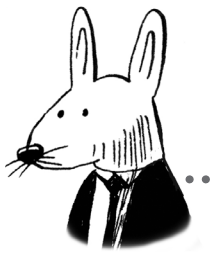
推薦序



曾經某段時間，我幾乎入侵這個星球上各類在走動或爬行的東西。從 1989 年第一次破解某位系統管理員的 root 密碼（當然經過授權），到 RSA 2012 資安大會的主題演講舞台接管胰島素泵浦並釋放所有胰島素，我的目標是揭露對手（駭客）的思維模式和攻擊手法。畢竟，教育及訓練是防範網路攻擊的最後希望堡壘。

我在 1999 年寫出第一本《Hacking Exposed: Network Secrets and Solutions》（黑客大曝光：網路安全機密與解決方案）時，深深體會到管理員多麼渴望瞭解對手的實力與情報。因此，很快又和其他人合著第一本關於網路新世界的駭客技術之教科書：《Web Hacking: Attacks and Defense》（網路入侵技巧：攻擊與防禦），並於 2002 年出版，在那本書中，我和共同作者使用相同的標準步驟來教學和演示，指導防禦者如何防範網路攻擊、保護網路資產。然而，我們當時並未意識到軟體開發人員竟具有左右攻擊成敗的影響力，簡而言之，軟體開發人員攸關網路攻擊的成敗 —— **因為，所有網路攻擊都是從程式碼開始，也是以程式碼結尾。**

網際網路的每個部分都靠軟體運行，從網路路由器、交換器到伺服器和終端節點，再到工業控制技術，用來分享、通訊和傳播資訊的一切都是靠程式碼寫成的，找到的漏洞，終究可以追溯到原始碼裡。

本書作者在書中介紹被成功攻擊的真實範例，並告訴讀者如何避免成為下一位受害者。

程式碼導致安全漏洞的兩個核心問題：存在安全缺陷，以及缺乏防止邏輯缺陷的保護功能。當這些條件碰在一起，就能夠引發 100% 的網路攻擊，只有

開發人員才能從根本阻擋攻擊，其他防禦手段都只是門面裝飾，全球開發人員對彼此大聲疾呼：「必須靠你才擋得住網路攻擊！」

防禦者想要一勞永逸阻絕對手，唯一方法就是從根源 —— 原始碼 —— 解決問題，軟體工程師必須成為精通安全的專家，能夠預測對手將如何惡意利用他們開發的程式（也許不需取得程式碼）來打穿漏洞，所以才說只有開發人員才能解決網路安全問題。

為此，我們需要一本簡單、直覺、易於理解、由開發人員用開發人員的語言寫給開發人員的書，Malcolm 所撰寫的這本書正符合這些條件，他將寶貴的建議和知識化作容易吸收的文字，不僅提供開發人員撰寫安全程式碼所需的知識，並釋出程式碼。此外，也指導寫程式的人如何管理安全缺口，這些實務作法有助於人們瞭解開發人員的工作性質，減少開發人員因誤解而受到的指責。假如只能閱讀一本有關網路安全的書，那麼本書就是不二之選。

本書可讓各種程度的開發人員瞭解網路攻擊的原因，以及如何修補或減輕程式裡的安全風險，Malcolm 清楚解釋入侵攻擊的手法，讓開發人員有信心解決在程式裡找到的問題。就本質上，本書可看作瞭解程式碼漏洞的重要基礎教材，每位開發人員（及每個人）皆可因熟讀本書而受益匪淺。

Stuart McClure Qwiet AI 執行長，
《Hacking Exposed》（黑客大曝光）系列的創始作者

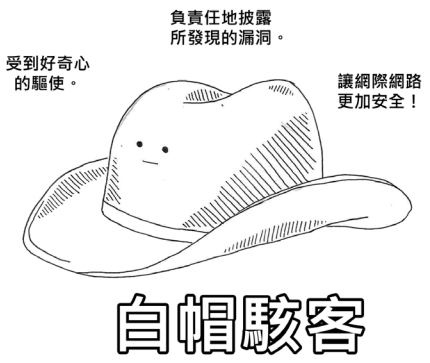
會選擇閱讀本書的讀者，應該是一位具有資安意識的開發人員，希望學會如何保護自己。這是一本詳盡的 Web 安全指南，讀者將學到如何從瀏覽器、網路、伺服器和程式碼等層面保護 Web App 的安全，筆者還會介紹可應用於各個層面的重要安全準則。

