NETWORK & MULTIMEDIA LAB

WEB SECURITY

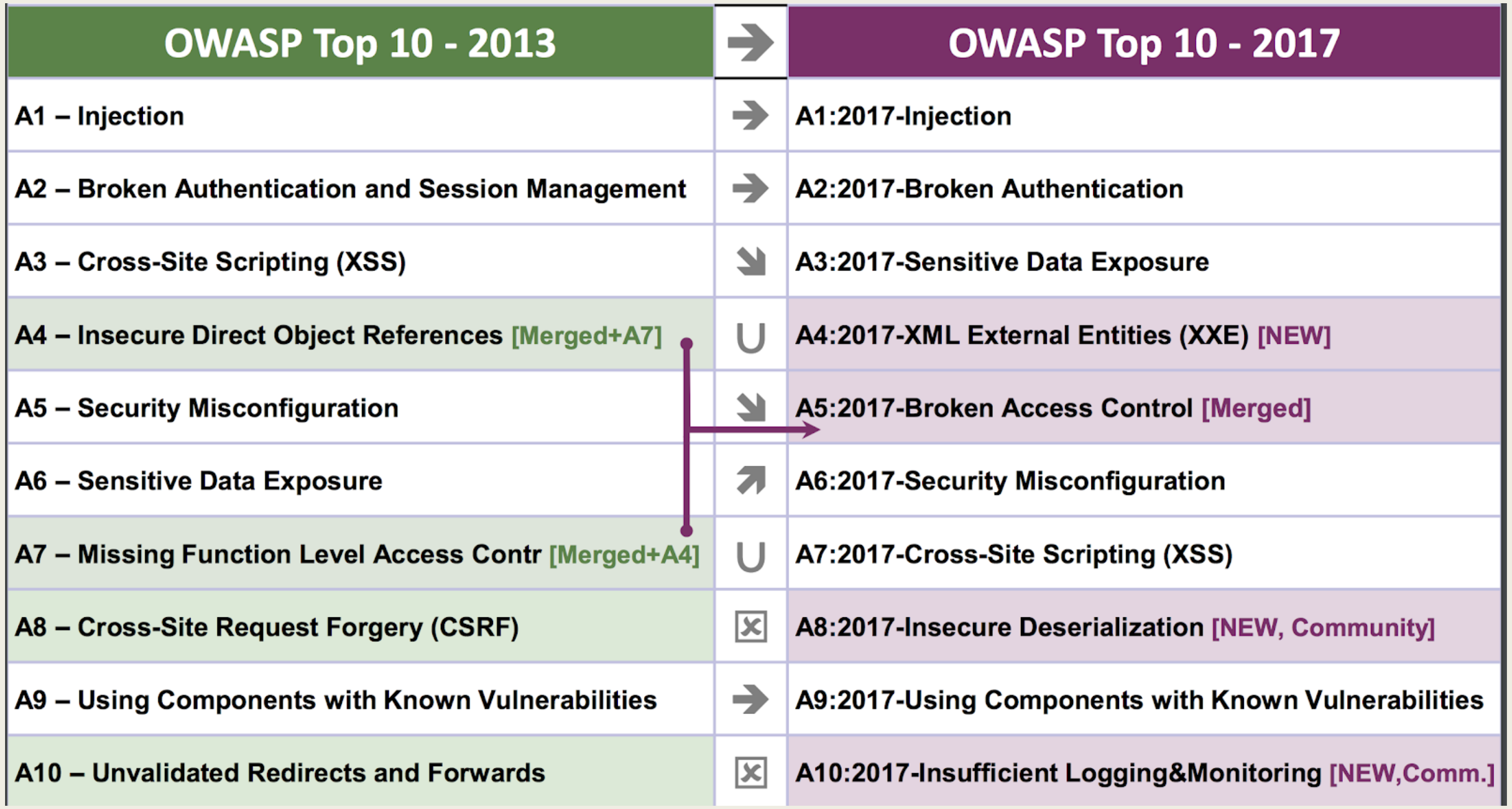
Fall 2020

Course Outline

- Cookie and Session
- Cross-Site Request Forgery (CSRF)
- Cross-Site Scripting (XSS)
- Server-Side Request Forgery (SSRF)

