出國報告（出國類別：開會）

出席OWASP 2023 Global AppSec Dublin
出國報告書

服務機關：數位發展部資通安全署
姓名職稱：高家祥科長
賴秀瑜分析師
派赴國家：愛爾蘭
出國期間：112年2月11日至18日
報告日期：112年5月2日

摘　要

OWASP(Open Web Application Security Project)是一個全球性的非營利組織，致力於推廣應用程式安全性的知識和做法；OWASP Global AppSec是由OWASP主辦的年度全球性應用程式安全性會議，主軸在於交流最新應用程式安全性趨勢和最佳做法，並藉由讓參與者可以相互學習和交流的方式，建立起一個全球性的應用程式安全性社群，以促進全球網路安全和強化應用程式的防護能力。

OWASP訂於112年2月13日至2月16日在愛爾蘭都柏林舉辦「2023 Global AppSec Dublin」，期間包含2天實作研習及2天研討會議。為厚植我國國際交流量能、協力共創網安環境、有效降低我國資安風險及挑戰，爰派員參加本次會議進行國際交流，以進一步瞭解資安相關國際標準與規範推動進程，並掌握最新攻擊手法和因應對策。

目錄

壹、目的

OWASP(Open Web Application Security Project)是一個全球性的非營利組織，聚集了來自各個國家和地區的資安人員、軟體開發人員和學術界人士，共同致力於推廣應用程式安全性的知識和做法，並提供開放且免費的資源，以協助人們瞭解及改善應用程式的安全性。

OWASP Global AppSec是由OWASP主辦的年度全球性應用程式安全性會議，主軸在於交流最新應用程式安全性趨勢和最佳做法，並藉由讓參與者可以相互學習和交流的方式，建立起一個全球性的應用程式安全性社群，以促進全球網路安全和強化應用程式的防護能力，爰派員參加本次會議進行國際交流，以進一步瞭解資安相關國際標準與規範推動進程，並掌握最新攻擊手法和因應對策。

貳、過程

本次會議自112年2月13日至2月16日止，共計4日，參加場次如下：

日期	參與場次	
2月13日	Web Application Security Essentials	Mobile Security Testing Guide Hands-on
2月14日	Web Application Security Essentials	Mobile Security Testing Guide Hands-on
2月15日	GitHub Actions: Vulnerabilities, Attacks, and Counter-measures	JavaScript Realms - The Blank Spot In Web Application Runtime Security
	[T]OTPs are not as secure as you might believe	Don't let bug bounty kill your appsec posture
	Trusting Software - Runtime Protection Is the Third Alternative	Credential Sharing as a Service: the Dark Side of No Code
	Philosophizing security in a "mobile-first" world	Server Side Prototype Pollution
2月16日	Testability Patterns for Web Applications - a new OWASP project	

	OWASP SERVERLESS TOP 10	Hacking and Defending APIs - Red and Blue make Purple
	Automated Security Testing with OWASP Nettacker	Log story short: Chopping through forests of data
	Mobile Wanderlust”！Our journey to Version 2.0!	Let’s Cook: Contextual Vulnerabilities are the Ingredients and OWASP Top 10 Mapping the Seasoning

參、會議紀要

一、Web Application Security Essentials(主講人：Fabio Cerullo)

這個實作研習結合理論和實踐練習，提供評估網路應用程式安全所需的知識和資源。通過理解理論和強調實踐練習，能夠識別網路應用程式中的關鍵漏洞，了解利用的運作方式，並學習實施必要的修補措施。

第一天的內容主要在進行實作研習介紹、實作研習所需工具介紹與安裝以及OWASP TOP 10的風險弱點介紹及利用方式，第二天則是進行剩餘的OWASP TOP 10的風險弱點介紹及利用方式，內容如下：

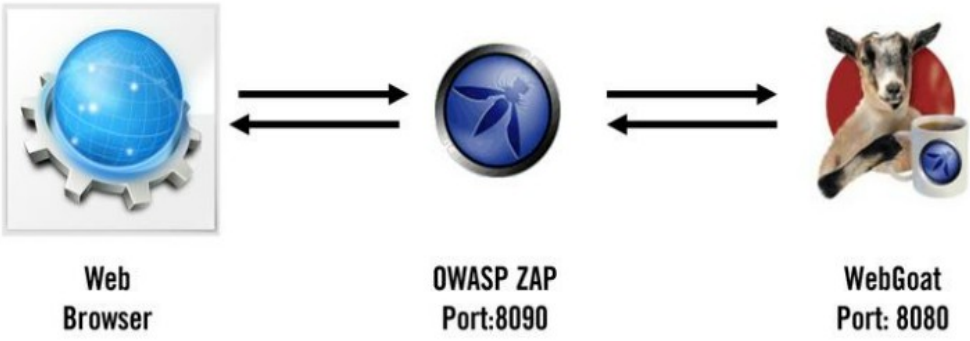
1.實作研習工具介紹

這個實作研習主要會使用到OWASP Zed Attack Proxy(ZAP)、OWASP WebGoat及瀏覽器開發人員工具。

OWASP Zed Attack Proxy（ZAP）是一個OWASP所開發及維護免費且開放程式碼的應用程式安全測試工具專案，可以協助資安及開發人員在測試和審查網路應用程式時發現漏洞和弱點，主要功能包括漏洞掃描、漏洞利用、攻擊模擬、自動化測試等，也支援各種語言和框架的應用程式測試，包括Java、Python、Ruby和PHP等，並且支援Windows、Linux和macOS作業系統。這個實作研習主要會將OWASP ZAP 作為代理伺服器來從中分析封包內容。

OWASP WebGoat是一個OWASP所開發免費和開放程式碼的網路應用程式，用於教授常見的網路應用程式漏洞和攻擊技術的教學平台，包含了一系列的網絡應用程式安全測試用例，包括SQL注入、跨站腳本（XSS）、文件包含、授權問題等，讓開發人員和測試人員可以學習如何測試、發現和利用網路應用程式漏洞。這個實作研習主要會讓參與者藉由OWASP WebGoat來了解及練習攻擊手法。

實作研習所使用的工具架構如下：

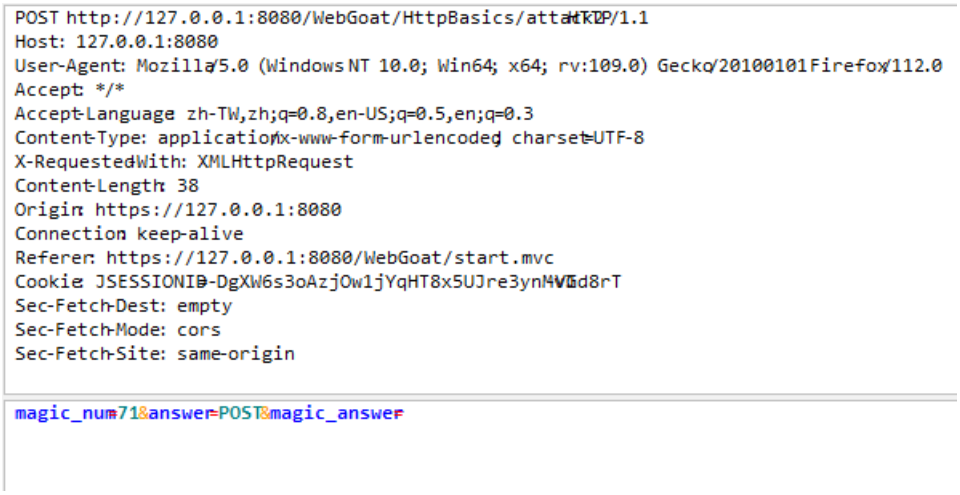


圖：研習工具架構(資料來源：實作研習簡報)

2.網路應用程式所使用的技術(Technologies used in Web Applications)

HTTP是一個客戶端和伺服器端的架構，客戶端通常是瀏覽器，而伺服器端通常是網站伺服器。當一個客戶端要求從伺服器端獲取一個網頁或文件時，它會使用HTTP協定發送一個請求，這個請求包括URL、HTTP方法（GET、POST、PUT等）、HTTP版本、請求標頭等。伺服器端接收到這個請求後，會根據請求中的資料來處理請求，最後回傳一個HTTP回應，包括HTTP版本、狀態碼、回應標頭等，以及所請求的網頁或文件的內容。

因為HTTP客戶端和伺服器端的架構，有心人士就可以在瀏覽器和網站伺服器之間插入自己的代理伺服器，就如同這個實作研習的架構，進而可以查看、修改或攔截傳輸的資料，在WebGoat HTTP Basic及HTTP Proxies的練習中可以利用OWASP ZAP來進行攔截或修改資料，如下圖所示：



圖：利用OWASP ZAP來進行攔截資料

3. 網路應用程式的關鍵領域(Critical Areas in Web Applicaitons)

網路應用程式中的關鍵領域通常擁有最機敏的資料，攻擊者通常會針對這些領域進行攻擊，因此，開發人員和資安人員要進行網路應用程式防護時必須先盤點出這些關鍵領域，並分析相關風險後實施相應的安全措施，如身份驗證和授權機制、加密通信、安全的控制存取和資料傳輸等，而OWASP也針對這些關鍵領域有可能發生的風險盤點出OWASP TOP 10。

要盤點網路應用程式的關鍵領域可以使用攀爬程式、人工瀏覽和強制瀏覽。使用攀爬程式可自動發現及收集應用程式中的所有頁面和功能，再以人工瀏覽的方式去補足攀爬程式不足的地方，來發現應用程式中的隱藏頁面和功能。而強制瀏覽通常透過使用自動化工具來嘗試不同的URL去發現應用程式中隱藏或未公開的頁面和資源，以及暴露可能存在的安全漏洞和風險。

在盤點出應用程式關鍵領域後就可以進行風險分析，OWASP 開發了一套OWASP Risk Rating Methodology用於評估應用程式安全風險的方法論，以幫助資安人員快速地識別和評估應用程式中的風險，進而更好地保護應用程式和其使用者。OWASP Risk Rating Methodology 可根據應用程式中可能存在的風險因素，包括應用程式的機密性、完整性和可用性，以及其他因素，例如風險的概率和影響等，進行評分，這些風險因素通常與安全漏洞和攻擊有關。

OWASP 也根據各種統計資料和研究，評估全球網路應用程式的安全性風險，並將結果定期發表為十大網路應用程式安全風險排名(OWASP TOP 10)，可用來幫助企業、開發者和使用者在設計、開發、測試和使用網路應用程式時了解其風險，並為企業和組織提供評估和遏止這些風險的指導方針。目前最新的OWASP TOP 10為2021年發布，項目如下表所示：

RISK	DESCRIPTION
A1:2021	BROKEN ACCESS CONTROL
A2:2021	CRYPTOGRAPHIC FAILURES
A3:2021	INJECTION
A4:2021	INSECURE DESIGN
A5:2021	SECURITY MISCONFIGURATION
A6:2021	VULNERABLE AND OUTDATED COMPONENTS
A7:2021	IDENTIFICATION AND AUTHENTICATION FAILURES
A8:2021	SOFTWARE AND DATA INTEGRITY FAILURES
A9:2021	SECURITY LOGGING AND MONITORING FAILURES
A10:2021	SERVER-SIDE REQUEST FORGERY

表：OWASP TOP 10

4. 不安全的存取控制(Broken Access Control)

不安全的存取控制指的是未能正確限制用戶對系統資源的訪問權限，導致未經授權的用戶可以訪問、修改或刪除應該受限制的資源。簡單來說，當一個應用程式的訪問控制機制被破壞，攻擊者就能夠繞過該機制，並以超出其正常權限的身份進行操作。例如，一個攻擊者可以通過修改URL、Cookie或其他資訊來存取應該只能被授權使用者存取的資源，或者直接利用暴力破解攻擊來獲取管理員權限。

不安全的存取控制可能導致嚴重的安全問題，例如敏感資料外洩、破壞系統完整性、未授權的資源存取及資料竊取等。為了保護應用程式不受這種類型的攻擊，可以建立白名單機制以及建立存取控制矩陣來管理和控制系統中的資源和權限、使用中央管理式的存取控制機制(如AD、LDAP等)、特別機敏的資源存取可用限制IP存取的方

式。

5. 加密失敗(Cryptographic Failures)

加密失敗指的是在網路應用程式中，因為加密演算法保護措施無法發揮作用，導致機敏資料的保護失敗。攻擊者可能能夠透過這些加密失敗，繞過應用程式的安全控制，並讀取、修改或破解機敏資料。

加密失敗的原因，往往來自於使用弱密碼、使用過舊或是加密強度不高的加密演算法、使用預設或是重複使用的密碼以及沒有在適當的層級進行加密；而預防加密失敗的做法可以使用高強度的密碼及安全的演算法、使用高強度的雜湊函數來儲存密碼以及確認在每個適當的層級都有進行加密，以確保所有的機敏資料都可以安全的傳輸及儲存。

6. 注入(Injection)

注入指的是攻擊者將惡意腳本或指令插入到應用程式中的資料輸入欄位，並透過這些欄位向資料庫或其他後端伺服器傳送請求，從而獲得應用程式未經授權的存取權限或進行其他危險操作的一種攻擊方式。

在 Web 應用程式中，最常見的注入攻擊是 SQL 注入攻擊和 XSS攻擊。在 SQL 注入攻擊中，攻擊者會利用表單、Cookie、HTTP標頭等輸入欄位的漏洞，將惡意的 SQL 語句注入到網路應用程式中，並以此方式繞過身份驗證，來獲取應用程式的機敏資料或直接對資料庫進行操作。

預防 SQL注入攻擊需要採取複合式的保護措施，包括輸入過濾及驗證、資料庫最小權限原則及避免顯示資料庫錯誤資訊框架等。

在 XSS 攻擊中，攻擊者利用網頁應用程式未能有效過濾使用者輸入的資料，在輸入欄位中插入惡意的腳本，當其他使用者訪問該頁面時，該腳本將被執行，並可能導致使用者的敏感資訊，如用戶名、密碼、信用卡號等資訊被竊取。

預防XXS攻擊的方法可以對使用者輸入的資料進行適當的過濾和驗證、在網站中使用HTTPOnly cookie來防止攻擊者利用JavaScript取得cookie的值以及在HTTP表頭中設置Content Security Policy來限制網頁中可以載入的資源和允許執行的腳本等。

7. 不安全的設計(Insecure Design)

不安全的設計指的是在網路應用程式的設計階段，未考慮安全性風險而造成的安全漏洞。常見的不安全設計包括未將安全性考慮納入需求分析、缺乏權限控制機制、未將敏感資料進行加密、未考慮攻擊者可能利用的安全漏洞等。

我們可以透過威脅建模(Threat Modeling)的方式來預防不安全的設計。威脅建模是網路應用程式開發人員、資安人員和其他相關人員從系統的安全目標開始，通過識別威脅、分析威脅、評估威脅及處理威脅等步驟來識別和分析系統中可能出現的威脅，將可能存在的安全威脅降到最低，提高系統的安全性。

8. 不安全的組態設定(Security Misconfiguration)

不安全的組態設定指的是應用程式或伺服器配置不當，如未刪除或停用預設帳戶、密碼或金鑰，未更新軟體或漏洞版本，以及未設置密碼複雜度、憑證或權限等，導致安全弱點或風險。

要預防不安全的組態設定，我們可以刪除不必要網站功能以確保網站功能最小化、使用最新版本的應用程式和網站伺服器來解決已知的漏洞和弱點、每個領域都配置適當的安全組態設定以及定期進行安全測試和審查來發現和修復任何安全漏洞和配置錯誤。

9. 脆弱及過舊的原件(Vulnerable and Outdated Components)

脆弱及過舊的原件表示應用程式或系統中使用的原件（如框架、套件、函式庫等）有安全漏洞或版本過舊，使得攻擊者能夠利用這些漏洞進行攻擊。

要預防脆弱及過舊的原件，我們可以移除應用程式中未使用的原件來減少應用程式中潛在的漏洞、從官方網站下載和使用原件以及建立原件管理機制，來對原件版本進行控管、針對原件更新及漏洞告警進行追蹤和管理，以及時發現和修補漏洞。

10. 識別及驗證失效(Identification and Authentication Failures)

認證及驗證失效在應用程式中識別使用者身份和進行身份驗證時可能出現的錯誤。識別是指應用程式中識別用戶身份的過程，而驗證則是在識別的基礎上，通過驗證用戶提供的資訊確認用戶身份是否有效的過程。當識別和驗證失敗時，攻擊者可能