然而，在深入研究具體細節之前，有必要調查一下網際網路上有哪些惡意行為者？其動機為何？使用哪些手段或工具？就來介紹一下駭客吧！

1.1 駭客攻擊的原因和手法

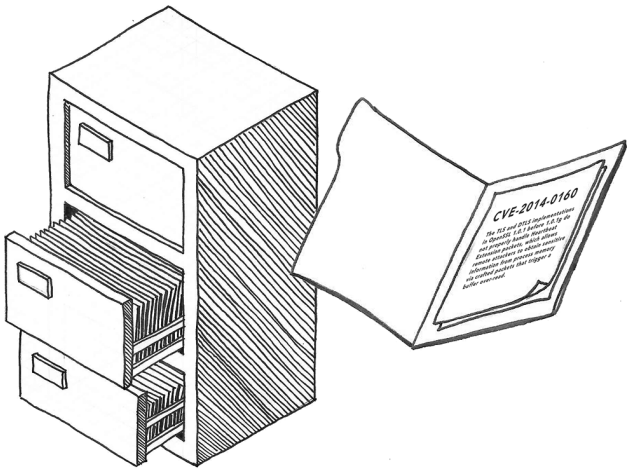
從字面上而言，**駭客攻擊**（Hacking）是指在未經授權情況，嘗試取得軟體系統的存取權。然而，此定義對網路的各種惡棍和搗蛋鬼並不公平，這其中包括一些吾人認為不屬於駭客行為的灰色地帶（與家人分享 Netflix 的登入資訊，會讓家人變成駭客嗎？看看 Reed Hastings 怎麼回答這個問題）。

相對地，應該將視野投向覬覦你的 Web App 之網路犯罪分子，這才是我們要找的駭客。自從網際網路出現後，這群人就一直利用網路從事犯罪。這群攻擊者大致可分為：為了金錢或政治利益而執行惡意（非法）行為的**黑帽**駭客；或試圖搶在黑帽之前找出漏洞，避免被黑帽利用的**白帽**駭客。大公司經常付給後者所謂的**錯蟲賞金**，獎勵任何能夠搶在惡意行為者之前找出安全缺陷的人，這種作法導致**灰帽**駭客崛起，如果通報漏洞有利可圖，就會通報，否則就利用漏洞獲取不法利益。



兩種顏色的駭客都會使用自動化工具和腳本來探測漏洞，而這些工具通常是開源且容易取得的。**Kali Linux** 是許多駭客愛用的攻擊平台，這是一種客製的 Linux 版本，包含受歡迎的數位鑑識和駭客工具。白帽駭客利用 Kali 從事**滲透測試**、或進行安全稽核時，用來掃描客戶的系統以尋找有漏洞的端點；黑帽也會使用相同工具來尋找可供利用的漏洞。

安全研究人員也算白帽駭客的一種，他們會探索常見軟體的漏洞資訊，並撰寫和分享研究成果。例如，某位研究人員可能找到流行的 Java Web 伺服器（如 Apache Tomcat）的漏洞，然後向該軟體開發者展示如何讓系統暴露漏洞；當發布此軟體問題的修補程式時，這些漏洞就會被編入**通用漏洞和披露**（CVE）資料庫，此資料庫由專門從事網路安全研究的美國非營利組織 MITRE 公司維護，因此，常可看到許多漏洞會參照到 CVE 編號。



一旦發布新的 CVE（有時甚至在發布之前），利用漏洞的**概念驗證**（PoC）程式碼也會伴隨出現。**漏洞利用**（Exploit）是一組程式碼片段，示範如何利用漏洞執行惡意行為，例如將惡意程式碼偷送進有漏洞的系統裡。過不久這些漏洞也會被收納到 **Metasploit** 等駭客工具中，黑帽和白帽駭客也常利用 Metasploit 探測網站的漏洞。而黑帽駭客會囤積所發現的漏洞知識，並盡可能避免漏洞暴露，以免被修補。

網路犯罪分子的工具包裡並不是只有軟體漏洞利用工具，**社交工程**是獲取目標的信任並說服他們洩露機密資訊（如登入帳密）的過程，社交工程可以是面對面、透過電話或簡訊。讀者可能聽過**網路釣魚**郵件，是試圖誘騙目標提供密碼的電子郵件，駭客發現利用**魚叉式網路釣魚**有很高的成功率，這種社交手法會事先調查目標人員（通常是公司的會計部門人員）的背景。在即時通（如 Line）及社群媒體（如 Facebook）上也常見這種詐騙手法。

近年來一些重大網路犯罪是由**惡意內部人員**協助完成的，這些人員包括心生不滿的員工或協力廠商，因而出售或洩露公司機密或智慧財產權，或者對公司造成其他類型危害。千防萬防，家賊難防，內部不良行為者是最難防範的，公司面對此風險，必須依照所知情勢，限制人員對資料的存取。

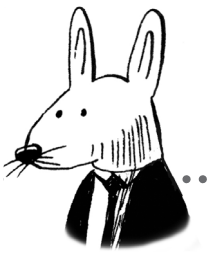
為何網路犯罪如此猖獗？不用驚訝，就因為有利可圖、報酬豐厚。駭客會在**暗網**（dark web）這類地下經濟網站兜售所竊取的資料、信用卡資訊、漏洞，甚至被俘擄的伺服器。報酬則透過加密貨幣支付，故很難追蹤。由於只有透過 Tor 瀏覽器才能拜訪暗網，並以匿名形式存取內容，執法機構很難破獲，所以這些違法交易幾乎不受懲罰。