OWASP top 10

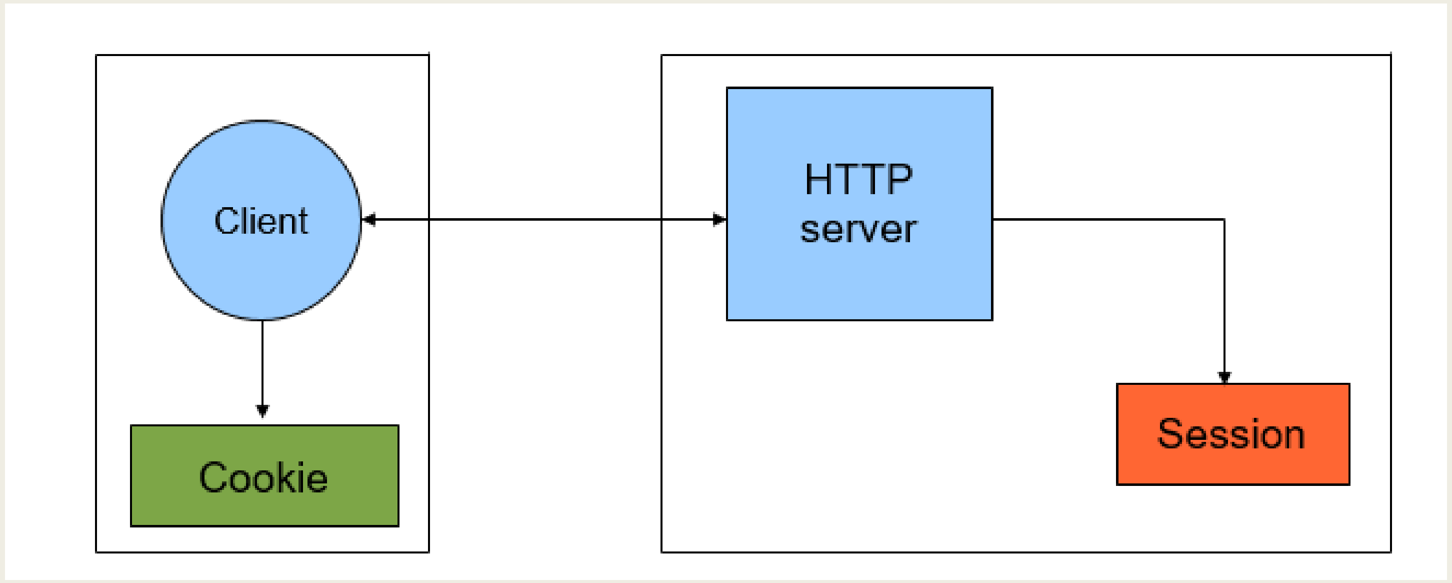
- Open Web Application Security Project
- OWASP 主要蒐集各種網頁安全漏洞，彙整為十大資安問題、排名、防範措施。



OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Cookie and Session

- HTTP 協定本身是無狀態的 **(Stateless protocol)**，亦即是說伺服器不會知道用戶上一次做了什麼。
- 如果一個網站要登入才能瀏覽，每次訪問該網站時(包含訪問不同頁面)，都需要將帳密再輸入一次



Introduces **state** into HTTP (remember: **HTTP is stateless**)

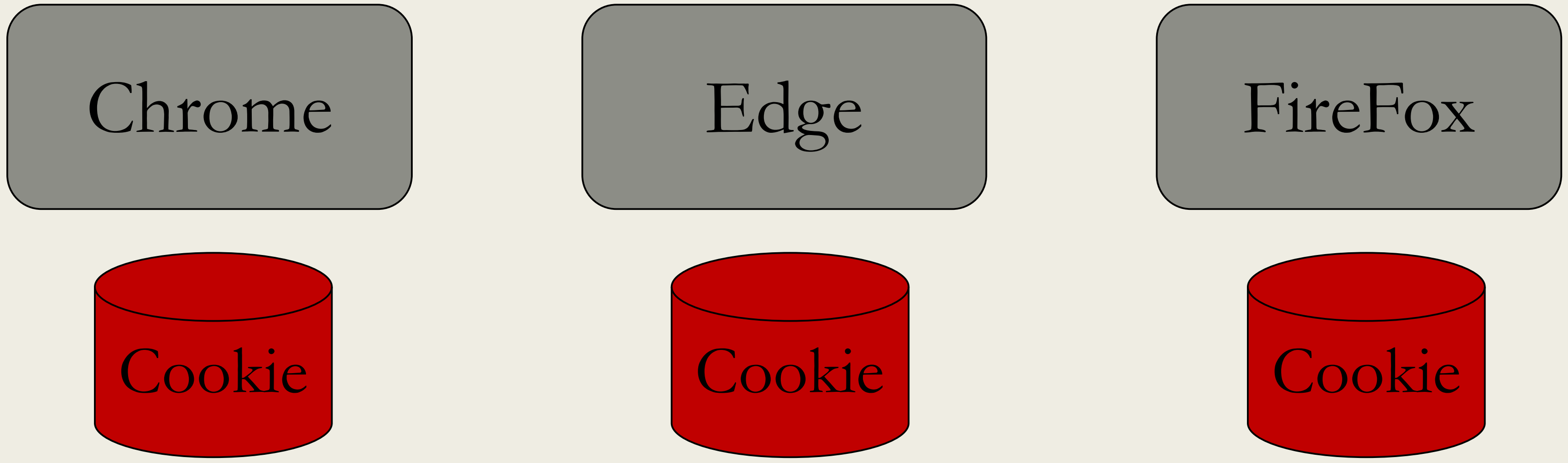
Cookie

- 網站為了辨別用戶身分而儲存在用戶端（Client Side）上的資料（通常經過加密）
- 瀏覽器會儲存它並且在瀏覽器下一次發送要求的時候將它送回原本送來的伺服器
- 可能包含用戶資訊
- 可以設定有效時間
- Doman-A 無法存取 Doman-B 的 cookies

