內容

[壹、 工作內容 2](#_Toc183337973)

[一、 工作詳述 2](#_Toc183337974)

[二、 上半年完成進度 2](#_Toc183337975)

[三、 下半年完成進度 2](#_Toc183337976)

[貳、 學習 3](#_Toc183337977)

[一、 技術面 3](#_Toc183337978)

[*( i )、* ios, android app 介紹 3](#_Toc183337979)

[*( ii )、* app 測試準備 8](#_Toc183337980)

[*( iii )、* 檢測工具 10](#_Toc183337981)

[*( iv )、* 檢測 app 心得 12](#_Toc183337982)

[二、 非技術面 17](#_Toc183337983)

[( i )、 如何進行清楚的報告 17](#_Toc183337984)

[壹、 自我評估及心得感想 17](#_Toc183337985)

# 工作內容

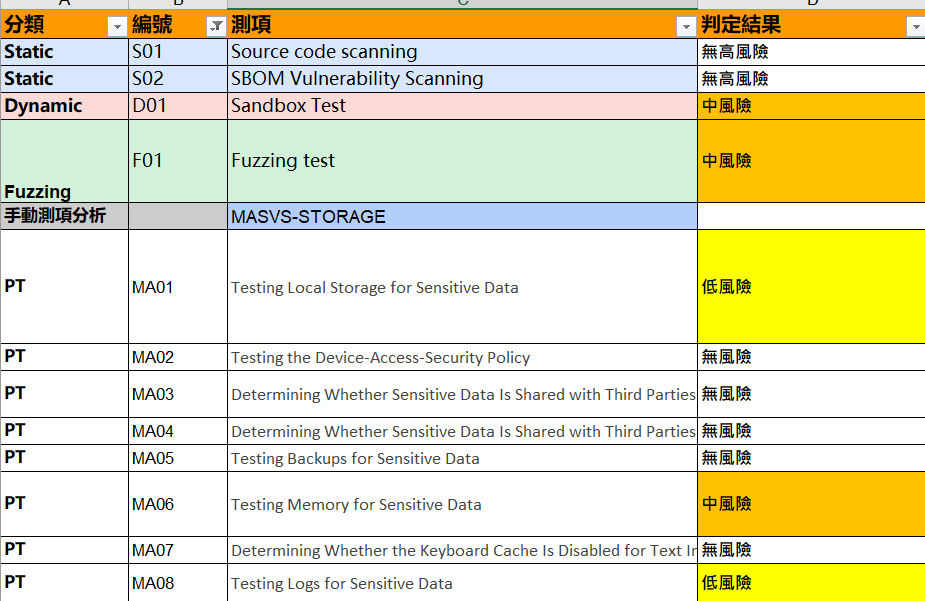
## 工作詳述

* 熟悉 OWASP MASTG Moblie Testing Guide，了解 app 的攻擊向量及對應的攻擊手法，並學習撰寫 app 的 best practice
* 撰寫中文版本測試手冊
* 加入公司的 app 檢測專案，主導ios app檢測，撰寫測試報告，跟主管會報測試進度