除了在暗網上出售竊取的資料外，網路犯罪分子也透過勒索手段，直接從被害者那裡取得贖金。**勒索軟體**（Ransomware）是一種惡意軟體，會加密被害者的檔案，禁止其存取，直到以加密貨幣向攻擊者支付贖金為止。石油管線業、醫療保健業、肉品供應商和連鎖飯店等都曾受到重大攻擊，被迫支付贖金才得以解鎖伺服器。由於這類軟體開發者透過抽佣方式，將工具授權給黑帽駭客團體使用，才讓勒索軟體如此普遍。攻擊者有時為了替支付贖金的受害者解密檔案系統，還會提供「支援管道」。

值得注意的是，並非所有駭客行為都是為了金錢。**駭客行動主義**是指激進份子為了推動特定政治目的所做的駭客行為，其行動常常獲得讚許。例如透過**人肉搜索**（doxing）摧毀極右派使用的社群網站、擾亂專制政權或洩漏避稅天堂的文件等。

由國家支助的強大駭客組織常透過網路間諜活動左右現代戰爭的局勢，這類駭客組織能夠利用複雜的監視技術來對付受害者。資安研究人員透過如**進階持續威脅**（APT）的技術特徵追蹤此類威脅，資安社群為每種 APT 起一個有趣的代號，例如 Cozy Bear（安逸熊；俄羅斯駭客組織）或 Charming Kitten（迷人貓；伊朗政府的網路戰組織），這些名稱和它造成的混亂形成鮮明對比。

安全是一種程序 | 5



本章重點

- 何以關鍵系統須由兩個人執行變更。
- 如何透過限制組織成員的權限來確保安全。
- 如何利用自動化和程式碼複用（reuse）來防止人為錯誤。
- 為什麼自動化測試和部署是安全發布的關鍵。
- 稽核軌跡對檢測安全事件有何重要性。
- 從安全錯誤中學習是多麼重要。

福斯橋是一座位於蘇格蘭愛丁堡以西，長約 2.5 公里，橫跨福斯河的懸臂鐵路大橋，在它竣工後被認為是一項工程奇蹟，是英國第一座以鋼鐵建造而成的大型建築，但選用的材料也帶來維護問題，為了保護鋼材免受蘇格蘭嚴冬的影響，全長 2.5 公里的橋樑都需要油漆。

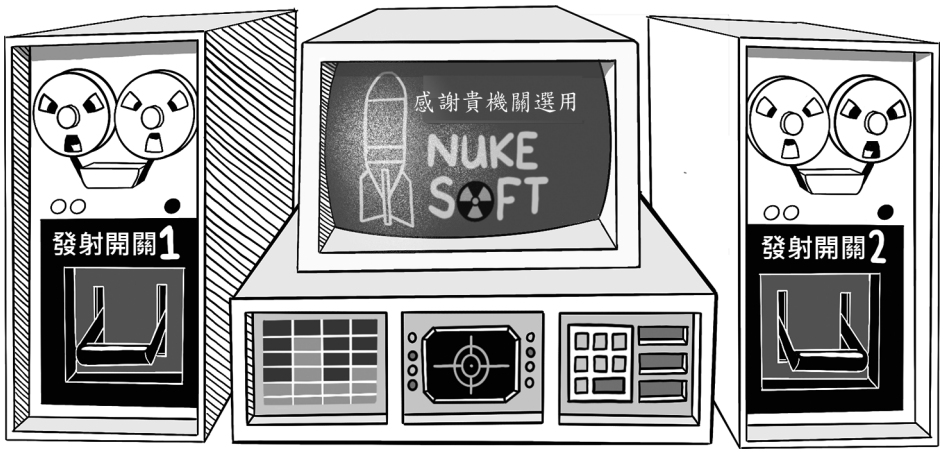
此橋從完工起就開始上漆，鑑於橋的長度，一支常駐的油漆團隊持續進行維護工作，對蘇格蘭人來說「油漆福斯橋」成為一種形容永無止境任務的口頭禪，人們認為油漆工人刷到另一端，又必須回到這一端重新粉刷。

維護 Web App 有點像在粉刷福斯橋，很少有 Web App 是十全十美的，因此，瞭解如何安全地修改和維護運轉中的應用程式，是一種反覆的過程，只擁有程式層級的潛在漏洞之各類制式知識依然不夠，還需知道如何在修改時確保安全。

對多數開發人員來說，撰寫程式是一項團隊活動，因此，先來討論如何實作版本變更。

5.1 應用四眼原則

高度安全的系統常會實施**四眼原則**，這是一種控制機制，重要變更須經兩個人審核通過才能實施。舉個極端例子，為了避免核彈誤發，須兩名操作人員在發射室的兩側轉動鑰匙之後，才能輸入發射密碼（也許當他們成功完成操作時，發射裝置會播放國歌，祝他們末日快樂）。