會以受害者的身份存取應用程式中的資源，例如機敏資訊、個人資訊等，甚至完全掌控受害者的帳戶。

要預防認證及驗證失效，我們可以將密碼以加密的方式儲存或傳輸、設定密碼複雜度、減少過多登錄失敗訊息的揭露以及使用多因子驗證來確保安全性等。

11. 軟體與資料整合錯誤(Software and Data Integrity Failures)

軟體與資料整合錯誤指的是攻擊者能夠修改應用程式或資料庫中的資料，以及在系統中插入惡意的程式碼或數據，主要原因是應用程式中缺乏適當的驗證和控制機制，讓攻擊者可以透過不當使用網頁表單或API來進行駭侵。這種攻擊可能會導致資料損壞、網站異常、機敏資料外洩等問題。

為了預防軟體和資料完整性錯誤，我們可以實施適當的身份驗證和授權機制、實施資料驗證和過濾機制來防止攻擊者通過注入攻擊等手段修改資料以及測試和評估應用程式的安全性，包括進行程式碼審查、弱點掃描和滲透測試等，以發現和修復可能存在的安全漏洞。

12. 安全的日誌及監控失效(Security Logging and Monitoring Failures)

安全的日誌及監控失效指的是系統未能在發生安全事件時及時記錄和監控這些事件。這種失敗可能會導致安全事件不被發現、無法追踪事件的起源、誰存取了系統以及他們存取了哪些資料等問題，使攻擊者可以繼續攻擊，或者導致對事件的調查不完整，無法了解發生了什麼事情以及進行源頭追查。

為了避免安全日誌和監控失敗，我們可以定期進行安全測試來確保安全監控措施實際運作、以加密或其他適當方式確保相關日誌的完整性以避免被刪除或竄改、確保日誌收集的完整性以利相關事件的追查以及規劃完整的安全監控機制等。

13. 伺服器端要求偽冒(Server-Side Request Forgery)

伺服器端要求偽冒指的是攻擊者可以利用應用程式允許發起對其他應用程式、內部網絡和資源的HTTP請求的功能，將攻擊者發起的HTTP請求發送到受害應用程式，利用受害應用程式的權限進行攻擊。

為了預防SSRF漏洞，我們可以實作針對應用程式能夠訪問的外部資源進行嚴格

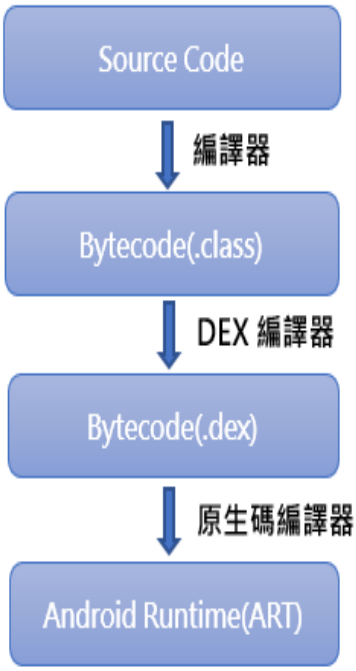
限制，像是限制應用程式能夠發起的請求協議、使用白名單限制應用程式能夠訪問的URL、過濾所有從用戶端提交的URL請求以避免攻擊者利用訪問區分符號(如@、//等)繞過訪問限制、實施高強度的身份驗證和授權機制以限制未經授權的用戶訪問應用程序等。

二、Mobile Security Testing Guide Hands-on(主講人：Sven Schleier)

第一天實作研習以介紹Android系統為主，第二天則是介紹iOS系統、平台及安全架構(硬體安全、原始碼簽章、沙盒、安全模式及安全隔離區等)，內容如下：

1. Android平台及安全測試簡介

Android是以Linux為基礎的開源軟體，使用在各式各樣的設備及主機板規格上，應用程式(APP)可以用Java或是Kotlin進行開發，撰寫的原始碼在經過編譯成bytecode後，就可以在Android Runtime執行。每個應用程式都有一個獨一無二的ID(憑證)，可以讓行動裝置及Google Play Store去辨識是否為同一個應用程式，ID如果改變即會被視為是一個新的應用程式。



圖： Android 程式編譯流程

Android的安全測試可以依照測試方式分為靜態測試(Static Application Security Testing, SAST)以及動態測試(Dynamic Application Security Testing, DAST)，靜態測試只單純分析應用程式的原始碼，不會執行，而動態測試則是包含應用程式在runtime的測試行為。

另外安全測試也可以分別透過實體設備或是模擬設備進行，實體設備速度較快，但會因為不同硬體設備所可以測試的功能不同(例：指紋掃描器)，導致如果需要測試多種存在在不同硬體設備上的功能，就需購置多種設備，成本較高，且依設備不同會決定設備是否能進行root，但對於硬體外加設備的測試則較為容易進行。使用模

擬設備測試則是速度較慢，但有免費的模擬平台可以使用(如本實作研習使用的Corellium)，通常模擬設備預設為已被root狀態，另外對於附加硬體設備則會依照模擬平台所提供硬體輸入而有一定限制。常用測試工具有：(Host端)MobSF、Android SDK、Frida、Objection、(Android設備)Busybox、Frida、Magisk。

2. Android應用程式架構介紹

一個APK裡面通常包含：AndroidManifest.xml、classes.dex、resources.arsc、res/、assets/、Lib/ 及META-INF/folder：

圖： Android 應用程式內容

(1).AndroidManifest.xml：以XML格式編成的manifest檔案。

(2).classes.dex：編譯成DEX格式的原始碼。

(3).resources.arsc：包含被編譯的資源檔，如顏色、風格等。

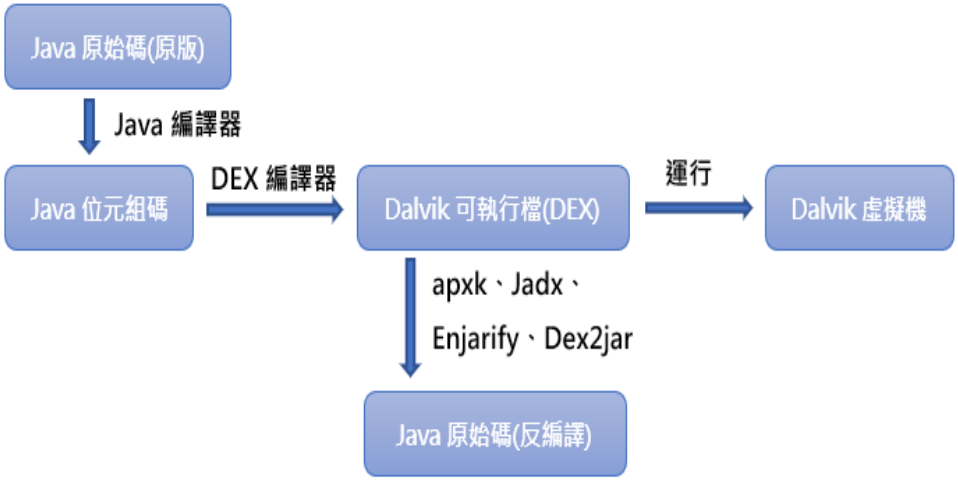
(4).res/：包含APP所需要的資源的資料夾，如圖片、xml等。

(5).assets/：包含可以被資產管理員讀取的資產資料夾。

(6).Lib/：包含被編譯的原始碼資料夾。

(7).META-INF/：包含儲存metadata以及APK的簽章資料等的資料夾。

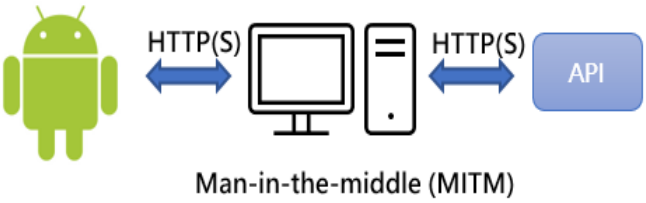
應用程式無法直接透過解壓縮APK來分析，因為AndroidManifest無法直接被讀取且Dalvik executable format(DEX)並非是可以直接讀懂的編碼，因此需要使用jadx或jadx-gui來反編譯DEX bytecode為java classes，並對AndroidManifest.xml及resources.arsc進行解碼，才能進行分析。另外，也可以透過各種工具結合(apktool解碼、jadx,Enjarify,Dex2jar反編譯dex為java)或是Mobile Security Framework(MobSF)來進行靜態分析。



圖：Android編譯與反編譯流程圖

3. 攔截透過行動裝置應用程式架構送出的流量

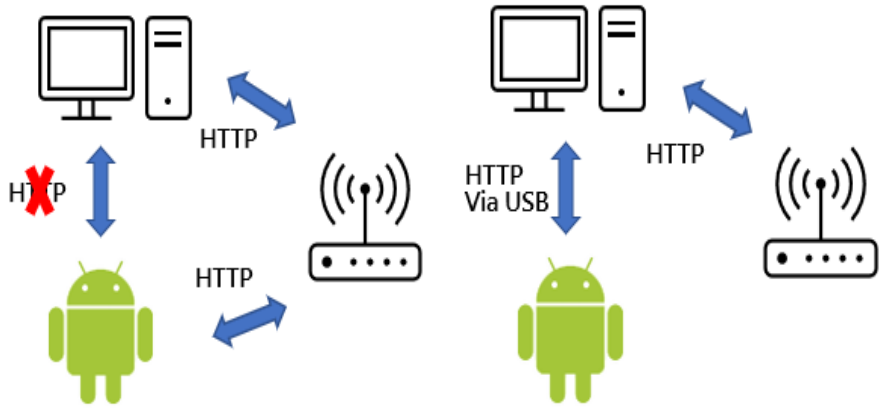
主流的應用程式都是透過HTTP(S)來傳送資料，如果需要攔截流量就必須要成為Man-in-the-middle，將資料攔截下來之後再轉發，因此會受到Android系統上網路安全設定限制。



圖：資料攔截示意圖

針對這個問題有兩種方式可以使用，一種是將Burp的憑證安裝在Android系統憑證上，另一種方式則是將網路安全設定變更後再重新包裝應用程式，但第2種方式會需要重新包裝所有需要被測試的應用程式，較為繁瑣。此外也可能遇上一些問題，例如：用戶使用獨立的WiFi、Google Flutter或Xamarin等應用程式框架或使用非Http埠等問題。

如果用戶未經過流量攔截設備直接連接WiFi，就會無法進行流量分析，所以需要在被測試的行動裝置裝設定流量重新導向，將流量導向流量攔截設備後，再透過該設備攔截行動裝置的流量。而用戶使用Google Flutter或Xamarin等應用程式框架或使用非Http埠等問題，則可以開啟使用支援隱藏代理的功能來進行流量攔截。



圖：設定手機流量導向

4. 辨識及利用深度連結(Deep links)弱點

所謂深度連結是一串可以從應用程式裡直接將使用者導向特定內容的網址，Android應用程式連結(App Links)則是一種特別型態的深度連結，可以利用網址將使用者導向特定的應用程式內容。

深度連結是一種可以在應用程式中觸發APP至特定頁面位置的網址，例如可以用fb://來開啟Facebook APP、用fb://profile/33138223345來透過Facebook的APP開啟維基百科中Facebook的檔案等。深度連結技術本身並非是弱點，但深度連結技術開啟了另一個攻擊可能性，駭客可以透過一個應用程式去使用另一個應用程式的內容，並且引導使用者開啟錯誤的應用程式。例如：攻擊者可以透過引導使用者去點擊惡意的深度連結，將使用者重新導向攻擊者控制的伺服器來接收使用者令牌(token)，造成令牌外洩。

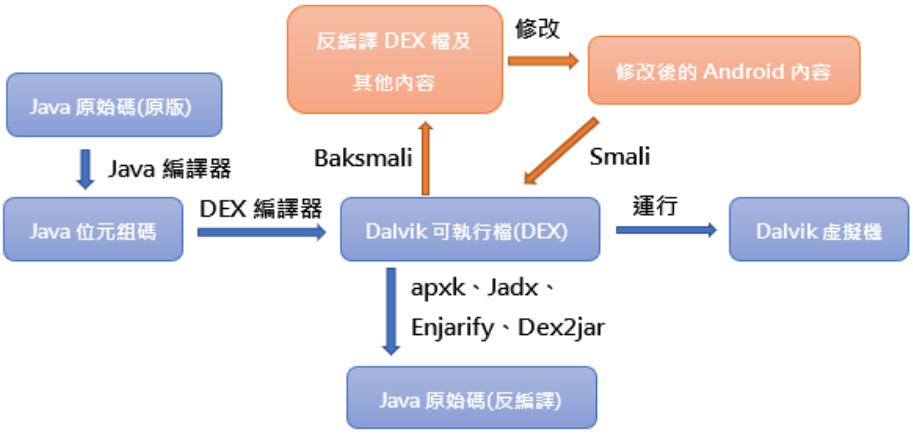
Android應用程式連結是其中一種可以強化資安的做法，Android應用程式連結可以透過驗證讓所使用的HTTP連結只能連到自己所擁有的網域，其他應用程式無法使用相同的連結，來避免惡意連線。

5. 探討逆向工程

逆向工程可以使用在繞過客戶端的安全控制機制上，例如root偵測機制等。root偵測機制可以讓應用程式更加難以在被root的設備上運行，其常見的偵測方法

有：確認相關檔案是否存在、確認執行中的程序、確認已安裝的應用程式、確認可以寫入的區塊與系統檔案夾及確認客製化應用程式建置等。

新增應用程式時，APK檔案中的classes.dex檔案包含可以執行在Android Runtime的binary bytecode，但因為二元碼(binary code)難以閱讀，所以可以先透過Smali將DEX格式轉換成較容易理解的組合語言後再進行修改。攻擊者可以透過這種方式在熱門應用程式加上惡意內容後，重新包裝上傳至APP Store來達成攻擊目的。



圖：Smali逆向工程流程圖

另外也可以使用開源的套裝軟體，常見的有：MagiskSU(提供應用程式的root存取)、Magisk Modules(變更限制讀取)、MagiskHide(從root偵測及系統完整性偵測隱藏Magisk)、MagiskBoot(解包裝及重新包裝Android啟動映像檔的工具)，來自動化整個修改流程。

除了直接去修改應用程式的靜態內容外，也可以使用Frida在runtime進行修改，Frida是一個開源的runtime測試框架，可以在Windows、 Linux、 macOS、 iOS、Android等平台程序中添加JavaScript Code在library裡，並透過各種功能的API在程序中添增一個完整的JavaScript runtime。

運作Frida有兩種方式，一種是在行動裝置上運行Frida server，但這種方式只能運作在已root的設備上，它可以讓Frida自動處理程序添加；另一種則可以運作在未被root的設備上，但需要重新包裝應用程式，將frida-gadget.so包進需要添增程序的

應用程式中，本次實作研習是使用第一種方法，在虛擬行動裝置上運行Frida Server。

Frida的執行方式有兩種，一種是用Frida CLI，類似Ipython可以透過命令和應用程式進行互動，另一種是用Java script或是python來撰寫，直接添加在應用程式程序上。

Frida CLI有幾項基礎功能：

(1).Hook：類似設定breakpoint功能，可以在函式中設定斷點來進行檢視修改等動作。

(2).Inspect：檢視傳入函式中的參數。

(3).Intrument：添加額外的指令來幫助除錯。

(4).Manipulate：直接修改函式流程或完全覆蓋函式。

用撰寫Java script或是python添加程序，可以透過附加或是新增程序的方式進行。新增或添加程序前必須先用frida-ps去查詢運行中應用程式pid以及應用程式id後，再用flag -F進行程序添加，或使用-f <app id>進行程序新增，例如frida -f com.app.name。

然而，為了不被Frida修改程序，許多應用程式也會用一些方法去偵測Frida是否存在，例如偵測記憶體、確認Frida Server是否正在運行、預設埠是否開啟或是簽章是否被竄改等。

確認 Frida 是否在執行序中執行

```
ProcessName = list.get(i).process;
If(processName.contains( "fridaserver" )){
   retVal = True
}
```

測試 Frida 預設通訊埠 27047

```
sa.sin_port = htons(27047);
inet_aton( "127.0.0.1" ,&(sa.sin_addr));
int sock = socket(AF_INET,SOCK_STREAM,0);
if(connect(sock,(struct sockaddr*)&sa,sizeof sa)!=-
1){
reVal = True
}
```

確認 Frida 的程式庫是否存在記憶體中

```
Fp = fopen( "/proc/self/maps" ," r" );
If(fp){
while(fgets(line,512,fp))
{ if(strstr(line," frida" )){
   retVal = True}}}
```

如果預設通訊埠已經被使用，可以透過送 D-BUS AUTH 命令給所有通訊埠來偵測，假設有回應則 Frida server 可能正在運行

```
send(sock," \x00" ,1,NULL);
send(sock," AUTH\r\n" ,6,NULL);
```

