OAuth 2.0 筆記 (6) Bearer Token 的使用方法

Sep 30, 2013

這篇不屬於 OAuth 2.0 規格書（RFC 6749）本身，而是屬於另一份 spec *RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage* 。我認為它存在的目的是「示範一下 Token 的用法，並且定義下來，讓大家可以參考」，因為 OAuth 2.0 規格書沒有明確規定「Token 長什麼樣子」，甚至「Resource Server 如何拒絕非法的 Token」（指 API）都沒定義，只規定了怎麼拿取、怎麼撤銷、怎麼流通。

實際上，即使有定義這個 Bearer Token ，各大網站的 API 也並非都使用這種 Token ，我看到有明確說明使用 Bearer Token 的像是 Twitter API，其他的要不是非使用 "Bearer" 關鍵字，就是沒有明確指出何種 Token （其實也不需要，因為在那些網站 Token 只有一種用途）。

不過即使如此，對於我打算實作的 API ，我也是準備使用 Bearer Token 的，因為夠 naïve 。如果你跟我一樣沒有自己刻 Token 的能力，就用 Bearer Token 就好了。

當然， RFC 6750 我也有轉成 Markdown 好讀版。

Bearer Token 的用途

OAuth 2.0 (RFC 6749) 定義了 Client 如何取得 Access Token 的方法。Client 可以用 Access Token 以 Resource Owner 的名義來向 Resource Server 取得 Protected Resource ，例如我 (Resource Owner) 授權一個手機 App (Client) 以我 (Resource Owner) 的名義去 Facebook (Resource Server) 取得我的朋友名單 (Protected Resource)。OAuth 2.0 定義 Access Token 是 Resource Server 用來認證的唯一方式，有了這個， Resource Server 就不需要再提供其他認證方式，例如帳號密碼。

然而在 RFC 6749 裡面只定義抽象的概念，細節如 Access Token 格式、怎麼傳到 Resource Server ，以及 Access Token 無效時， Resource Server 怎麼處理，都沒有定義。所以在 RFC 6750 另外定義了 Bearer Token 的用法。Bearer Token 是一種 Access Token ，由 Authorization Server 在 Resource Owner 的允許下核發給 Client ，Resource Server 只要認這個 Token 就可以認定 Client 已經經由 Resource Owner 的許可，不需要用密碼學的方式來驗證這個 Token 的真偽。關於 Token 被偷走的安全性問題，此 Spec 裡面也有提到。

本段參考 Abstract 及 Section 1

Bearer Token 的格式

```
Bearer XXXXXXXX
```

其中 `XXXXXXXX` 的格式為 b64token ，ABNF 的定義：

```
b64token = 1*( ALPHA / DIGIT / "-" / "." / "_" / "~" / "+" / "/" ) *"
```

寫成 Regular Expression 即是：

```
/[A-Za-z0-9\-\._~\+\/]+=*/
```

本段參考 Section 2.1

Client 向 Resource Server 出示 Access Token 的方式

三種

(1) 放在 HTTP Header 裡面

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

Resource Server 必須支援這個方式。

本段參考 Section 2.2

(2) 放在 Request Body 裡面（Form 之類的）

```
POST /resource HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

access_token=mF_9.B5f-4.1JqM
```

前提：

- Header 要有 `Content-Type: application/x-www-form-urlencoded`。
- Body 格式要符合 W3C HTML 4.01 定義 application/x-www-form-urlencoded。
- Body 要只有一個 part （不可以是 multipart）。
- Body 要編碼成只有 ASCII chars 的內容。
- Request method 必須是一種有使用 request-body 的，也就是說不能用 `GET` 。

就是送表單嘛，但不可以是 `multipart/form-data` 這種（通常用來上傳檔案）。

Resource Server 可以但不一定要支援這個方式。

本段參考 Section 2.3

(3) 放在 URI 裡面的一個 Query Parameter （不建議）

規定要使用 `access_token` 這個 parameter ，例：

```
GET /resource?access_token=mF_9.B5f-4.1JqM HTTP/1.1
Host: server.example.com
```

然而因為 URL 可以被 proxy 抄走（如 log）或存在瀏覽器的歷史記錄裡面，為了防 replay ，最好這樣做：

- Client 送 `Cache-Control: no-store` header
- Server 回 `2xx` 的時候，送 `Cache-Control: private` header