- Cookie 的缺陷:
  - Cookie 會被附加在每個 HTTP 請求中，所以無形中增加了流量。
  - 由於在 HTTP 請求中的 Cookie 是明文傳遞的，所以安全性成問題，除非用超文字傳輸安全協定。
  - Cookie 的大小限制在 4KB 左右，對於複雜的儲存需求來說是不夠用的

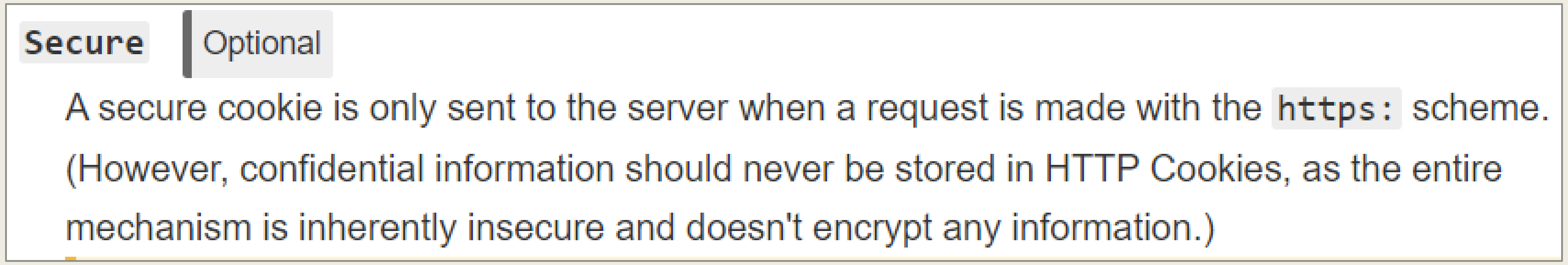
Cookie and Browser

- 每個瀏覽器都會以獨立的空間存放 Cookie
- 同一台電腦上使用同一瀏覽器的多用戶群，Cookie 不會區分他們的身分



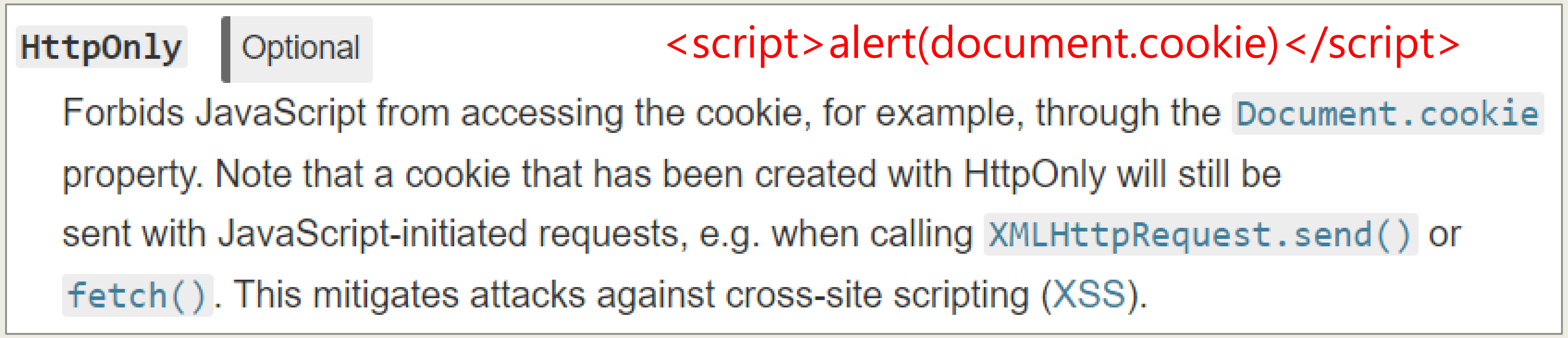
How Cookies are implemented

- 透過 “Set-Cookie” headers
  - *Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; Secure; HttpOnly; SameSite=<samesite-value>*



`Secure` Optional

A secure cookie is only sent to the server when a request is made with the `https:` scheme. (However, confidential information should never be stored in HTTP Cookies, as the entire mechanism is inherently insecure and doesn't encrypt any information.)