圖：偵測Frida是否存在

Frida、Magisk及Smali Patching比較如下：

	Magisk	Frida	Smali Patching
部署	第三方復原	Android Debug Bridge(ADB)+操作電腦	Android Debug Bridge(ADB)+操作電腦
侵入方式	修改作業系統	修改目標程序	變更編譯的應用程式
變更層級	方法(Method)	方法(Method)	指令(Intruction)
更新方式	重啟應用程式或設備	即時	重新編譯

表：Frida、Magisk及Smali Patching比較表

6. TLS及SSL 綁定(Pinning)介紹

TLS(Transport Layer Security)是透過公私鑰加密技術來對傳輸資料進行非對稱式加密技術，需要將SSL憑證安裝在伺服器上。當使用者向伺服器連線時，就會透過SSL憑證取得伺服器公鑰對資料加密後傳輸給伺服器，伺服器端收到資料後會再以自己的私鑰對資料進行解密，而達到保密的效果。

圖：Android憑證示意圖

SSL綁定(SSL Pinning)是在應用程式中使用的一種TLS加密技術，開發者會將SSL憑證事先包裝在應用程式中，當需要連線時，就會在TLS handshake之後使用事先包裝在應用程式中的SSL憑證去和目前網站正在使用的SSL憑證去做比對是否一致，若不一致則會拒絕連線。

另外，SSL綁定又可以分為憑證綁定(Certificate Pinning)跟公鑰綁定(Public Key Pinning)，一個是將整個憑證包裝進APP中，另一個則只是將Public Key包裝進APP。

	憑證綁定	公鑰綁定
安裝	實作簡單	實作簡單
時效	當憑證過期時即過期	停止使用該公鑰時即過期
挑戰	1.可能有多個憑證	1.如果使用同一個公鑰安全性可以維持多久

	2.必須經常性更新	2.自簽憑證

表：憑證綁定跟公鑰綁定比較表

SSL綁定是一種在客戶端控制的機制，只要有足夠的時間與技術，都有可能透過逆向工程被破解，因此，開發者可以使用root偵測或是偵測APK是否有被進行動態操作來讓破解SSL綁定的行為變得更加困難。

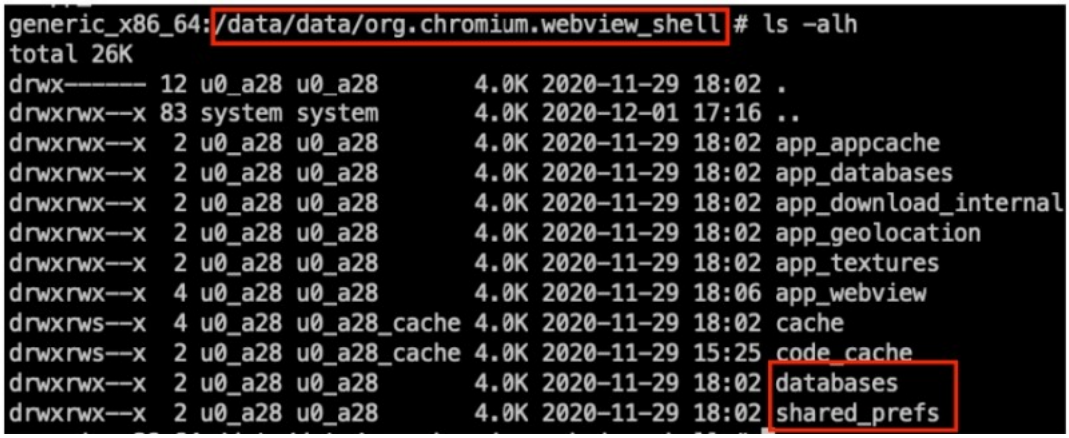
7. 分析APP的本地儲存內容

因為敏感資料的洩漏可能會提供攻擊者更多攻擊手段，例如社交工程或是盜帳號等 ，所以一般來說會建議將敏感資料盡可能永久地儲存在本地，除此之外也有一些必須儲存在本地的資料，例如session cookie、access tokens、encryption keys以及離線需要使用的資料等。使用在Android系統最常見的儲存資料技術為Shared Preferences、SQLite Databases以及Files(直接儲存)。

Android Files提供兩種形式的儲存方式，內部儲存(預設)以及外部儲存，各個應用程式會預設使用內部儲存，並在內部儲存新建用來儲存該應用程式資料的資料夾，該資料夾在Android保護機制之下各個應用程式間無法相互讀取寫入資料。外部儲存則為SD卡等擴充裝置，所有應用程式皆可進行讀取寫入，使用者也可以隨時移除，因此敏感資料不建議放在外部儲存中。

Shared Preferences是利用XML格式來儲存鍵值對(key value pairs)、SQLite Database則是Android SDK內建副檔名為.db 的資料庫。

所有應用程式資料的儲存位置都會在/data/data/PID之下。



圖：APP資料儲存位置(資料來源：實作研習簡報)

此外，這些儲存內容還可以利用Android KeyStore透過加密金鑰保護儲存內容，例如Android 4.3版所有應用程式都可以自行建立一組公私加密金鑰，可以用Android所提供公鑰API去加密儲存的資料，被加密的資料則可以用應用程式的私鑰進行解密，而這對公私鑰將透過使用者保護機制被儲存在Android KeyStore，例如PIN碼、密碼、圖形密碼、指紋等。大多數的行動裝置都有提供硬體支援的KeyStore，公私鑰只能在可信執行環境(Trusted Execution Environment, TEE)進行新建使用，作業系統無法直接使用，所以即使在已經root的設備，這些公私鑰還是會被加密保存，而無法被檢索。

另外，端對端加密(End-to-end encryption, E2EE)常被使用在伺服器端以及客戶端之間用來加密敏感資訊，攻擊者可以透過jadx-gui開啟並分析應用程式，找到加密的函式，就可以破解在用戶端的加密內容。但這些加密金鑰通常不會被直接以字串儲存，而是會在runtime或是使用Android KeyStore產生，所以在實務上並不容易達成。

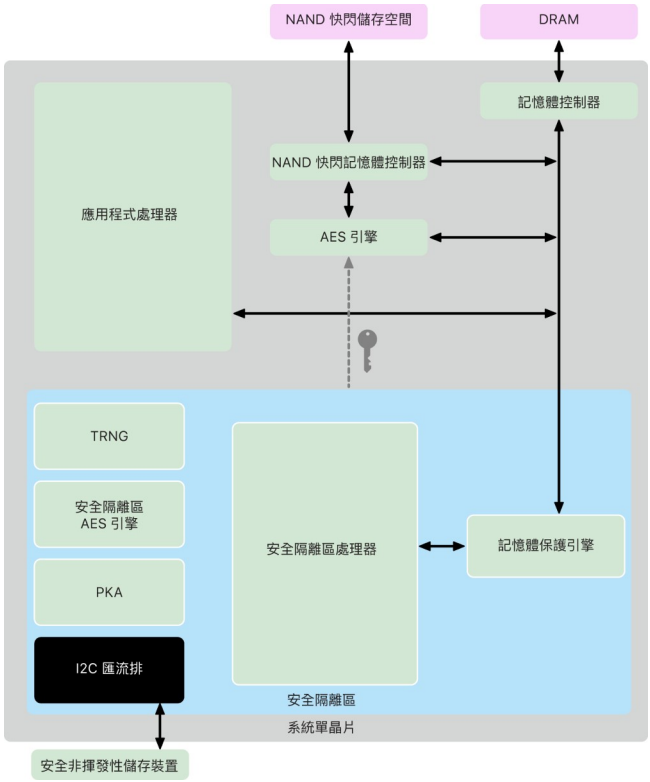
8. iOS平台及安全架構簡介

iOS是一種行動裝置作業系統，只使用在Apple自家的硬體設備上，如：iPhone、iPod Touch、Apple Watch、Apple TV、iPad等，另外，iOS的家族還包括：WatchOS、tvOS、iPadOS等。iOS Apps只能從封閉平台Apple App Store下載安裝，下載下來的應用程式都經過加密，在運行的時候才會被載入到設備上。比起Android，iOS擁有更高的同質性，96%的iPhone都是使用iOS 15或14。開發者可以預設所有軟硬體安全措施都可以使用，例如：Touch/Face ID、Secure Enclave(SE)、KeyChain。iOS Apps只能在macOS上進行，而且皆需要使用iOS SDK的Apple整合式開發環境Xcode，並使用Objective-C或是Swift語言進行開發。

iOS的安全架構包含：硬體安全(安全隔離區)、原始碼簽章及沙盒。

硬體安全(安全隔離區)是一個整合在SoC上的元件，包含一個以硬體為基礎的金

鑰管理，提供在設備休眠時加密資料以及生物辨識功能，它在設備被Jailbreak的情況下依舊可以維持著加密作業的完整性。



圖：iOS硬體架構圖(資料來源：蘋果公司官方網站 https://support.apple.com/zh-tw/guide/security/sec59b0b31ff/web)

原始碼簽章是一種Apple應用程式的簽章方式，因為iOS上只能運行由Apple認可的原始碼，在應用程式安裝或是上架Apple Store前都需要在Apple Developer Program或Enterprise Program註冊並進行Apple簽章，這個機制可以同時強迫用戶使用最新的iOS版本，這也是iOS設備版本差異較小的原因之一。

沙盒是使用在第三方iOS應用程式上的機制，讓第三方iOS應用程式在檔案系統中與其他應用程式獨立，可以避免各個應用程式間相互讀取或是修改。每個應用程式沙盒都有一個獨一無二的UUID，並使用非特權的使用者權限。

與Android平台類似，iOS平台可以依據不同版本的iOS進行不同方法的Jailbreak，Jailbreak可以讓部分安全架構失效。在設備被Jailbreak的情況下，iOS的安全架構只剩硬體安全(安全隔離區)會生效，原始碼簽章以及沙盒都會失去功能，因此就可以在iOS上安裝未被簽章的應用程式。已被Jailbreak的設備可以取得包含檔

案系統的完整權限並且運行未被Apple簽章的原始碼，也可以進行不被限制的除錯作業以及動態分析。

自iOS 15版本就開始使用封裝過的根檔案系統，通常Jailbreak都是透過修改根檔案系統來進行，但在iOS 15版本如果對根檔案系統進行修改，就會造成系統不停重新啟動。然而目前iOS 15版本已經可以進行Jailbreak了，而iOS 16版本則還沒有公開的方法可以進行Jailbreak。

被Jailbreak的設備可以使用軟體套件管理系統Cydia來對軟體套件進行管理，就如同被Jailbreak版本的App Store，這個系統提供管理、對設備進行微調以及安裝應用程式或主題。

9. iOS 應用程式架構介紹

IPA(IOS APP Store Package)為IOS APP的檔案類型，是一種壓縮格式，每個IPA裡都包含應用程式的binary檔，只可以被安裝在iOS設備上，從Apple App Store上下載的應用程式都會經過”Fairplay encryption”加密。另外，在ARM架構上編譯的IPA不能被安裝在運行x86架構的iPhone模擬器上。

IPA裡面包含可以執行的binary檔、動態連結函式庫、應用程式的描述檔以及開發商檔案。binary檔使用Mach-o檔案格式，主要分為Header、Load以及Data，一旦安裝後，就會被儲存在/var/mobile/Applications/{GUID}下。

10. 攔截透過行動裝置應用程式架構送出的流量

TLS會透過認證憑證簽署者、加密傳輸以及透過計算雜湊值等方式對傳輸進行保護。在iOS系統中存在一個Truststore，裡面保存根憑證，可以利用根憑證的私鑰去簽署中繼憑證，然後再使用中繼憑證去簽署使用者SSL憑證。

為了攔截流量，可以重新導向http(s)的流量到攔截流量用的VM，並且監視應用程式以及API之間的request，開發者可以透過修改requests來對伺服器端進行除錯。

但部分行動裝置框架，例如Google’s Flutter或是Microsoft’s Xamarin可能會忽略系統代理，所以如果在WiFi設定代理可能會沒有作用，這時可以透過ARP Poisoning或是DNA Spoofing(NoPE Proxy)來解決。在burp擴充裡就可以開啟設定DNS

Server及Non-HTTP MiTM攔截代理。

除此之外如果在已經被jailbreak的設備上，也可以透過設定/etc/hosts檔案中的domain進行中間人攻擊，burp則需要開啟隱形代理模式並聽取80及443埠。另外也可以透過設定Acceess Point來攔截流量，將所有流量重新導向burp。

技術	設備是否需要Jailbreak	困難程度	被攔截的流量
ARP Poisoning	否	中	所有來自iOS設備的流量
DNS Spoofing	否	易	所有基於DNS的流量
/etc/hosts	是	易	列在hosts的網域
Access Point	否	難	所有來自iOS設備的流量

表：攔截流量比較表

假設使用XMPP或其他non-HTTP埠或是非80或443埠，例如使用XMPP的內嵌chat functions或為了減少http headers的成本而使用原始TCP的應用程式等，可以利用rvictl及wireshark去進行所有流量攔截。

11. 用Frida進行iOS APP的動態檢視

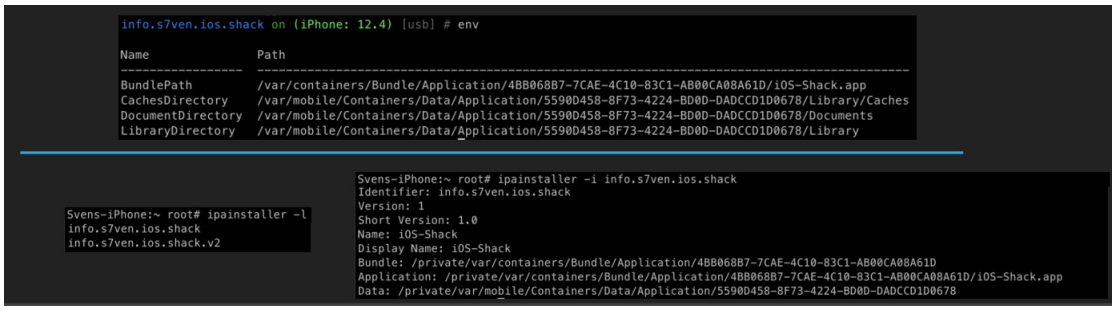
OBJC API是一種介於Frida scripts及 Objective-C runtime間用來對iOS進行動態檢測的介面，類似在Android系統的Java API，其常見的指令為ObjC.Classes及ObjC.Object。它可以將應用程式的執行流程重新導向Frida script，然後再導回原始應用程式，來對執行中的程序進行修改。可以在透過指令列出應用程式所有class後，從這些class中找尋需要修改的method、參數與回傳型態。就可以利用Frida script去重新改寫應用程式中的method。

12. 分析iOS應用程式的資料儲存

因為洩漏Cookie、JWT、加密金鑰以及使用在離線的敏感資料都可能會造成社交工程或是帳號被盜的可能性，因此一般來說都建議將敏感資料儲存在本地端。

iOS系統資料都儲存在/Application下，而使用者安裝的應用程式則放在/var/containers/Bundle/Application下。當安裝應用程式時，都會自系統取得一個128-bit的UUID作為資料夾名稱，並可以使用ipainstaller或是objection來列出所有

應用程式的資料夾資訊。



圖：iOS APP資料儲存資訊(資料來源：實作研習簡報)

Bundle資料夾裡包含以下內容：

(1)./var/containers/Bundle/Application/[Bundle-UUID]/[Application].app 包含APP所有的靜態資料。

(2)./var/containers/Bundle/Application/[Bundle-UUID]/[Application].app/App 包含可執行的binary檔。

(3)./var/containers/Bundle/Application/[Bundle-UUID]/[Applicaion].app/Info.plist 包含APP的設定檔。

(4).其中128 bit Bundle-UUID為一個36個亂數字母組成的獨一無二字串。

Data資料夾包含以下內容：(儲存應用程式所產生的資料)

(1)./var/mobile/Containers/Data/Application/[Data-UUID]/Documents 包含所有使用者產生的資料。

(2)./var/mobile/Containers/Data/Application/[Data-UUID]/Library 儲存非特定使用者的資料，快取、偏好設定、cookie、plist設定檔等。

(3)./var/mobile/Containers/Data/Application/[Data-UUID]/tmp 儲存無須在應用程式使用的暫存資料。

(4).其中128-bit Data-UUID為一個36個亂數字母組成的獨一無二字串。

iOS常用的資料儲存技術有Plist、File System、CoreData、Database，預設皆為非加密資料。Plist是一種儲存應用程式資料的格式，可以使用文字編輯器開啟，類似XML檔。而NSUserDefaults為使用者預設資料庫的介面，用來儲存應用程式運行中的key-value資料。CoreData是一種持續性框架，通常使用SQLite來儲存及取回結構化資