## 上半年完成進度

1. 了解webshell
2. webshell主題的文獻探討。
3. 確認接下來要往惡意加密流量進行研究。
4. 初步構思檢測系統

## 下半年完成進度

1. 熟悉OWASP MASTG檢測手冊
2. 加入公司的 APP 檢測專案，並主導IOS APP測試

# 學習

## 技術面

### ios, android app 介紹

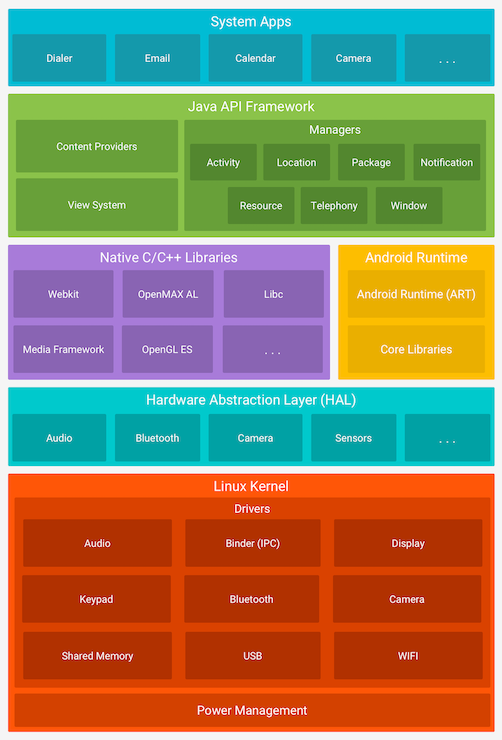
#### 執行環境

1. **Android**

**核心架構**

* **基於 Linux Kernel** ：Android 使用 Linux 核心進行底層系統管理，並進行修改以支持移動設備需求（例如電源管理、低記憶體處理）。
* **Dalvik Bytecode** ：Android 應用程式以 Java 或 Kotlin 編寫，並轉譯為 Dalvik Bytecode（.dex 檔案）。
* **硬體抽象層 (HAL)** ：抽象化硬體資源，提供應用層與硬體間的標準介面。

**應用沙盒**

* 每個應用運行於獨立的虛擬機環境中，透過沙盒機制保護其他應用與系統資源不被惡意應用影響。

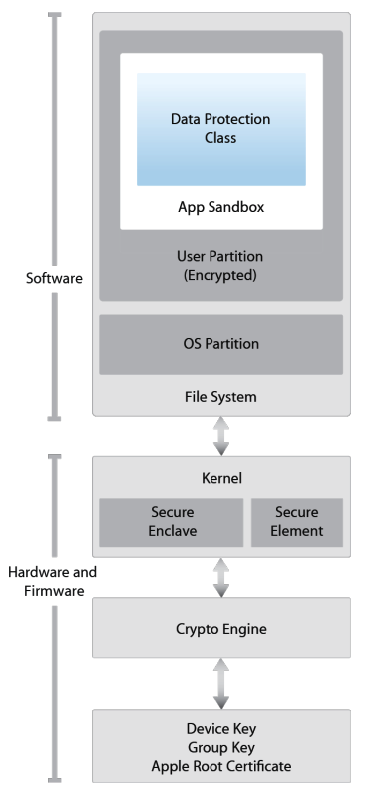
1. **iOS**

**核心架構**

* **基於 Darwin Kernel** ：基於 Unix 的 Darwin Kernel，具有 Apple 獨家修改（例如安全性與記憶體管理）。
* **Objective-C 與 Swift** ：
  + 使用 Objective-C 與 Swift 開發應用程式，所有程式碼必須通過 Apple 的簽名驗證（Code Signing）。
* **執行方式** ：
  + 編譯過程產生二進位檔案（Mach-O 格式），直接運行於硬體上，效能優化。

**安全機制**

* **安全啟動（Secure Boot）** ：通過 Boot ROM 驗證每個啟動階段的完整性，確保只能執行 Apple 簽名的程式碼。
* **沙盒機制** ：所有應用執行於沙盒內，限制對系統資源的訪問範圍，保護用戶資料。
* **硬體加密** ：使用 UID（裝置唯一 ID）進行資料加密，結合硬體層進一步保護敏感資訊。



#### 執行檔結構

##### Android 執行檔格式與結構

* **檔案格式** ：
  + **APK (Android Package Kit)** ：
    - Android 應用的標準封裝格式，包含應用程式的程式碼、資源與元數據。
  + **AAB (Android App Bundle)** ：
    - 新一代分發格式，Google Play 依裝置生成最佳化的 APK。
* **主要目錄結構** ：
  + /META-INF/：
    - 包含數位簽名資訊（CERT.RSA、CERT.SF）與應用清單。
  + /res/：
    - 應用資源檔案（圖片、佈局 XML、字串檔案）。
  + /lib/：
    - 存放針對不同 CPU 架構（如 ARM、x86）的原生程式庫。
  + /assets/：
    - 開發者自定義的應用資源。
  + /AndroidManifest.xml：
    - 定義應用基本資訊與權限。
* **執行檔** ：
  + **Dalvik Bytecode (.dex files)** ：
    - 由 Java/Kotlin 原始碼編譯生成，運行於 Android Runtime (ART)。
  + **原生程式庫 (.so files)** ：
    - 使用 C/C++ 寫成，提升效能。

##### iOS 執行檔格式與結構

* **檔案格式** ：
  + **IPA (iOS App Store Package)** ：
    - ZIP 壓縮格式，包含應用程式的執行檔與資源。
* **主要目錄結構** ：
  + /Payload/：
    - 應用的主目錄，包含所有執行檔與資源。
    - /Payload/Application.app：
      * 包含以下內容：
        + **Mach-O 執行檔** ：

