Open Banking API 存取控制風險探討

何承遠[1]、毛翊蓁[2]、蔡宇倫[1]、陳柏言[1]、楊佳真[1]

[1] 國立台灣大學資訊管理學系

[2] 國立台灣大學資訊工程學系

{tommyho, B10902103, B08705056, R11725004, R11725040}@ntu.edu.tw

壹、 緒論

一、 研究背景與動機

（一） 研究背景

在科技快速進展與 COVID-19 疫情的雙重推力下，數位轉型已成為各行業的發展重點，金融業也在數位化與科技創新中快速演進。金融創新教父布雷特・金恩（Brett King）於《Bank 4.0》一書中指出，隨著技術與使用者行為轉變，銀行將從實體存在的服務提供者轉型為數位時代中的無所不在（Banking Everywhere, Never at a Bank）服務平台（孫一仕等譯，2018）。「開放銀行」（Open Banking）便是在這樣的背景下迅速興起，作為實現新型態金融服務的核心架構之一（IBM 藍色觀點，2018）。

全球各國陸續推動開放銀行政策，以促進資料共享與產業創新。台灣自 2018 年起亦跟進國際潮流，開啟開放銀行制度規劃，逐步建置開放 API 機制（張婉君，2021）。2020 年金管會公布的「金融科技發展路徑圖」提出八大發展主軸，涵蓋法規調適、資料共享、技術創新等項目，並於 2023 年推出 2.0 版本（蘇文彬，2023），進一步深化政策實踐。

根據政治大學與勤業眾信於 2021 年聯合發表的《台灣金融科技趨勢展望》報告指出，在疫情與數位浪潮衝擊下，金融業不僅需加速數位轉型，更需重塑金融服務策略，其中「身份驗證」與「資料共享」被視為推動創新的兩大關鍵技術（遠見雜誌整合傳播部，2021）。此觀點亦反映於監理機關與產業界對 API 經濟與新金融生態的高度期待。

然而，在開放銀行制度不斷拓展的同時，資安風險問題亦浮上檯面。當 API 成為金融資料交換與服務提供的核心通道，存取控制的安全性將直接關係到整體金融體系的信任基礎。若授權機制設計不當、驗證方式存在漏洞，便可能導致未授權存取、敏感資料外洩、甚至交易偽冒等嚴重後果（李靜宜，2020）。

（二） 研究動機與問題界定

本研究的發想動機，來自於開放銀行在實務推動過程中所面臨的資訊安全挑戰，尤其集中於 API 存取控制中的授權與身份驗證技術。此技術不僅牽涉使用者資料權限與存取範圍，也關係到交易安全與法遵要求。隨著開放銀行第二階段推動（提供帳戶資訊服務與交易啟動服務），第三方服務業者（TSP）對金融資料的請求與存取日益頻繁，如何確保這些行為具備高度安全性與合規性，是業界與主管機關的關注重點。

目前國際主流標準多以 OAuth 2.0 作為授權機制，並搭配 OpenID Connect（OIDC）進行身份驗證與 claims 管理。但實務上常見包括 access token 洩漏、redirect URI 被濫用、中介人攻擊（MITM）、token 存活期設定不當等資安問題。為強化這些基礎協定的安全性，OpenID Foundation 進一步推出 Financial-grade API（FAPI）規範，導入更嚴格的安全要求與技術指引，作為金融情境中高保護需求的替代方案。

然而，儘管 FAPI 具備一定的保護力與規格設計，目前仍缺乏系統性的研究分析其是否真正能解決過往協定的漏洞。特別是在不同地區（如台灣、英國 OBIE、歐洲 Berlin Group）對協定採用與安全配置方式不盡相同之下，其相容性與實用性仍有待驗證。

本研究旨在釐清以下三大問題意識：

1. OAuth 2.0、OIDC、FAPI 三大協定是否能有效回應既有資安挑戰？其演進是否實質提升風險防護能力？

2. 台灣現行標準與國際開放銀行實作（如 OBIE、Berlin Group）之間，是否在技術與政策層面上具備相容性與互通性？

3. 面對「零信任架構」與「去密碼化」等資訊安全新趨勢，這些協定是否具備足夠的延展性與長期可行性？

在研究設計上，本文不僅以文件比較與風險分析為基礎，更聚焦於實務上的差距與落差，提出制度面與技術面之具體建議，作為我國後續政策調整與實作參考。

二、 研究範圍與目的

本研究以「開放銀行 API 存取控制機制」為研究主軸，探討 OAuth 2.0、OIDC 與 FAPI 三項核心協定在授權與身份驗證過程中的應用與風險。具體研究範圍與目的如下：

（一）分析現行協定架構與應用情境：

整理 OAuth 2.0、OIDC 與 FAPI 的核心功能、架構設計與技術應用，釐清其在開放銀行系統中所扮演的角色與流程差異。

（二）評估潛在風險與實作困難：

針對實務常見資安風險進行盤點與解析，包括 redirect URI 濫用、token 洩漏、中介攻擊等場景，並比較協定間對這些風險的處理方式與保護能力。

（三）探討改善方向與制度建議：

從協定分析出發，提出可行的風險對應方案與標準強化方向，協助金融業者提升資安保護層級，也提供政府機關在政策擬定上的參考依據。

本研究致力於從開放銀行實務與資安挑戰出發，檢視目前標準的適用性與防護力，並透過邏輯清晰、內容嚴謹的比較分析，提出具體建議，以促進安全可控的開放金融發展。

三、 研究方法

本研究採用文獻分析法，以系統性方式蒐集與研究主題相關之國內外資料，包含正式出版文獻、官方技術文件、學術報告、政策規範、實作案例以及業界實務操作指引。透過對 OAuth 2.0、OIDC 與 FAPI 三項主流授權與驗證協定的比對分析，進一步探討其在開放銀行 API 存取控制中的應用情境與潛在風險。

研究過程將從以下三個層面進行：

（一）技術協定結構分析：

首先對 OAuth 2.0、OIDC、FAPI 的協定架構、核心流程、授權類型與安全保護機制進行歸納與比較，釐清其概念基礎與實務差異，並對各協定支援的功能層面進行統整。

（二）資安風險評估：

其次針對上述協定在實務部署中的弱點與已知攻擊案例進行整理與分析，

聚焦於如 access token 洩漏、中介人攻擊、redirect URI 濫用、token 存活期配置不當等問題，從使用者端、服務端與攻擊者視角建構可能風險場景。

（三）制度與實作落差觀察：

最後，從台灣與歐美等地的開放銀行政策與標準出發，對照台灣財金資訊公司（FISC）所制定之開放 API 平台架構與國際標準（如 OBIE、Berlin Group）之異同，辨識制度設計與實際部署間可能的落差與挑戰，並整理可行性強化方向。

透過上述層層交錯之技術分析與政策對照，期能提出系統性的實務觀察與風險判讀，提供未來政策推進與業界應用的重要參考依據。

貳、 開放銀行制度與技術基礎：發展、機會與挑戰

一、 開放銀行（Open Banking）與開放 API（Open API）概述

開放銀行是一種金融服務模式，它促使銀行和第三方服務業者合作，透過 API 共享金融數據，以提供個人化且多元化的金融服務。其中，Open API 是開放銀行運作中非常重要的技術，它讓不同的銀行和第三方業者能夠進行資料共享和交換，並且讓第三方業者能夠開發各種應用程式或服務。簡而言之，開放銀行通過 Open API 促進金融業的創新和協作，為消費者帶來更豐富的金融體驗（國立政治大學金融科技研究中心，2019）。

Open API 指的是使用開放標準來制訂的 API，以提供 TSP 業者明確的 API 規格標準，如此可以減少 TSP 業者與銀行之間串接的阻礙，使推出新的金融服務時可大幅降低重新設計 API 的難度。因此 Open API 是建立開放銀行金融生態系最關鍵的環節，讓平台之間的數據和服務可以重新組合，創造新的商業模式與價值，間接促成更強大的金融服務生態系。故 Open API 主要有兩大特色，其中第一個特色為「標準化」，即格式統一，這樣就能夠使用相同的方式來存取和處理資料。第二個特色是「規模化」，也就是讓更多潛在客戶使用第三方服務，來擴大銀行客戶群、增加銀行收益（國立政治大學金融科技研究中心，2019）。

二、 國內開放銀行進展與階段

隨著開放銀行的趨勢，台灣也進行了相應的規劃。採用不修法的香港模式，讓銀行與第三方服務公司（以下簡稱 TSP 業者）合作推動。金融監督管理委員會（金管會）核定了開放銀行的發展進程，分成「公開資料查詢」、「消費者資料查詢」、「交易面資訊」三大階段，並委由財金資訊股份有限公司制定技術與資安標準（高敬原，2019）。以下將對這三個階段進行詳細說明：

（一）第一階段「公開資料查詢」

第一階段已於 2019 年 10 月 16 日上線，並於 2019 年 06 月完成該階段的技術與資安標準制定。一階段的主要目標是開放非交易相關的金融產品或服務資訊查詢，並不涉及消費者個人資料。這些資訊包括新台幣或外幣存款利率、外幣匯率、分行和 ATM 據點、房貸利率、定存利率、貨幣匯率、信用卡產品等等。舉例來說，民眾可以透過第三方業者的應用程式直接比較房貸利率、信用卡、外幣

匯率等資訊，而不需要逐一進入各個網頁查詢。總共有 25 家金融機構和 7 家第三方服務業者（TSP）加入了開放程式介面（Open API）(姚惠茹，2021）。

表 1

第一階段金融機構與 TSP 參與概況

對象	金融機構	第三方服務業者（TSP）
第一階段參與機構與業者	合庫、華南、國泰、兆豐、臺企銀、元大、玉山、土銀、第一、彰銀、富邦、永豐、中郵、台新、瑞銀、新光、星展等 25 家。	集保公司「集保 e 存摺」、遠傳電信「friDay 理財+」、麻布記帳、CW Money、錢管家「保險小存摺」APP、好好投資 FundSwap 基金下單平台等 7 家。

註：

修改自“開放 API 服務”，財金資訊股份有限公司，2023。

（二）第二階段「消費者資料查詢」

目前國內的開放銀行已進入第二階段，該階段的主要目標是開放消費者資訊查詢。在經過客戶同意的前提下，第三方服務業者（TSP）提供帳戶整合服務，其中包括銀行帳戶的餘額和交易明細資訊、信用卡的歷史帳單資訊以及消費者的個人資料。這使得消費者能夠將在 A 銀行填寫過的個人金融資料直接應用於 B 銀行的帳戶申請中，而無需重複填寫相同的資料（姚惠茹，2021）。

表 2

第二階段金融機構與 TSP 合作概況

TSP 業者	第二階段合作業者	上線時間
集保公司「集保 e 存摺」	華南、元大、中信、兆豐、第一、國泰	2021 年 04 月 27 日
遠傳電信「friDay 理財+」	遠銀、中信、台新	2021 年 01 月 15 日