料。第三方資料庫可以是關聯資料庫(SQLite)或是以key-value方式(YapDatabase)儲存，有Firebase、Couchbase及Realm等多種第三方資料庫。

本地儲存可以透過Xcode(non-Jailbroken)、SSH(Jailbroken)、Frida(jailbroken and non-jailbroken)等幾種方法進行分析。

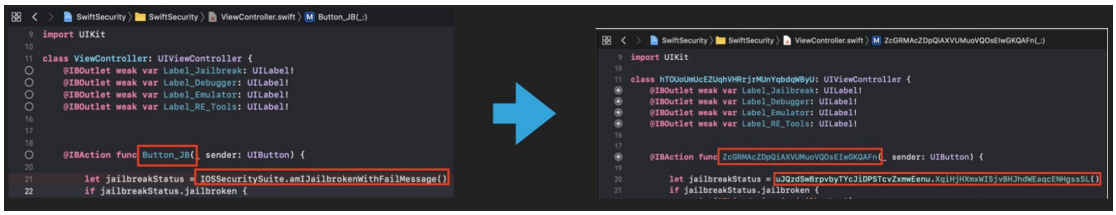
Xcode可以開啟被自己簽章的應用程式，直接利用UI介面開啟應用程式後，選擇”Download Container”，下載下來的檔案副檔名為xcappdata，可以直接改成zip解壓縮，或是使用”Show Package Contents”來開啟。如果是被Jailbreak的設備，則可以利用SSH透過Cydia安裝ipainstaller後，透過CLI Command去開啟並檢視已安裝的應用程式清單。

13. iOS的應用程式安全控制措施

確認使否有使用逆向工程的工具(Frida) ，可以藉由確認應用程式的完整性來確認應用程式是否有被frida-gadget函式庫修改、確認環境是否有被加工來確認Frida-server是否有在設備上運行、確認開啟的TCP埠是否有使用Frida預設埠27042、確認埠對D-Bus Auth的回應來確認Frida server是否存在以及掃描正在運行的記憶體確認是否有存在已知Frida的函式庫(例如LIBFRIDA有使用在所有版本的frida-gadget及frida-agent)。

確認設備是否被Jailbreak：偵測Jailbreak可以讓運行應用程式在被Jailbreak的設備上更加困難，進一步阻止部分逆向工程會使用的工具及技術。偵測Jailbreak的方法常見有確認檔案副檔名、確認埠處理程序及檔案權限確認，通常在偵測到Jailbreak行為後，會通知使用者正在使用一個非安全的環境或是停止應用程式運行。

代碼混淆：代碼混淆是一種將原始碼轉換成較難以理解及拆解的方法，通常用在軟體保護方案中，例如Swift-Shield會在編譯之前將原始碼中的classes、methods及fields名稱轉換成亂數值。



圖：代碼混淆(資料來源：實作研習簡報)

確認應用軟體是否被重新包裝(Objection/Frida-Gadget)：可以從比較Bundle ID、比較mobileprovision檔案的hash值或比較執行檔的hash值進行。

確認debugger是否被使用：Ptrace是一個system call常使用在tracing以及debugging上，可以使用ptrace(PT_DENY_ATTACH,0,0,0)來拒絕ptrace依附，告訴作業系統不希望被追蹤或是除錯。SYSCTL則可以透過追蹤kinfo.kp_proc.p_flag去確認是否有被debug過。

Anti-debugging及Jailbreak偵測等資安控制可能會大幅度降低測試效率，為了增加測試效率，可以建置release及debug(停用資安控制)兩種版本的應用程式來進行測試，雖然可能不適用於所有測試腳本，但這是最有效率的測試方式。

14. 探討逆向工程

iOS 應用程式是使用Object-C或Swift撰寫，所以需要被拆解，和Android的反編譯非常不同，在沒有被原始碼混淆的情況下，Android應用程式的反編譯可以取回大部分的原始碼，但iOS應用程式的拆解只能取回ARM CPU的指令集。為了進行IPA的逆向工程，可使用IDA Pro(可使用在主流平台)、Hopper(使用在macOS及Linux)及Ghidra(開放原始碼，可使用在主流平台)。

Ghidra及Hopper等拆解程式可以去修改可執行程式(Mach-O)，透過二進位修補功能修改程式邏輯，變更回傳值(true to false)或讓邏輯無效化來繞過IPA的Jailbreak偵測機制，再重新包裝IPA後Jailbreak偵測機制就會失效。

根據應用程式開發使用的語言，可以用不同方式進行逆向工程，例如Objective-C可以在run time進行method的調用，所以能夠簡單的對函式進行掛勾，但在Swift中並沒有以相同的方式實作這個功能，造成Swift在技術上比Objective-C更加難以執行，需先將函式名稱還原之後，才能進行掛勾。

三、GitHub Actions: Vulnerabilities, Attacks, and Counter-measures(主講人：Magno Logan)

GitHub是一個大型的Git Server，開發人員可以使用GitHub來創建和共享代碼庫，進行版本控制、分支管理、問題追蹤、代碼審查、整合等多種開發活動；GitHub也提供了廣泛的開源代碼庫，開發者可以參考和貢獻這些代碼庫，這樣可以加速程式開發進程，提高開發效率。

GitHub Actions Marketplace是一個包含多個自動化工作流的集合，這些自動化工作流是由第三方提供，並且可以免費使用，能幫助使用者更輕鬆地設置和執行自己的GitHub Actions。使用者可以在GitHub Actions Marketplace中找到各種不同的自動化工作流，例如自動部署、自動測試、自動建立和自動發布等，並從中選擇一個適合自己的自動化工作流，然後按照指示進行設置和使用。

雖然GitHub Actions Marketplace是一個便捷的平台，可以幫助使用者更輕鬆地發現、使用和分享各種自動化工作流，提高他們的開發效率和產品質量，但是在這個研討會議中主講人提出三點GitHub Actions可能存在的資安風險。第一種可能存在的資安風險為Pull Requests from forked repos。GitHub上的Action fork機制可以讓使用者將其他使用者的Action代碼庫複製到自己的帳戶中，並對其進行修改和使用，從而節省了自己開發Action的時間和精力。fork其他使用者的Action代碼庫後，如果進行修改，這些變更不會自動同步到原始代碼庫。其他使用者可以自由地在fork的代碼庫中進行更改和修改，而不會影響原始代碼庫的狀態。

如果想要將修改後的Action與原始代碼庫進行同步，可以通過提交pull request的方式將修改的Action推送回原始代碼庫，這樣原始代碼庫的擁有者就可以檢查這個變更，如果他們認為變更有益，就可以將其合併到原始代碼庫中。

也因為這樣的機制，讓攻擊者可以利用具損害性的 pull requests (PRs) 或提交請求，以進行攻擊和破壞。攻擊者可能會提交帶有惡意程式碼的 PRs 或提交請求，這些程式碼可能會被包含在自動化的 GitHub Actions 工作流程中執行，從而獲得對

代碼庫和系統的控制權。攻擊者也可能會使用惡意的 PRs 或提交請求來引誘原始代碼庫擁有者執行敏感操作，例如解密密鑰、授予權限或揭示敏感信息，從而危及代碼庫和系統的安全。

第二種可能存在的資安風險為Command Injection via untrusted inputs。GitHub 的 Command Injection via untrusted inputs 是一種常見的安全漏洞，這種漏洞可以允許攻擊者注入有害的命令來攻擊和破壞 GitHub 存儲庫或系統。這種漏洞通常出現在使用不受信任的輸入來執行命令的場景中，例如在 GitHub Actions 工作流程中使用來自 pull requests 或其他未經驗證的來源的輸入。

攻擊者可能會通過提交帶有惡意代碼的 PR 或提交請求，例如執行特定命令或修改內容，從而觸發該漏洞。當您在 GitHub Actions 工作流程中使用這些輸入時，攻擊者可以注入特定的命令或代碼，從而獲得對您的存儲庫或系統的控制權。

第三種可能存在的資安風險Malicious Github Actions from the Marketplace。由於GitHub Actions Marketplace可自由創建和共享的特性，攻擊者也可以藉由創建惡意的GitHub Acions，將帶有後門或是其他惡意行為的程式來偽裝成合法的操作，以繞過安全管控措施來獲得機敏資訊或是系統控制權限。

要防止以上可能存在的資安風險，主講人建議只使用信任的開發人員所建立的GitHub Actions、確保自己的GitHub Actions已建立以最小需求權限為原則的權限設定、不執行forked repos上的Actions或是在執行前要檢視其安全性及善加保護自己的機敏資訊並驗證無法信任的輸入資料。

四、JavaScript Realms - The Blank Spot In Web Application Runtime Security(主講人：Gal Weizman)

隨著越來越多的獨立開發者出現，供應鏈攻擊越來越常見，有許多行業引入各種解決方案，但在瀏覽器執行環境的保護卻還是很缺乏，最糟的是在Realms(瀏覽器內嵌架構)下，很容易被攻擊卻很難去做保護。

以前的網站(website)通常使用簡單的靜態資料，少有暴露原始碼的問題，也很

少需要進行變更，對於第三方原始碼的依賴性較小。現在的網站(web apps)包含動態應用程式的功能，較以往更為複雜，但優點是有自動建置部屬網站等功能，且比起開發者自行撰寫，通常會大量使用外部第三方的原始碼，省去開發成本，並加入廣告或是付費等元素增加收益。但是另一方面也可能會降低網站的安全性以及能見度，例如會在自動部署的過程中暴露原始碼、因為快速的開發週期而降低安全性或因對第三方原始碼的依賴而降低能見度。

能見度系指是否能看到系統內部運作流程，所以能見度越低越難以維持應用程式的安全性，另外高頻率的修改以及使用不明來源的第三方原始碼也很容易造成安全性風險。而近期也出現越來越多的第三方服務可以提高網站(web apps)的安全性以及能見度，例如透過監控敏感指令及異常行為、禁止應用程式和惡意內容通訊及存取敏感資料來增加安全性或是透過擷取錯誤訊息及日誌、追蹤使用者介面使用流程以及監控效能的改變來增加能見度。

第三方服務通常是透過使用者加入他們的腳本或是用覆寫的方式進行，這種方式需要允許這些第三方服務取得應用程式運作環境的完全控制。雖然這些服務可以監控錯誤訊息並針對敏感資訊進行防護措施，但卻缺少了針對JavaScript Realms的能見度控管。

Realm是當JavaScript運作時就會存在的生態系統，裡面的元素包含全域執行環境及可以取得平台物件存取權限的全域物件。每個網站(web apps)會預設在同一個主Realm執行，可以使用”top”來進行存取，子Realm則預設會在同一個來源，相同來源間的子Realm可以互相進行存取編輯，但子Realm也可以設定為不同來源，這樣兩個不同來源的子Realm就會無法相互存取編輯，這也是主要的安全防禦手段，另外，Realm也可以被設定為工作用的子Realm，就會被視為是和所有其他子Realm不同來源的Realm以達到安全性的效果。

在這樣的情況下Realm在執行環境依然存在著一些盲點，例如：Realm可以在第三方服務下執行，透過主Realm去取得子Realm的資料，或是Realm可以在不被第三方服

務攔截的情況下影響主Realm，直接改變主Realm的資料。

為了解決盲點，必須要找一個簡單且自動化的方式同步執行在所有可能產生的Realm上，例如：Snow JS。Snow JS是一個簡單的API，可以以同步的方式執行，並且支援Safari、Firefox或是以Chromium為基礎開發的瀏覽器，另外也支援主流網站(web apps)Facebook、Instagram、Google、TikTok、Twitter、Youtube及Netflix等，且已被市面上多次測試。

Snow JS可以影響所有新建立的Realm，包括在DOM插入新的節點或HTML、開啟新的頁籤、重新導向、巢式Realm等，目前Snow JS已經被實用在MetaMask錢包瀏覽器所使用來防護供應鏈的工具LavaMoat上。未來Snow JS也會持續增進它的安全性並且調整能見度等功能。

五、[T]OTPs are not as secure as you might believe(主講人：Santiago Kantorowicz)

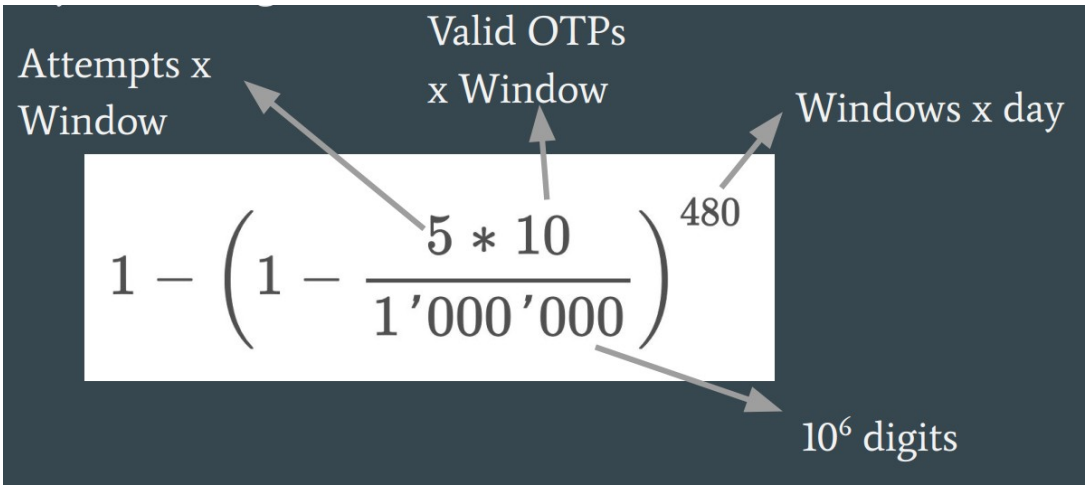
OTP(One-Time Password，一次性密碼) 是指系統自動產生只能使用一次的密碼，通常用於進行安全認證時驗證用戶身份。當使用OTP進行身份驗證時，系統會生成一個密碼，並將其發送到用戶的設備上，用戶可以使用此密碼來完成身份驗證過程。由於OTP只能在一次身份驗證中使用，因此它提供了更高的安全性和保密性，即使密碼被洩露，攻擊者也無法在未來使用它。

OTP作為身份驗證的一種方式，雖然可以提高帳戶的安全性，但若是在設計或是實作上發生錯誤，如太長的失效時間、沒有限制的嘗試次數、OTP的長度太短、[T]OTP的重複使用率過高、並不是以雙因子認證的方式使用、不安全的亂數生成等，仍會讓OTP變得不安全，所以在設計和實作上須要避免這些錯誤，才能減少OTP可能出現的漏洞和問題。

針對單一OTP攻擊的機率，以4位元的OTP來說，嘗試1次命中的機率為0.01%，若機制設計嘗試的上限為5次的話，則有0.05%的命中機率。假設一個4位元的OTP是藉由簡訊或電子郵件來傳送，有效的時間(Window)為3分鐘，可以嘗試的次數為5次，可以

要求重送的次數為10次，嘗試命中此單一OTP的機率為5*10/10000=0.5%。假如我們重複上述針對單一OTP攻擊的模式持續進行，1小時內我們就會有20個可供攻擊的有效時間，1天就會有480個可供攻擊的有效時間，而嘗試命中此單一OTP的機率為1-(1-0.05%)480約等於91%。

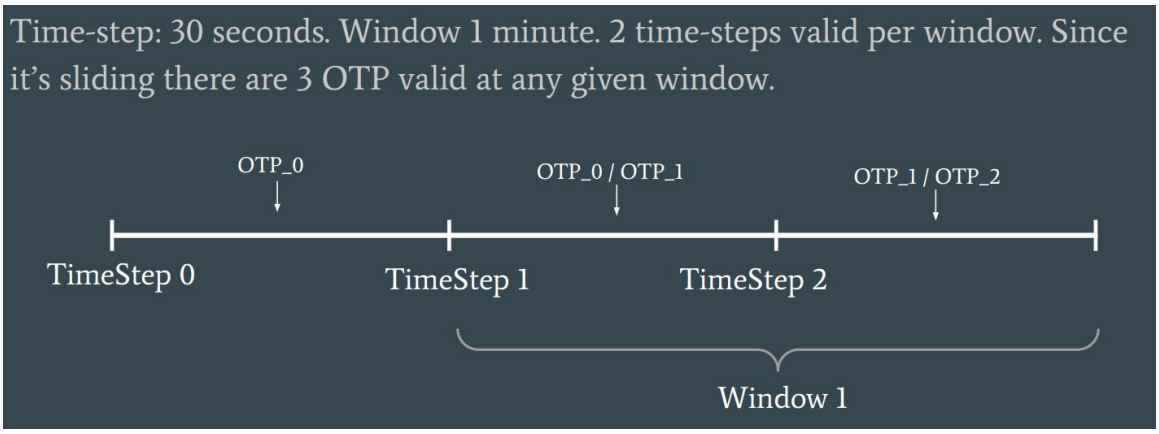
當我們把OTP的位元數增加到6 位元時，1天之內嘗試命中OTP的機率就會如下圖所示：