直接運行於 iOS 裝置硬體上的原生程式碼。

* + - * + **Info.plist** ：

存放應用的配置檔案（如 Bundle ID、版本號）。

* + - * + **資源檔案** ：

包括應用圖標、圖片、聲音檔案、字串檔案。

* + - * + **nib 或 storyboard** ：

視覺化定義應用的介面佈局。

* + /iTunesArtwork：
    - 應用圖標檔案（512x512 PNG 格式）。
  + /iTunesMetadata.plist：
    - 包含應用相關的元數據（如開發者名稱、版權資訊）。

#### sandbox

##### iOS 與 Android 沙盒機制

|  |  |  |
| --- | --- | --- |
| 特性 | iOS | Android |
| 沙盒基礎 | 硬體加密 + App Sandbox | Linux 進程隔離 + 文件系統權限 |
| 數據目錄結構 | 每應用一個沙盒目錄 | /data/data/<package-name> |
| 權限聲明 | Info.plist 定義權限 | AndroidManifest.xml 定義權限 |
| 跨應用數據共享 | 預設不允許，僅通過系統 API 實現 | 可通過共享 UID 或 Content Provider |

#### IPC

**Android 和 iOS 的 IPC（Inter-Process Communication，進程間通信）**  
IPC 是操作系統中用於不同進程之間傳遞數據或訊息的機制。Android 和 iOS 分別基於其系統架構實現了不同的 IPC 方法。

|  |  |  |
| --- | --- | --- |
| 特性 | Android | iOS |
| 核心機制 | Binder | XPC |
| 通信方式 | Intent, AIDL, Messenger, Content Provider | URL Scheme, App Groups, XPC, Keychain |

### app 測試準備

#### root

1. 什麼需要android/ios device需要root/jailbreak
2. **解除系統限制**

* **應用沙盒隔離** ：現代操作系統（如 Android 和 iOS）會將應用運行在沙盒中，限制其與其他應用和系統文件的交互。獲得特權可以：
  + 訪問應用的數據目錄。
  + 查看應用生成的文件和配置。
* **繞過安全機制** ：許多安全機制會限制應用的某些行為（如數據加密或通信過程），需要特權來繞過這些限制，方便分析。

1. **進行動態分析**

* **調試應用行為** ：特權允許使用高級調試工具來觀察應用的運行時行為，例如函數調用和變數變化。
* **動態注入與鉤取（Hooking）** ：
  + **Android** ：通過工具如 Frida、Xposed，攔截應用內部函數，分析加密邏輯或支付接口。
  + **iOS** ：可以禁用 Apple 的程式碼簽名限制，注入自定義程式碼或調用未公開的系統 API。

1. **分析網絡流量**

* 繞過應用對 HTTP/HTTPS 請求的加密驗證，抓取和分析網絡流量。
  + 使用工具如 **Burp Suite** 或 **Wireshark** 。
  + 截獲 SSL/TLS 加密流量，分析敏感數據的傳輸是否符合安全標準。

#### Android Get Privilege (Rooting) Rooting 是指在 Android 系統中獲取最高權限（root user 的權限），允許使用者繞過系統限制，進一步掌控裝置的所有操作。以下是關於 Android 獲取特權的概述： 方法

1. **Unlock Bootloader** :

* 要 root Android，首先需要解鎖設備的 **bootloader** 。
* 不同廠商（如 Google、Samsung）有不同的解鎖程序，詳細方法通常可以在 XDA 開發者論壇找到。

1. **使用 Root 工具** ：

* **Magisk** :
  + Magisk 是目前最流行的 Android root 工具。
  + 支持 **systemless root** （不修改系統分區），因此更容易隱藏 root 狀態，適合敏感應用（如銀行和遊戲）。
  + 可安裝自定義模組（如 Xposed Framework 的 Systemless 版本）。

#### iOS Get Privilege (Jailbreaking)

在 iOS 中，獲取特權被稱為 **Jailbreaking** （越獄），它是禁用蘋果設備安全限制（如程式碼簽名檢查）的過程，從而允許執行未經授權的程式碼。  
**方法**