註：

修改自“開放銀行第二階段 10 家參戰”，朱漢崙，2020，*工商時報*。

（三）第三階段「交易面資訊」

第三階段的主要目標是開放交易資訊查詢。在獲得客戶同意後，透過第三方服務業者（TSP）將帳戶整合，進一步透過應用程式連結帳戶並進行消費支付，同時也能夠調整不同帳戶之間的資金，包括貸款清償和扣帳授權等操作。然而，由於涉及資金移轉，故第三階段需要更高規格的資訊安全措施，目前正積極努力中（蘇文彬，2023）。

開放銀行的第二和第三階段，涉及消費者的個人資訊和交易資料，因此需要嚴格的技術和資訊安全標準（高敬原，2019）。

三、 開放銀行帶來的機會與金融科技轉型

金融機構和第三方業者合作，並透過 API 共享數據，為開放銀行帶來許多優點。以下說明開放銀行對三者所帶來的好處：

（一）傳統銀行：透過與第三方業者的合作，傳統銀行能夠串接 Fintech 公司的創新技術，解決既有營運痛點，並專注於金融服務的創新和研發，從而為客戶提供更多元的產品和服務，改善客戶體驗，提升客戶黏著度。這種合作模式使得傳統銀行能夠拓展金融服務的範圍，觸及過去無法涉足的領域，並提供更個人化、更多元、更貼近市場需求的金融產品與服務，同時擴展服務的客群，從而增加營業利潤。

（二）第三方服務業者（TSP）：第三方服務業者能夠運用創新科技，協助解決消費者的痛點。同時，透過資料共享，也能夠改善金融數據壟斷的問題，並推動金融科技的發展，提高銀行服務的價值。這樣的合作模式能夠提供更貼近消費者需求的服務，對於經營關鍵客戶和擴大普惠金融都具有幫助，並且也有助於建立一個有利於新創 Fintech 發展的環境。

（三）消費者：開放銀行所帶來的變革，能夠讓消費者獲得「資料數據主導權」，並體驗到更快速、敏捷和多元化的金融和增值服務，滿足他們的個人需求。

開放銀行為消費者、傳統銀行和第三方服務業者創造了三贏的局面。藉由安全共享金融數據，進行跨領域的資料和服務整合，實現了一站式開放金融生態系統。在建立開放金融生態系的過程中，開放銀行只是個開始，未來甚至會擴展至

開放證券投資和開放保險服務，形成更完整的開放金融生態系（國立政治大學金融科技研究中心，2019）。

圖 1

Open Banking 機會與優勢

註：

本研究自行繪製（參考自"從開放銀行走向開放金融"，國立政治大學金融科技研究中心，2019。）

四、 開放銀行實施的資安挑戰與風險背景

開放銀行雖能促進更多創新多元的金融服務，並為金融生態帶來新樣貌，然而開放銀行的發展可能會伴隨一些資安風險，主要有以下四個風險：

（一） API 管理及安控風險：API 管理及安控風險涉及開放銀行中 API 的管理和安全控制，其關注確保 API 的正確管理，包括訪問控制、身份驗證、授權和 API 安全性。

（二） 資訊安全風險：資訊安全風險涉及整個開放銀行系統和數據的安全性，其關注確保系統的機密性、完整性和可用性，以防止未經授權的訪問、數據泄露和其他安全威脅。

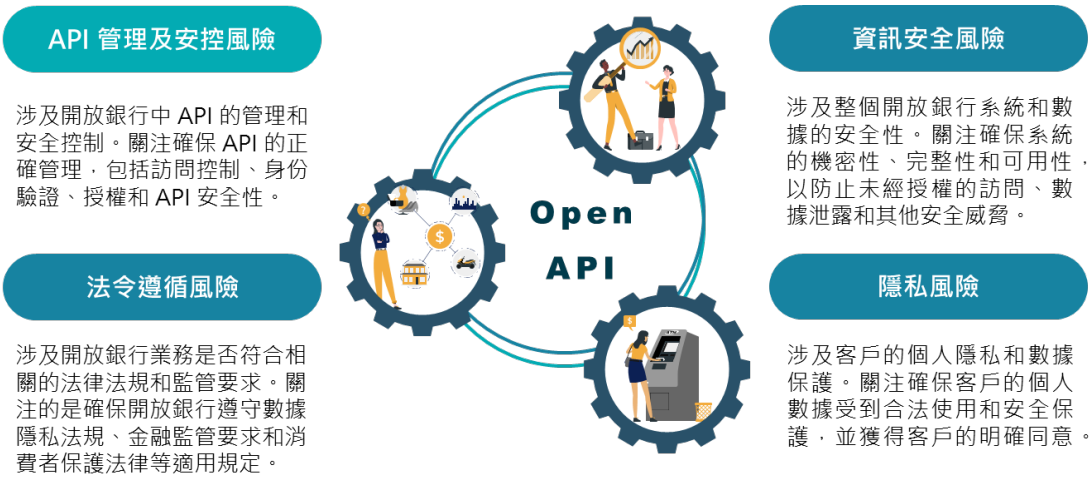
（三） 法令遵循風險；法令遵循風險涉及開放銀行業務是否符合相關的法律法規和監管要求，關注的是確保開放銀行遵守數據隱私法規、金融監管要求和消費者保護法律等適用規定。

（四） 隱私風險：隱私風險涉及客戶的個人隱私和數據保護，其關注確保客戶的個人數據受到合法使用和安全保護，並獲得客戶的明確同意。

在 API 管理及安控風險的議題中，「資料授權和身份驗證」風險涉及到消費者的資料隱私和安全，對於開放銀行的任何參與者而言非常重要，並且備受關注（林岑俞、吳家瑄，2019）。

圖 2

Open Banking 四大風險與挑戰



註：

本研究自行繪製（參考自"邁向開放金融打造穩定安全的資料共享環境"，林岑俞、吳家瑄，2019。）

參、 開放銀行授權與驗證技術標準評析

一、 OAuth 2.0、OIDC 與 FAPI 協定比較與應用解析

為確保分析邏輯之完整與系統性，本研究首先針對 OAuth 2.0、OIDC 與 FAPI 三項授權與驗證協定進行橫向比較，涵蓋技術設計、功能目的、安全風險對應機制及當前實務應用等層面。完成比較後，進一步分別解析各協定的架構特點與潛在資安挑戰。

表 3

OAuth 2.0、OIDC 與 FAPI 在開放銀行存取控制中的比較分析

比較項目	OAuth 2.0	OpenID Connect（OIDC）	Financial-grade API（FAPI）
標準制定機構	IETF（RFC 6749）	OpenID Foundation	OpenID Foundation（FAPI WG）
設計目的	資源存取授權 (Authorization delegation）	身分驗證（Authentication）+ 基本授權	高風險金融交易之強化授權與身份保護
使用場景	第三方應用程式代表用戶存取資源（如授權 Google Drive 給 App）	附加身份驗證功能（如單一登入 SSO）	金融場景下資料與交易之高度安全存取（如銀行帳戶資訊與轉帳）
核心流程	Authorization Code / Implicit / Client Credentials / Resource Owner Password Credentials	基於 OAuth，加入 ID Token 與 Discovery/ Registration	基於 OAuth + OIDC，採更強認證/授權約束（如 PAR、JARM、MTLS、DPoP）
主要身份資訊處理	無（僅提供 access token）	傳回 ID Token（JWT 格式）描述使用者身分與 claims	採明確使用者身份驗證機制，結合可信身份來源與聲明控制
Token 傳輸保護	依實作決定，通常透過 HTTPS	同上，但可結合 nonce, state 等防重放參數	強制使用 MTLS 或 DPoP，避免 token 重播與中介人攻擊
針對攻擊類型防護能力	較弱，易受 token 洩露、redirect 攻擊、中介人攻擊	增強型防護，但實作不當仍可能有 impersonation、replay 等風險	引入 cryptographic binding、PAR、JARM 等手段，有系統性防護設計

密碼移除機制支援	否	否（部分擴充支援 FIDO2）	支援與 FIDO2、WebAuthn 整合之無密碼機制（需另建 eKYC 流程）
實務部署複雜度	最低，支援度廣、模組成熟	中等，需要額外 ID Provider 與 metadata 配置	高，涉及證書管理、key rotation、API registration 等額外需求
與開放銀行相容性	基礎協定，需加強補強保護	常用於登入/claims 驗證，但不足以獨立應用於金融交易	完全符合 OBIE、Berlin Group 等開放銀行框架之資安要求
國際應用現況	廣泛用於一般 Web 應用與第三方授權場景	廣用於企業 SSO、部分銀行登入系統	英國 OBIE、澳洲 CDR、日本 FSA 等均採 FAPI 為標準，台灣部分金控試行中
標準穩定性與未來展望	穩定，更新為 OAuth 2.1 正式草案	穩定，逐步與 FIDO、VC 整合	發展中（FAPI 1.0 Finalized，2.0 草案進行中），與可信身分證明、EUDI Wallet 整合為趨勢

二、 OAuth 2.0 授權協定架構與適用情境

（一） OAuth 2.0 定義

OAuth 2.0 是一個開放的資料授權框架（或稱協議），提供資源擁有者允許並授權第三方應用程式，取得儲存在的資源伺服器的使用者資料，可支援網站、桌面和手機等應用程式的開發（D. Hardt, Ed., 2012）。

（二） OAuth 2.0 發展歷程

在傳統用戶端-伺服器認證方法中，用戶端（第三方應用程式）需要使用資源擁有者的憑證（Credentials），向伺服器發送存取受限（受保護）資料的請求。然而，為了讓第三方應用程式存取受限資料，資源擁有者不得不向第三方共享其憑證，其中便會產生一些問題和限制，包含：

1. 第三方應用程式需要保存資源擁有者的憑證，通常以「明文」形式保存憑證密碼，以便未來使用。

2. 伺服器需要支援密碼認證，然而密碼本身就存在安全弱點。
3. 第三方應用程式取得的存取權限過於廣泛，使得資源擁有者無法限制存取時間和存取範圍。
4. 資源擁有者無法單獨撤銷個別第三方的存取權限，必須撤銷所有第三方的存取權限，並且只能透過更改該第三方的密碼來執行。
5. 若第三方應用程式受到侵害，使用者的密碼和受保護的資料就會受到威脅。

為了解決這些問題，OAuth 協議出現。OAuth 引入了授權層概念，將用戶端和資源擁有者的角色區分開來。在 OAuth 中，用戶端須向資源擁有者發出存取受保護資料的請求，而資料由資源伺服器保管。隨後，用戶端會獲得一組與資源擁有者不同的憑證，稱為存取權杖（Access Token），該權杖包含了特定的範圍、時長和其他存取屬性。這些存取權杖由授權伺服器在獲得資源擁有者的批准後，才發放給用戶端，之後用戶端便使用存取權杖來存取由資源伺服器保管的受保護資料。