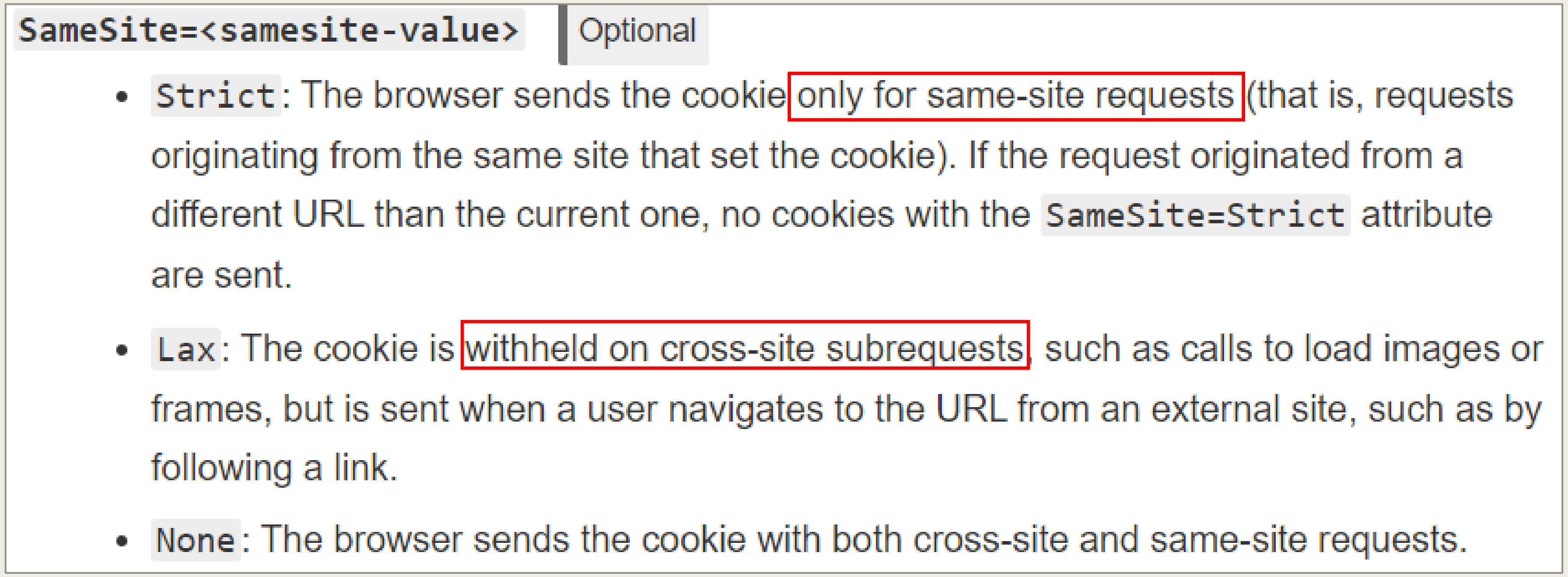
`HttpOnly` Optional

<script>alert(document.cookie)</script>

Forbids JavaScript from accessing the cookie, for example, through the `Document.cookie` property. Note that a cookie that has been created with HttpOnly will still be sent with JavaScript-initiated requests, e.g. when calling `XMLHttpRequest.send()` or `fetch()`. This mitigates attacks against cross-site scripting (XSS).

How Cookies are implemented

- 透過 “Set-Cookie” headers
  - *Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; Secure; HttpOnly; SameSite=<samesite-value>*