圖：OTP為6位元時於1天之內嘗試命中的機率(資料來源：研討會議簡報)

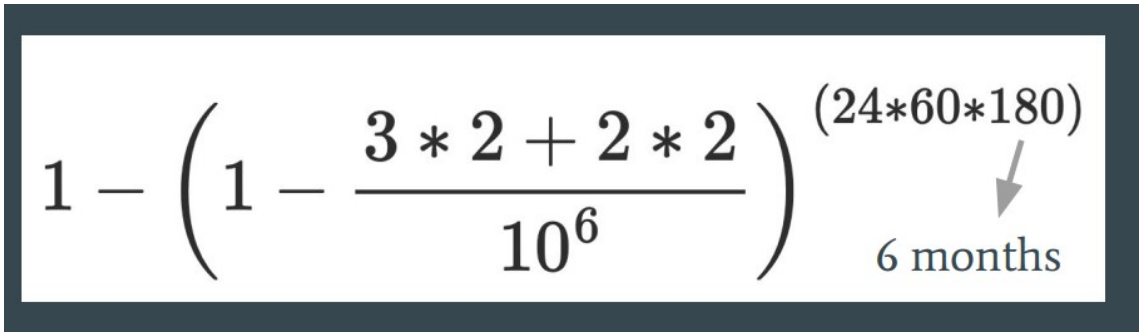
1天之內嘗試命中OTP的機率為2.3%，100天嘗試命中的機率為91%，而在6個月內嘗試命中的機率為99%，相關機率可藉由GitHub上的OTP暴力破解模擬器計算(GitHub - kantos/otp-brute-force-simulator: Simulates in real time attacks to OTPs and gives you the probability of success)。

若改成TOTP(Time-based One-Time Password)，假設有效區間(Time-step)為30秒，有效時間為1分鐘，1段有效時間內有2段有效區間，所以在1段有效時間內會有3個有效的OTP，如下圖示：



圖：TOTP示意圖(資料來源：研討會議簡報)

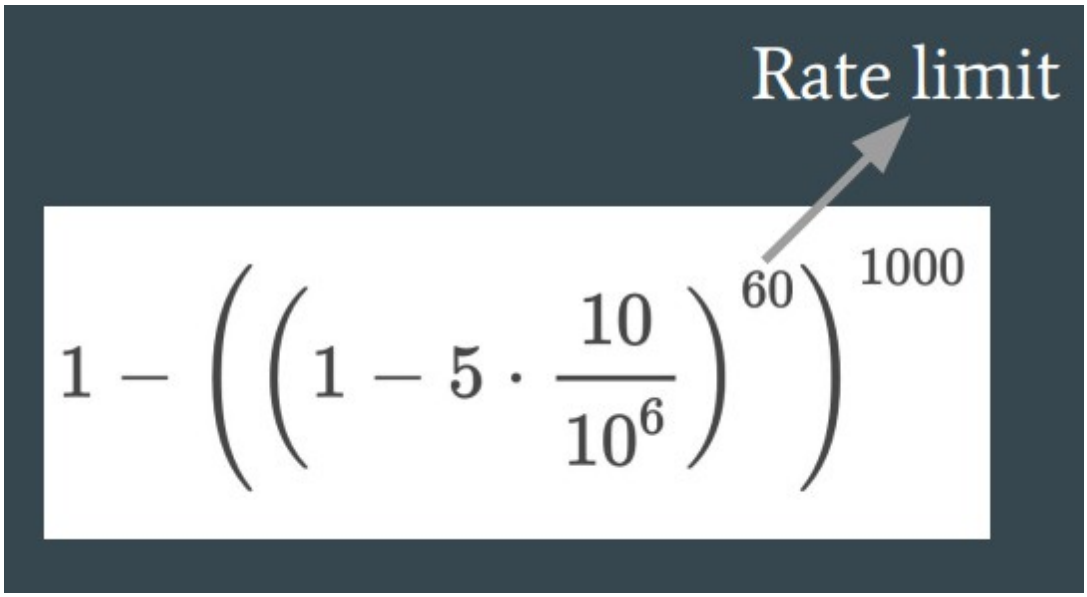
經計算後TOTP在6個月之內嘗試命中的機率約為93%，機率計算請參考下圖：



圖：TOTP在6個月內嘗試命中機率(資料來源：研討會議簡報)

之前的情境都是假設只有1個人發動攻擊，但是當有100個人同時發動攻擊時，以上述單一OTP攻擊的例子，若OTP有6位元，則1天內嘗試命中的機率為91%。若我們將OTP位元數提高到8位元後，在100天內嘗試命中的機率為91%，若有1000人同時發動攻擊時，在15天內嘗試命中的機率就有97%。

針對以上例子，若要強化OTP的強度，以限制嘗試次數的方式，如以1個月嘗試60次為限(1天只能嘗試2次)，1個月內有1000人嘗試命中的機率為95%，機率計算請參考下圖：



圖：1個月嘗試60次為限且有1000人嘗試命中的機率(資料來源：研討會議簡報)

若將限制嘗試次數再加上OTP以英數組合，則在10,000人同時攻擊的情況下，在6個月內嘗試命中的機率僅0.47%。

OTP是目前常用作雙因子認證的身分驗證方式之一，雖然其一次性身份驗證的特性提供了更高的安全性和保密性，但也因此讓人忽略了部分潛在的漏洞；因此，在強化OTP方面，應針對機制設計上多加考量，像是增加嘗試次數的限制、不揭露太詳細的錯誤訊息、用更好的傳輸方式、增加驗證的方式或是寄發告警信給使用者，以降低OTP遭嘗試命中的風險。

六、Don't let bug bounty kill your appsec posture(主講人：Zohar Shchar)

除錯賞金在過去幾年成為了業界被關注的項目，有越來越多的組織提供經費資源執行多項相關計畫。但是除錯賞金另一方面也可能會暴露應用程式的安全策略，當除錯賞金越來越熱門，比起花大量時間精力在研究特定議題的深入技術，越來越多的研究者將關注放在可以被自動找尋的常見問題上。他們或許可以簡單的找到像是子域名劫持(subdomain takeovers)或是XSS等衝擊較小的議題，但卻不再關注需要深入技術了解的高風險議題。這個問題或許可以透過定義除錯賞金範圍或是教育研究者來解決，但可能會讓你比起花時間去精進除錯賞金計畫，更希望把時間花在解決實際的問

題上。

Wix.com是SaaS網站建置者，有超過2千5百名以上的開發者、5千個以上的微型服務、2萬個節點以及2億5千萬以上名使用者，每年花費在除錯賞金計畫上的金額超過25萬美金。

除錯賞金計畫常常被外界誤會，認為可以用來取代滲透測試。最初的除錯賞金其實並非是為了安全控制，而只是為了讓研究者告訴外界無論投資多少資源在安全控制上，仍舊會有人找到弱點，所以最好提供外界一個可以通知主辦方弱點的管道，並鼓勵外界當發現弱點時可以即時回報。

在除錯提供賞金之前，對於那些花費時間精力找尋弱點的人來說，他們只是有興趣去研究這些技術，想了解這些錯誤發生的原因。現今雖然除錯賞金越來越流行，有越來越多經費被投入在除錯賞金上，有許多人甚至將除錯賞金當作他們主要的收入來源，但卻漸漸失去了原本找尋弱點的初衷。當除錯賞金成為收益之後，找尋弱點的動機也因此而改變，比起弱點所造成的衝擊他們更在乎賞金的多寡。另一方面，不只是除錯的人因除錯賞金計畫改變作為，一些公司在進行除錯賞金計畫的時候，也漸漸忘記了除錯賞金的背景，將除錯賞金變成在其他安全控制之前的首要目標。

如果你向Google通報一個高風險的議題，你可能可以獲得1萬美金以上的賞金，但這需要具備一定的技術並且需要花費大量時間在上面，然而你也有可能在花費了大量時間後卻沒有找到任何弱點。可是如果你向Google通報一個子域名劫持(subdomain takeovers)弱點，你雖然只能收到1百美金的賞金，但這個弱點可以透過寫腳本自動化去測試所有目標域名，這也是現在多數人的做法。也因此越來越少人投入時間進行深入研究，反之卻有越來越多人投入時間研究開發工具進行範圍性測試。

有一間公司很長一段時間都沒有在除錯賞金計畫中收到SQL注入攻擊的通報，一開始被認為是因為不存在SQL注入議題，所以沒有被找出來，但當這間公司將SQL注入議題的賞金拉高之後一周，就有3件有效的SQL注入議題被通報，所以事實上這間公司是存在SQL注入議題的，只是因為給予的動機不夠強烈，所以沒被通報。

除錯賞金計畫可能無法像我們想像的那樣有效，在上面的例子裡是因為他們知道自己有SQL注入議題，所以拉高SQL注入議題的獎金來提高被找尋出來的動機，但除錯賞金計畫目的是希望能夠找到當前所不知道的錯誤而不是已知的錯誤，所以除錯賞金計畫可能只能找出重複發生的議題，而無法找出真正的風險

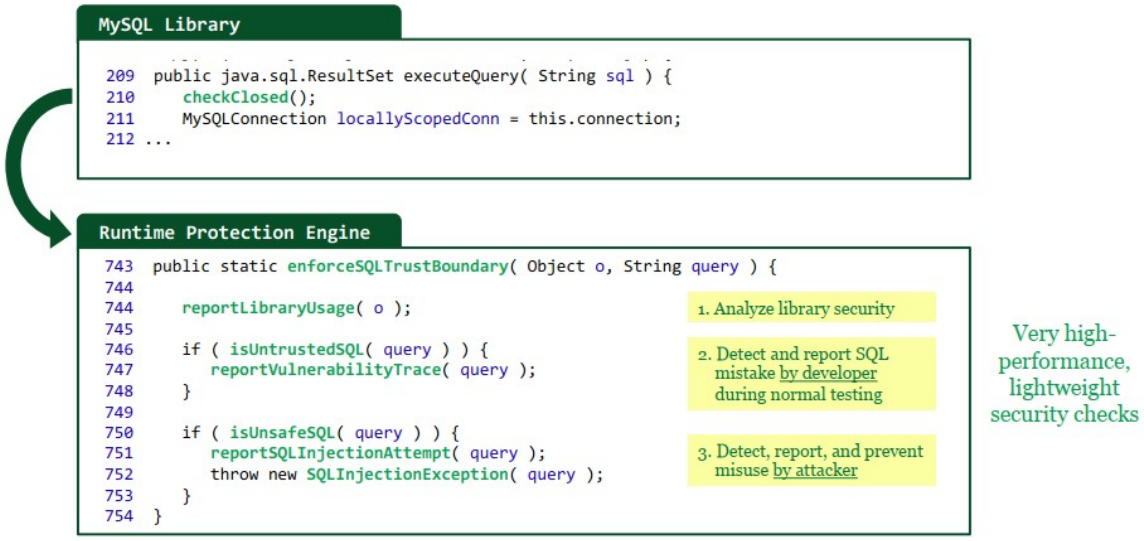
除錯賞金是有必要存在的，但這只是其中一種除錯的方法，它可以提供客觀性，可是並不是最終的目標，因此還是需要持續針對計畫進行滲透測試、威脅模式分析或是內部研究。

七、Trusting Software - Runtime Protection Is the Third Alternative(主講人：Jeff Williams)

要達成可信賴軟體的方法目前常見的有兩種，一種是人員，另一種是邊界防禦。在人員方面可透過開發人員安全程式設計訓練、漏洞測試、威脅建模、安全架構等方式來使開發人員創建安全的軟體；邊界防禦方面可以透過相關網路防禦設備(如網路防火牆、入侵防禦系統等)建立邊界防禦線，藉此監控網絡流量、阻擋未經授權的訪問及利用漏洞的攻擊來保護內部網路、系統和應用程式免受外部攻擊。

本次研究會議的主講人認為以前述的兩種方式仍不足以達成可信賴軟體，應該要加入如同ASLR(Address Space Layout Randomization)和DEP (Data Execution Prevention)的機制一樣，能夠在程式運行期間所執行的防禦機制，也就是這個研討會議所強調的運行時防護(Runtime Protection)，來達成可信賴的軟體；而依據Forrester的報告顯示，65%的公司已採用運行時防護，而17%的公司正在規劃採用。

運行時防護是一種在軟體運行期間對其進行保護的技術，可以協助減少軟體運行時可能出現的安全漏洞和攻擊，如不安全的反序列化編譯(Unsafe Deserialization)、代碼注入(SQL Injection)、表達式語言注入(Expression Language Injection)、XML外部實體攻擊(XML eXternal Entity)、類別載入器修改攻擊(Classloader Manipulation)等。運行時防護最重要的是識別出信任邊界，將防禦措施建置在信任邊界上，以代碼注入為例，如下圖所示：



圖：將代碼注入防禦措施建置在信任邊界上(資料來源：研討會議簡報)

當程式已經運行到這個階段時表示如果當中有代碼注入攻擊，在這個階段人員或是邊界防禦的手法都已經無法阻止這個攻擊，但是如果我們在程式碼第209行及210行之間，加入enforceSQLTrustBoundary這個輕量化的檢測函式，就可以在代碼注入攻擊進到資料庫前進行把關。

運行時防護在主講人的案例分析中，可以防護95%的攻擊，也因此在案例分析中的公司將運行時防護列為每個應用程式所必備的防護機制。由於運行時防護可以應用於各種不同的程式語言和平台、不需要改變程式開發流程或架構且幾乎不會對性能產生影響的特性，在傳統的人員於開發階段及測試階段所進行漏洞測試和修復無法發現，或是邊界防禦網路監控阻擋機制有漏洞時，可以做為軟體安全的最後一道防線。

八、Credential Sharing as a Service: the Dark Side of No Code(主講人：Michael Bargury)

低程式碼應用程式已廣泛被使用在企業中，調查中顯示許多企業應用程式使用IT以外缺乏資安實作的手段建置，攻擊者們也找出了各種手段去攻擊這些平台，例如讓使用者簡單點擊一個按鍵就盜走使用者的帳號、在沒有網路流量的情況下升級帳號權限、留下無法被追蹤的後門或是自動外流資料等。

企業中使用的低程式碼應用程式數量在近年來呈現指數型成長，但其實過去就

已經有很多類似的程式，例如Excel的巨集或是統計分析系統。低程式碼應用程式也可以簡單將各種功能進行自動化整合，例如如果使用者想要在一趟旅程中拿到一些補助，就可以使用應用程式來上傳收據並取得補助。目前已經有許多企業在使用低程式碼應用程式，像是微軟、safeforce、outsystems、Appian、Zapier及mendix等，所以低程式碼應用程式的資安問題已經成為所有人的問題。低程式碼應用程式通常使用SaaS方式來提供服務，所以難以被監測，也經常被低估資安對它的重要性。

當創建一個新的低程式碼應用程式，可以從雲端或是本地端選擇你需要的應用程式範本，當你選擇之後你需要登入你的帳號，這時你所選擇的範本可能就已經被分享給其他使用者使用，這也是低程式碼應用程式之所以會快速成長的原因，假設在使用範本時都需要取得授權，那就不可能有這樣的發展性。例如公司想要禁止員工將公務郵件自動轉發到個人郵件信箱，原本需要在郵件伺服器及客戶端做一些設定來阻止，但透過低程式碼應用程式已經建好的範本就可以很簡單的過濾郵件的地址。

在創建新的低程式碼應用程式時，一開始就會要求使用者授權一些必要存取權限，如果要求授權的是像是微軟這樣的大公司，大多使用者都會相信並給予授權，所以攻擊者只需要建立一個看起來很實用的誘餌範本，就可以引誘使用者去使用並建立連結，使用者並無法知道連結背後運作的內容可能包含一些惡意行為，只要這些連結是被建立在微軟的網域下，使用者就會無條件相信這些範本內容的安全性。

