能否 Fallback。

表四: DoH Fallback 在不同應用程式上的實驗結果

用戶端應用程式	Desktop / Android	版本	Fallback ?	備註
Firefox	Desktop / Android	74.0 / 68.6.0	yes/yes	設定 DoH : “about:preferences” 或 “about:config”
Chrome	Desktop / Android	86.0.4240.183 / 86.0.4240.185	yes/yes	Desktop：需在系統端的 TCP/IPv4 欄位修改 DNS IP
Brave	Desktop / Android	1.8.56 / 1.5.5	yes/NA	Desktop：需在系統端的 TCP/IPv4 欄位修改 DNS IP
Opera	Desktop / Android	67.0.3575.115 / 57.1.2830.52480	yes/NA	Desktop：需在系統端的 TCP/IPv4 欄位修改 DNS IP
Vivaldi	Desktop / Android	2.11.1811.49 / 2.11.1813.18	yes/NA	Desktop：需在系統端的 TCP/IPv4 欄位修改 DNS IP
Edge	Desktop / Android	80.0.361.69 / 77.0.3865.116	yes/NA	Desktop：需在系統端的 TCP/IPv4 欄位修改 DNS IP
Bromite	Android	81.0.4044.76	no	-
curl	Desktop	7.62+	no	$ curl -dohurl https://cloudflaredns.

				com/dnsquery https://example.com/

由本章節以及先前說明，從 2018 年年底到現今，DoH 還在實驗階段，尚有許多因素待考量。在未來，如果應用程式或系統端有設置 DoH 解析器的網址，我們可以更新基於 Domain 的黑名單過濾列表，而且為了穩定性，DoH 通常有 Fallback 機制，使廣告阻擋軟體能利用此繞過 DoH 並過濾廣告；如果是直接使用 DoH 解析器的 IP，則根據埠號 443 和 IP 阻擋。

Source	Destination	Protoco	Le	Info
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x1873 A config.uca.cloud.unity3d.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x41f6 A launches.appsflyer.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x1a0f A ms.applovin.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x10c3 AAAA d.applovin.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x6518 AAAA rt.applovin.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x2264 AAAA mads.amazon-adsystem.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x3b59 A reports.crashlytics.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x3e04 AAAA cdn2.inner-active.mobi
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x482c A i16-tb.isnssdk.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x3b58 AAAA adc3-launch.adcolony.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x24de AAAA config.unityads.unity3d.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x025e AAAA init.supersonicads.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x7b1a AAAA ads.api.vungle.com
10.1.10.1	127.0.0.1	DNS	…	Standard query 0x6bb8 AAAA googleads.g.doubleclick.net

圖六：阻擋遊戲 APP 裡內嵌廣告的 DNS 封包截圖

4.5 實測廣告阻擋與 DoH/DoT Fallback

在 3.4 章節已說明如何把廣告阻擋功能整合至 androidPF，因此，本章節將進行兩階段的測試。測試用的手機型號為 HTC U Play，Android 6.0。在第一階段，單純測試廣告阻擋功能 (無 DoH 環境) 在不同類型的 APP 上。如圖五所示，透過點選"Use AdBlock List"，會從我們的 Web 伺服器下載過濾列表，所使用的列表為 Domain_AdAway + Domain_Peter_Lowe_List。此合併列表的規則數量較少 (15,284 個)，並且沒有遺漏前 20 名行動廣告網域，以及在表二實驗中與基於 URL 列表相比只差 12 ~ 17%。我們手動測試瀏覽器 Chrome、遊戲 Jewel Hunter Lost Temple 和記事本工具 CUMO Note，透過前後開啟 androidPF 的 VPN 來對比阻擋廣告的效果，皆有成功阻擋並仍能正常使用 APP。圖六是阻擋遊戲 APP 裡內嵌廣告的 DNS 封包截圖範例，所有有關廣告的 DNS 請求，其目的端都被導到 127.0.0.1。

第二階段為廣告阻擋在 DoH/DoT 的環境下測試：我們在 Firefox 和 Chrome 分別設

置不同解析器：Cloudflare (https://mozilla.cloudflare-dns.com/dns-query) 以及 CleanBrowsing (https://doh.cleanbrowsing.org/doh/family-filter/) 。透過 Wireshark 確認，能成功阻擋解析器的網域，因此之後也能阻擋明文的 DNS 廣告請求並正常上網。