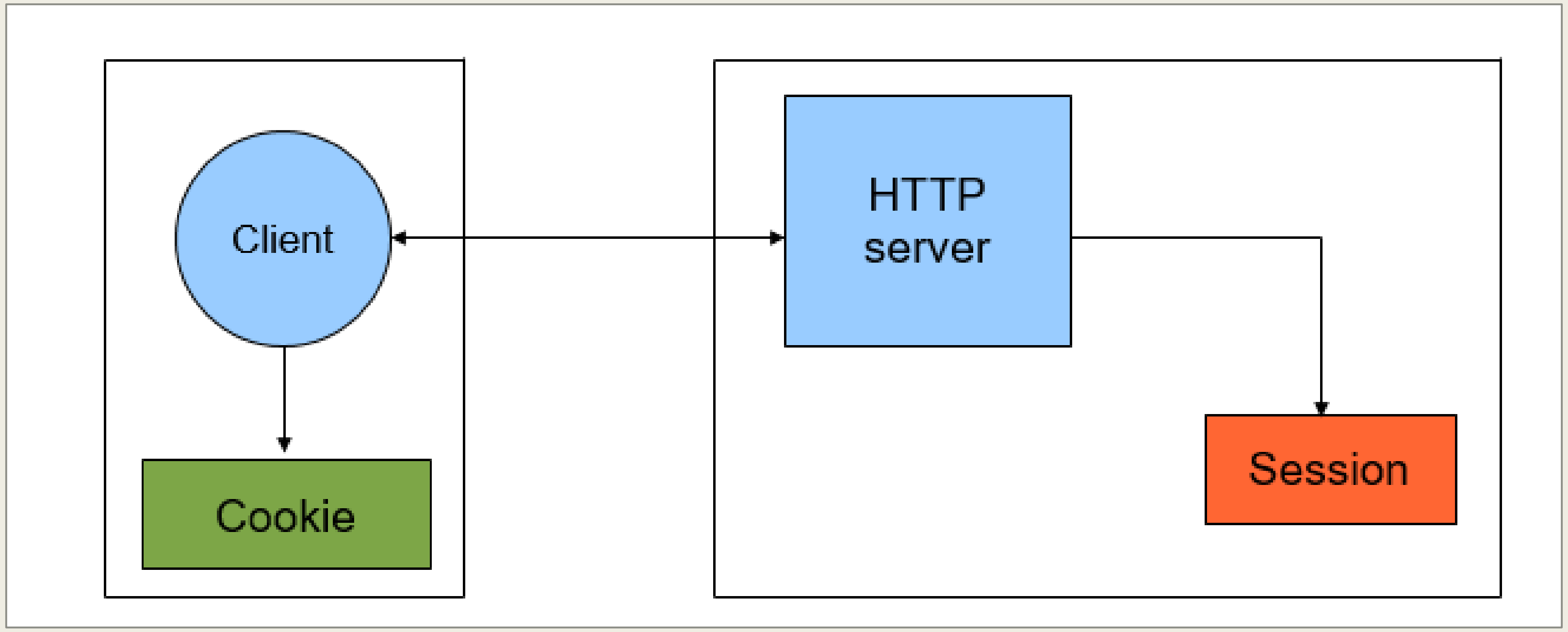


`SameSite=<samesite-value>` Optional

- `Strict`: The browser sends the cookie only for same-site requests (that is, requests originating from the same site that set the cookie). If the request originated from a different URL than the current one, no cookies with the `SameSite=Strict` attribute are sent.
- `Lax`: The cookie is withheld on cross-site subrequests, such as calls to load images or frames, but is sent when a user navigates to the URL from an external site, such as by following a link.
- `None`: The browser sends the cookie with both cross-site and same-site requests.

Session

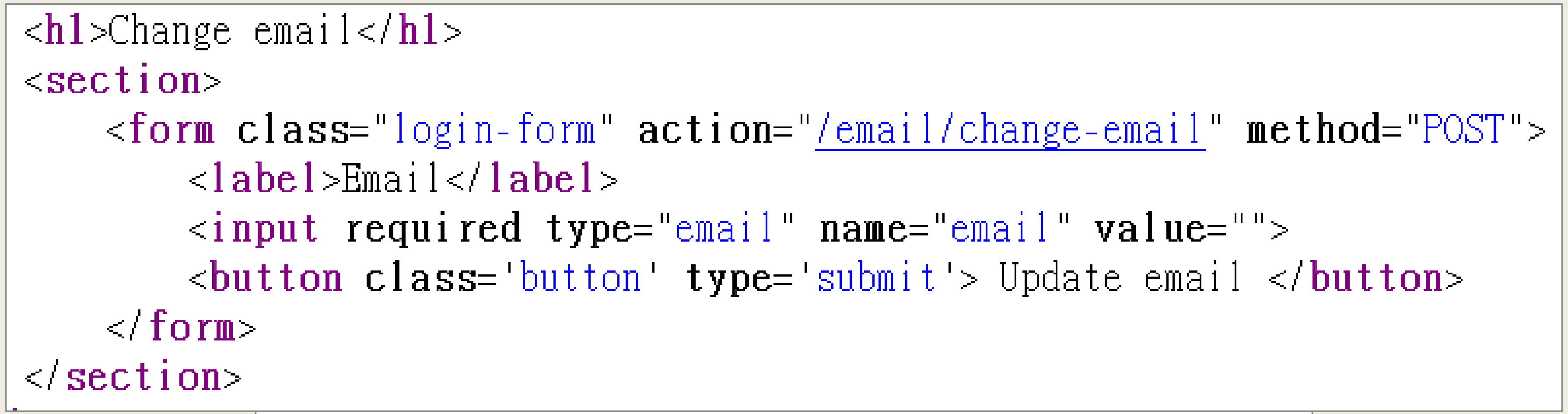
- Session 負責紀錄在 server 端上的使用者訊息，會在一個用戶完成身分認證後，存下所需的用戶資料，接著產生一組對應的 ID，以 cookie 的形式存在用戶端
- 之後使用者的 request 只要包含這個 cookie，server 便會認為這是一個已登入的使用者所發出的請求