簡而言之，OAuth 解決了傳統的用戶端-伺服器認證模型中存在的問題，引入了授權層概念，將用戶端的角色和資源擁有者的角色區分開來。通過使用存取權杖（Access Token）而不是資源擁有者的憑證，使第三方應用程式可以在未共享資源擁有者憑證的情況下，存取受限資料。

在 OAuth 的基礎下，OAuth 1.0 協議文件於 2010 年首次開放。隨著時間推移，對於身份驗證和授權的需求不斷發展，OAuth 2.0 協議文件於 2012 年開放，為 OAuth 1.0 的更新版本，具備更好的安全性和更廣泛的應用性（D. Hardt, Ed., 2012）。

（三） OAuth 2.0 協議授權流程

1. OAuth 2.0 授權流程中的四種角色

（１）資源擁有者（Resource Owner）

授予應用程式取得受保護資料的實體（人），通常為終端使用者（End-User）（D. Hardt, Ed., 2012）。在使用 Google 帳密登入其他應用程式時，資源擁有者為應用程式使用者。資源伺服器可能為 Google 的資源伺服器 API；若在開放銀行的場景中，資源擁有者可能為消費者。

（２）資源伺服器（Resource Server）

為保管受保護資料的伺服器，負責接受並回應帶有 Access Token 的應用程式請求（D. Hardt, Ed., 2012）。在使用 Google 帳密登入其他應用程式時，資源伺服器可能為 Google 的資源伺服器 API；若在開放銀行的場景中，資源伺服器可能為金融機構之資源伺服器 API。

（３）用戶端（Client）

為想要取得受保護資料的應用程式，其必須在取得資源擁有者的同意授權後，才能代表資源擁有者請求受保護的資料（D. Hardt, Ed., 2012）。在使用 Google 帳密登入其他應用程式時，用戶端為該應用程式。若在開放銀行的場景中，用戶端可能為 TSP 第三方服務業者的應用程式。

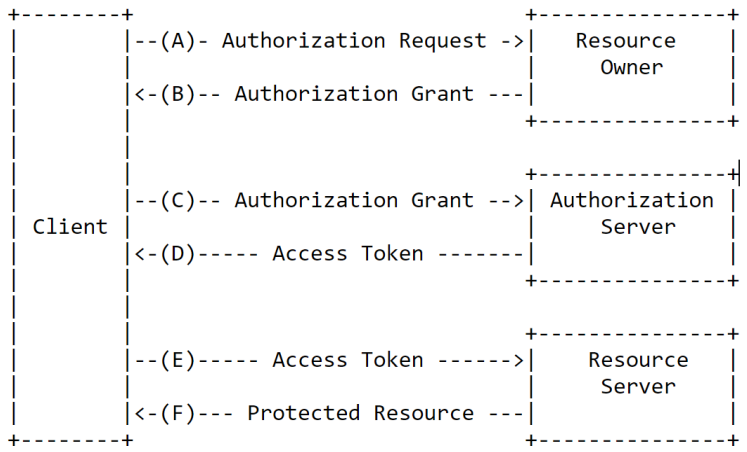
（４）授權伺服器（Authorization Server）

負責驗證資源擁有者和用戶端，並在取得資源擁有者的同意授權後，發送 Access Token 給用戶端（D. Hardt, Ed., 2012）。在使用 Google 帳密登入應用程式時，授權伺服器可能為 Google 的授權伺服器 API；若在開放銀行場景中，授權伺服器可能為金融機構之授權伺服器 API。

2. OAuth 2.0 授權流程和角色互動

圖 3

RFC-6749 文件提供之 OAuth 2.0 基本流程圖



註：

參考自"The OAuth 2.0 Authorization Framework（RFC-6749）"，D. Hardt, Ed.，2012。

（１）圖中（Ａ）步驟：Client（應用程式）可選擇直接向 Resource Owner（使用者）發送資料授權請求，或選擇以 Authorization Server（API）作為媒介，向 Resource Owner 發送授權請求。上圖是以 Client 直接向 Resource Owner（使用者）發送授權請求作為示範。

（２）圖中(Ｂ)步驟:接著,在 Client 向 Resource Owner 發送授權請求、Resource Owner 同意授權後,Client 會接收到一個 Authorization Grant。Authorization Grant 為應用程式可取得一定範圍（Scope）資料的「證明」，代表著使用者的同意授權。而 Authorization Grant 擁有不同的類型（Type），取決於 Client 請求使用者授權的方法及授權伺服器所支持的授權類型。

（３）圖中（Ｃ）步驟：Client 並不能直接拿著 Authorization Grant 向 Resource Server 取得資料，仍必須帶著 Authorization Grant 向 Authorization Server 請求換取 Access Token。

（４）圖中(Ｄ)步驟:在 Authorization Server 驗證完 Client 身分和 Authorization Grant 後，才會發放 Access Token 給 Client。

（５）圖中（Ｅ）步驟：Client 取得 Access Token 後，向 Resource Server（API）要求取得一定範圍的受保護資料。

（６）圖中（Ｆ）步驟：最後 Resource Server 驗證 Access Token 並且確認有效後，便提供資料給 Client。

以上是 OAuth 2.0 基本流程，然而實際上的授權流程有不同類型，但以上流程可協助理解 OAuth 2.0 的流程並進行變化。在 OAuth 2.0 協議中，包含 Authorization Code、Implicit、Resource Owner Password Credentials 和 Client Credentials 四種不同的授權流程（D. Hardt, Ed., 2012），分別對應不同的 Authorization Grant Type，開發人員可根據不同的使用情境和需求，彈性選擇合適的流程。由於四種流程當中，Authorization Code 流程的應用較為常見且最為安全，故本研究再以 Authorization Code 流程，進行 OAuth 2.0 流程的詳細介紹。

3. 用戶端註冊 OAuth 之流程及相關名詞說明

在介紹 Authorization Code 流程之前，本研究先介紹應用程式在使用 OAuth 協定開發授權流程時，註冊的方式及相關的名詞概念，以方便理解 Authorization Code 流程。

首先，應用程式（用戶端）在使用 OAuth 之前，需向 Authorization Server（API）的機構或平台（例如 Google 等）進行註冊，並提供應用程式名稱、應用程式網站和 Redirect URL（或稱 Callback URL），其中 Redirect URL 是使用者同意授權或拒絕授權用戶端後，Authorization Server 將使用者重新導向的位置。

接著，應用程式註冊完畢後，Authorization Server 的機構或平台會發送 Client ID 和 Client Secret 給應用程式，其中 Client ID 是 Authorization Server 用來辨識應用程式身份，為一個公開的字串，而 Client Secret 是 Authorization Server 用來驗證應用程式身份，為非公開字串，在應用程式和 Authorization Server 的機構或平台之間保密（Mitchell Anicas, 2021；Mike Huang, 2021）。

4. OAuth 2.0：Authorization code flow

本研究參考 Aaron Parecki、Mitchell Anicas 和 Mike Huang 對於 Authorization Code 流程的介紹進行 Authorization Code 流程的說明。

在 Authorization Code 流程中，Authorization Server 為 Client 和使用者之間的媒介，Client 不會直接向使用者請求授權，而是透過 User Agent（如網頁瀏覽器）將使用者定向到 Authorization Server 的位置後，Authorization Server 會對使用者進行身份驗證並詢問是否同意授權，若使用者選擇同意授權後，Authorization Server 將使用者帶回客戶端提供的 Redirect URL 位址，並將 Authorization Code 給予 Client。

雖然這些步驟多半在前端、使用者的 User Agent（如網頁瀏覽器）以頁面跳轉、Query String 方式傳遞訊息，但由於使用者是和 Authorization Server 進行驗證，不須將其憑證給予 Client，因此確保了一定的安全性。而後續在 Authorization Server 驗證 Client、發送 Access Token 和向 Resource Server 取得資料的步驟上，通常以後端、POST 方式進行訊息傳遞，不通過前端、使用者的 User Agent（如網頁瀏覽器），以此避免較為機密的資訊洩漏出去。因此，Authorization Code 流程提供了較高的安全性較高。

以上為 Authorization Code 流程的初步介紹，接下來本研究詳細說明 Authorization code 流程及步驟：

（１）使用者在應用程式（Client）中，點擊授權相關連結後，Client 會透過使用者的 User Agent，將使用者導向 Authorization Server（API）的位置。以下為範例連結：

圖 4

範例連結圖

```
https://authorizor-server.com/oauth2/authorize?
response_type=code&
client_id=549751264684113165 46&
redirect_uri=CALLBACK_URL&
scope=profile&
```

其中

client_id=5497512646841131654 6：設定為 Client 註冊時取得的 Client ID，以讓 Authorization Server 進行 Client 的身分辨識

redirect_uri=CALLBACK_URL：設定為 Client 註冊時提供的 Callback URL，讓 Authorization Server 在使用者同意或不同意授權後，將使用者重新導向 Callback URL

response_type=code：授權方式者設定為 authorization code grant，即採用 Authorization Code 流程

scope=profile：設定應用程式的資料存取範圍，如只能存取使用者個人資訊

註：

參考自

"What is the OAuth 2.0 Authorization Code Grant Type?"，okta（by Aaron Parecki），2018。

"An Introduction to OAuth 2"，DigitalOcean（by Mitchell Anicas），2021。

"[筆記] 認識 OAuth 2.0：一次了解各角色、各類型流程的差異"，Medium（by Mike Huang），2021。

（２）當使用者前往 Authorization Server 後，須登入該 Authorization Server 之平台（如 Google）進行驗證，接著 Authorization Server 會詢問使用者是否授權應用程式取得其隱私資訊

（３）使用者選擇同意授權後，Authorization Server 會透過使用者的 User Agent 將使用者重新導向 Redirect URL 並附上 Authorization Code，若使用者不同意，便會附上存取否定的資訊。Redirect URL 範例如下：

圖 5

Redirect URL 範例圖

```
https://example-application.com/redirect
?code=9E5fG7H3IjKlM6n8OpQ2rStUvW0xY1Z
```

註：

參考自

"What is the OAuth 2.0 Authorization Code Grant Type?"，okta（by Aaron Parecki），2018。

"An Introduction to OAuth 2"，DigitalOcean（by Mitchell Anicas），2021。

"[筆記] 認識 OAuth 2.0：一次了解各角色、各類型流程的差異"，Medium（by Mike Huang），2021。

（４）應用程式取得 Authorization Code 後，以 POST 方式向 Authorization Server 請求 Access Token，並提供 Client ID、Client Secret 等資訊給 Authorization Server

（5）如果應用程式身分驗證成功並且 Authorization Code 有效，Authorization Server 便回傳 Access Token 等資訊給應用程式，回傳資訊範例如下：

圖 6

回傳資訊範例圖

```
{
 "access_token": "3A9bDc7F5E2gH1IjK4lM6n8oP0qR2sT",
 "token_type": "bearer",
 "expires_in": 3600,
```

```
 "scope": "profile"
}
```

註：

參考自

"What is the OAuth 2.0 Authorization Code Grant Type?"，okta（by Aaron Parecki），2018。