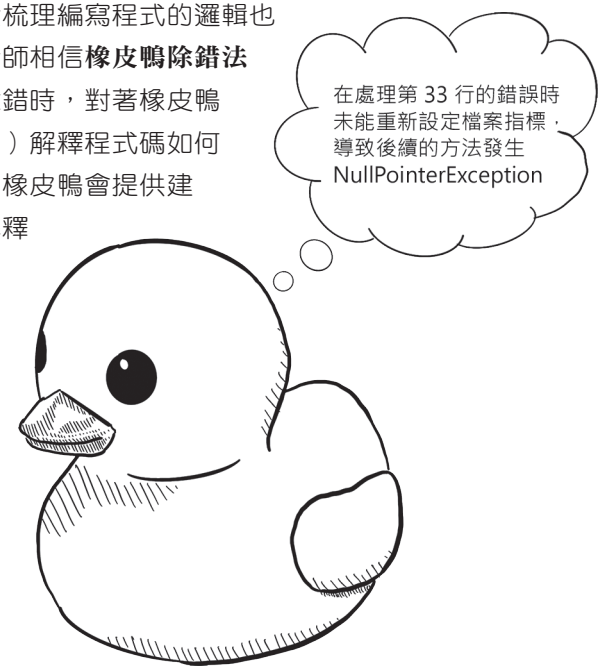
幸好，Web App 的風險沒那麼高，但採用四眼原則可以協助確保系統安全。對重要系統的變更（發布程式碼、更新組態、遷移資料庫等）應由一個人編寫變更程式，並經另一個人審核同意。開發人員之外的第二雙眼睛，可以事先發現潛在的安全漏洞，批准作業通常藉由工單系統完成，並產生書面紀錄，在排除故障時，有助於支援工程師確認問題。碰到 Web App 出現意外錯誤時，

支援工程師會問的第一個問題是「最近做了什麼變更？」最近的改版清單和嚴謹的源碼控制策略（稍後介紹）有助解答這個問題。

審核人員也必須認真看待他們的任務，並非只是加蓋橡皮圖章，當批准重要系統變更時，表示他們有自信此次變更不會造成系統混亂。如果對變更內容有疑慮，他們有權拒絕批准，在同意進行變更之前，要求申請人員提供額外保證和安全措施。審核人員應該接受充分教育訓練，培養良好判斷力；可考慮讓資深工程師或專門的安全團隊成員擔任審核工作，應該會有所幫助。

注意

在實施變更管理控制（如四眼原則）時，會強迫你的團隊提前做好變更紀錄文件。記下要做的事情，本身就有一定助力：可以驅使你確認如何變更、為何需要變更、可能遭遇什麼風險，以及怎樣才是成功的變更。釐清事情的能力，對於梳理編寫程式的邏輯也有幫助，某些程式設計師相信**橡皮鴨除錯法**的效用，這是一種在除錯時，對著橡皮鴨（或其他無生命的對象）解釋程式碼如何運行的作法，重點不是橡皮鴨會提供建議，而是在自言自語解釋有問題的功能時，時常會突然意識到問題出在哪裡！



5.2 在流程中套用最小權限原則

第 4 章提到最小權限原則，規定只授予主體完成其任務所需的最小權限集，並看到這項原則如何應用於系統上，例如網頁伺服器和資料庫帳戶，當然，它也可以（且應該）適用於機構內的人員。

限制團隊成員的權限，能夠降低員工有意或無意所造成的破壞性變更之風險，另一方面，外部駭客企圖竊取或猜測機構成員的身分憑據時，這些限制也可以減低可能造成的損害。依照團隊規模，將職責劃分為多個不同角色，會有不一樣的好處。下圖是軟體開發組織裡的常見角色。

根據團隊規模和文化，同一人可能扮演多個角色，最小權限原則也可以協助建立**限時**（time-box）權限：敏感權限（如變更伺服器或升級資料庫綱要的權限）應限制權限的有效期，以降低惡意行為者劫持帳戶而造成破壞性變更的風險。

5.3 盡可能採用自動化作業

前面已討論過如何實施變更控制來降低人為錯誤的風險，但讀者知道什麼比人類更可靠嗎？就是電腦！將手動流程自動化，可以降低出錯的機率。以下是管理 Web App 時應自動化的一些流程：

- **建置程式**：編譯程式碼和產生資產（如 JavaScript 和**階層式樣式表**〔CSS〕）應透過可從命令列或開發環境觸發的自動建置程序來執行。
- **部署程式**：程式應能以一條指令從源碼控制系統進行部署。儘管資料庫等有狀態的系統在執行版本回退時，常需要多餘復原工作或程序，仍應盡量簡化版本回退作業。
- **新增伺服器**：需要增加執行 Web App 和輔助服務的伺服器時，應盡可能採用自動化腳本處理。使用 DevOps 工具和容器化，可以讓部署新伺服器的工作更加輕鬆。
- **測試**：在建置程式的流程裡加入單元測試，使用自動化的瀏覽器測試工具來找出每個版本的重大變更，藉由自動化的滲透測試工具協助，在駭客攻擊之前先找出及修補安全漏洞。

根據經驗，作業說明文件裡會劃分成許多執行步驟的流程，都很適合交給自動化腳本或建置工具來執行。在第 13 章會看到許多安全問題是因為伺服器配置錯誤或部署的意外事故所引起的，減少開發生命週期裡的手動作業，將可降低發生這些問題的風險。