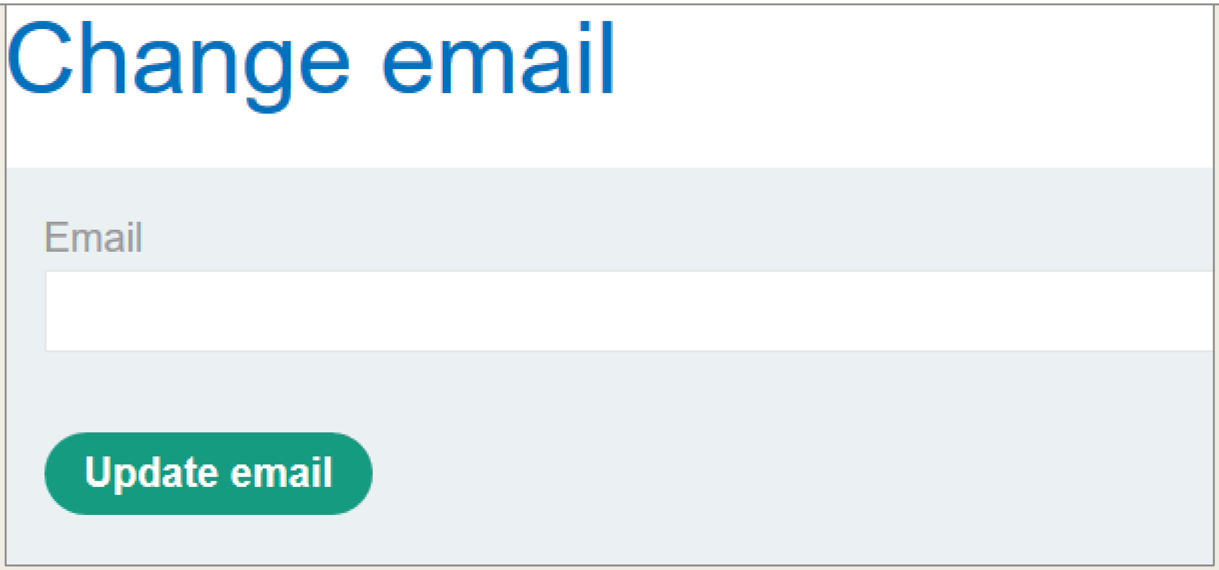
Cross-Site Request Forgery (CSRF)

- **跨站請求偽造**，也被稱為 **one-click attack** 或者 **session riding**
- 一種挾制用戶在當前已登入的 Web 應用程式上執行非本意的操作的攻擊方法

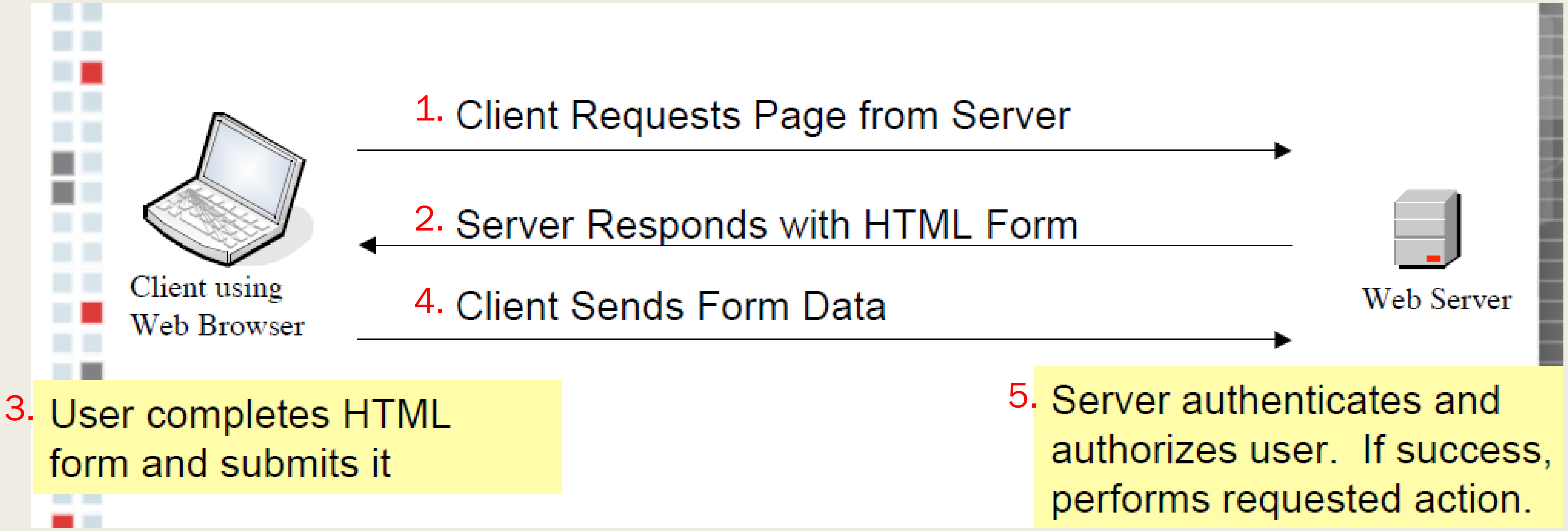
HTTP request methods



```
<h1>Change email</h1>
<section>
    <form class="login-form" action="/email/change-email" method="POST">
        <label>Email</label>
        <input required type="email" name="email" value="">
        <button class='button' type='submit'> Update email </button>
    </form>
</section>
```