"An Introduction to OAuth 2"，DigitalOcean（by Mitchell Anicas），2021。

"[筆記] 認識 OAuth 2.0：一次了解各角色、各類型流程的差異"，Medium（by Mike Huang），2021。

應用程式取得以上 Access Token 後，可帶著 Access Token 向 Resource Server（API）存取一定範圍的使用者資料，直到 Access Token 失效。

（四） OAuth 2.0 特色及優點

綜合上述內容，OAuth 2.0 的應用和優點在於能讓使用者，授權第三方應用程式從另一平台或機構取得隱密的資料，且使用者可不用將驗證資訊（憑證）分享給第三方應用程式，解決了傳統使用者-伺服器方式的問題，確保使用者的資料隱密性和安全性較不受第三方應用程式端的威脅影響。

若應用程式端採用 Authorization Code 授權流程，使用者同意授權應用程式端後，應用程式端仍需向授權伺服器進行身份驗證，並以後端方式傳遞身份訊息，因此採用 Authorization Code 授權流程，不僅可降低應用程式端的身份訊息洩漏出去，也可提升使用者資料的隱密性。

OAuth 2.0 的應用相當廣泛，不僅可運用在網頁、電腦和手機的應用程式中，亦提供不同類型的授權方式和擴充功能，供應用程式和授權伺服器端視使用需求和使用場景，彈性選擇合適的授權方法（D. Hardt, Ed, 2012；Aaron Parecki, 2018；Mitchell Anicas, 2021；Mike Huang, 2021）。

三、 OpenID Connect 身份驗證協定設計與應用

（一） 介紹 OpenID Connect

1. 定義 OpenID Connect

OpenID Connect（OIDC）是一個基於 OAuth 2.0 協議的身份驗證協議。它提供了一個標準化的方法，讓用戶可以使用他們在網際網路上的身份（例如 Google 或 Facebook）來登錄不同的網站或應用程式，而無需在每個網站上創建新帳戶。

儘管 OIDC 的目的與其前身 OpenID 1.0/1.1/2.0 相同，但其機制有很大不同。主要的差異在於 OIDC 建立在 OAuth 2.0 之上，而 OpenID 1.0/1.1/2.0 則是獨立的協議（ibm, 2023）。

2. OpenID Connect 的發展歷程

OpenID Connect 最初由 OpenID 基金會開發，旨在提供一個更安全、可靠且易於使用的身份驗證協議。該協議的第一個版本於 2014 年發布，並已成為一個開放標準，受到廣泛支持和使用（frontegg, 2023）。

3. OpenID Connect 的基本工作原理

OpenID Connect 使用了一種名為 JSON Web Tokens（JWTs）的標準，其中包含了關於用戶身份的基本信息，如用戶 ID、姓名和電子郵件地址。當用戶嘗試登錄一個網站或應用程序時，該網站或應用程序會要求用戶使用其 OpenID Connect 提供者進行身份驗證。

當用戶成功進行身份驗證後，OpenID Connect 提供者會向網站或應用程序發送一個 JWT，其中包含用戶身份的基本信息。該 JWT 可用於網站或應用程序驗證用戶身份並決定是否授予其訪問權限。

4. OpenID Connect 的主要特點和用途

（1）簡單易用：OpenID Connect 使用 JSON Web Token（JWT）作為身份驗證和授權的標準格式，使開發者能夠輕鬆在不同的應用程式間進行身份驗證和授權。

（２）支援單一登入：OpenID Connect 支援單一登入（Single Sign-On，SSO）功能，讓使用者只需在第一次登入時驗證身份，便可在整個生態系統中使用相同的身份驗證資訊。

（３）提供可靠的身份驗證：OpenID Connect 使用加密技術和簽名驗證機制確保身份驗證資訊的安全性，同時提供多種授權方式（如 implicit flow 和 code flow）以滿足不同應用場景需求。

（４）跨平台支援：OpenID Connect 支援多種應用場景，包括 Web 應用程式、原生移動應用程式、單頁應用程式等。

（５）可擴展性：OpenID Connect 是一個可擴展的協議，能夠滿足不同業務需求和應用場景，同時支援自定義的擴展功能。

（二） OpenID Connect 和 OAuth 2.0 的差別

OpenID Connect 是 OAuth 2.0 的一個擴展協議。OAuth 2.0 主要用於授權存取受保護資源的用戶端應用程式，並不包含身份驗證（Authentication）的部分。而 OpenID Connect 在 OAuth 2.0 的基礎上定義了身份驗證的流程，讓應用程式可以根據授權伺服器執行的身份驗證結果來確認使用者的身份，並透過 API 取得使用者相關的資訊。

OpenID Connect 彌補了 OAuth 2.0 在實務上的一些不足之處，主要擴充了以下幾個元素（Petertc, 2019）：

1. ID Token：ID Token 主要攜帶驗證資訊和存取 Identity Profile API 的相關資訊，並以加密的方式透過 JOSE（JSON Object Signing and Encryption）傳送，確保認證資訊的機密性和不可否認性。ID Token 包含的內容（claims）有：發布者 URL、subject identifier（通常用於代表特定 End-user 的識別碼）、token 逾傳送的對象、ID Token 到期時間、發布時間、防止重播攻擊的 nonce 等。
2. UserInfo Endpoint：除了 ID Token 提供基本用戶資訊外，OpenID Connect 還要求認證提供者提供一個 API 界面，讓應用程式能夠獲取更完整的用戶資訊。
3. Standard Set of Scopes：在 OAuth 2.0 中，Scope 用於劃定特定 token 能夠存取受保護資源的範圍。OpenID Connect 也定義了一組相關的 Scope，包括 profile、email、address、phone 等，這些 Scope 被定義用於回傳一組 claim。此外，一個 Access Token 也可以映射到多個 Scope。

OpenID Connect 在 OAuth 2.0 的基礎上提供了更完整的身份驗證機制，並擴展了 ID Token、UserInfo Endpoint 和一組標準 Scope 的定義，以提供更豐富的用戶身份資訊和授權範圍的支援。

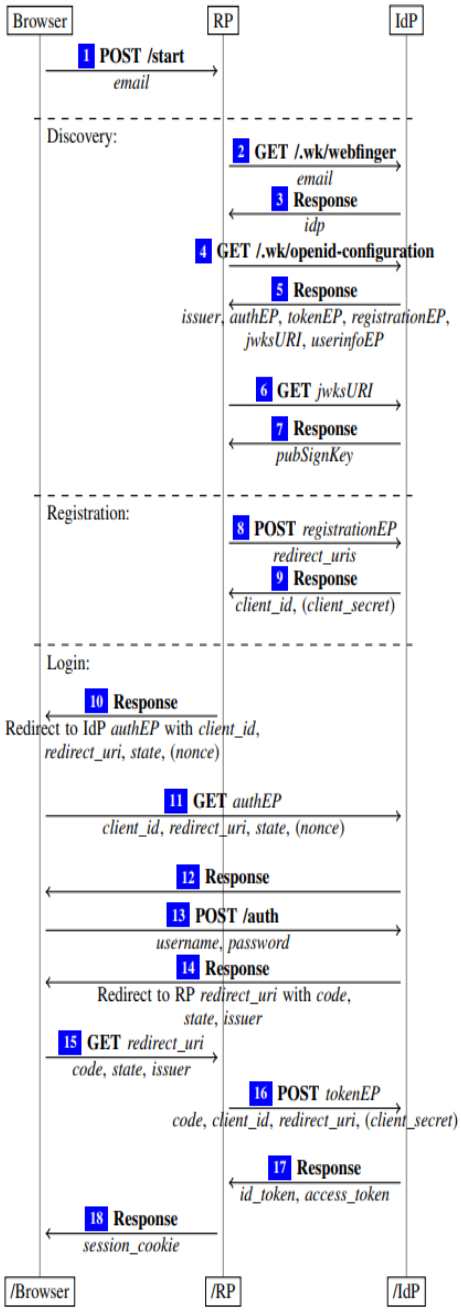
（三） OpenID Connect 詳細流程說明

本研究探討了 OpenID Connect（OIDC）協議中的詳細流程，OIDC 通過允許 Relying Party（RP）（即客戶端應用程序）向 Identity Provider（IdP）（即身份提供者）驗證和獲取用戶身份信息，實現了單點登錄和安全授權的功能。以下說明和流程圖為 OIDC 的 18 個流程步驟：

1. 使用者向 Relying Party（RP）發送 POST / start 請求。
2. Relying Party（RP）向 Identity Provider（IdP）發送 GET /.well-known/webfinger 請求，以查詢使用者的 email。
3. IdP 回應 RP 的請求，提供 IdP 資訊。
4. RP 向 IdP 發送 GET /.well-known/openid-configuration 請求，以獲取 OpenID Connect 相關資訊。
5. IdP 回應 RP 的請求，提供 issuer、authEP、tokenEP、registrationEP、jwksURI 和 userinfoEP 等資訊。
6. RP 向 jwksURI 發送 GET 請求，以獲取公開簽名金鑰。
7. IdP 回應 RP 的請求，提供 pubSignKey 資訊。
8. RP 向 registrationEP 發送 POST 請求，以註冊新用戶或更新現有用戶的資訊。其中包括 redirect_uris 等資訊。
9. IdP 回應 RP 的請求，提供 client_id 和 client_secret 等資訊。
10. RP 導向使用者到 authEP 並要求使用者授權。其中包括 client_id、redirect_uri、state 和 nonce 等資訊。
11. 使用者向 authEP 發送 GET 請求，以授權使用者。
12. IdP 回應使用者的請求，提供 client_id、redirect_uri、state 和 nonce 等資訊。
13. 使用者向 IdP 發送 POST /auth 請求，以提供使用者名稱和密碼等資訊。
14. IdP 回應使用者的請求，導向 RP 的 redirect_uri 並提供 code、state 和 issuer 等。

圖 7

OIDC 流程圖



註：

參考自"The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines", D. Fett, R. Küsters and G. Schmitz, 2017.