1. **安裝越獄工具** ：

* 越獄工具依賴於 iOS 的漏洞來禁用安全機制。
* 常見越獄工具：
  + **Checkra1n** :
    - 支持基於 **checkm8** 漏洞的越獄（僅適用於 A11 或更早版本的晶片）。
    - 支援 iOS 12 至 14.x。
  + **unc0ver** :
    - 支持 iOS 11 至 14.x，基於多種漏洞進行越獄。

1. **選擇越獄類型** ：

* **Tethered** （連接式越獄）：
  + 每次重啟設備都需要連接電腦重新應用越獄。
* **Semi-Tethered** （半連接式越獄）：
  + 設備可自行重啟，但重啟後進入非越獄模式，需使用工具再次越獄。
* **Untethered** （完全越獄）：
  + 最方便的越獄方式，僅需進行一次操作，越獄狀態永久保留。

1. **安裝越獄後工具** ：

* **Cydia** :
  + 非官方的 App Store，允許安裝未經蘋果簽名的應用程式。
* **Sileo** :
  + 替代 Cydia 的工具，同樣用於管理和安裝未簽名的應用。

#### Android 和 iOS Root/Jailbreak 的優勢與風險

#### 優勢

|  |  |  |
| --- | --- | --- |
| 功能/目的 | Android Root | iOS Jailbreak |
| 完全訪問權限 | 獲得系統最高權限（root），可修改系統文件、調整核心設定。 | 獲得文件系統的 root 權限，訪問和修改系統文件與核心設置。 |
| 第三方應用安裝 | 支援從未受 Google Play 保護的應用市集下載應用程式，例如破解應用或工具。 | 可以從 Cydia 或其他第三方應用商店下載未經 Apple 認證的應用程式。 |

**風險**

|  |  |
| --- | --- |
| 風險/限制 | Root |
| 保固失效 | 可能違反製造商政策，導致保修失效。 |
| 安全風險 | 移除系統安全限制，容易遭受惡意程式或病毒攻擊。 |
| 設備磚化風險 | 如果操作不當（如錯誤刷機或修改），可能導致設備無法啟動（Bricking）。 |

### 檢測工具

1. **Xcode**

* **用途** ：iOS/macOS 應用開發環境。
* **功能** ：
  + 提供 iOS 模擬器進行應用測試。
* **應用場景** ：ios app testing

1. **Visual Studio Code (VSCode)**

* **用途** ：通用程式碼編輯器，適合多種開發語言。
* **應用場景** ：android, ios testing

1. **ADB (Android Debug Bridge)**
   * **用途** ：用於連接電腦與 Android 設備的調試橋樑。
   * **功能** ：
     + 支援透過 USB 或 Wi-Fi 連接設備。
     + 可用於部署應用程式、查看日誌、傳輸檔案及遠端控制。
   * **範例** :

adb devices: 列出目前連線裝置清單  
adb version: 印出版本號  
adb shell: 進入裝置裡的 shell 模式  
adb shell <指令>: 讓 adb shell 去下你想要下的指令，例如 adb shell ls 會列出目錄  
adb install <apk>: 安裝 apk，當已經安裝過舊版本的程式時，可以使用 -r 去覆蓋。  
adb pull <檔案或目錄> [<放到哪個路徑>]: 拉檔案或目錄出來  
adb push <檔案或目錄> <推到哪個路徑>: 推檔案或目錄進去  
adb logcat: 印出系統log日誌訊息  
adb logcat | grep "$(adb shell ps | grep <package-name> | awk '{print $2}')":輸出關於某個app的日誌

1. **Jadx (Dex to Java Decompiler)**

* **用途** ：將 Android 的 DEX 檔案反編譯成 Java 原始碼的工具。
* **功能** ：
  + 提供 CLI 和 GUI 介面。
  + 能夠快速檢視 APK 中的程式碼結構。
  + 幫助分析應用程式內部邏輯及檢測漏洞。

1. **Frida**
   * **用途** ：動態分析工具，用於攔截和修改應用程式行為。
   * **功能** ：
     + 可進行 API 攔截、函式注入及調試。
     + 支援多平台（Android、iOS、Windows 等）。
     + 利用 JavaScript 編寫腳本來操作目標應用程式。
   * **範例** :

frida-ps -aiU #列出裝置上所有的app的名稱，還有他的package name  
frida-trace -U -n <process> #對指定進程進行動態跟蹤。  
frida-trace -U -i <class> #對特定類或方法進行動態跟蹤（Hook）。  
frida -U -n <process> -s <script> #加載本地腳本文件進行分析。