少數尚未被破解，即使優秀的安全專家所創造的加密演算法，也常被證明存在缺陷，因此，Web 開發人員表現出謙遜並遵循專家的指導是有意義的。

要謹慎考慮是否自己編寫解決方案的第二個領域是 session 管理。應該還記得第 4 章提到的 **session**，它在 Web 伺服器完成身分驗證後回傳給使用者，這個過程可以是在 cookie 設置 session ID，以便伺服器能從它的 session 儲存區查找與該使用者相關的資訊，或者實作用戶端的 session 保存機制，將整個 session 狀態寫在用戶端。

表面上看起來，為 Web App 實作 session 好像很簡單，實際上，session 管理要做得好並不容易，第 9 章將介紹駭客如何利用可預測或不夠隨機的 session ID 來挾持使用者的 session，以及如何利用不安全的用戶端 session 來提升權限。要實現安全的 session 管理絕非易事，導致 session 經常成為攻擊目標，建議直接使用 Web 伺服器內建的 session 實作機制，不要嘗試自己另外編寫一套。

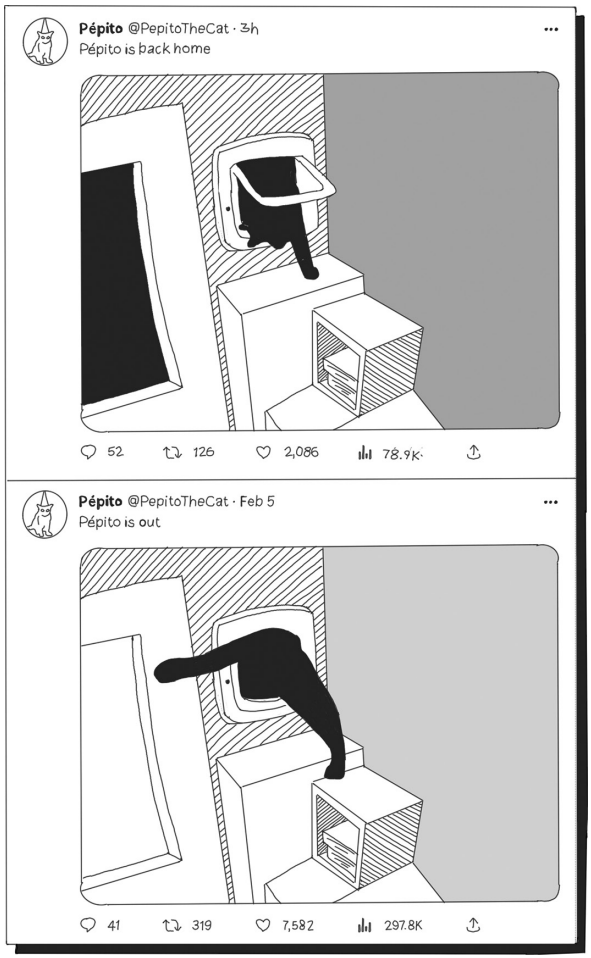
5.5 保留稽核軌跡

知道誰在什麼時候做了什麼事，是確保 Web App 安全的關鍵，就像有安全意識的機構會留存訪客日誌一樣，當然，也應該追蹤應用程式和流程的重要活動軌跡。當發生資安事件時，稽核軌跡可以協助找出事件期間的可疑活動，也是事後鑑識時，釐清事件始末的關鍵。以下是使用稽核軌跡強化應用程式安全的常見作法：

- **程式碼變更**：程式碼庫（codebase）的更新歷程應該儲存在源碼控制系統裡，以便查看由誰變更了哪些程式碼；將變更後的程式碼儲存至源碼控制系統時，應該執行數位簽章。
- **部署**：應該記錄每部伺服器所部署的程式版本，以及這些版本在何時由何人發行出去的。
- **HTTP 存取日誌**：Web 伺服器會記錄網站上的 URL 被誰存取、HTTP 回應代碼、來源 IP 和 HTTP 動詞及存取時間等資料，這些日誌檔若含有**個人識別資訊**（PII），請確認保存方式符合當地法規要求。

- **使用者行為**：應記錄使用者的重要操作，如註冊、登入和編輯內容，以供客服人員和技術服務人員在必要時參考。如果要求使用者提供真實姓名，則揭示 PII 附帶條款就具有雙重意義。
- **資料更新**：對資料庫紀錄的變更應該有稽核軌跡，至少要保留每筆紀錄的建立和修改時間，對於機敏資料，應該保留執行程序或使用者對此資料的異動紀錄。
- **管理活動**：應該記錄管理權限存取系統的活動，以便檢測異常行為或意外變更事件。
- **SSH 存取日誌**：如果允許遠端人員透過 SSH 存取伺服器，除了在這部伺服器上保留存取日誌外，也要將日誌紀錄傳送到其他地方集中保管。

廣受喜愛的 Twitter/X 帳戶 @PepitoTheCat，在那隻貓進出門簾時，會為牠拍張照片，這個帳戶是稽核軌跡的不錯例子，如果讀者想要知道 Pepito 的下落，可以去查看這個帳戶的貼文。