15. RP 向 redirect_uri 發送 GET 請求，以獲取 code、state 和 issuer 等資訊。
16. RP 向 tokenEP 發送 POST 請求，以獲取 id_token 和 access_token。其中包括 code、client_id、redirect_uri 和 client_secret 等資訊。
17. IdP 回應 RP 的請求，提供 id_token 和 access_token 等資訊
18. RP 向使用者發送 session_cookie，以確保使用者在未來的請求中可以被識別和驗證。

四、 Financial-grade API 之安全強化策略與規範

（一） FAPI 簡介

Financial-grade API（FAPI）是一個安全協議，旨在保護客戶的金融數據和資源免受各種攻擊。FAPI 建立在 OAuth 2.0 和 OpenID Connect 之上，並包括了最近由 IETF 和 OpenID 基金會開發的 OAuth 2.0 安全擴展。FAPI 提供了兩種主要模式，即唯讀訪問模式和讀寫訪問模式，以便客戶可以授權第三方訪問其銀行帳戶信息或發起付款等操作。FAPI 支持公共客戶端（如 JavaScript 瀏覽器應用程序）和保密客戶端（如 Web 服務器），並提供多種不同的配置選項以適應不同的使用情況。此外，FAPI 還包括一些新的防禦機制，例如強制要求使用 TLS 1.2 或更高版本、限制重定向 URI、使用 JWT 簽名等。這些防禦措施可以有效地保護客戶免受各種攻擊，例如中間人攻擊、偽造請求攻擊等。因此，FAPI 是一個非常重要的安全協議，可以幫助銀行和金融機構保護客戶的數據和資源（OpenID Foundation, 2023；D. Fett et al., 2019）。

（二） FAPI 具備的三大安全性質

1. 授權（Authorization）：FAPI 確保只有經過授權的用戶才能訪問受保護的資源。這意味著，當用戶試圖訪問受保護的資源時，FAPI 會驗證其身份和權限，以確定其是否有權訪問該資源。
2. 身份驗證（Authentication）：FAPI 確保用戶身份得到驗證，以防止未經授權的用戶訪問受保護的資源。當用戶試圖訪問受保護的資源時，FAPI 會要求其提供身份驗證信息，例如使用者名稱和密碼、生物特徵等。
3. 會話完整性（Session Integrity）：FAPI 確保在用戶與伺服器之間進行通信時，通信內容不被竄改或篡改。這可以通過使用加密技術來實現。如果攻擊者

嘗試竄改或篡改通信內容，則 FAPI 將檢測到此類攻擊並拒絕相關請求（OpenID Foundation, 2023；D. Fett et al., 2019）。

（三） FAPI 具有以下六個方面的安全措施

1. 強制要求使用 TLS 1.2 或更高版本：FAPI 要求所有通信都必須使用 TLS 1.2 或更高版本，這可以防止中間人攻擊和其他類型的攻擊。
2. 限制重定向 URI：FAPI 限制了重定向 URI 的使用，以防止攻擊者將用戶重定向到惡意網站。
3. 使用 JWT 簽名：FAPI 使用 JWT（JSON Web Token）簽名來保護數據完整性和身份驗證。這可以防止偽造請求攻擊和其他類型的攻擊。
4. Code and Token Binding：FAPI 支持 OAuth 2.0 擴展功能 Code and Token Binding（背後使用 mTLS 技術），支持授權碼和 Access Token 與特定的客戶端和授權伺服器綁定，可以防止令牌劫持和其他類型的攻擊（OpenID Foundation, 2023；D. Fett et al., 2019）。
5. JWS Client Assertions：FAPI 支持 JWS Client Assertions，這可以提供更強大的身份驗證和授權功能。
6. Proof Key for Code Exchange：FAPI 支持 OAuth 2.0 擴展功能 Proof Key for Code Exchange（PKCE），這可以提供更強大的安全性能，以防止授權碼劫持等攻擊。

（四） Token Binding 和 PKCE 補充

Token Binding 和 PKCE 同為 OAuth 2.0 的擴展功能，在 FAPI 中可提升授權和驗證的安全性，相當重要，因此本研究再次補充其運作概念。

1. Token Binding

Oauth 2.0 的 Token Binding 目的是將 Access Token 和（或）Authorization Code 綁定至特定的 TLS 連線上。當客戶端（應用程式）和授權伺服器建立 TLS 連接時，會生成一對公鑰（Token Binding ID）和私鑰，公鑰傳遞給授權伺服器作為客戶端的辨識值，而客戶端使用私鑰將 TLS 連線中 Session 的唯一值（EKM 值）加密為簽署值，以證明其擁有私鑰，這樣後續授權伺服器就可以確認該連接的來源是否為正確的客戶端。

當客戶端需要取得授權碼或 Access Token 時，會在相應的 HTTP 請求中附上 Token Binding 相關的資訊，包括客戶端的公鑰（Token Binding ID）和簽署值，接著授權伺服器會驗證客戶端的公鑰（Token Binding ID）和簽署值，以此確認授權碼或 Access Token 的請求是由正確的客戶端發出的。因此，Token Binding 可將 Access Token 和（或）Authorization Code 綁定至特定的 TLS 連線上，來確保授權碼和 Access Token 不輕易被非法的客戶端持有以取得使用者的機密資料（D. Fett et al., 2019）。

2. PKCE

客戶端在每次授權請求時都會生成一個秘密金鑰，稱為「code verifier」，並將它保存下來。然後，客戶端會將這個 code verifier 轉換成「code challenge」，並在授權請求中將 code challenge 和轉換方法一起發送到授權伺服器。

授權伺服器會記錄下 code challenge 和轉換方法，並回應授權請求，給予客戶端授權碼。接著，客戶端在發送存取權杖請求時，會將最初生成的 code verifier 包含在請求中。當授權伺服器接收到 code verifier 後，會使用客戶端最初提供的轉換方法來驗證 code verifier，確保生成的 code challenge 與之前記錄的一致。如果不一致，授權伺服器將拒絕客戶端的訪問（Mitchell Anicas, 2021）。因此，PKCE 可以防止攻擊者攔截授權碼，以提高應用程式的安全性，保護用戶的資料。

肆、 三大協定之存取控制風險分析與因應策略

一、 OAuth 2.0 實務風險與防護措施

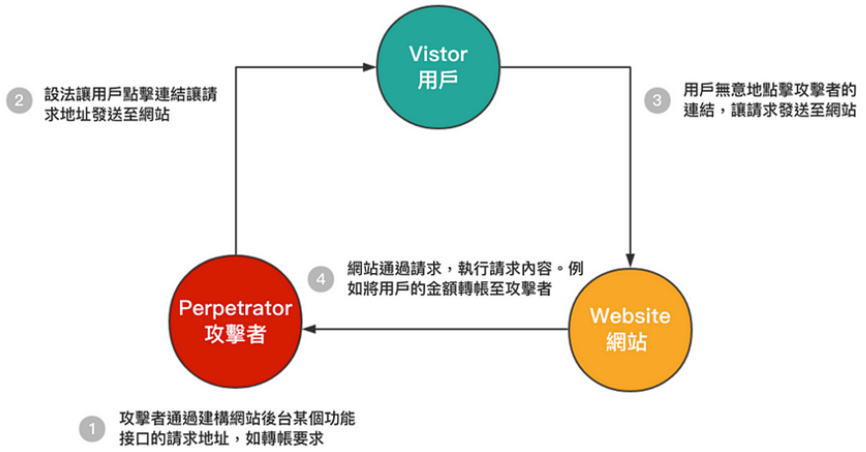
（一） CSRF 攻擊

1. CSRF 攻擊說明

Cross-Site Request Forgery（CSRF），又稱跨站請求偽造，是 OAuth 2.0 中最常被提起的攻擊之一，其攻擊行為大致為：使用釣魚等手法誘騙使用者點擊連結，以此達到竊取用戶的 Access token 之目的。大致流程如下所示：

圖 8

CSRF 攻擊流程



註：

參考自"CSRF 攻擊是什麼？簡述"，Roy Kwok，2018。

在這張圖中粗略地講述了 CSRF 的攻擊方式。一般而言，User 授權給 Client 時，允許其使用部分隱私資料時，會先傳送一個授權許可給這個 Client，此時 Client 便可以此授權向 Authorization Server 要求 Access token。因為在登入 IDP 後，會有一段時間是 Cookies 仍未過期，若有一個攻擊者在用戶尚未登出 IDP 時，惡意地將其製作的假網頁連結置入圖片等中，一旦使用者點擊此連結，攻擊者便會啟動攻擊流程。

由於使用者已登入 IDP，IDP 啟動自動授權機制後將 Authorization Code 傳送到攻擊者的 Redirection URI，攻擊者再其轉送合法的 Client；Client 將收到的

Authorization Code 傳給 IDP 換取使用者的 Access Token，由於此時攻擊者已登入 Client，Client 於攻擊者的 session 中又收到來自攻擊者傳來的 Authorization Code，造成攻擊者的 Client 帳戶與使用者的 IDP 帳戶進行連結，攻擊者就可以藉此非法存取 IDP 上的使用者資料（OAuth 2.0 系列（七）--- OAuth 2.0 可能存在的安全漏洞及預防方法, 2021）。

2. CSRF 攻擊之防禦手法

Clien 的開發者能採取部分措施來預防此類攻擊發生。如：在 OAuth 2.0 認證過程中加入 state 參數。在將用戶 redirect 至 Authorization Server 時，為用戶提供一個與 user session 相關的亂數作為 state，並在服務提供者返回 Authorization Code 請求時，驗證 state 之正確性。如果是正確合法的請求，那麼此時接受到的參數值應該和上一步提到的為該用戶生成 stat 參數值完全一致，否則就是異常請求（蔡峻福，2016）。

（二） 授權碼失竊

1. 授權碼失竊說明

一般而言，若一個使用者對第三方的 Client 進行授權並產生了一個 Authorization codeA，這時沒有 codeA 的另一個軟體並不能取得這個使用者的資訊。但若這個 codeA 受到了竊取，那麼便有可能產生令並非被授權的軟體取得使用者資訊的機會。

其中，重定向 URI 遭竄改是一種常見的竊取 Authorization code 的方法。若一個網頁使用子目錄匹配，那麼攻擊者便能偽造一個授權請求的連結，並重定向至自己的頁面，如圖九所示。

攻擊者若將其連結藏在偽造的連結中，並引誘使用者點擊。由於子目錄匹配的匹配方式，這會讓攻擊者的連結被認定為合法連結，進而向攻擊者的 server 發出請求，並以此獲得 Authorization code（OAuth 2.0 系列（七）--- OAuth 2.0 可能存在的安全漏洞及預防方法，2021）。

圖 9

授權碼失竊攻擊模式

註：

本研究自行繪製（參考自"OAuth 2.0 系列（七）--- OAuth 2.0 可能存在的安全漏洞及預防方法"，bug 改了我，2020）

2. 授權碼失竊之防禦方法

為了防止受到攻擊，首先需要注意避免使用安全性較低的 Implicit Grant 模式，也可以利用 state 參數，確保 Authorization code 與授權請求是由同一個用戶發起。此外，盡量在註冊 redirect_uri 時具體，也可以避免讓攻擊者有偽造連結的可能。

（三） XSS 攻擊

1. XSS 攻擊說明

Cross-Site Scripting（XSS），又名跨站腳本攻擊，是一種常見的攻擊技術。通常，XSS 是指在網站讀取時，執行攻擊者提供的腳本程式。一般來說，XSS 攻擊分為數種形式：