Spec 不建議使用這種方法，如果真的沒辦法送 header 也沒辦法透過 request-body 送，再來考慮這種。

Resource Server 可以但不一定要支援這個方式。

本段參考 Section 2.4

Resource Server 向 Client 提示「認證不過，拒絕存取」的方式

拒絕存取的情況，例如沒給 Access Token 或是給了但不合法（如空號、過期、Resource Owner 沒許可 Client 拿取此資料），則 Resource Server 必須在回應裡包含 `WWW-Authenticate` 的 header 來提示錯誤。這個 header 定義在 RFC 2617 Section 3.2.1。`WWW-Authenticate` 的值，使用的 auth-scheme 是 `Bearer` ，隨後一個空格，接著要有至少一個 auth-param 。

範例：

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example",
                  error="invalid_token",
                  error_description="The access token expired"
```

以下這些 auth-params 是 `WWW-Authenticate` 會用到的：

參數名	必/選	填什麼/意義
realm	選用	見下文
scope	選用	提示所需權限，見下文
error	選用	有出示 Access Token 則最好有這個

realm

用 `realm` 來指出需要授權才能存取的範圍，意義跟 HTTP Authentication 的 `realm` 一樣。`realm` 只能出現一次。

scope

用 `scope` 來指出「要拿這個 Resource 需要出示具有哪些 scope 的 Access Token」：

- 要區分大小寫。
- 要以空白分隔。
- 可以用哪些 scope ，是看 Authorization Server 怎麼定義，Spec 不定義，也沒有登錄中心。
- 順序不重要。
- 是給程式看的，不是設計給使用者看的。

`scope` 還可以在向 Authorization Server 索取新 Access Token 的時候使用。

`scope` 值只能出現一次。實際寫在 `scope` 裡面的單一個 scope 必須只能用以下的字元，定義在 RFC 6749 附錄 A.4 ：

```
\x21, \x23-\x5b, \x5d-\x7e
```

即可見的 US-ASCII 字元裡面，除了雙引號 " (\x22) 和反斜線 \ (\x5c) 以外。空格當然也不能用，因為是用來區分不同 scopes 的。

error

如果 Client 出示了 Access Token 但認證失敗，則最好加上 `error` 這個 auth-param ，用來告訴 Client 為何認證失敗。此外還可以加上 `error_description` 用自然語言來告訴開發者為什麼錯誤，但這個不該給使用者看到。此外也可以加上 `error_uri` 用來提供一個網址，裡面用自然語言解釋錯誤訊息。這三個 auth-param 都只能最多出現一次。

如果 request 沒有出示 Access Token （例如 Client 不知道需要認證，或是使用了不支援的認證方式（例如不支援 URI parameter）），則 response 不應該帶 `error` 或任何錯誤訊息。

`error` 的值的意義以及推薦使用的 HTTP response code 如下：

值	Status Code	意義/用途
invalid_request	400 Bad Request	沒提供必要的參數、提供了不支援的參數、提供了錯誤的參數值、同樣的參數出現多次、使用一種以上的方法來出示 Access Token （如放在 header 裡又放在 form 裡）、或是其他無法解讀 request 的情況。

值	Status Code	意義/用途
invalid_token	401 Unauthorized	Access Token 過期、被收回授權、無法解讀、或其他 Access Token 不合法的情況。這種情況下， Client 可以重新申請一個 Access Token 並且用新的 Access Token 來重試 request 。
insufficient_scope	403 Forbidden	這個 request 需要出示比 Client 出示的 Access Token 代表的 scopes 還要更多的 scopes 。這種情況下，可以另外提供 `scope` auth-param 來具體指出需要哪些 scopes 。

`error` 和 `error_description` 的值必須只能用以下的字元，定義在 RFC 6749 附錄 A.7 和 RFC 6749 附錄 A.8：

```
\x20-\x21, \x23-\x5b, \x5d-\x7e
```

即空格 (\x20) 再加上可見的 US-ASCII 字元裡面，除了雙引號 " (\x22) 和反斜線 \ (\x5c) 以外。

`error_uri` 的值必須符合 RFC 3986 的定義，即是只能用以下的字元（同 `scope` 裡面的單一 scope）：