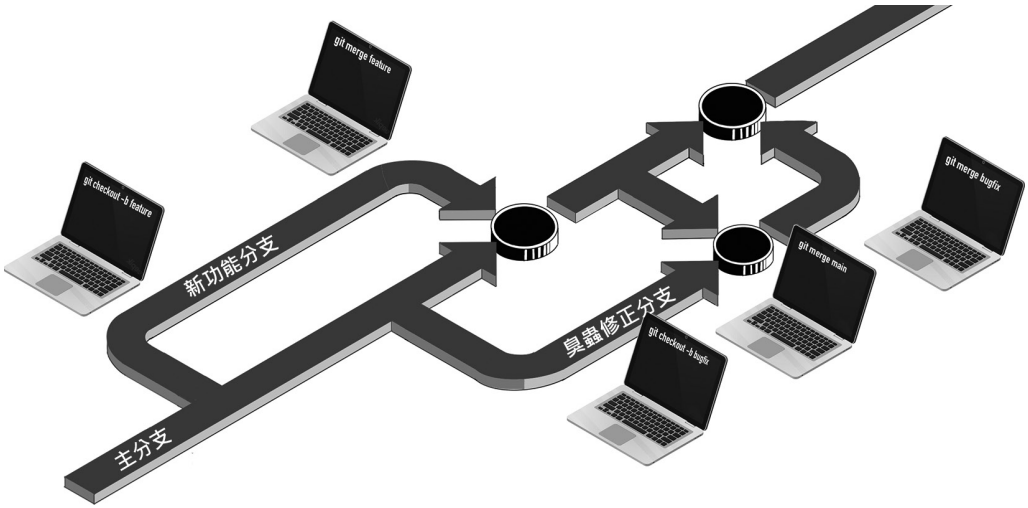
5.6 撰寫安全的程式碼

到目前為止，本章提供的建議主要屬於組織管理層面，這些建議都不錯，然而，讀者會選讀本書，代表你的主要工作可能是開發程式，而不是為機構裡的人員分配角色，還是應該花些時間討論應用於**軟體開發生命週期**（SDLC）的安全原則，亦即撰寫和發行程式碼的安全過程。

實施源碼控制

源碼控制是開發人員的重要工具，利用 `git` 之類工具追蹤碼庫（codebase）的變更，對於記錄何時為 Web App 加入新功能至關重要。

若團隊成員遵循 GitHub 作業流程（由同名公司推廣），會為正在開發的新功能建立分支，並在功能完善，準備發行時，將它們合併回主分支。合併時點是審查程式碼的好機會，應該要求團隊成員謹慎審查要合併回主分支的任何內容。



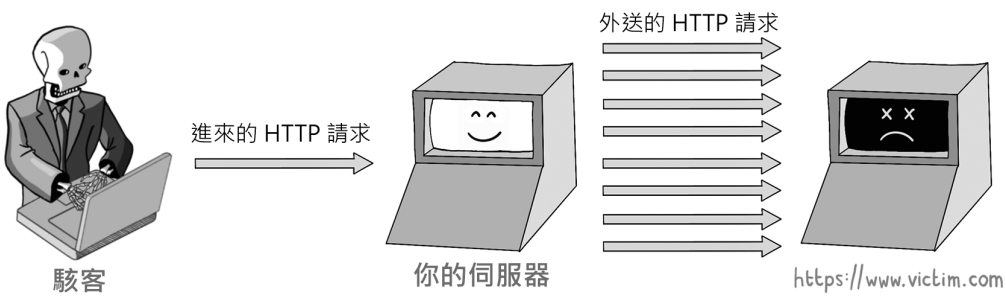
14.1 伺服器端請求偽造（SSRF）

網際網路是一種用戶端 - 伺服器模型，瀏覽器和行動 App 等用戶端向 Web 伺服器發送 HTTP 請求，Web 伺服器則提供 HTTP 回應作為回報，但有時伺服器也需要向其他 Web 伺服器提出 HTTP 請求，把自己當作用戶端。你的 Web App 可能基於多種原因而發出 HTTP 出站請求，包括：

- 呼叫外部 API 來處理付款、發送電子郵件、查找資料或執行身分驗證。
- 向 CDN 或雲端儲存體讀寫資料。
- 透過 Webhooks 向用戶端程式發送重要通知。
- 為了處理圖片上傳請求而存取託管圖片的遠端 URL。
- 透過尋找網頁 HTML 裡的**開放社交關係圖**（open graph）的詮釋資料來建立連結預覽。

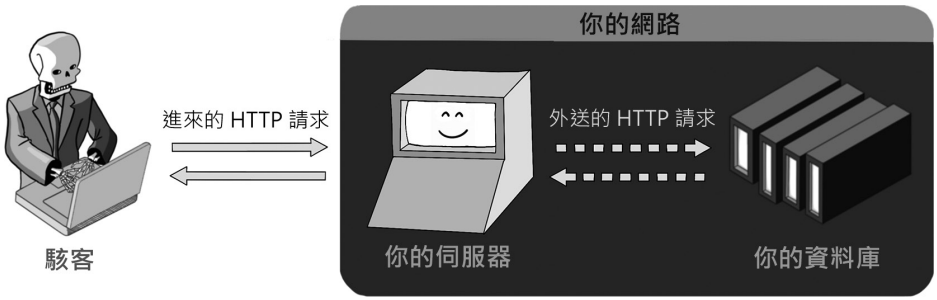
這些都是可能出現的應用場景，但如果 Web App 會因惡意用戶端程式的觸發而對任意 URL 發出 HTTP 請求，就可能形成**伺服器端請求偽造**（SSRF）漏洞。