（1）**Reflected/DOM 型 XSS**：反射型 XSS 中，攻擊者將惡意的程式會藏在網址列裡，範例如下：

http://www.example.com/upload.asp?id=<script>alert(1）;</script>

DOM 型則是在 URL 中輸入 DOM 物件，把物件嵌入網頁程式碼，如：*<img src=# onerror=”alert(123）”>*

大部分攻擊者會利用釣魚技巧令使用者點擊，並利用短網址等技巧偽裝。

（2）**Stored XSS**：與前兩者不同，儲存型 XSS 並不需要社交工程釣魚，也能使 User 受到攻擊。攻擊者會將 JS 程式碼儲存在伺服器的資料庫中，因網頁載入了當下頁面的惡意程式，便會因此受到攻擊。例如，使用者將 JS 程式注入某網頁的留言板中，而下一位使用此網頁的使用者在載入留言板時便會因此受害（Amit Klein, 2005）。

在 OAuth 2.0 中，大多數情況下系統需做校驗時，受保護的資源都是把輸入的內容，如 user id 或 access token 等，直接返回到使用者的頁面上，若攻擊者傳入一些蒐集用戶數據的 JS 程式碼，那麼當用戶觸發這些程式時就會受到攻擊。

2. XSS 攻擊之防禦手法

目前而言，針對 OAuth 2.0 的 XSS 防禦思路大致上是減少惡意程式碼執行的機會以及增加對受保護資源，如 access toke 等的保護，例如在存取受保護資源的 server 中增加各種 referrer 以規定受保護的資源必須返回正確的類型，以此拒絕執行 JS 程式碼，亦或是不允許用參數傳遞 access token 等（王思翰，2009）。

（四） 水平越權問題

1. 水平越權問題說明

越權問題是指一個使用者可以擅自訪問另一個並不屬於自己權限的使用者的資料，大多時候是因應用程式未校驗請求數據的用戶與擁有數據的用戶為相同而產生，一般分為水平越權及垂直越權。在 OAuth 2.0 中，水平越權是一個較嚴重的問題。

水平越權是指一個軟體使用者 A，未經另一個使用者 B 許可便能查看存於軟體內且屬於 B 的數據。在開放平台的環境下，受保護資源的歸屬關係很容易被遺漏。一方面，開放平台的作用是將使用者授權以後的 access token 轉換成真實的使用者資訊，再傳遞到 API 提供者，如此一來歸屬的判斷只能在 API 提

供者這一方處理；另一方面，數據提供者往往會認為開放的接口是「與自己公司相同的系統調用的」，一來二去便容易忽略水平校驗的重要性，水平越權問題便可能因此產生（王新棟，2020）。

2. 水平越權問題之防禦手法

相對來說，水平越權的防禦手法並不需要太多的技術實現，只需要增加有關於受保護資源的歸屬判斷便能很大程度地解決。因此，更重要的是開發人員的安全認知與意識的培養。

二、 Open ID Connect 潛在攻擊面與對應機制

（一） OAuth 2.0 問題及對應的 OIDC 解決辦法

OAuth 2.0 為廣泛使用的授權機制標準，在認證機制方面相對較弱，缺乏對於網路通訊協定中更嚴謹的安全防範措施。為了彌補這個缺點，後來發展出了 OpenID Connect（OIDC）。OIDC 在 OAuth 2.0 的基礎上進一步發展，引入了 ID Token 的概念，用於提供部分使用者資訊給授權者，以進行身份驗證，避免中間人的冒用風險。OIDC 旨在解決 OAuth 2.0 中的六個主要問題，以下是這些問題以及 OIDC 提供的解決方法：

1. Access Token 僅作為授權象徵，不具備可利用的資訊。

（１）OpenID Connect 對應解決方法

引入 ID Token，包含用戶的身份信息，確保用戶身份得到驗證，減少冒充攻擊的風險。

（２）OpenID Connect 的解決方法詳細說明

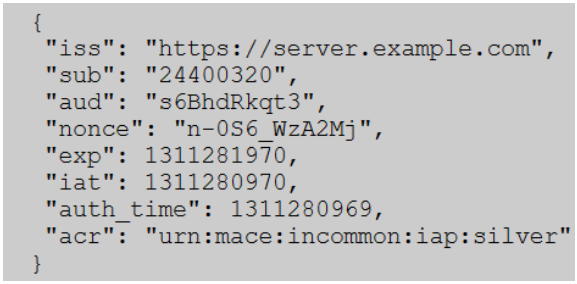
ID Token 是一個 JSON Web Token（JWT），包含有關用戶身份的信息，例如用戶 ID 和電子郵件地址等。

當用戶成功驗證並授權後，IdP 將生成 ID Token 並返回給 RP。RP 可以使用公鑰驗證 ID Token 的簽名，確保其來自可信的 IdP。

通過引入 ID Token，OpenID Connect 確保了用戶身份得到驗證，減少了冒充攻擊的風險。攻擊者無法輕易地偽造有效的 ID Token，因為它需要由可信的 IdP 簽署。此外，RP 可以使用 ID Token 中包含的信息來確認用戶身份和授權範圍，從而更好地保護用戶數據和資源。

圖 10

ID Token 的結構



```
{
 "iss": "https://server.example.com",
 "sub": "24400320",
 "aud": "s6BhdRkqt3",
 "nonce": "n-0S6_WzA2Mj",
 "exp": 1311281970,
 "iat": 1311280970,
 "auth_time": 1311280969,
 "acr": "urn:mace:incommon:iap:silver"
}
```

註：

參考自"OpenID Connect Core 1.0 Incorporating Errata Set 1." Sakimura. 2014.

2. 取得 Access Token 時，有時無需取得 End-user 的即時授權，無法確保 End-user 是否存在。

（１）OpenID Connect 對應解決方法

OpenID Connect 引入即時授權驗證，要求在獲取 Access Token 時，需要 End-user 進行授權驗證，確保 End-user 在獲取 Access Token 前已通過驗證並確認其存在。

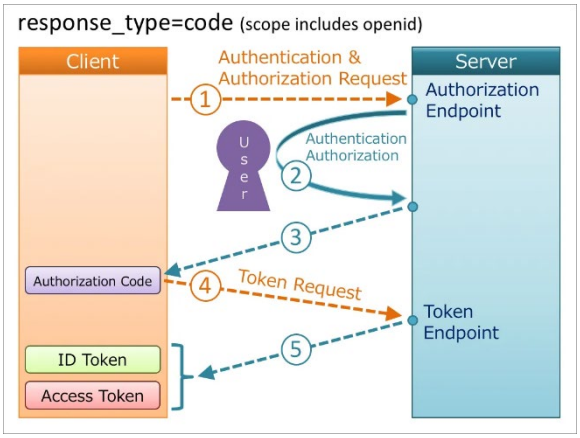
（２）OpenID Connect 的詳細解決方法說明（D. Fett, R. Küsters and G. Schmitz, 2017）

當用戶試圖訪問 RP 的受保護資源時，RP 將重定向用戶到 IdP 的授權端點。在那裡，End-user 將被提示進行身份驗證和授權。如果 End-user 通過了驗證和授權，IdP 將生成一個 ID Token 和一個 Access Token，並返回給 RP。RP 可以使用 ID Token 驗證用戶身份，並使用 Access Token 訪問受保護資源。

通過引入即時授權驗證，OpenID Connect 確保了 End-user 在獲取 Access Token 前已通過身份驗證和授權。這有助於減少冒充攻擊和其他安全風險。

圖 11

即時授權驗證流程



註：

參考自"Diagrams of All The OpenID Connect Flows." Kawasaki, T. 2017.

3. Access Token 的注入攻擊，攻擊者可能透過調包 Access Token 來訪問無關的資源。

（１）OpenID Connect 對應解決方法

OpenID Connect 在 ID Token 中存放 Access Token 的 hash 值，確保資料完整性，防止 Access Token 的注入攻擊。

（２）OpenID Connect 的詳細解決方法說明

當 IdP 發送 ID Token 給 RP 時，ID Token 包含了 Access Token 的 hash 值，如圖 12 所示。RP 可以使用這個 hash 值來驗證 Access Token 是否有效並防止注入攻擊。如果 Access Token 被修改或替換，其 hash 值也會相應地改變，因此 RP 可以通過比較 hash 值來檢測到任何變化。

這種方法可以提高安全性，因為即使攻擊者能夠截取 ID Token 和 Access Token，他們也無法修改或替換 Access Token 而不被 RP 檢測到。

4. 無法辨識 Access Token 的發送目的，Client 無法確認 Access Token 是否真的是自己所用。

（１）OpenID Connect 對應解決方法

OpenID Connect 引入發送目的（audience）的概念，如圖 13，在 ID Token 中包含客戶端的識別符，區分身份驗證是針對哪個應用程式，確保身份驗證是針對自己的應用程式。

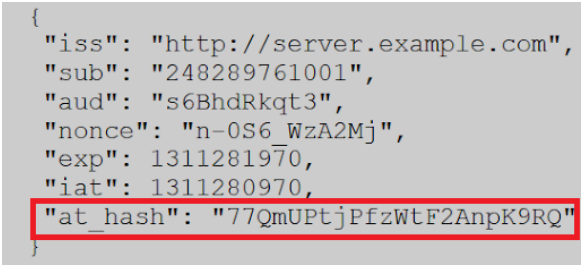
（２）OpenID Connect 的詳細解決方法說明

這種方法確保了身份驗證是針對特定的應用程式而不是其他應用程式。如果攻擊者試圖使用同一個 ID Token 來訪問另一個應用程式，RP 將會拒絕該請求，因為 ID Token 中包含了不正確的客戶端識別符。

因此，在 OpenID Connect 中使用發送目的（audience）可以幫助防止跨站請求偽造（CSRF）攻擊和其他類型的安全漏洞。

圖 12

Access Token 的 hash 值

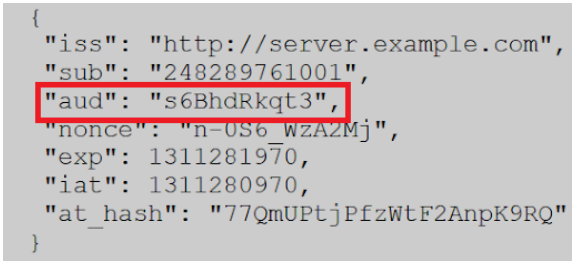


註：

參考自"OpenID Connect Core 1.0 Incorporating Errata Set 1." Sakimura. 2014.

圖 13

發送目的



註：

參考自"OpenID Connect Core 1.0 Incorporating Errata Set 1." Sakimura. 2014.