因為這些範本實際上是運作在微軟雲上，所以也很難被察覺或是監控，另外攻擊者可能還需要可以被遠端執行、能加載任意內容、可以維持存取權限、避免被偵測、避免被查出來源以及不留下日誌紀錄的方法來運作惡意行為。攻擊者可以利用HTTP連結來執行惡意行為，因為HTTP連結不需要被授權、可以被遠端執行、可以避免被偵測、避免被查出來源，所以只剩下加載任意內容以及不留下紀錄的問題需要被解決。這些問題攻擊者可以使用Power Automation這個低程式碼應用程式來解決，這個低程式碼應用程式可以新增、刪除、編輯任何資料，攻擊者的惡意連結只需要當作觸發的板機去自動執行低程式碼應用程式，他就可以在使用者建立流程的時候偷偷將惡

意內容加入流程裡，然後再刪除他，這樣就可以不留下任何紀錄並執行惡意內容。

低程式碼應用程式已經廣泛的運用在企業中，但卻被資安團隊低估，攻擊者可以在眾目睽睽下隱藏惡意行為或是從外面洩漏可能的錯誤設定，使用者在使用低程式碼應用程式時建議最小化連結的使用、檢視並且監測外部到資料庫的連線及環境設定並檢視分享給整個企業的連結內容以及OWASP LCNC Top 10弱點，來減少資安風險的發生。

九、Philosophizing security in a "mobile-first" world(主講人：Sergiy Yakymchuk)

在一份統計數據顯示，網路安全的投資逐年增加，但是因為網路犯罪的損失也是逐年增加，這就啟發我們去思考，是什麼樣的原因造成這樣的結果？會不會是因為工程師的偏見所產生的路燈效應，讓工程師針對網路安全僅針對已經有明確定義的問題去研擬解決的方式，而研擬出的解決方式也僅侷限在工程師容易解決且已經或是有類似技術運用的方案，忽略了從發生的現象去定義出問題，再從問題的各個面向去思考出解決問題的方法。

從技術的觀點上去思考往往會產生盲點，像是思考攻擊者將利用正在發展的量子電腦來進行密碼破解，或是利用機器學習來發展攻擊或是防禦的手段，但是我們發現大部分的攻擊者還是會使用傳統的社交工程手段，雖然傳統，但是時至今日卻還是依然有用。依據IBM的研究顯示，造成網路安全漏洞的原因有95%是因為人為錯誤所造成的，所以使用者是網路安全環節中最弱的一環，而這個部分就需要以認知及資訊安全的教育訓練來強化。

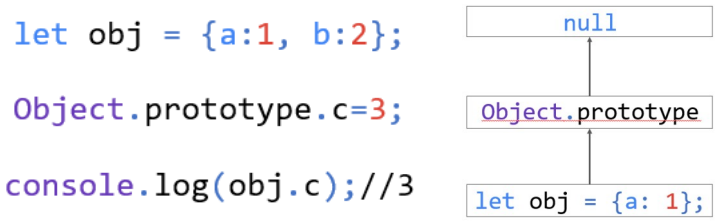
網路安全與使用者的自由及安全感環環相扣，有時候雖然我們部署了一系列的防護措施，比如說行為監控系統、機器學習的防護機制等，但這種措施有時候反而造成了一些使用者感覺上的不安全感或是限制了使用者的自由，有些人則否，所以安全感就必須在網路安全與使用者的自由之間取得一個平衡。在傳統的社會裡使用者會以契約的方式同意以放棄一些自由來換取安全的防護措施，而在數位世界了，如同

Cookie在GDPR所規定的一樣，也有經過使用者的同意而讓使用者以放棄一些自由來換取安全的防護措施或是服務。

現在人可能沒有意識到，我們在近幾年的時間，把資訊服務的提供從網路應用服務程式移轉到了手機軟體，但是因為網路應用服務與手機軟體的架構不同，在資訊安全上無法如此快速的移轉，導致手機軟體在資訊安全方面還有很大的進步空間，我們可以把手機軟體的世界想像成中古世紀的情況，iOS是一個王國，而Android則是一個聯邦，而每個手機軟體可以視為一座城堡，那我們要如何讓這個中古世界更安全？這涉及到了安全及安全感選擇上兩難的困境，以城堡的角度來看，當然是希望以安全感為主，讓人民覺得有安全感後，人民才會願意進駐這個城堡，城堡的人多了，相對來說稅收也增加了，才會有更多的錢去投資安全措施，至於要如何才能增加安全感，建議可以讓安全相關訊息可以隨時通知使用者並且讓使用者知道要怎麼做、 讓使用者可以簡單地回報問題、訓練使用者自我防護的能力及收集弱點資訊作為機器學習的分析資料。

十、Server Side Prototype Pollution(主講人：Gareth Heyes)

檢查伺服器端原型鏈污染的合法性是一件很困難的事，因為在伺服器上變更物件的原型鏈可能會引起DoS發生。



圖：一個a為1、b為2的物件(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

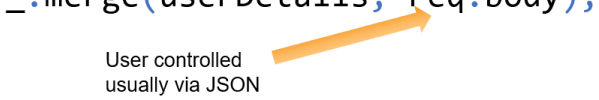
假設這裡有一個物件設定a為1、b為2，可以另外用物件的原型鏈去增加指定c為3，這時這個物件就會增加一個c為3的資料，這是因為幾乎所有物件都是繼承自原型

鏈，原型鏈又繼承自null所致。

```
let obj = {a: 1};
obj.__proto__ === Object.prototype
obj.hasOwnProperty('__proto__'); // false
let json = JSON.parse('{"__proto__":"WTF"}')
json.hasOwnProperty('__proto__'); // true!
```

圖：物件a為1且_proto_為Object.prototype(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

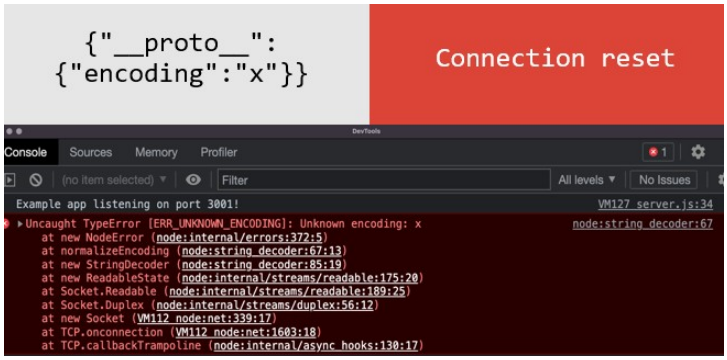
如果設一個物件a為1，然後將他的_proto_設定為Object.prototype，再利用函式檢測這個物件是否不是繼承物件的時候，就會顯示這個物件為繼承物件，但如果是用JSON.parse直接指定一個內容的時候，再次檢測這個物件就會變成非繼承物件。



圖：右邊由使用者輸入的內容和左邊的合併(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

像是merge這個函式是將右邊的內容和左邊的內容合併，通常右邊的內容都是透過JSON由使用者給予，攻擊者就可以利用這點將惡意內容合併進去。原型鏈汙染可以改變應用程式的設定並改變應用程式的行為，因此可能會導致遠端執行漏洞(RCE)。

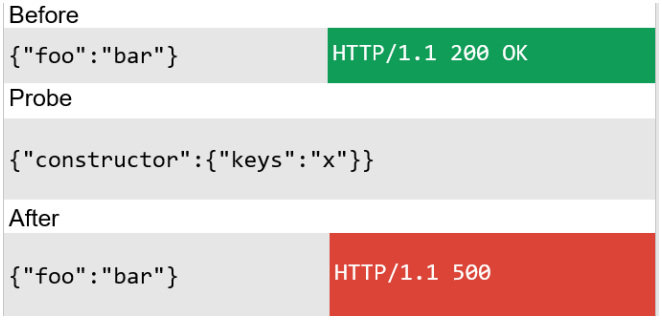
合法性測試是一件很困難的事，探究這些問題可能會造成DoS，但如果不造成DoS就很難確認伺服器的行為已經被改變，所以必須找到一個非破壞性的技術去改變伺服器行為來進行測試。



圖：透過原型鍊指定參數(資料來源：

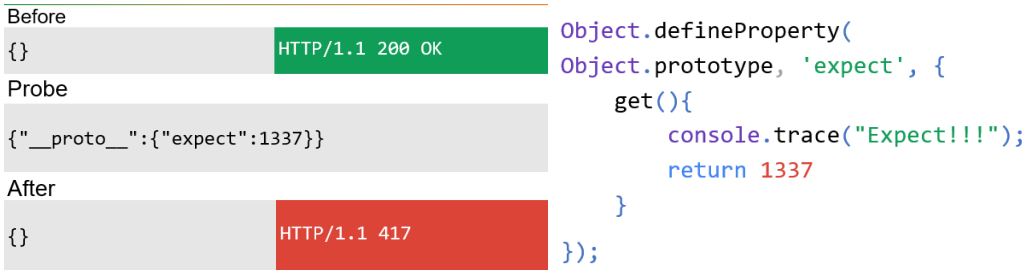
https://portswigger.net/research/server-side-prototype-pollution)

就如同前面所述，可以透過原型鏈去指定參數，改變伺服器的行為，因為x並非原本程式碼裡面有的參數，所以在沒有接取Exception的情況下就會造成DoS。



圖：用Constructor指定Object的Key物件(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

上面這個例子一開始送一個JSON參數收到伺服器回應正常，這時可以用Constructor這個物件去指定Object的keys物件為x，因為原本的keys函式被改成字串，所以這時重新傳送跟一開始一樣的JSON時就會出現錯誤。但這個例子其實並沒有涉及原型鏈汙染，因為他改變的是全域物件而非原型鏈物件。



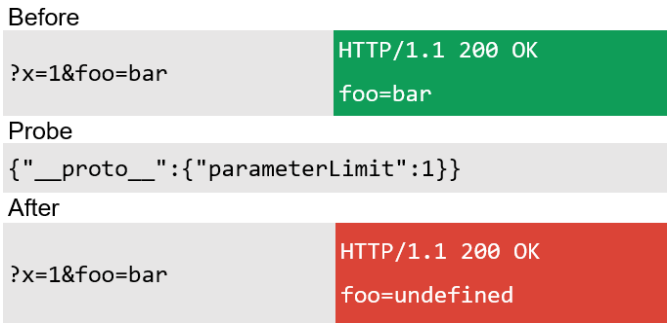
圖：將expect物件指定一個數字(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

上面這個例子一開始也送一串JSON收到伺服器正常回應，這時將expect這個物件指定為一個數字，重新傳送JSON時就會出現417 Exception錯誤。在Chrome的開發者模式中使用除錯用的console.trace()就可以找到這個expect參數是用在Node Http Server中，作為確定expect是否為undefined使用，原本的流程是如果有發生Exception這個參數才會是undefined，並進行後續Exception處理，但因為在沒有發生

Exception的情況下把expect改成undefined，其他為了處理後續Exception的參數卻沒有資料，造成最後進入Exception錯誤的判斷式中。

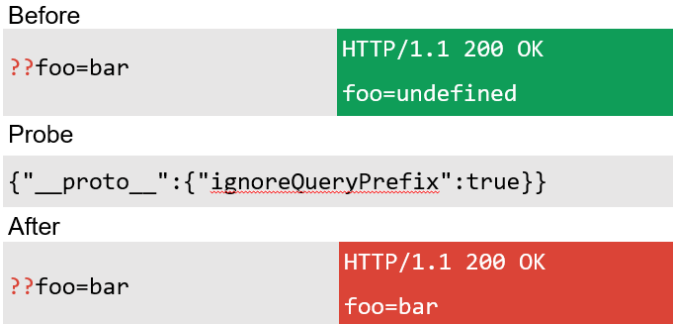
圖：將_body參數改成true(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

上面這個例子是將_body參數改成true、body改為<script>evil()，因為_body這個參數被改為true可以讓攻擊者跳過body參數內容的檢查，進而將body的內容直接加進伺服器回應的內容中。這個例子就是一個可以達成測試的目的技術，在不使伺服器DoS的情況下改變伺服器的行為，並且是可以被復原的修改。



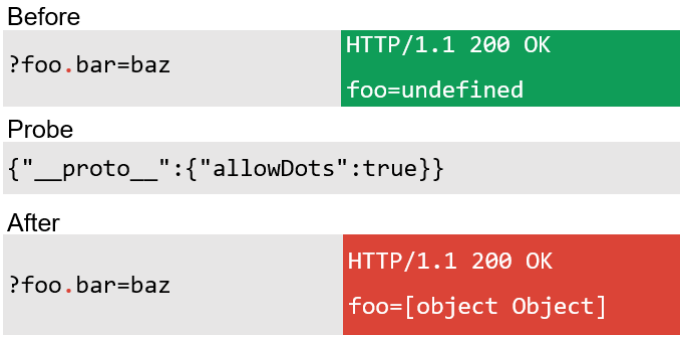
圖：修改parameterLimit參數(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

上面這個例子是修改參數上限，透過修改parameterLimit參數，讓接收的參數只剩下一個。



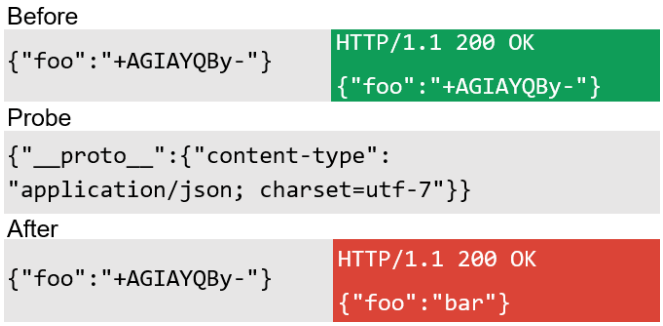
圖：修改ignoreQueryPrefix參數(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

這個例子是修改ignoreQueryPrefix參數，原本送出的參數包含兩個「?」，導致foo無法被設定為bar，將ignoreQueryPrefix修改為true之後，即使有兩個「?」，foo還是可以被設定為bar。



圖：將allowDots改為true(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

這個例子是將allowDots改為true，就可以透過傳送參數建立新的物件。



圖：變更編碼規則(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

UTF-7是一種類似base64的編碼規則，攻擊者可以透過變更編碼規則規避特殊字

眼，繞過伺服器檢測。

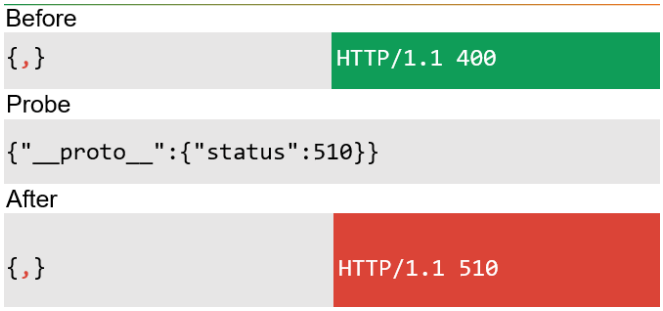
除了手動方式去檢測原型鏈汙染外，也有可以自動化檢測的方式。

圖：改變json spaces參數(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

利用改變json spaces參數，在json回應格式中增加一個空白，來檢測是否有原型鏈汙染情形，但這個情形目前已經被修復。

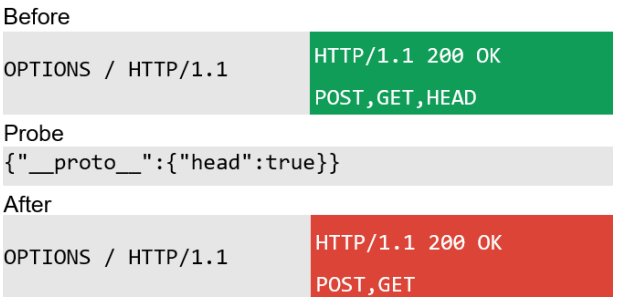
圖：修改exposedHeaders參數(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

修改exposedHeaders參數，讓回應的標頭中增加Access-Control-Expose-Headers參數。



圖：將狀態碼400修改為510(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

這個例子是將狀態碼400修改為510。



圖：將head參數改為true(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

將head參數改為true，伺服器回應就少了HEAD。

圖：利用原型鏈汙染取得伺服器端原始碼(資料來源：https://portswigger.net/research/server-side-prototype-pollution)

除了改變參數以外，也可以利用原型鏈汙染去取得伺服器端的原始碼，做一個包含名為creative-cloud-loc的cookie並給予constructor參數的request，從回應可以看出cookie的值被用做一個屬性的名稱。

最簡單防止原型鏈污染的方式就是避免使用object相關字眼在應用程式之中，使用Set/Map來取代object可以有效的防止原型鏈污染，如果是無法使用Set/Map的情況下，也可以使用Object.create()，這可以確保你所新增的物件並非是繼承自Object.prototype，如果一定要使用object，你可以用__proto__屬性創建一個使用繼承自null並初始化為null的object。

