tags: `PortSwigger-note` `is1ab-note`

PortSwigger：About SSRF

Introduction

- 目標：寄送 request 到 非預期的地方
- 一般來說：連接至 internal-only 類型的服務
- 其他情況：強迫 server 連接至任意的 external 系統 → 敏感資料洩漏

Impact

- Result in：
  - (1) 未經授權的動作
  - (2) 組織內部資料存取權
  - (3) 可執行任意指令

Common SSRF attacks

SSRF attacks against the server

- 目標：透過網路介面的 loopback (回送，一種檢驗機制)來向應用程式所在的 server 回傳 HTTP 請求
- PortSwigger Lab：https://hackmd.io/@AndyShen/S1jFhatF1x
- 應用程式的行為緣由？
  - (1) 存取控制的檢查是由前端 server 實作：

- **示例**

  假設系統架構如下：

  1. **前端組件（API Gateway）** 負責身份驗證，只有授權的請求才會被轉發到後端伺服器。
  2. **後端應用程式伺服器** 接收 API Gateway 轉發的請求，並提供服務。

  正常情況下：

  - 用戶 → **API Gateway**（檢查權限）→ **應用程式伺服器**（執行請求）

- 但如果攻擊者發現應用程式伺服器沒有額外進行身份驗證，那麼攻擊者可以 **直接向應用程式伺服器發送請求**，從而繞過 API Gateway 的存取控制，例如：

  ```nginx
  curl http://backend-server/secret-data
  ```

  - (2) **基於災後重建的目標： 當管理員遺失憑證時，可以提供給管理員修復系統**
  - (3) **管理員介面可能會監聽不同的 ports (通常無法被使用者直接接觸到)**

SSRF attacks against the other back-end systems

- **特徵：**
  - **應用程式的 server 能夠與使用者無法接觸的後端溝通**
  - **這些系統通常配置了私有且不可路由(路由 : route)的 IP 位址**
  - **由於後端系統通常依賴網路拓撲進行保護，所以安全防禦較為薄弱**
- **PortSwigger Lab：https://hackmd.io/@AndyShen/rJHrsRtKkg**

Circumventing (規避) common SSRF defenses

SSRF with blacklist-based input filters：

- 利用以下技巧規避黑名單輸入過濾：
  - 利用 `127.0.0.1` 的不同表示法
    (such as `2130706433`, `017700000001`, or `127.1`)
    (`127.0.0.1` = (01111111 00000000 00000000 00000001)$_2$ = (2130706433)$_{10}$ = (0177 0000 0001)$_8$)
  - 註冊一個自訂的網域並讓它指向 `127.0.0.1`
    (Register your own domain name that resolves to 127.0.0.1)
  - 利用 URL encoding 或是 case variation 混淆 (Obfuscate) 遭到過濾的字串 (blocked strings)
  - 提供一個可控的 URL 會重新導向到目標 URL
    (舉例來說：在重新導向的過程中，將 URL 從 `http:` 轉換為 `https:` ，用以繞過 SSRF 的過濾)
- PortSwigger Lab：https://hackmd.io/@AndyShen/rJg-n8lqye

SSRF with whitelist-based input filters

- 有些應用程式只允許白名單輸入
- 輸入很可能從一開始就被過濾，或是檢查有沒有包含特定字串。可以利用 URL 解析不一致性去繞過這些過濾方法
- URL 的規範中有許多容易被忽視的細節，尤其是在採用臨時實作的方式來解析，或是用下列方法驗證 URL ：
  - 在 URL 中透過 @ 字元，在主機名稱前插入帳號與密碼等憑證資訊
    - 例如：`https://expected-host:fakepassword@evil-host`
  - 在 URL 中透過 # 字元 來表示 URL 的片段識別符 (混淆或欺騙伺服器處理的 URL)
    - 例如：`https://evil-host#expected-host`
  - 利用 DNS 的命名階層，把需要的輸入嵌入自己控制的 DNS 名稱中
    - 例如：`https://expected-host.evil-host`