"DoT" Fallback 測試則是直接用系統端的 DoT 設置，但因實驗用手機是 Android 6.0，必須 Android 9 以上才有支援，因此我們改用模擬器 (API 30, Android 11.0) 測試。利用 VPN 的特性，它使用函數 addDnsServer (8.8.8.8) 來修改裝置中的 DNS 設置，故 DoT 的 "Automatic"模式會使用當前這個 DNS IP (4.4 章節有說明) 。因 8.8.8.8 支援 DoT，基於網域名稱的阻擋列表已不管用，所以需透過 androidPF 的功能 (圖五) 設定規則以阻擋 Port 853，測試的遊戲 APP 便能順利阻擋廣告。

伍、結論

本研究探討目前全系統廣告阻擋軟體在不同面向的考量，包括隱私、安全和效能。因其實作環境上的限制，透過 VPN 攔截並查看明文內容的封包，故只能檢查 DNS 請求，無法查看 HTTPS 裡的 URL 欄位。基於安全因素，我們並不建議利用中間人手段 (MITM) 去解密 HTTPS，來達到更準確的阻擋成效。透過我們設計的一套自動化瀏覽網頁以生成大量的 HTTP/S 請求，並經由不同來源的列表過濾出廣告相關的網址和網域。在進行交叉比對與分析後，我們發現基於網域名稱的過濾列表與基於 URL 過濾列表，兩者間的阻擋結果只差 9.8%～17%。證明基於網域名稱過濾列表雖然無法細粒度地檢查網址完整的參數，但仍有良好的阻擋成效。

另一方面，由於近幾年推行的 DoH 技術，使得透過 HTTPS 加密的 DNS 請求也無法查看明文的網域名稱來阻擋廣告，因此我們設計一套實驗來檢驗 DoH Fallback 機制，透過阻擋 DoH 伺服器的網域或阻擋其 IP/Port，仍能使它退回傳統的 DNS 請求。我們將 DoH/廣告阻擋功能整合至封包過濾器 androidPF 上以證明實際運行在 Android 裝置上是可行的。

參考文獻

[1] A. Razaghpanah, R. Nithyanand, N. VallinaRodriguez, S. Sundaresan, M. Allman, C. Kreibich, and P. Gill, “Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem,” in Network and Distributed Systems Security (NDSS) Symposium, Feb. 2018.

[2] B. Hu, Q. Lin, Y. Zheng, Q. Yan, M. Troglia, and Q. Wang, “Characterizing location-based mobile tracking in mobile ad networks,” in 2019 IEEE Conference on

Communications and Network Security (CNS) . IEEE, 2019, pp. 223–231.

[3] K. Borgolte, T. Chattopadhyay, N. Feamster, M. Kshirsagar, J. Holland, A. Hounsel, and P. Schmitt, "How dns over https is reshaping privacy, performance, and policy in the internet ecosystem," Performance, and Policy in the Internet Ecosystem (July 27, 2019) , 2019.

[4] S. Son, D. Kim, and V. Shmatikov, "What mobile ads know about mobile users," in Network and Distributed Systems Security (NDSS) Symposium, 2016.

[5] M. Ikram and M. A. Kaafar, "A first look at mobile adblocking apps," in 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA) . IEEE, 2017, pp. 1–8.

[6] A. Shuba, A. Markopoulou, and Z. Shafiq, "Nomoads: Effective and efficient crossapp mobile adblocking," Proceedings on Privacy Enhancing Technologies, vol. 2018, no. 4, pp. 125–140, 2018.

[7] C.Y.Lan, "androidpf: An efficient and flexible packet filter on android device," Master's thesis, National Chiao Tung University, 2020.

[8] P. Snyder, A. Vastel, and B. Livshits, "Who filters the filters: Understanding the growth, usefulness and efficiency of crowdsourced ad blocking," Proceedings of the ACM on Measurement and Analysis of Computing Systems, vol. 4, no. 2, pp. 1–24, 2020.

[9] S. S. Hashmi, M. Ikram, and S. Smith, "On optimization of adblocking lists for mobile devices," in Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, 2019, pp. 220–227.