```
\x21, \x23-\x5b, \x5d-\x7e
```

即可見的 US-ASCII 字元裡面，除了雙引號 " (\x22) 和反斜線 \ (\x5c) 以外。

本段參考 Section 3 及 Section 3.1

Authorization Server 給 Client 核發 Access Token 的範例

既然是 OAuth 2.0 的 access token ，就通常是循 OAuth 2.0 的 spec 來核發，範例如下：

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"mF_9.B5f-4.1JqM",
  "token_type":"Bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

本段參考 Section 4

安全性問題與對策

RFC 6750 是基於 OAuth 2.0 RFC 6749 來寫的，所以在該 spec 裡面提過的安全性問題就不再提及。

原文將問題與問題對策分開成 Section 5.1 和 Section 5.2 ，我為了方便筆記，所以合併在一起。以下「壞人」的原文是 *attacker*。

一般對策

大部份的安全性問題可以透過數位簽章或是 MAC (Message Authentication Code) 來防護。

Authorization Server 必須有實作 TLS，版本則隨時間推移而不同。 spec 作成的時候，TLS 最新版是 1.2 ，但實務上很少使用，1.0 才是最為廣泛利用的。

偽造或竄改 Access Token 的問題

壞人可能會偽造或竄改既有的 Access Token （竄改指的是修改授權範圍或授權參數），讓 Resource Server 給予 Client 不適當的存取權。例如，壞人可能會延長 Token 的過期時間。或是惡意的 Client 變造聲明來看到它不應該看到的東西，例如，告訴使用者只拿取公開的個人資料，卻在取得授權時，另外拿了朋友名單。

Section 5.1 > Token manufacture/modification

對策

對於 Bearer Token ，可以只加一個參照用的 id 來間接指到真正的授權資訊，而不是直接燒在 Token 裡面。這種間接參照用的 id ，必須要難以被猜到；但使用間接參照，因為要間接檢查授權資訊，所以可能會導致 Resource Server 和 Authorization Server 之間有額外的動作※。這種機制的細節，spec 裡面沒有定義。

Spec 沒有定義 token 的編碼方式，所以不提及保護 Token 的完整性 (integrity) 的詳細建議。若要實作保護完整性的措施，則該實作方式必須要可以防止 Token 被竄改。

※：原文是 *"between a server and the token issuer"*

Access Token 傳輸過程外洩、曝露敏感資料的問題

Token 傳輸過程可能被監聽而外洩，或 Token 本身可能會包含敏感資料※。

※在 Section 5.1 原文提及 "token disclosure" 的時候，僅提及曝露敏感資料，沒提及傳輸過程的外洩，然而 5.2 裡面關於 "token disclosure" 的對策，有一併提及傳輸過程外洩（中間人攻擊、監聽等），所以我寫這一段時，同時提及傳輸外洩以及曝露敏感資料。

Section 5.1 > Token disclosure

對策

為了防範 Token 在傳輸過程外洩，必須用 TLS 來實作機密防護，且該實作方式必須要使用有提供機密防護和完整性防護的加密方式，如此就能要求 Client 與 Authorization Server 和 Client 與 Resource Server 之間的通訊要有機密防護和完整性防護。因為 TLS 是這份 spec 裡面規定一定要實作的，所以利用 TLS 來達成通訊過程的機密防護和完整性防護，是比較偏好的做法。

如果要防止 Client 取得 Token 的內容，那麼除了 TLS 之外，還必須實作 Token 加密。

要進一步防範 Token 外洩，則 Client 在發 request 的時候，還必須要驗證 TLS 的憑證鏈 (certificate chain) ，包括檢查憑證有沒有被撤銷（Certificate Revocation List, RFC 5280）。

Cookie 通常是明文傳輸的 (in the clear)，所以任何寫在裡面的資訊都有外洩的風險。所以， Bearer Token 絕對不可以存放在明文傳輸 cookie 裡面。詳見 RFC 6265 (HTTP State Management Mechanism) 裡面關於 cookie 的安全性問題。

某些部署方式，好比說利用 Load Balancer 的，TLS 傳輸在抵達 Resource Server 之前就結束了。這樣子會導致 Token 在前端 Load Balaner 和後端實體 Resrouce Server 之間，沒有加密保護。這種情況下，必須實作足夠的手段※，來確保前端和後端 server 之間的資料保密。Token 加密也是一種方式。