5. 注入假使用者資訊，攻擊者可能在 OAuth 2.0 授權請求過程中竄改資訊。

（１）OpenID Connect 對應解決方法

OpenID Connect 要求使用加密的 ID Token，由身份提供者在生成 ID Token 時提供使用者資訊並使用私鑰進行簽名，確保資訊的完整性和真實性，防止注入假使用者資訊。

（２）OpenID Connect 的詳細解決方法說明

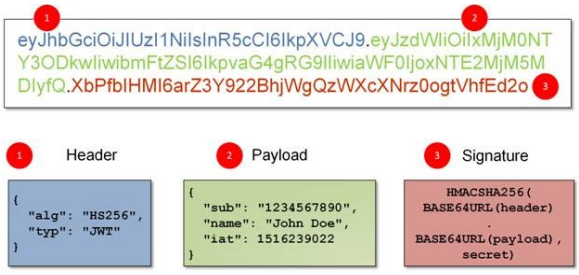
當使用者進行身份驗證時，身份提供者會生成一個 ID Token，其中包含使用者的資訊和其他元數據。然後，身份提供者會使用其私鑰對 ID Token 進行簽名。

當 RP 收到 ID Token 時，它可以使用公鑰來驗證簽名是否有效。如果簽名有效，RP 就可以信任 ID Token 中包含的資訊是真實和完整的。

這種方法可以防止攻擊者注入假使用者資訊或其他惡意內容。因為只有擁有正確私鑰的身份提供者才能夠對 ID Token 進行有效簽名，所以攻擊者無法輕易地修改或偽造 ID Token。

圖 14

JSON Web Tokens 結構



註：

參考自"Access Token Vs ID Token Vs Refresh Token - What? Why? When? Medium." Anupam, M. 2020.

6. 缺乏統一的標準，不同認證系統可能有不同結果。

（１）OpenID Connect 對應解決方法

OpenID Connect 提供一組統一的標準，包括 ID Token 的結構和內容，

以及身份提供者和客戶端之間的互操作性，降低不同身份提供者實現和解讀方式的差異，提高整個系統的安全性和一致性。

（２）OpenID Connect 的詳細解決方法說明

OpenID Connect 定義了 ID Token 的格式和內容，包括使用者資訊、發行者資訊、RP 資訊等。這使得不同身份提供者生成的 ID Token 具有相同的格式和內容，從而使得 RP 可以更容易地解讀和驗證 ID Token。

此外，OpenID Connect 還定義了身份提供者和客戶端之間的互操作性。例如，它定義了如何進行授權請求、如何獲取 ID Token 等。這些標準可以確保不同身份提供者和客戶端之間可以順利地進行通信，從而降低整個系統出現錯誤或漏洞的風險。

表 4

ID Token 的格式與內容

Payload 資訊	解釋
iss	OpenID Provider 的 url
sub	被驗證的使用者的 identifier
aud	Relying Party 的 ID
exp	到期時間
iat	token 的發行時間
auth_time	使用者通過驗證的時間

註：

修改自"*OpenID Connect Core 1.0 incorporating errata set 1*", The OpenID Foundation, 2014.

表 5

OAuth 2.0 的問題與 OpenID Connect 的解決方法重點整理

OAuth 2.0 的問題	OpenID Connect 的解決方法
Access Token 僅作為授權象徵，不具備可利用的資訊。	OpenID Connect 引入 ID Token，包含用戶的身份信息，確保用戶身份得到驗證，減少冒充攻擊的風險。

取得 Access Token 時，有時無需取得 End-user 的即時授權，無法確保 End-user 是否存在。	OpenID Connect 引入即時授權驗證，要求在獲取 Access Token 時，需要 End-user 進行授權驗證，確保 End-user 在獲取 Access Token 前已通過驗證並確認其存在。
Access Token 的注入攻擊，攻擊者可能透過調包 Access Token 來訪問無關的資源。	OpenID Connect 在 ID Token 中存放 Access Token 的 hash 值，確保資料完整性，防止 Access Token 的注入攻擊。
無法辨識 Access Token 的發送目的，Client 無法確認 Access Token 是否真的是自己所用。	OpenID Connect 引入發送目的（audience）的概念，在 ID Token 中包含客戶端的識別符，區分身份驗證是針對哪個應用程式，確保身份驗證是針對自己的應用程式。
注入假使用者資訊，攻擊者可能在 OAuth2 授權請求過程中竄改資訊。	OpenID Connect 要求使用加密的 ID Token，由身份提供者在生成 ID Token 時提供使用者資訊並使用私鑰進行簽名，確保資訊的完整性和真實性，防止注入假使用者資訊。
缺乏統一的標準，不同認證系統可能有不同結果。	OpenID Connect 提供一組統一的標準，包括 ID Token 的結構和內容，以及身份提供者和客戶端之間的互操作性，降低不同身份提供者實現和解讀方式的差異，提高整個系統的安全性和一致性。

註：

修改自"*The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines*", D. Fett, R. Küsters and G. Schmitz, 2017.

綜上所述，ID token 以及 JWT 標準的引入，使得 OpenID Connect 較傳統的 OAuth 2.0 來說能防禦更多的惡意攻擊。也因此，在我們需要驗證使用者身分及授權時，應善加運用 OpenID Connect 這個工具，更能保護使用者的資料以及維持系統的安全。

（二） OpenID Connect 風險與相應之解決方法

雖然 OpenID Connect 作為 OAuth 2.0 的進一步發展，能夠解決 OAuth 2.0 中的大部分問題並提供更強的認證機制，但它本身仍存在一些風險和挑戰。以下是 OpenID Connect 可能面臨的 10 種攻擊以及相應的防範方法（D. Fett, R. Küsters and G. Schmitz, 2017）：

1. Server-side Request Forgery（SSRF）攻擊

在 Authorization Request 中，OpenID Connect 允許使用者透過間接的方式來傳送參數，使用者可傳送一個名為 request_uri 的參數給 IDP，讓 IDP 透過 GET 去向這個位置獲取其他的參數。這帶來的便利是，若要傳的參數過大，超出 URI 的長度限制，就可以透過這個方法來傳送資訊給 IDP。

然而，這帶來了 SSRF 攻擊的機會。使用者可以向 IDP 傳送任意網址作為 request_uri，從而使 IDP 向該網址發出請求。因此，IDP 必須使用適當的過濾跟限制機制，來避免惡意的 request。

2. Injection 攻擊

攻擊者透過在 request 中注入惡意指令，使得使用者以及 IDP/RP 上的機密資訊外流。攻擊者可將額外參數接在已存在的參數後面，由於在 OIDC 中資料的傳輸相當頻繁，這很容易被忽視。如果 IDP/RP 直接使用了這個參數，惡意程式碼/指令就會被執行到。

因此，在 OIDC 的 implementation 中，除了必須對所有從 untrusted sources 來的參數進行檢查及對輸出進行過濾以外，也要盡量避免將重要、危險的資訊跟 OIDC 的 endpoint 放在同一個 domain。

3. Cross-Site Request Forgery（CSRF）攻擊

攻擊者通過利用已登錄用戶的身份發送未經授權的請求。防範方法包括在請求中添加 CSRF token，以確保請求是由合法用戶發送的。

4. Redirect URI 攻擊

攻擊者通過修改 redirect_uri 來重定向到惡意站點。防範方法包括驗證 redirect_uri 是否與事先註冊的 URI 相符。

5. Token 竊取攻擊

攻擊者通過竊取 access_token 或 id_token 來冒充合法用戶。防範方法包括使用 HTTPS 來保護傳輸中的 token，並在 token 中添加適當的有效期和範圍限制。

6. Token 篡改攻擊

攻擊者通過修改 access_token 或 id_token 的內容來冒充合法用戶。防範方法包括使用數字簽名或加密來保護 token 的完整性和機密性。

7. Token Reuse 攻擊

攻擊者通過重複使用已經失效的 token 來冒充合法用戶。防範方法包括在 token 中添加適當的有效期和範圍限制，並在 RP 和 IdP 之間進行必要的同步。

8. Cross-Site Scripting（XSS）攻擊

攻擊者通過注入惡意腳本來竊取使用者的身份驗證信息。防範方法包括使用 proper escaping 和 Content Security Policies（CSP）作為第二道防線，並將 OIDC 端點放在與其他可能更易受攻擊的網頁不同的域名下。

9. SQL 注入攻擊

攻擊者通過注入惡意 SQL 語句來竊取使用者的身份驗證信息。防範方法包括使用預處理語句和綁定變量等技術來防止 SQL 注入。

10. CSRF 攻擊

攻擊者通過利用第三方登錄啟動功能來發起 CSRF 攻擊。防範方法包括限制第三方登錄啟動功能的使用，驗證發送方和接收方之間的關係，以及實施適當的 CSRF 防護措施，例如同步令牌或雙重提交 cookie。

從上述的介紹可以發現，在使用 OIDC 時，仍然需要考慮許多可能存在的危險。即使這些攻擊都是可防範的，但只要使用者有疏忽，忘記對使用者輸入資料進行檢查，或以不安全的方式傳送資料，私人資訊仍然有機會外洩。

三、 Financial-grade API 實作挑戰與資安落差

Token Binding 和 PKCE 為 OAuth 2.0 的擴展功能，同時也是 FAPI 中的重要資安防禦機制，可提升授權和驗證流程的安全性，因此本研究選擇針對這兩個機制的攻擊進行介紹，並參考 D. Fett et al（2019）提出的文獻進行說明：

1. Token Binding 攻擊：Token Binding 是一種用於保護 Web 應用程序的安全協議，但是文獻中提到了兩種攻擊方法可以繞過 Token Binding 的保護。第一種攻擊方法是利用 TLS 1.3 中的一個漏洞，將 Token Binding ID 注入到 Session Identifier 中，從而使攻擊者能夠使用受害者的 Token Binding ID 進行身份驗證。第二種攻擊方法則是利用 Cookie 注入漏洞，將 Token Binding ID 注入到

Cookie 中，從而使攻擊者能夠使用受害者的 Token Binding ID 進行身份驗證。

2. PKCE 攻擊：PKCE 是 OAuth 2.0 協議中的一個安全機制，用於防止授權碼竄改攻擊。由於 PKCE 協議中對授權碼的使用方式存在一些假設，攻擊者可以利用這些假設來繞過 PKCE 的保護，使得攻擊者可以繞過授權碼流程並獲取訪問令牌。也就是，攻擊者可以通過在授權請求中使用已知的 code_verifier 值來模擬正確的 PKCE 驗證，從而繞過授權碼流程並獲取訪問令牌，導致安全問題（D. Fett et al., 2019）。

Token Binding 和 PKCE 對於 FAPI 的安全性相當重要，因此，以下亦參考 D. Fett et al（2019）所提出的文獻，說明上述兩種攻擊的可能防範方法：

1. 防範 Token Binding 攻擊的可能方法：

（１）增加綁定到特定應用程序上下文的綁定密鑰

為了增加綁定到特定應用程序上下文的綁定密鑰，可以引入一個新的參數來表示應用程序上下文，例如使用 URI 或其他唯一標識符。然後，在客戶端和伺服器之間建立 TLS 連接時，可以使用這個參數生成一個新的綁定密鑰。這樣可以確保每個應用程序都有自己的綁定密鑰，從而增加了攻擊者對該協議進行攻擊的難度。

（２）在使用 Token Binding 時要求客戶端驗證伺服器提供的公開金鑰。

在建立 TLS 連接時，客戶端可以要求伺服器提供其公開金鑰並對其進行驗證。如果驗證失敗，則客戶端可以拒絕與該伺服器建立連接。這樣可以防止攻擊者使用偽造的公開金鑰來進行攻擊。