駭客有很多種利用 SSRF 漏洞的方式。首先，可以利用這些漏洞對受害者發動**阻斷服務**（DoS）攻擊，嘗試透過 HTTP 請求淹沒受害者，使其應用程式離線。



對於這種場景，駭客躲在你的應用程式後面，所有攻擊流量都來自你的伺服器，如果駭客向伺服器發出一個請求，就能觸發伺服器向受害者發出多個請求，因而放大攻擊力道，則這種方法就特別有效。

SSRF 漏洞的第二個常見用途是用來偵測內部網路，由於 Web App 通常部署於特權環境，可能有權存取未暴露於網際網路的資料庫和快取等機敏資源，駭客便可透過 SSRF 漏洞去探測此類資源並嘗試取得它們的控制權。



無可否認地，駭客必須運氣夠好，才能讓這種攻擊可以成功。一般需要透過 HTTP 回應將錯誤訊息傳回給駭客，因此要有效執行此種攻擊方法，必須伴隨資訊洩露的錯誤訊息。駭客擅長混搭安全漏洞，且隨著軟體系統老化，漏洞多年未被發現的情況並不少見，直到合適的組合出現後才被利用。

管制伺服器可造訪的網域

緩解 SSRF 漏洞的最簡單方法是，避免直接以原始 HTTP 請求所提供的網域作為伺服器發出 HTTP 請求的對象。假設你的伺服器向 Google 地圖 API 發出請求，則每個出站 HTTP 請求的網域名稱應該定義在伺服器端的程式碼裡，而不是從傳入的 HTTP 請求中提取，安全呼叫 API 的簡單方法是使用 Google 地圖的**軟體開發套件**（SDK），在 Java 的範例如下：

```
DirectionsResult result =
    DirectionsApi.newRequest(ctx)
        .mode(com.google.maps.model.TravelMode.BICYCLING)
        .avoid(
            RouteRestriction.HIGHWAYS,
            RouteRestriction.TOLLS,
            RouteRestriction.FERRIES)
        .region("au")
        .origin("Sydney")
        .destination("Melbourne")
        .await();
```

SDK 會以你的身分安全地建構 HTTP 請求，確保駭客無法控制想存取的網域。常見的 API 都有 SDK，由 API 擁有者發行或由第三方維護，這些套件通常可以透過依賴項管理員取得，並能防止 SSRF 漏洞滲透到你的程式裡。

只為真實使用者發出 HTTP 請求

有些網站**確實**需要向不特定的第三方 URL 發出請求，例如，社群媒體網站可共享網路連結，並經常從這些 URL 讀取 open graph 的詮釋資料以產生連結預覽（當使用者在社群媒體頁面分享連結時，社群媒體利用此功能產生縮圖和標題），若你的 Web App 也提供類似服務，必須保護自己免受 SSRF 攻擊：

- 只在通過身分驗證的使用者操作時，伺服器才為該使用者發出 HTTP 請求。
- 對於社群媒體網站，應限制使用者在某段時間內可以分享的連結數量，以避免此服務被濫用。
- 建議在使用者分享連結時，要求他們輸入正確的 CAPTCHA 驗證碼。

檢查要造訪的 URL

為防止駭客刺探網路架構，應該確保伺服器只能向可公開的 URL 發送存取請求，可透過下列作法實現此規則：

- 與網路團隊討論如何管制 Web 伺服器可存取的內部網路裡之伺服器。
- 驗證提供的 URL 含有適當的 Web 網域名稱，而非使用 IP 位址。
- 禁止使用非標準端口的 URL。
- 確保所有 URL 均使用 HTTPS 存取，並使用有效的憑證。

下列是以 Python 實作檢查的程式範例：

```
import requests
from urllib.parse import urlparse
from IPy import IP

def validate_url(url):
  parsed_url = urlparse(url)
```

```
if parsed_url.scheme != 'https':
  return False, "此URL未使用HTTPS"

if parsed_url.port and parsed_url.port != 443:
  return False, "此URL未使用標準HTTPS連線端口"

if not parsed_url.hostname:
  return False, "此URL沒有指定網域"

try:
  IP(parsed_url.hostname)
  return False, "主機名稱不能是IP位址"
except ValueError:
  pass

try:
  response = requests.get(url, verify=True)
  response, "有效的連線加密憑證"
except requests.exceptions.SSLError:
  return False, "此URL無法提供有效的TLS加密憑證"
except requests.exceptions.RequestException:
  return False, "無法連線此URL"
```

注意

道行高深的駭客能夠設定指向私有 IP 的 DNS 紀錄，因此只驗證 URL 是否具有網域是不夠的。

使用網域黑名單

如果 Web App 須向不特定第三方的 URL 發送 HTTP 請求（也許是提供連結共享服務），可以考慮維護一份網域黑名單，限制伺服器端向這些目標發送請求，這種作法可以阻斷駭客觸發惡意請求，且能避免產生任何意圖的 DoS 攻擊。