    - 利用 URL-encode 的技巧來混淆負責解析 URL 的 code ，這種方法對於「處理 URL-encoded 字符過濾的 code」 與 「處理後端 HTTP req 的 code」 有所差異時特別有效。也可以利用 double-encoding 的技巧造成不一致性 ，這種方法專門用來應對那些對接收資料重複 URL-decoding 的伺服器

    - 嘗試將以上方法混和使用！！

- PortSwigger Lab：https://hackmd.io/@AndyShen/BkcEZWLkee

Bypassing SSRF filters via open redirection

- 有時候可以利用開放重新導向漏洞來繞過基於 filter 的防禦

- 可以藉由構造一個符合過濾條件的網址，讓請求被重導向到想要的後端目標

    - 以下例子會重新導向至 `http://evil-user.net`

        - `.../product/nextProduct?currentProductId=6&path=http://evil-user.net`

    - 把上面的例子延伸，就能用來繞過針對 DNS name 的 URL filter

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118

stockApi=http://weliketoshop.net/product/nextProduct?currentProductI
```

    - 上述例子要成功最重要的是提供的stockAPI網址是否屬於允許的網域

- PortSwigger Lab：https://hackmd.io/@AndyShen/B1OrmtPkxe

Blind SSRF vulnerabilities

- Blind SSRF 漏洞通常發生在「使用者能讓應用程式向你提供的 URL 發送後端 HTTP request ，但是該 request 的 response 不會出現在前端(的 response) 當中」

- 這種類型的漏洞本身難以利用，但是一旦遭到利用就有可能在 server 或其他後端元件中造成 RCE

How to find and exploit blind SSRF vulnerabilities?

- 偵測有無 blind SSRF 漏洞最可靠的方法是用 out-of-band(OAST=Out-of-band Application Security Testing) 的技巧，主要是利用向攻擊者控制的外部系統發送 HTTP request ，然後監控與該系統的網路互動行為

- 在測試 SSRF 漏洞時，常會觀察到針對提供的特定域名進行 DNS 查詢，但後續卻無 HTTP request 的現象。這種情況通常發生於應用程式嘗試向目標 DNS name 發起 HTTP request 而觸發初始 DNS 查詢，但實際的 HTTP 連線卻被網路層過濾機制(在 OSI 模型的第三層（網路層）或第四層（傳輸層）進行的封包過濾)阻擋

Finding hidden attack surface for SSRF vulnerabilities

- 由於應用程式的正常流量會包含帶有完整 URL 的請求參數，大部份的 SSRF 漏洞很容易被找到

Partial URLs in requests

- 有時候應用程式只會將 hostname 或是部分的 URL 放入 request 參數當中，這些值隨後會在 server 端被組合成一個完整的 URL 並發送請求。因此，儘管這些值很容易被辨識為 hostname 或是 URL 路徑(潛在攻擊面很廣)，仍會因為無法控制最終被請求的整個 URL 導致 SSRF 的可利用性受到限制

URLs within data formats

- 有些應用程式會以允許包含 URL 的格式來傳輸資料，而這些 URL 可能會被該格式的資料的解析器發出請求，從而導致某些安全漏洞。像是 XML 資料格式，它在網頁應用中被廣泛用來在用戶端與伺服器之間傳遞結構化資料，當應用程式接收 XML 格式的資料並進行解析時就有可能出現 XXE Injection ，也可能因 XXE 而導致 SSRF

SSRF via the Referer header

- 某些應用程式使用伺服器端分析軟體來追蹤訪客行為。這類軟體通常會記錄請求中的 Referer 標頭，以便追蹤外部連結來源。分析軟體常會主動訪問 Referer 標頭中出現的第三方 URL ，目標是分析來源網站的內容，其中包括外部連結所使用的錨點文字。因此，Referer 標頭往往成為 SSRF 的重要攻擊面

- 註：錨點文字是 `<a>` 標籤中的可視文字內容，常見的像是 `<a href="#top">↑<\a>` 中， `↑` 就是錨點文字 (`#top` 屬於片段識別符， SSRF 繞過白名單過濾的技巧中有提到過)

PortSwigger SSRF cheatsheet

- **URL：https://portswigger.net/web-security/ssrf/url-validation-bypass-cheat-sheet**