2. 防範 PKCE 攻擊的可能方法：

（１）增加對 PKCE 代碼的保護

可以引入一個新的參數來表示應用程序上下文，例如使用 URI 或其他唯一標識符。然後，在客戶端和伺服器之間進行交互時，可以使用這個參數生成一個新的代碼。這樣可以確保每個應用程序都有自己的代碼，從而增加了攻擊者對該協議進行攻擊的難度。

（２）限制 PKCE 代碼的使用次數

在客戶端和伺服器之間進行交互時，可以限制每個代碼只能使用一次或者在特定時間內只能使用一次。這樣可以防止攻擊者重複使用同一個代碼來

進行攻擊。

以上為 FAPI 中，針對 Token Binding 和 PKCE 兩種 OAuth 2.0 擴展功能的攻擊和防範手法介紹，然而在實際上，執行以上防範方法時，仍需要考慮到不同場景下可能存在的風險和安全需求，評估其對系統的影響和風險，並根據實際情況進行調整和優化（D. Fett et al., 2019）。

伍、 結論與建議

一、 研究結論與建議

本研究以開放銀行 API 存取控制風險中的資料授權和身份驗證為主軸，探討並說明 OAuth 2.0、OpenID Connect 和 FAPI 的資安問題及相對應的解決方法，可幫助金融機構和第三方服務業者了解相關的內容，確保開放銀行服務的安全性和可靠性。本研究亦建議相關金融機構和服務業者以具備更高安全性的 FAPI 為發展目標，並持續使用更加安全的資安標準和技術，提升資料授權和身份驗證的安全與效能，為消費者帶來更安全便利的的金融服務。

二、 研究價值

綜觀過往國內關於開放銀行的文獻，較少針對資料授權和身份驗證進行整理與深入探討，而台灣開放銀行發展處於第二階段，如何運用安全的授權和驗證措施，確保消費者資料的隱私安全，便是台灣開放銀行良好發展的重要關鍵。因此，本研究結果可作為國內學術人員或研究機構之參考，提升國內學術界對於資料授權、身份驗證和其他開放銀行相關資安議題的探討與重視，亦可提供政府機構、金融機構和第三方應用服務業者參考，以制定更安全的資料授權和身份驗證措施，並增進開放銀行的 API 存取控制安全，進一步提升消費者的資料隱私與安全性，同時促進更安全、更多元的開放銀行服務。

三、 問題討論

1. 為何 OIDC 10 種攻擊會挑選這兩個進行介紹？

由於版面關係，我們從 10 個 OpenID Connect 的攻擊中，就針對性和廣泛性兩種角度，分別選擇了 SSRF 攻擊和 Injection 攻擊進行介紹，期望能提升大家對於更多面向 OIDC 攻擊的瞭解。

其中，SSRF 攻擊屬於較為針對 OpenID Connect 協定的攻擊，因為 OpenID Connect 允許使用者透過間接的方式來傳送參數，使用者可傳送一個名為 request_uri 的參數給 IDP，讓 IDP 透過 GET 去向這個位置獲取其他的參數，這就給 SSRF 攻擊者機會，向 IDP 傳送惡意網址作為 request_uri，從而使 IDP 向該網址發出請求產生問題。因此，IDP 必須使用適當的過濾跟限制機制，來避免惡意的 request。

而 Injection 攻擊屬於攻擊對象較廣泛的攻擊，在許多網路協定、授權和身份驗證協定中，皆能看到它的蹤影，OpenID Connect 也是它的攻擊對象，Injection 攻擊者可將額外參數接在已存在的參數後面，由於在 OIDC 中資料的傳輸相當頻繁，這很容易被忽視。如果 IDP/RP 直接使用了這個參數，惡意程式碼/指令就會被執行到，使得使用者以及 IDP/RP 上的機密資訊外流。因此，在 OIDC 的實施中，除了必須對所有從 untrusted sources 來的參數進行檢查及對輸出進行過濾以外，也要盡量避免將重要、危險的資訊跟 OIDC 的 endpoint 放在同一個 domain。

綜上所述，我們以針對性和廣泛性兩種角度，特別向大家介紹了 SSRF 攻擊和 Injection 攻擊，以及他們的防範方法，期望能在期末報告發表時，提升大家對於更多面向 OIDC 攻擊的瞭解。

2. TLS 1.3 有 Session、Cookie 漏洞的問題，那其他 1.2 版本以上的 TLS 也有此問題嗎？

關於其他 1.2 版本以上的 TLS 是否也存在 Session、Cookie 漏洞，導致 Token Binding 攻擊發生，這部分我們仍須透過蒐集相關的資料，瞭解不同版本 TLS 的運作並比較不同版本之間的風險，來確認其他 TLS 版本是否仍存在 Session、Cookie 漏洞問題。因此，這項關鍵問題，可以作為未來的研究探討之一，以更加瞭解 FAPI 的 Token Binding 攻擊及其防範方法，作為學術界、金融機構和相關業者的參考。

參考文獻

一、 中文文獻（按筆畫排序）

bug 改了我（2021，11 月 11 日）。*OAuth 2.0 系列（七）--- OAuth 2.0 可能存在的安全漏洞及預防方法*。博客園。https://www.cnblogs.com/hellowhy/p/15533625.html

Brett King（2018）。*Bank4.0：金融常在，銀行不再？*（孫一仕、周羣英、林凱雄）。財團法人台灣金融研訓院。(原著出版於 2018)

Mike Huang（2021，4 月 11 日）。[筆記] 認識 OAuth 2.0：一次了解各角色、各類型流程的差異 [Medium 貼文]。Medium。https://medium.com/麥克的半路出家筆記/筆記-認識-oauth-2-0-一次了解各角色-各類型流程的差異-c42da83a6015

Petertc [Petertc]（2019，4 月 27 日）。*每當點選社群帳號登入，背後發生了什麼事？* [Medium 貼文]。Medium。https://petertc.medium.com/openid-connect-a27e0a3cc2ae

王思翰（2009）。*合作式跨網站指令碼攻擊之防禦機制* [碩士論文，中原大學]。臺灣博碩士論文知識加值系統。

王新棟（2020，7 月 16 日）。*實踐 OAuth 2.0 時，使用不當可能會導致哪些安全漏洞？*極客時間。https://time.geekbang.org/column/article/261403?utm_source=related_read&utm_medium=article&utm_term=related_read

李靜宜（2020，8 月 21 日）。*【臺灣資安大會直擊】政大金融科技研究中心副主任陳恭：API 有 3 大資安風險 2 大挑戰，靠 AI 即時偵測異常是新對策*。iThome 文章。https://www.ithome.com.tw/news/139531

林岑俞、吳家瑄（2019）。*邁向開放金融打造穩定安全的資料共享環境*。勤業眾信金融服務產業團隊。https://www2.deloitte.com/tw/tc/pages/financial-services/articles/Open-Finance-insight.html

姚惠茹（2021，9 月 2 日）。*開放銀行是什麼？台灣正值第二階段「客戶資料查詢」*。財經新報報導。http://ftrc-ob.nccu.edu.tw/

高敬原（2019，9 月 23 日）。*開放銀行第一階段月底上線，銀行業坦言：初期消費者感受不大*。數位時代報導。http://ftrc-ob.nccu.edu.tw/

國立政治大學金融科技研究中心（2019）。*開放銀行(Open Banking）是什麼？*國立政治大學金融科技研究中心官網介紹。http://ftrc-ob.nccu.edu.tw/origin

國立政治大學金融科技研究中心（2019）。*什麼是 Open API？*國立政治大學金融科技研究中心官網介紹。http://ftrc-ob.nccu.edu.tw/origin

國立政治大學金融科技研究中心（2019）。*開放銀行創造三贏*。國立政治大學金融科技研究中心官網介紹。http://ftrc-ob.nccu.edu.tw/

張婉君（2021）。*從生態圈角色及資安、法遵等風險面向論台灣推動開放銀行之重要因素* [碩士論文，世新大學]。臺灣博碩士論文知識加值系統。

智慧年代 IBM 藍色觀點（2018）。*無所不在的銀行*。天下實驗室文章。https://event.cw.com.tw/2018ibm/article/index2/article1.html

遠見雜誌整合傳播部（2021，12 月 30 日）。*邁向疫情後金融業新常態：擁抱 API 經濟與資料共享，以生態系創造多贏*。遠見雜誌產經文章。https://www.gvm.com.tw/article/85654

蔡峻福（2016，1 月 14 日）。*雲端個人身分鑑別* [碩士論文，明新科技大學]。臺灣博碩士論文知識加值系統。

蘇文彬（2023，4 月 1 日）。*金管會揭我國金融科技發展路徑圖推動現況，今年將發表路徑圖 2.0*。iThome 文章。https://www.ithome.com.tw/news/156236

二、 英文文獻（A - Z 排序）

Aaron Parecki（2018, April 10）. What is the OAuth 2.0 Authorization Code Grant Type?. okta. https://developer.okta.com/blog/2018/04/10/oauth-authorization-code-grant-type#exchange-the-authorization-code-for-an-access-token

Amit Klein（2005, July 4）. DOM Based Cross Site Scripting or XSS of the Third Kind. http://www.webappsec.org/projects/articles/071105.shtml

Anupam , M.（2020, November 11）. Access Token Vs ID Token Vs Refresh Token - What? Why? When? Medium. https://www.c-sharpcorner.com/article/accesstoken-vs-id-token-vs-refresh-token-what-whywhen

OpenID Connect.（2023, February 7）. WebSphere Application Server Liberty.https://www.ibm.com/docs/zh-tw/was-liberty/base?topic=liberty-openid-connect

D. Fett, R. Küsters and G. Schmitz, "The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines," 2017 IEEE 30th Computer Security Foundations Symposium（CSF）, Santa Barbara, CA, USA, 2017, pp. 189-202, doi: 10.1109/CSF.2017.20.

D. Hardt, Ed.（2012, October）. The OAuth 2.0 Authorization Framework（RFC-6749）.

Kawasaki, T.（2017, October 31）. Diagrams of All The OpenID Connect Flows. Medium. https://darutk.medium.com/diagrams-of-all-the-openid-connect-flows-6968e3990660

D. Fett et al.（2019）. "An Extensive Formal Security Analysis of the OpenID Financial-Grade API". IEEE Symposium on Security and Privacy（SP）. pp. 453-471

Mitchell Anicas（2021, July 29）. An Introduction to OAuth 2. DigitalOcean. https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

OpenID Foundation（n.d., Retrieved June 08, 2023, from）. https://openid.net/wg/fapi/

Sakimura, N., Bradley, J., Jones, M., De Medeiros, B., & Mortimore, C.（2014）. Openid connect core 1.0. The OpenID Foundation, S3.

Sakimura, N.（2014）. OpenID Connect Core 1.0 Incorporating Errata Set 1. OpenID. https://openid.net/specs/openid-connect-core-1_0.html