[10] U. Iqbal, P. Snyder, S. Zhu, B. Livshits, Z. Qian, and Z. Shafiq, "Adgraph: A graph-based approach to ad and tracker blocking," in 2020 IEEE Symposium on Security and Privacy (SP) . IEEE, 2020, pp. 763–776.

[11] D. Gugelmann, M. Happe, B. Ager, and V. Lenders, "An automated approach for complementing ad blockers'blacklists," Proceedings on Privacy Enhancing Technologies, vol. 2015, no. 2, pp. 282–298, 2015.

[12] T. Böttger, F. Cuadrado, G. Antichi, E. L. Fernandes, G. Tyson, I. Castro, and S. Uhlig, "An empirical study of the cost of dns-over-https," in Proceedings of the Internet Measurement Conference, 2019, pp. 15–21.

[13] A. Hounsel, K. Borgolte, P. Schmitt, J. Holland, and N. Feamster, "Comparing the effects of dns, dot, and doh on web performance," in Proceedings of The Web Conference 2020, 2020, pp. 562–572.

[14] K. Bumanglag and H. Kettani, "On the impact of dns over https paradigm on cyber systems," in 2020 3rd International Conference on Information and Computer Technologies (ICICT) . IEEE, 2020, pp. 494–499.

[15] S. Patil and N. Borisov, “What can you learn from an ip?” in Proceedings of the Applied Networking Research Workshop, 2019, pp. 45–51.
[16] A. Guttmann, “Mobile advertising spending worldwide 2007-2022,” 2019, https://www.statista.com/statistics/303817/mobile-internet-advertising-revenue-worldwide/
[17] TrendMicro, “Disguised adware infect 9 million google play users,” Jan. 2019, https://blog.trendmicro.com/trendlabs-security-intelligence/adware-disguised-as-game-tv-remote-control-apps-infect-9-million-google-play-users/
[18] AdblockPlus, https://adblockplus.org
[19] AdGuard, https://adguard.com/en/adguardandroid/overview.html
[20] DNS66, https://github.com/julianklode/dns66
[21] Adaway, https://adaway.org/
[22] Dns Over Https (DoH) , https://tools.ietf.org/html/rfc8484
[23] Mozilla, “Firefox continues push to bring dns over https by default for us users,” 25 Feb. 2020, https://blog.mozilla.org/blog/2020/02/25/firefox-continues-push-to-bring-dns-over-https-by-default-for-us-users/
[24] “Dns over tls support in android p developer preview,” 13 Apr. 2018, https://androiddevelopers.googleblog.com/2018/04/dns-over-tls-support-in-androidp.html
[25] Dns Over TLS (DoT) , https://tools.ietf.org/html/rfc7858
[26] AdblockMobile, https://nomobileads.com
[27] personalDNSfilter, https://zenzsolutions.de/personaldnsfilter/
[28] EasyList, https://easylistdownloads.adblockplus.org/easylist.txt
[29] Mozilla, “Firefox nightly secure dns experimental results,” Aug. 2018, https://blog.nightly.mozilla.org/2018/08/28/firefox-nightly-secure-dns-experimental-results/
[30] Pihole, https://pihole.net
[31] J. Livingood, M. Antonakakis, B. Sleigh, and A. Winfield, “Centralized dns over https (doh) implementation issues and risks,” 2019, https://tools.ietf.org/html/draft-livingood-doh-implementation-risks-issues-04
[32] Godlua, “An analysis of godlua backdoor,” Jul. 2019, https://blog.netlab.360.com/an-analysis-of-godlua-backdoor-en/
[33] Selenium, “Selenium with python,” https://seleniumpython.readthedocs.io/
[34] BrowsermobProxy, https://github.com/lightbody/browsermobproxy
[35] AdblockParser, https://github.com/scrapinghub/adblockparser
[36] Appium, https://appium.io/

[37] AppBrain, “Android ad network statistics and market share,” 2020, https://www.appbrain.com/stats/libraries/adnetworks?sort=apps

[38] Microsoft, “Windows will improve user privacy with dns over https,” Nov. 2019, https://techcommunity.microsoft.com/t5/networking-blog/windows-will-improve-user-privacy-with-dns-over-https/ba-p/1014229