1. **Wireshark**

* **用途** ：網路封包攔截與分析工具。
* **功能** ：
  + 支援攔截並檢視實時網路流量。
  + 允許過濾協議（如 HTTP、TCP 等）進行深入分析。

1. **Burp Suite**

* **用途** ：Web 和移動應用安全測試平台。
* **功能** ：
  + 提供代理工具，用於攔截和修改 HTTP/HTTPS 流量。

### 檢測 app 心得

#### app 檢測概述

* 防治app的機敏資料，經由本地端儲存，網路傳輸，IPC機制傳輸，透過加密，確保資料在保存或傳輸的過程中不會外洩，被更改  
  ，在讀取或接收資料時，也可以透過HAMC對資料的完整性，真實性進行驗證。
* 防止app的程式被進行修改，或破解，造成商業機密，處理機敏資料的程式片段外洩，暴露攻擊者可利用的漏洞。

#### app 攻擊向量分析

|  |  |  |  |
| --- | --- | --- | --- |
| 攻擊面 | Android 攻擊向量 | iOS 攻擊向量 | 整合防禦措施 |
| IPC 通信與 URL Scheme | - IPC 機制（Intent、Service、Content Provider）可能導致敏感數據洩露或功能被濫用  - 深層鏈接（Deep Links）可能觸發未授權操作 | - 自定義 URL Scheme 可能導致敏感功能被濫用  - IPC 請求未驗證可能暴露敏感功能 | - 嚴格驗證 URL Scheme 和 IPC 請求的輸入  - 實施權限檢查，限制未授權的功能調用 |
| WebView 的安全性 | - JavaScript 支援不當可能導致 XSS  - addJavascriptInterface 可能被濫用- 加載未受信任內容可能執行惡意程式碼 | - JavaScript 支援可能引發程式碼注入  - 暴露原生方法（如 addScriptMessageHandler）可能被利用 | - 禁用不必要的 JavaScript 支援  - 僅允許加載可信來源內容-  測試並限制暴露的原生方法 |
| 網絡通訊 | - 使用 HTTP 傳輸數據導致攔截或 MITM- 接受自簽名憑證或無效憑證可能被攻擊- 敏感數據暴露於請求或標頭中 | - HTTP 傳輸數據可能被攔截- 無效憑證導致 MITM 攻擊- 敏感數據暴露於請求中 | - 強制實現 HTTPS，並使用憑證鎖定- 嚴格驗證 SSL/TLS 憑證  - 避免在標頭或 URL 中傳輸敏感數據 |
| 數據存儲 | - Shared Preferences 或外部存儲不安全- 敏感數據未加密或未檢查完整性 | - 未加密的本地數據可能被訪問  - 從未受信任來源加載數據可能導致安全風險 | - 使用 Android Keystore 或 iOS Keychain 存儲敏感數據  - 確保數據加密並檢查完整性- 避免加載未受信任的數據 |
| 防篡改與逆向工程 | - APK 容易被反編譯和修改- 使用 Frida 等工具進行動態攻擊 | - 程式碼被反編譯或重打包- 未授權的應用分發於第三方應用商店 | 使用 ProGuard/R8 或 SwiftShield 混淆代碼- 實施防篡改檢測，檢查應用簽名  - 定期檢查未授權應用分發 |

#### Android App 檢側實例

##### 介紹 webview

* + 簡介
    - 在app中開啟網頁
    - 功能比一般瀏覽器要少，不能輸入網址，也不能開啟多個分頁
    - 因為是在app中開啟，所以可以讀取app儲存空間內部的檔案
  + 可設置選項及對應攻擊向量
    - setAllowFileAccess()
      * 功能:設定WebView是否可以存取本地檔案
      * 預設可以存取
      * 風險:file Access in WebViews
        + app有android.permission.READ\_EXTERNAL\_STORAGE權限，可以透過file URL scheme(file://path\_to\_file)讀取任意檔案
      * file access setting(不同設定的層級不同)
        + setAllowFileAccessFromFileURLs(boolean)

default:

Android 4.0.4 以下 -> true

Android 4.1 以上 -> false

功能:設定WebView是否可以透過universal scheme(http, https)載入檔案

* + - * + setAllowFileAccessFromFileURLs(boolean)

default:

Android 4.0.4 以下 -> true

Android 4.1 以上 -> false

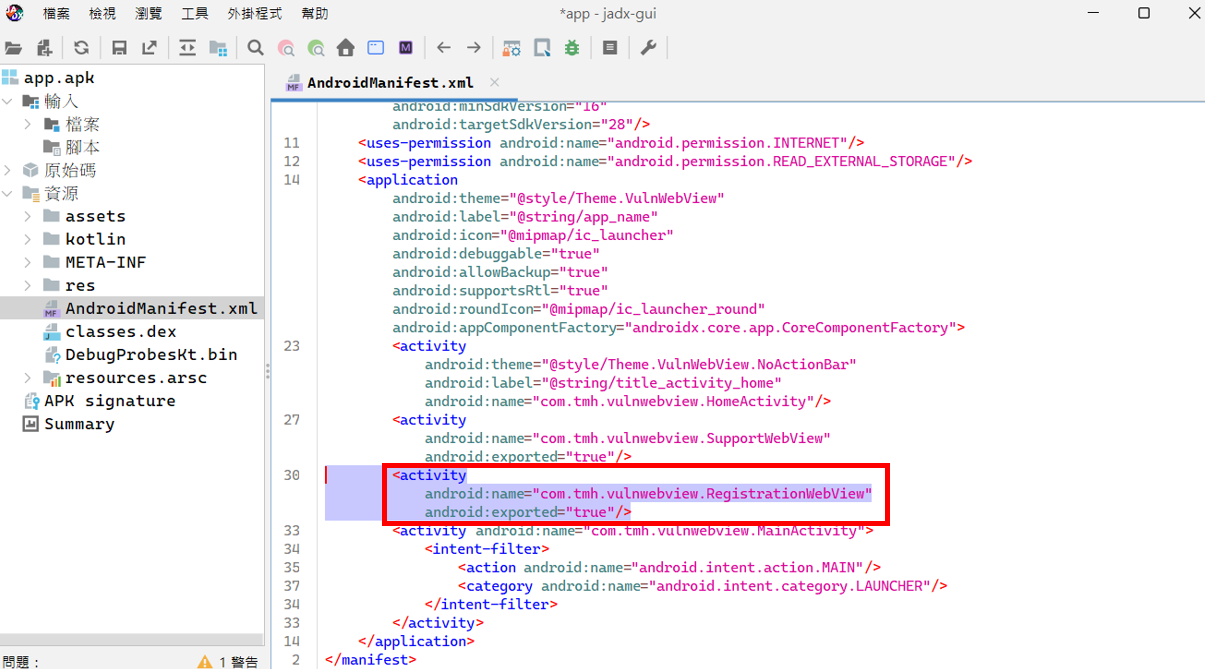
功能:設定WebView是否可以透過file URL scheme(file://)載入檔案

* + - * + setAllowFileAccess()

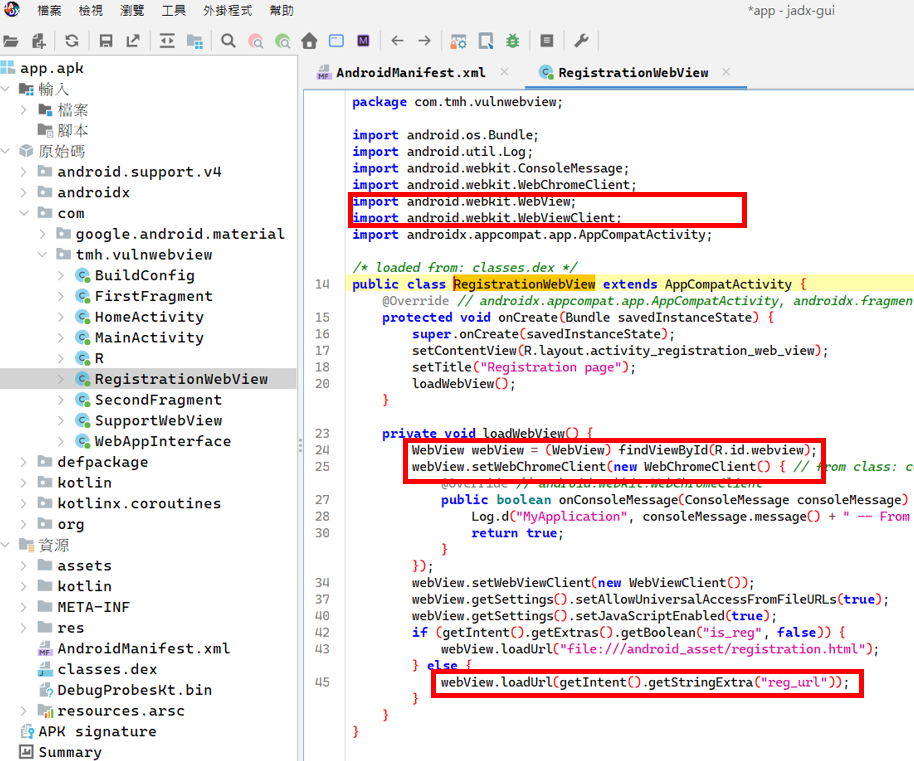
功能:禁止WebView存取file system(app 外部檔案)，但不包含file of app in assets and resources