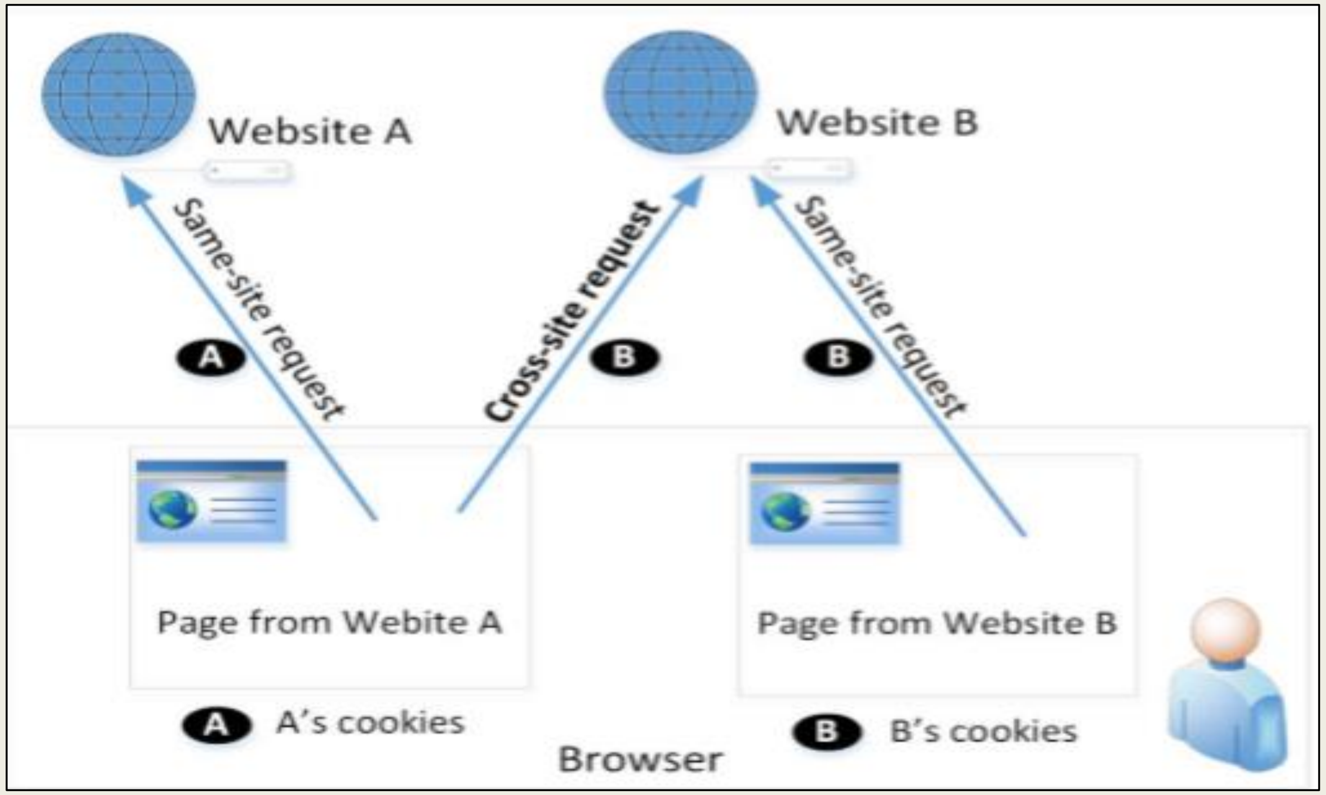


HTML Forms



1. Client Requests Page from Server
2. Server Responds with HTML Form
4. Client Sends Form Data
Client using Web Browser
Web Server
3. User completes HTML form and submits it
5. Server authenticates and authorizes user. If success, performs requested action.