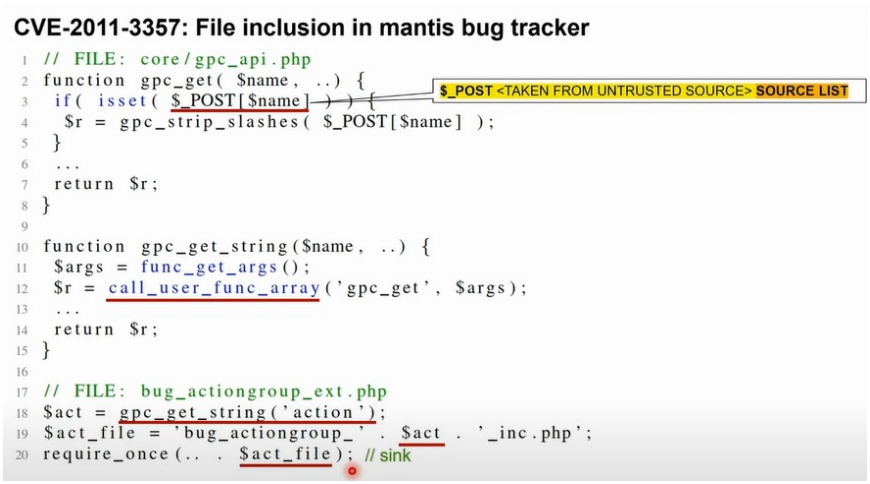
十一、Testability Patterns for Web Applications - a new OWASP project(主講人：Dr. Luca Compagna)

介紹一個關於web應用程式可測試的模型以及在web應用程式中安全性及隱私性

可以被測試的方式的計畫。

靜態應用程序安全性測試(SAST)的可測試模型是一組原始碼設計的指引，讓程式碼可以用簡單的靜態分析工具進行分析，目的是為了讓程式碼的自動化更新可以更簡單得找出潛在的錯誤以及議題。

靜態應用程序安全性測試在企業中已經被普遍用來測試弱點，例如可以測試注入攻擊的弱點。但這樣的偵測並不容易，如果分析工具無法完整了解所有原始碼之間的流程，就會無法偵測出弱點，因此如果沒有被偵測出弱點也不代表是真的沒有弱點存在。



圖：測試障礙範例(資料來源：研討會議簡報)

譬如這個例子，實際有問題的地方會出現在最後執行的檔案名稱$act_file，但這個檔案名稱經過多次的改變，所以要追朔到實際會執行的檔案名稱非常困難，使用很多SAST軟體也沒辦法檢測出這樣的問題。經猜測這可能是因為原始碼中間有經歷動態呼叫的關係，所以這時如果將測試的模型改為靜態呼叫，大多SAST軟體就可以偵測出這樣的弱點，因此這個計畫就是要以PHP、JavaScript以及Java為目標找出可測試的模型。

目前的模型經過測試超過3000個以上在Github上的開源程式碼，約每二十行原始碼就會出現測試障礙無法繼續測試，所以現階段還必須持續進行改寫測試模式、加強SAST工具或是提供客製化規則，來減少測試障礙。

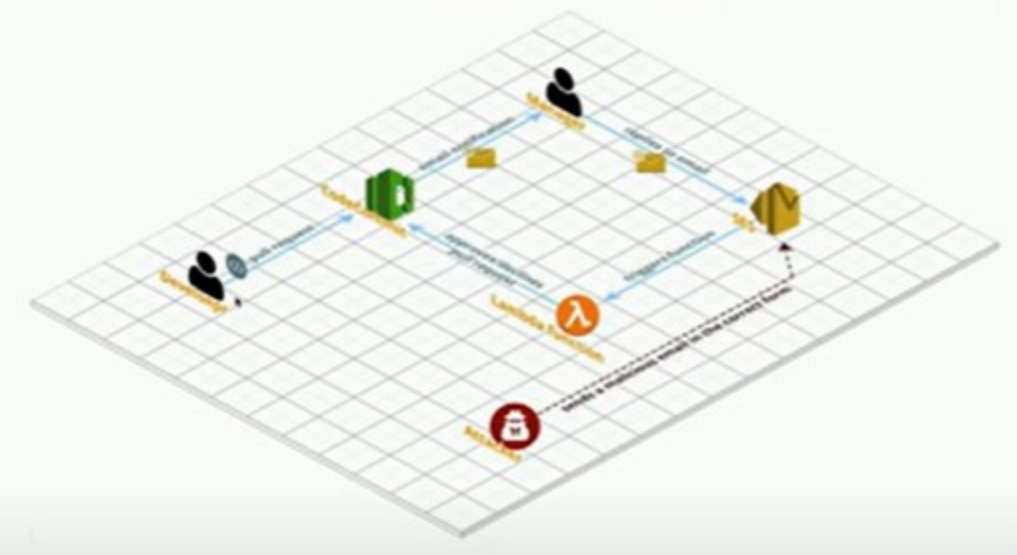
十二、OWASP SERVERLESS TOP 10

Serverless是指一種雲端運算架構，應用程式的開發人員不必考量運行應用程式所需的基礎設施和資源(如伺服器、作業系統、資料庫等)的設置、管理、維護等問題，只需專注於編寫應用程式，就可以在雲端上快速部署和運行應用程式，而基礎設施和資源則由雲端供應商提供和管理，並且按使用量計費。採用Serverless技術後，我們也將一些資安風險移轉給雲端供應商。然而，Serverless即使不需要提供或管理伺服器，仍然執行程式碼。如果該程式碼以不安全的方式撰寫，仍然可能容易受到傳統應用層攻擊的漏洞影響。

OWASP 比照原有的OWASP Top 10專案，提出了OWASP Serverless Top 10，在這個研討會議中主講人針對下列6項風險提出說明：

1. 事件注入(Event Injection)：因為在雲端上所有的輸入都是以事件的方式運作，所以我們應該對於所有來源的資料輸入採取零信任的態度做驗證，以及針對輸入資料的格式進行有效性驗證，並且使用最小權限執行函式以縮小攻擊面，如果可以的話也可以使用商用解決方案在函式中偵測弱點。

2. 失效的身分驗證(Broken Authentication)：因為函式是以無狀態的方式執行，並具有多個的執行點、服務或事件，且無固定的資料流，所以攻擊者很容易就繞過正常的管道去觸發執行惡意程式，如下圖所示：



圖：攻擊者繞過正常管道觸發執行惡意程式(資料來源：研討會議簡報)

所以我們應該多利用雲端供應商所提供的驗證機制(如AWS Cognito)，或是在函式中規劃像是JWT的token身分驗證機制，並且使用最小權限及進行輸入資料驗證，如果有必要的話也可以將狀態儲存至資料庫中。

3. 敏感資料暴露(Sensitive Data Exposure)：如同其他的雲端儲存資料一樣，如果沒有做好管理機制，在雲端上的資料很容易就被竊取，所以我們應該隨時針對暫存資料夾(如/tmp)進行刪除或加密，並使用雲端供應商提供的加密機制(如AWS KMS)來加密環境變數或敏感資料，還要針對儲存空間進行安全組態設定，以及針對敏感資料存取設定最小權限，或是使用AWS Marcie去識別潛在敏感資料來評估是否有必要進行修改許可權或加密資料等必要操作。

4. 不當的權限設定(Over-Privileged Functions)：主講人認為在他看過的函式中，高達90%的函式都有不當權限設定的問題，像是下圖所示的函示：

圖：函式不當權限設定(資料來源：研討會議簡報)

這個函式只是一個簡單的讀取資料並放入資料庫表格中的函式，但是因為僅使用預設的權限設定，讓資料庫及表格後面的接著萬用字元「*」，表示這個函式是可以觸及到跟這個函式無關的資料庫表格，修正後的權限設定如下圖：

圖：修正後的權限設定(資料來源：研討會議簡報)

所以我們應該要盡可能地檢視每個權限設定，或是使用自動化的方式去紀錄程式執行的狀態來定期執行檢視分析。

5. 不安全的第三方元件(Vulnerable Dependencies)：在Serverless的架構中程式通常都是輕量化，但是會呼叫許多的第三方元件，有些是雲端供應商提供，有些則是開發人員自己匯入的，所以我們應該要在程式佈署上線先針對第三方元件進行掃

描，或是使用安全版本的第三方元件並定期檢視是否有安全上的漏洞來進行更新。

6. 不足的日誌和監控機制(Insufficient Logging & Monitoring)：日誌和監控機制是由雲端供應商所提供，但是開發人員必須要去規劃告警的機制來有效運用雲端供應商的日誌及監控機制。

在Serverless的架構下，攻擊模式和防範技術與傳統的應用程式世界並不相同，很多細節都必須要由開發人員來規劃與執行，藉由OWASP Serverless Top 10，開發人員就可以針對相關的風險進行補強與管理，以強化服務的安全性。

十三、Hacking and Defending APIs - Red and Blue make Purple(主講人：Matt Tesauro)

API是應用程式的基本技術，漸漸成為了攻擊者主要的攻擊目標，例如：攻擊者會透過典型的滲透測試方法進行不安全的物件授權。API是一個應用程式介面作為連結電腦或是電腦程式之間的溝通，例如後端及應用程式網頁、手機應用軟體與資料中心、雲端或是第三方提供者之間的溝通等，所以即使API在概念上是一個簡單的溝通介面，但實務上並不單純，即使你有強大的應用程式安全計畫，還是只能覆蓋部分的內容，就像是用手接水一樣。資料就像是石油，而API就像是移動這些石油的運輸船，也因此API是資訊處理上非常重要的一環，在瀏覽器逐年增強控制機制讓登入越來越困難的時候，API卻沒有類似的機制發展。當攻擊者知道API缺乏控制時，就會在網頁無法侵入時轉而攻擊API。

API的測試方式有在沒有任何資訊下的黑箱測試、在知道所有資訊並移除部分控制之下的白箱測試、比黑箱測試多擁有一些資訊的灰箱測試及知道所有資訊並只保留API控制的水晶箱測試。API安全性有3個主要定義：(1).API的安全樣態需盤點所有API並了解API的來源、呼叫者以及被傳送接收的資訊、(2).API運行環境安全性需觀察API的流量並分辨出非正常流量，進行異常偵測以及警報、(3).API安全性測試需評估API的安全狀態並經常性進行動態應用程式安全測試，將結果回饋給開發團隊。以下針對攻擊模式進行攻擊者及防禦者之行為分析：

被動偵測：攻擊者無須與目標進行互動，資訊來源來自公開來源情報，例如：利用Google搜尋、DNS/OWASP資料、Shodan(連接網路的設備搜尋引擎)、Github議題及Stack Overflow(技術討論論壇)上的討論等。因為防禦者並沒有提供攻擊者任何資料，所以所能做的手段有限，而且為了要提供API服務，API提供者通常不會隱藏API的使用方法，這也間接幫助了攻擊者更了解所提供的API內容，建議可以將相關文件資料放在登入之後。再提供

主動偵測：攻擊者會和目標進行互動，嘗試去尋找相關線索，例如去竊聽http/https埠的傳輸內容、掃描網站路徑等。防禦者容易受到其他正常流量的干擾，很難將異常流量過濾出來，建議可以檢視會指向API的相關文件例如：robots.txt，並且監視異常流量，一旦發生攻擊者進行主動偵測時就能夠及時發現。

探索：攻擊者會嘗試取正常授權，向API進行合法的請求，攻擊者會尋找API的相關文件以及使用方式、尋找API規格檔案、竊聽一般使用者的流量並蒐集路徑等資料。防禦者可以透過觀察一些嘗試學習API的流量或是一些沒有被文件紀錄的API請求失敗紀錄來判斷是否遭受攻擊，建議可以定義限定內部使用的API或是用偵測特殊狀況的方式來進行防禦。

不安全的物件授權：攻擊者會觀察API架構，嘗試使用其他ID或資源來呼叫API，譬如用A使用者的API將參數改變為B使用者的資料進行呼叫，或是透過觀察API的回應來進行攻擊。防禦者需要透對API呼叫進行深度封包檢測或是觀察異常增加的失敗授權呼叫來偵測攻擊，例如從同一個客戶端出現2個只有ID參數不同的類似API請求。建議可以將目光放在高風險的API並偵測不安全的物件授權的攻擊行為。

無效身分認證：攻擊者會針對沒有反自動化重設密碼或是多重身分認證機制的API進行暴力破解身分認證，以及利用密碼潑灑方式使用一組簡單的密碼嘗試登入不同使用者帳號，除此之外也會利用JWT弱點進行攻擊，例如沒有SSL/TLS加密或是簽章保護的API。防禦者如果有正常的運行環境保護機制，應該可以辨識出暴力破解攻擊，另外防禦者應該遵循RFC文件進行API開發並確保加密機制有正常運作。

資料不當暴露：攻擊者會尋找API回應中有提供額外資訊的標的，例如行動裝置應用程式所使用的API傾向在客戶端進行資料過濾，所以會有大量的資料可以從API中取得。防禦者無法將單一異常請求從正常流量中分辨出來，所以建議防禦者可以利用靜態應用程序安全性測試來測試原始碼並且每支API中只提供必要資訊，避免在客戶端才進行資料過濾。

資源缺乏及速率限制：攻擊者會利用不同使用端及不同IP來向API要求大量資料來癱瘓API服務。對防禦者來說，這些攻擊看起來只是會需要大量回應的正常請求，所以只能透過使用量的異常來判斷是否被攻擊，建議可以增加API的限制，並偵測異常流量。

無效功能權限控管：攻擊者會將目光放在有多種角色控管權限的API，測試沒有被紀錄在API規格文件上的HTTP方法，並透過不同角色權限來和API互動，例如使用管理者權限來向API請求。防禦者可以觀察到一些因不支援而請求失敗的紀錄或是同一個用戶端在短時間內扮演多個角色，建議可以確認各種角色使用API的權限等級，並偵測用戶端變換角色且請求失敗的異常行為。

批量配置不當：攻擊者會透過請求的內容猜測是否有其他未送出的資料，並觀察不同角色權限間的請求差異來變更請求內容，進而取得他們不應該取得的資料。防禦者可以從大量無效以及來自不同角色的請求觀察到批量配置不當攻擊，建議將目光放在有多重角色權限控制以及有敏感資料的API進行防護，並觀察是否有出現索要額外資料的請求。

安全設定缺陷：攻擊者會觀察TLS設定、從標頭洩露的資訊、預設授權，或是透過除錯模式及內網設備去呼叫內部函式。防禦者可以利用弱點掃描找到這些問題，或是透過被動流量監測來觀察攻擊行為。

注入攻擊：攻擊者會在Token、金鑰、query或是資料中注入一些字串來進行攻擊。防禦者可以在API輸入參數中增加檢查機制並且針對輸出進行加密來防止注入攻擊。

版本控管不當：攻擊者會找尋一些錯誤設定的議題，例如內網API可以從外網存取或是存取還在開發中的API。防禦者必須了解所有API資料，針對所有API傳輸的資料進行分級以及對內外部API進行控管，並建議當環境改變時及時更新API來抵禦版本控管不當攻擊。

紀錄與監控不足：因為模糊測試並不會被視為需要被阻擋的對象，所以攻擊者會利用這點進行攻擊，例如注入不合理的資料。防禦者無法察覺攻擊，所以建議確認每個API都要有合理的授權機制，並確認授權機制可以正常運行。

模糊測試：攻擊者會測試參數的極值、負數、在文字參數輸入數字、在數字參數輸入文字或是特殊字符，並觀察API回應碼、大小、時間及錯誤資訊來進行攻擊。防禦者可以從同一個客戶端在短時間送出大量請求來觀察是否被攻擊。

API安全性工具： http://owasp.org/www-community/api_security_tools

十四、Automated Security Testing with OWASP Nettacker(主講人：Sam Stepanyan)