##### 攻擊向量演示

* + 實驗環境:
    - Android Virtual Device
    - Vulnerable app:[Vulnerable WebView](https://github.com/t4kemyh4nd/vulnwebview)
    - Malicious app
* 觀察:
  + AndroidManifest.xml
    - RegistrationWebView.activtiy 設置 android:exported=true
    - ->可以由別的 app 開啟這個頁面



* + RegistrationWebView.activtiy
    - 有使用 webview 功能
    - webview載入的URL(reg\_url)是藉由IPC機制傳輸到RegistrationWebView -> 可以被控制



攻擊情境:

* + Malicious app 開啟 Vulnerable app 的 RegistrationWebView，同時傳送payload.html的URL
  + RegistrationWebView 載入payload.html頁面
  + payload.html讀取vulnerable app 內部的檔案，並傳送給c2 server(Command and Control Server)
  + 駭客得到機敏資訊

##### 檢測實作

* 測試項目名稱:Testing WebView Protocol Handlers
* 內容:
  + 檢查是否有使用WebView
  + 評估WebView是否有存取遠端或本地資源的需求
  + 如果有需要，確認是否符合最佳實踐
  + 最佳實踐
    - webview會使用到的HTML頁面，需要放在app中的儲存空間，不能放在外部儲存空間，以避免被加入惡意程式片段
    - 必須確保載入的URL沒有機會被更改
    - 創建白名單(可以被載入的網頁)，並在載入URL時進行檢查
    - 每當app開啟時，確認要載入的 HTML/Javascript file 沒有被修改
* webView.getSettings().setAllowFileAccess(false);  
    
  webView.getSettings().setAllowFileAccessFromFileURLs(false);  
    
  webView.getSettings().setAllowUniversalAccessFromFileURLs(false);  
    
  webView.getSettings().setAllowContentAccess(false);

## 非技術面

### 如何進行清楚的報告

報告內容需要寫的簡單易懂，因為除了要報告給主管以外，還需要跟客戶報告，除了文字以外，還需要截圖當作證據，證明影使用哪些不安全配置或服務。寫完測試報告，跟主管報告之前，我會先檢查三遍，第一遍，讀一遍自己的測試報告，確認邏輯是否通順，並思考格式或表達方式是否需要修改，第二遍，再次進行檢查，第三遍，嘗試自己報告一遍，確保自己報告時的流暢度。

# 自我評估及心得感想

雖然上面提到學了那麼多東西，但是我覺得我還是有不足的地方。像是時間管理方面，雖然我可以自己思考，並安排自己接下來要做甚麼，但是規劃的範圍基本上不超過兩個禮拜，也就是我目前沒辦法做長期規劃，所以我規劃的進度不是根據我這個大目標要甚麼時候完成，並把這個大目標又切成多個小任務，並且把每個小任務要完成的時間定下來，這就導致我感覺我自己比那些在公司實習，並且有接產學合作的同學還要閒的原因。

我最近有在嘗試改變，使用notion做自己的任務規劃，並且notion的功能，把對我來說重要的事件，拉一個timeline，讓我可以一目瞭然自己在甚麼時候，有甚麼重要的事，並即時調整自己的進度。最好把完成的時間定早一點，方便檢查、修改自己的成品。

我的情況比較特別，是到公司做專題，但是我也因此學到不少，除了可以藉由接觸業界觀察職場的互動方式，也有機會參與我這個部門舉辦的分享會，了解現在業界在關注甚麼議題，進而做出更符合現實世界的系統，很感謝系上，還有王老師，還有資策會的智偉主任，提供我這麼好的實習機會。