※ 原文為 *sufficient measures*，我不會翻譯…

挪用 Access Token 的問題

壞人可能會把某個專門給 Resource Server A 的 Access Token ，挪用到 Resource Server B ，使得 B 誤信該 Access Token 可以拿來存取 B 的資料。

Section 5.1 > Token redirect

對策

要防止 Token 被挪用，則這件事很重要：Authorization Server 核發的 Token 裡面要附上被核發人的資訊（通常是一或多部 Resource Server）。同時，也建議限制 Token 可以使用的 scope 範圍。

二度利用 Access Token 的問題

壞人使用之前就存在的 Access Token 來存取 Reseouce Server （即：Token 被偷去用）。

Section 5.1 > Token replay

對策

要防止 Token 被偷走並且拿來二度利用，建議採用以下方案：

1. Token 的存活時間必須被限制住。一種手段是在 Token 受保護的區段裡面，設一個合法期間。使用短時效的 Token （如一小時以下）可以降低 Token 外洩的風險。

- Token 在 Client ↔ Authorization Server 、 Client ↔ Resource Server 之間交換的時候，必須要實作機密防護。如此一來，就算在傳輸途徑上監聽，也無法獲得 Token ，也就無法二度利用。
- Client 要向 Resource Server 出示 Token 的時候，Client 必須驗證 Resource Server 的真實身份，如 RFC 2818 (TLS) 的 Section 3.1 裡面所述。注意，Client 必須要驗證 TLS 憑證的憑證鏈 (certificate chain)。若 Resource Server 未經授權且未通過認證，或是憑證鏈驗證失敗，這時候向它出示 Token ，會導致敵手取得 Token 並且得到未經授權的權限來存取受保護的 resource。

安全性建議的總結

Section 5.3 Summary of Recommendations

要藏好 Bearer Token

Client 實作必須確保 Bearer Token 不會外洩給無關人士，因為他們可以以此來存取受保護的 resources。利用 Bearer Token 時，這是首要的安全性考量，且優先於其他更細節的建議。

要驗證 TLS 的憑證鏈

當 Client 發 request 索取受保護的 resources 的時候，Client 必須驗證 TLS 的憑證鏈。若做不到的話，可能會引發 DNS 劫持，導致 Token 被壞人偷走。

全程使用 TLS (https)

當 Clients 利用 Bearer Token 發 request 時，Client 必須一直使用 TLS (RFC5246) (https) 或同等的安全傳輸。若做不到的話， Token 會曝露在各種攻擊方式，讓壞人可以得到意料之外的存取權。

不要把 Bearer Token 存在 Cookie

絕對不可以把 Bearer Token 存在可以明文傳輸 (sent in the clear) 的 Cookie 裡面（明文傳輸是 cookie 傳輸的預設方式）。若存在 Cookie 裡面，必須要小心 Cross-Site Request Forgery 。

要核發短時效的 Bearer Token

核發 Token 的伺服器最好是核發短時效的 Bearer Token （一小時以內），尤其是發給跑在瀏覽器裡面的 Client ，或是其他容易發生資訊外洩的場合。利用短時效的 Bearer Token 可以降低 Token 外洩時的衝擊。

要核發有區分使用範圍的 Bearer Token

Token 伺服器最好要核發包含 audience restriction, scoping their use to the intended replying party, or set of relying party 的 Token 。

不要用 Page URL 來傳送 Bearer Token

Bearer Token 最好不要從 URL 來傳送（例如 query parameter），而最好是從有保密措施的 HTTP header 或是 body 來傳輸※。瀏覽器、伺服器等軟體可能不會把歷史記錄或資料結構給妥善加密。如果 Bearer Token 透過 URL 傳輸，則壞人就有可能可以從歷史記錄取得之。

※ *"be passed in HTTP message headers or message bodies for which confidentiality measures are taken"*

OAuth 2.0 系列文目錄

Yu-Cheng Chuang's Blog © 2009 to 2023 Yucheng Chuang