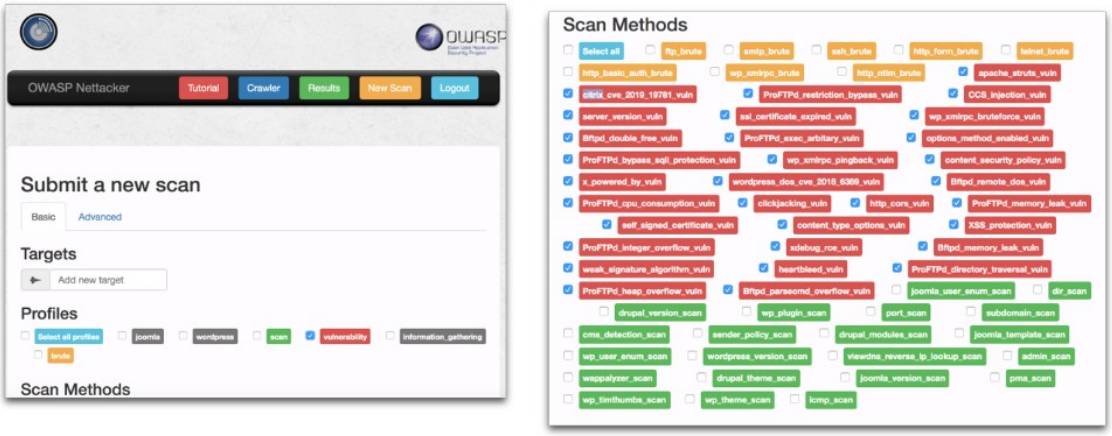
OWASP Nettacker專案是一個自動化的滲透測試工具，用Python完全寫成，是一個如同瑞士刀般功能強大且多元的工具。Nettacker能夠使用各種方法運行各種掃描，並為應用程序和網絡生成掃描報告，包括服務、漏洞、弱點、配置錯誤、預設憑證和其他功能，還提供了一個可擴展的架構，可以輕鬆添加自定義模塊和腳本，以滿足用戶特定的需求，目前已被廣泛用於漏洞掃描、滲透測試、安全評估和其他安全相關任務，甚至被包含在針對滲透測試人員和安全研究人員的專門Linux發行版中。

OWASP Nettacker與Burp Suite和OWASP ZAP的不同在於Burp Suite和OWASP ZAP主要是針對一個網站進行全網站的弱點掃描，而OWASP Nettacker可以針對多個IP、網路及子網域掃描開啟的服務或是特定的弱點以及進行暴力破解。且OWASP Nettacker在執行指令上十分簡便，僅須設定掃描標的以及使用的模組即可進行掃描，如下述指令：

```
nettacker -i <target> -m <module>
nettacker -i 192.168.1.149 -m port_scan
```

```
nettacker -i 192.168.1.0/24 -m port_scan
```

不熟悉CLI(Command-Line Interface)的操作者，OWASP Nettacker也具備WebUI供操作者使用，如下圖所示：



圖：OWASP Nettacker WebUI(資料來源：研討會議簡報)

除了上述功能外，OWASP Nettacker也具備IOT掃描及暴力破解IOT設備預設帳密的功能，主講人曾接受倫敦一家金融機構的委託，希望找出在機構裡所有使用預設帳號的網路攝影機，而使用OWASP Nettacker，僅在短短幾分鐘就找到了所有使用預設帳號的網路攝影機。在2021年微軟的MS Exchange Server出現CVE-2021-26855時，歐洲某個國家的電腦緊急應變中心也曾利用這個工具掃描全國的IP，找出存在這個CVE弱點的exchange server並連絡相關負責單位去處理。

圖：OWASP Nettacker 掃描MS Exchange Server CVE-2021-26855(資料來源：研討會議簡報)

綜上，利用OWASP Nettacker可以掃描全網段的特性，我們可以利用它來進行內外網的資產盤點以及攻擊面的發現，並掃描所有子網域所提供的服務及開放的埠，也可以用它來進行暴力破解攻擊，所以OWASP Nettacker可以幫助防禦者在攻擊者攻擊前，先行檢視風險所在，以強化自我防護的能力。

十五、Log story short: Chopping through forests of data(主講人：Moti Harmats)

錯誤與例外的日誌紀錄常常顯示出應用程式的弱點，只要有正確的方法就可以

從成品環境中找到弱點。

紅隊的攻擊模式可以分為兩種，一種是攻擊產品邏輯，例如：暴力破解及密碼潑灑，另一種是攻擊應用程式的運行環境以及框架，例如：SQL注入攻擊。

常見日誌有三種類型：從使用者網路傳送資料進來時，會先在網站伺服器留下存取日誌，接著在應用程式之中會留下應用程式的邏輯日誌以及技術性日誌，分別記錄系統工作流程的邏輯內容以及系統運行的訊息內容。當攻擊者進行攻擊時，一定會先經過網站伺服器，並寫下伺服器存取日誌，然而，伺服器存取日誌無法分辨出攻擊者的攻擊是否成功，而且百分之99的存取日誌都與攻擊無關。

假設攻擊者是透過弱點與SQL資料庫進行檢索，並出現語法錯誤的話，那就會被反應在伺服器存取日誌上，或是其他例外發生也會被伺服器存取日誌紀錄，所以大概有百分之1的可能性可以在伺服器存取日誌上找到被攻擊的線索，然後再透過應用程式的日誌去尋找攻擊內容。雖然大多人希望能夠事前預防系統被攻擊，而非事後再找出弱點，但並非所有弱點都可以被事前偵測出來，部分開發內容是不會發出例外處理訊息或是被記錄再日誌上。即使很難事前偵測出所有弱點，開發者仍可以嘗試在不影響產品運行的情況下找尋弱點、監控包含第三方的所有原始碼並最小化偵測錯誤情形。

在系統分散的架構下，因為沒有單一管道及儲存庫，所以要即時性收集所有日誌非常困難，而且日誌資料量龐大，任何搜尋都需要花費大量資源。為了處理龐大的資料，建議可以使用Vizsla，這個軟體可以簡單增加新的日誌來源、找出有關的日誌並有模組化規則。然而，日誌分析並非一朝一夕可以完成的事，目前仍有許多異常偵測的方法等待被發現並加入日誌分析工具，進一步強化這些工具對日誌分析的能力。

十六、Mobile Wanderlust”！Our journey to Version 2.0!(主講人：Sven Schleier)

隨著手機應用程式的蓬勃發展，利用標準來確認手機應用程式具備適當的安全保護措施是必要的，OWASP也針對手機應用程式訂定了OWASP MASVS(Mobile Application Security Verification Standard)及OWASP MASTG(Mobile App

Security Testing Guide)，手機應用程式可藉由驗證這兩項標準來確認其安全性，透過這兩項標準也可以讓開發人員知道手機應用程式所隱藏的潛在攻擊面並知道如何強化潛在攻擊面的安全控制措施，提升開發產品的資安防護。

OWASP MASVS是針對手機應用程式的安全驗證標準，主要在幫助開發人員、測試人員和資安人員評估和驗證手機應用程式的安全性，並提供了一個框架，可用於驗證手機應用程式是否符合安全最佳實踐和標準。OWASP MASTG則是一個針對手機應用程式安全測試的指南，主要在提供給測試人員、測試人員和資安人員一個完整且可重複使用的框架，以測試手機應用程式在不同方面的安全性。該指南覆蓋了廣泛的主題，包括應用程式結構、敏感數據存儲、身份驗證和授權、網路傳輸安全、使用者界面和使用者體驗等。

OWASP MASVS已更新至2.0版，但是OWASP MASTG則仍停留在1.5版，目前OWASP正在針對OWASP MASTG2.0進行編撰，使測試指南項目可以對應到OWASP MASVS，正在調整的架構如下圖所示：

圖：OWASP MASVS的架構架構示意圖(資料來源：研討會議簡報)

將安全的控制措施標準化可以讓驗證的標準一致，Google已經針對其所開發的手機應用程式，如Gmail、Google Doc和Google Drive等，都已經通過OWASP MASVS及OWASP MASTG的驗證而得到驗證標章，只要使用者越來越重視手機應用程式的安全議題，可以預見會有越來越多手機應用程式也加入OWASP MASVS及OWASP MASTG的驗證行列。

十七、Contextual Vulnerabilities are the Ingredients and OWASP Top 10 Mapping the Seasoning(主講人：Meghan Jacquot)

本研討將分享從開源資料及除錯賞金發現之弱點分類方式，並對這些弱點進行分類，分析出對防禦者及開發者有意義的內容。弱點將被分為兩個資料集介紹，一個是從開源資料(Google Project Zero及Open Source Intelligence, OSINT)所分析的弱點資料集、另一個是除錯賞金計畫以及滲透測試蒐集來的弱點資料集，本研討將對這兩個資料集中的弱點進行分析，盡可能歸納進OWASP Top 10中。

Google Project Zero通常會在四月發布報告，說明對於威脅的處置方式以及所找到的弱點，在去年他們首次發布了2份報告。根據每年發布的報告，在近年所偵測到的零日攻擊數量急遽增加，但這裡可能存在著偏差，因為當你越想找特定目標的時候，你會越容易找到這個目標，所以這個結果或許可以歸因於越來越多人在尋找零日攻擊、對於零日攻擊的偵測越來越多所致，因此就算零日攻擊被找出來的數量越來越多，也不能當作零日攻擊發生數量越來越多的證據。

根據Google Project Zero 2022年4月的報告，他們所偵測出的零日攻擊中有67%與記憶體損壞有關，這些弱點可以被歸納到OWASP第11項涉及記憶體管理的原始碼品質問題。另外，因為許多組織並不希望公布零日攻擊細節，所以很難將所有弱點都進行歸類，例如CVE-2021-31199只寫到「發生於 Microsoft Enhanced Cryptographic Provider，可用以提升執行權限」，所以只能猜測這可能與OWASP第2項密碼機制失效有關。零日攻擊也有可能是一連串的弱點組合，例如CVE-2021-25855就是使用CVE-2021-26857、CVE-2021-26858及CVE-2021-27065一連串的OWASP第10項伺服器端請求偽造的組合對微軟的Exchange伺服器進行攻擊。

根據Google Project Zero 2022年6月發布的另一份報告顯示，在2022找到的零日攻擊中有50%的弱點是變化自過去已經有修補的弱點、22%的弱點是變化自2021年找到的零日攻擊弱點。因此可以推估許多弱點的根本原因沒有被完全修補，所以當攻擊者稍微變化攻擊手段，就又可以再次進行零日攻擊。

另一項弱點資料集蒐集自除錯賞金計畫及滲透測試，最常見的弱點為OWASP第3項注入攻擊，這在各個除錯賞金計畫中也很常見，因為注入攻擊很容易被觀察到，也很容易造成後續遠端程式碼執行攻擊。但，無論弱點是否可以被歸類到OWASP Top10，重要的是甚麼樣的資訊是有用的。

對於這些弱點，可以透過威脅模型分析來釐清甚麼樣的資訊對你的系統來說是至關重要。威脅模型分析可以促進在設計階段進行主動的安全防護、及早辨識並修復威脅，減少在產品階段進行修復的成本或在被攻擊之前找到潛在的問題，此外，資安團隊也可以透過威脅模型分析來和開發團隊進行溝通。目前有很多威脅模型分析方法可以使用，例如微軟的STRIDE威脅模型分成6類，欺騙、竄改、否認、資訊外洩、阻斷服務攻擊以及特權提升，可以對應到OWASP Top 10的權限控制失效、注入式攻擊、加密機制失效以及安全設定缺陷。鑽石威脅模型分析則可以分為敵人、設施、能力以及目標等4個元素，也可以對應到OWASP Top 10的認證及驗證機制失效以及易受攻擊和已淘汰的組件弱點，除此之外還有很多種威脅模型分析方法可以選擇，另外也可以創建你自己的威脅模型來進行分析。

創建威脅模型的第一步是定義範圍，決定要使用哪種威脅模型來分析，並思考要針對甚麼應用程式或系統進行。第二步是辨識威脅，因為有很多不同的工具以及方法可以找尋威脅，例如應用軟體安全測試(SAST、DAST以及ISAT)和靜態原始碼分析，所以必須決定要使用的工具。第三步是處理威脅，決定要優先處理的威脅、進行風險評估以及思考可能產生的衝擊。第四步是對於發現的威脅進行交流，決定哪些關鍵人員需要知道這些威脅。第五步是評估模型的有效性，從結果去評估所建立的威脅模型是否能覆蓋正確的元件、是否有尋找到足夠的威脅、是否有處理所找到的威脅以及這個威脅模型是否發揮效用等。

肆、心得與建議事項

隨著資訊科技的迅速進步和不斷創新，人們的日常生活早已脫離不了這些新興科技，如智能手機的普及、人工智慧的應用、物聯網和智慧城市的興起等。然而這些新興科技帶

來便利的同時，也帶來了更多可供駭客利用的攻擊途徑，所造成的影響不光只在個人生活範圍，若是智慧城市的物聯網或是關鍵基礎設施的工控系統遭到攻擊，影響將擴大至國家安全層級。

本次參與OWASP 2023 Global AppSec Dublin期間，積極加入各項研討會議進行討論，部分會議議題所討論的想法可納入政策執行與法規制定的參考，以下列舉2項可納入參考的議題討論：

1.資通安全管理法修法程序規劃

在「Philosophizing security in a "mobile-first" world」中，主講人提到了網路安全與使用者的自由及安全感環環相扣，有時組織部署了一系列的防護措施，但這種措施有時候反而造成了一些使用者感覺上的不安全感或是限制了使用者的自由，所以安全感就必須在網路安全與使用者的自由之間取得一個平衡。這個情境亦可以對照到政策的制定者在制定相關規範時，出發點是為了讓民眾能夠在一個受到管控的安全環境下可以放心的生活，但是當民眾的生活受到高度管控時，反而造成了民眾的不安全感，讓原本應該是受到管控的安全環境在這種不安全感下變得不受民眾的信任。

如上述這種因為缺乏和民眾的溝通導致政策制定受到阻力的例子比比皆是，因此在接下來規劃的資通安全管理法修法程序中，除了針對國家資通安全政策進行整體規劃外，也必須要注重與專家學者、業界和民眾等利害關係者之間的意見交流，納入專家學者的專業意見及業界的實務經驗，並在意見交流的過程中讓民眾藉由充分的參與增加民眾對於修法的認同感，才能讓整個修法過程及結果獲得全體國民的支持。

2.網路攻防演練精進

在「Don't let bug bounty kill your appsec posture」及「Contextual Vulnerabilities are the Ingredients and OWASP Top 10 Mapping the Seasoning」中，兩場主講人分別針對除錯賞金提出了一些想法；在「Don't let bug bounty kill your appsec posture」中，主講人表示研究者為了有效率地獲得賞金，將心力放在可以被自動找尋的常見問題上，反而不再關注需要深入技術了解的高風險議題；在

「Contextual Vulnerabilities are the Ingredients and OWASP Top 10 Mapping the Seasoning」中，主講人認為應該要將除錯賞金或是滲透測試所蒐集來的弱點進行分類，來分析出對防禦者及開發者有意義的內容。

為提升我政府機關資安防禦及應變能力，行政院國家資通安全會報自102年起每年辦理網路攻防演練，其中包含資通系統實兵演練，以遴選方式選出符合資格的攻擊手，於機關外部網路搭配弱點掃描或滲透測試等方式，來協助機關發現及改善對外資通設備、系統及網站存在之弱點。

因發現弱點的攻擊手可以依弱點種類獲得獎勵，所以我國所辦理之網路攻防演練資通系統實兵演練執行形式與除錯賞金相似，在弱點的發現上也有可能出現於「Don't let bug bounty kill your appsec posture」中主講人所提到的攻擊手為求發現弱點的效率，而專注於可以被自動找尋的常見弱點，反而不願意多花心力深入需要時間與技術的高風險弱點上，造成國家資通安全的潛在風險，因此，在系統弱點的發現上，除了參考主講人的建議，由機關持續針對系統進行安全性檢測及風險識別、分析及處理外，也可以針對攻擊手獎勵辦法進行調整或是限縮攻擊手攻擊範圍，藉此引導攻擊手將攻擊量能投入在高風險弱點的發現上。

在「Contextual Vulnerabilities are the Ingredients and OWASP Top 10 Mapping the Seasoning」主講人所提的弱點種類分類上，則可以將近年來發現的弱點進行統計，針對發現頻率高的弱點在規則上以增加扣減分數或是進行專案檢討報告的方式，促使機關在系統發展生命週期設計、開發、測試及部署與維運階段都能自行發現並修正這類型的弱點，藉此提升系統防護能力並保留攻擊手攻擊量能。

最後，為因應資安攻擊手法與技術不斷翻新，國際社會正持續發展相關資安防護基準或參考指引讓組織能有效採取相關防護措施，如本次會議所提到常用於應用程式安全測試標準及風險分析的OWASP TOP 10、針對雲端運算架構所訂定的OWASP Serverless Top 10以及確保手機應用程式具備適當安全保護措施的OWASP MASVS及MASTG等，可提供組織在訂定資訊安全規範時的最佳做法建議。因此，建議後續應持續並積極地參與像是OWASP

Global AppSec、Black Hat Asia及美國RSA資訊安全大會等國際資安會議交流，藉此掌握國際資安趨勢、了解相關規範或標準未來訂定的方向，並獲得最新攻擊手法及因應對策等相關資訊，進而精進我國資安政策，提升資安技術能量，以有效保護國家安全。