維護黑名單的工作可能很繁瑣，想必也不會以手工打造，最好是利用可信任第三方維護的黑名單（例如 **https://github.com/StevenBlack/hosts**）會比較省時省力。

14.2 電子郵件詐欺

HTTP 並非駭客利用的唯一網路協定，駭客會利用**簡單郵件傳輸協定**（SMTP）主動寄送惡意電子郵件，透過網路釣魚攻擊來騙取身分憑據，或說服受害者下載惡意軟體。

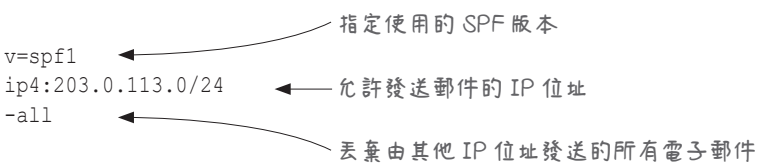
比爾蓋茲在2004年宣稱此類垃圾郵件攻擊將在「兩年後」得到解決，很不幸，他的預言並沒有實現。

在耐心等待比爾蓋茲完成他的任務時，還是需要採取必要措施，確保使用者能夠區分你的 Web App 所發送的合法電子郵件，還是駭客冒充 Web App 發送的惡意電子郵件，這項措施涉及兩個層面：公布哪些 IP 位址能夠以你的網域傳送電子郵件；讓收件者能夠檢測電子郵件在傳輸過程中是否遭到竄改。

寄件者策略框架（SPF）

透過 DNS 紀錄列出 **SPF**，明確指出哪些伺服器可從你的網域發送電子郵件，此方法會將偽裝成你的網域所發送的電子郵件標記為惡意行為者，即來自**欺騙**（spoof）的郵件網域。

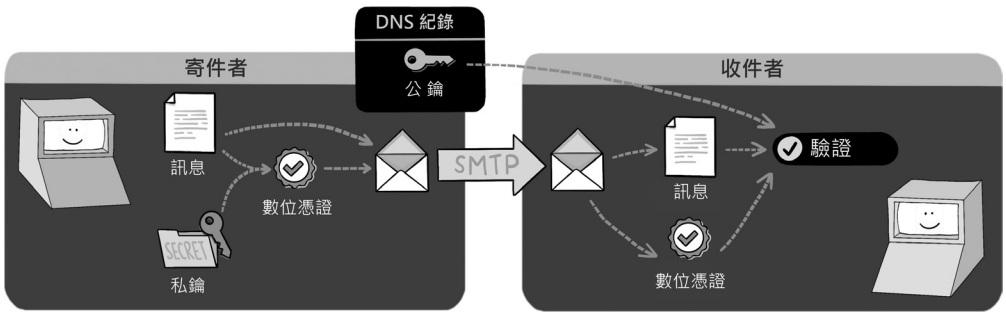
假設正確的網域是 **example.com**，只有 203.0.113.0 至 203.0.113.255 範圍內的 IP 位址會寄送電子郵件，便可在 DNS 的 `TXT` 紀錄新增下列文字來實作 SPF：



SMTP 是使用**傳輸控制協定**（TCP）在網際網路上傳輸電子郵件，要偽冒 IP 位址遠比偽冒 SMTP 的 `From`（寄件者）標頭要困難得多，SMTP 沒有任何機制可供收件人驗證寄件者的身分，因此，SPF 為電子郵件用戶端提供一種檢測詐騙電子郵件的簡單方法。

網域金鑰識別郵件（DKIM）

讀者可以實作 DKIM 來防止電子郵件在傳送過程中被竄改，方法是在 DNS 紀錄加進一把公鑰，且發送的每封電子郵件都使用配對的私鑰進行簽章。電子郵件用戶端在收到電子郵件時，可以重新計算簽章並拒絕簽章不符的郵件，因為，簽章不符表示郵件遭到竄改。



要為電子郵件增加 DKIM 標頭，並在發送時產生 DKIM 簽章，會比實作 SPF 更複雜，幸好郵件伺服器會為你處理大部分工作，讀者若迫不及待，可以先略過下一節，直接跳到「實施步驟」小節，然而，在討論該小節之前，筆者打算利用一些篇幅討論最後一個問題：那些未通過 SPF 或 DKIM 檢測而被拒絕的電子郵件，會發生什麼情況？

網域郵件身分驗證、回報及確認（DMARC）

要怎麼處理被拒絕的電子郵件是由 DMARC 策略決定（全名真的很長），以 **example.com** 網域的策略為例，如下所示，即子網域 **_dmarc.example.com** 上的 TXT 紀錄：

```
v=DMARC1;
p=quarantine;
rua=mailto:admin@example.com"
```

指定實作的 DMARC 版本

指示隔離（而非完全拒絕）電子郵件

向何處發送匯總報告（描述有多少電子郵件被隔離或拒絕）

指定 DMARC 策略，也能用以檢測因設定不當而被誤標為惡意的電子郵件。