Definition:
Cross-Site Requests & Same-Site Request



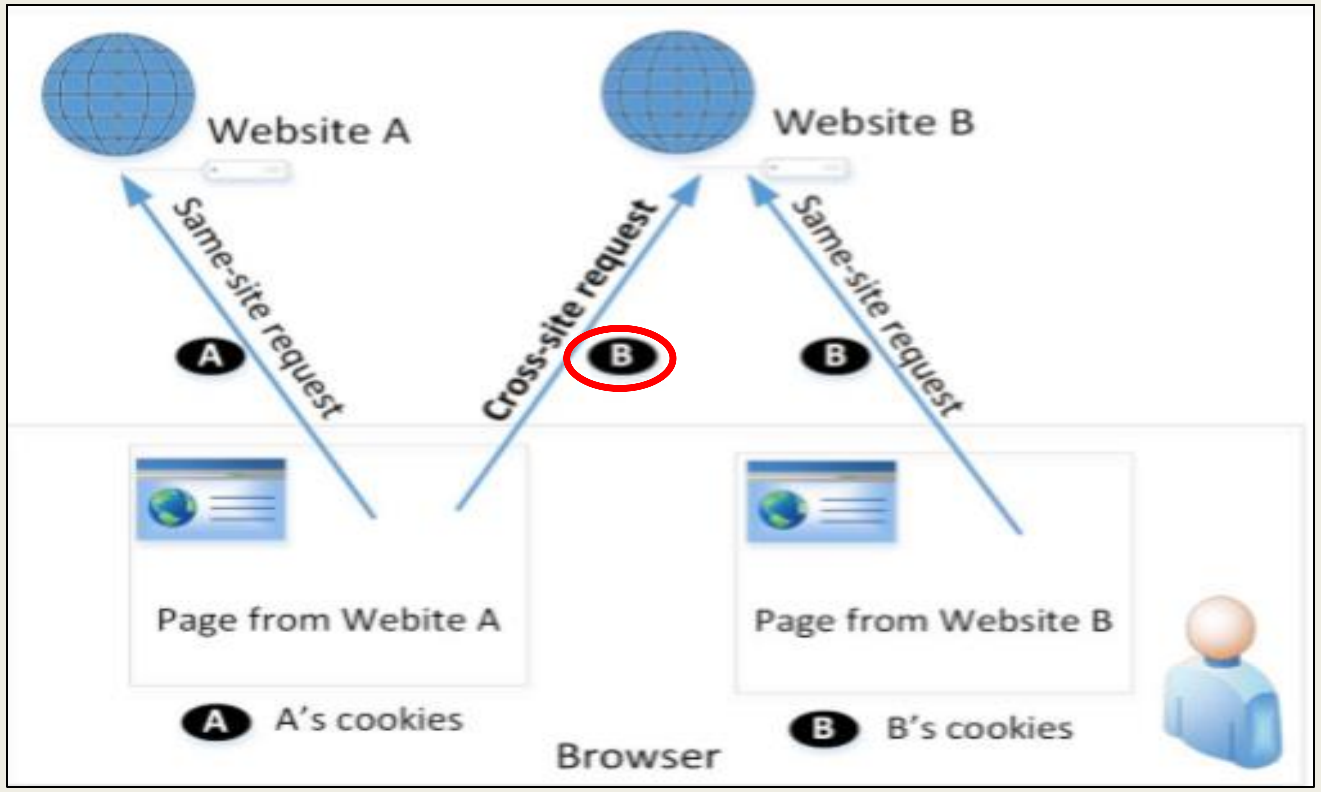
- same-site request:
  一個網站的頁面向自己發送請求

```
action="/email/change-email"
```

- cross-site request:
  A網站的頁面向B網站發送請求

```
action="http://www.example.com/action_post.php" method="post">
```

Cookies in Cross-Site Request

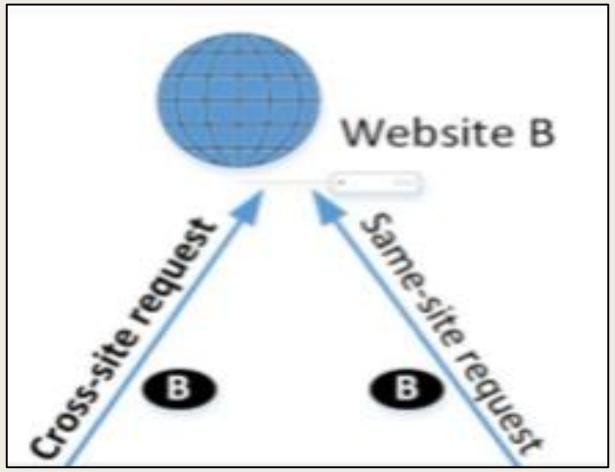


~~SameSite=Strict~~

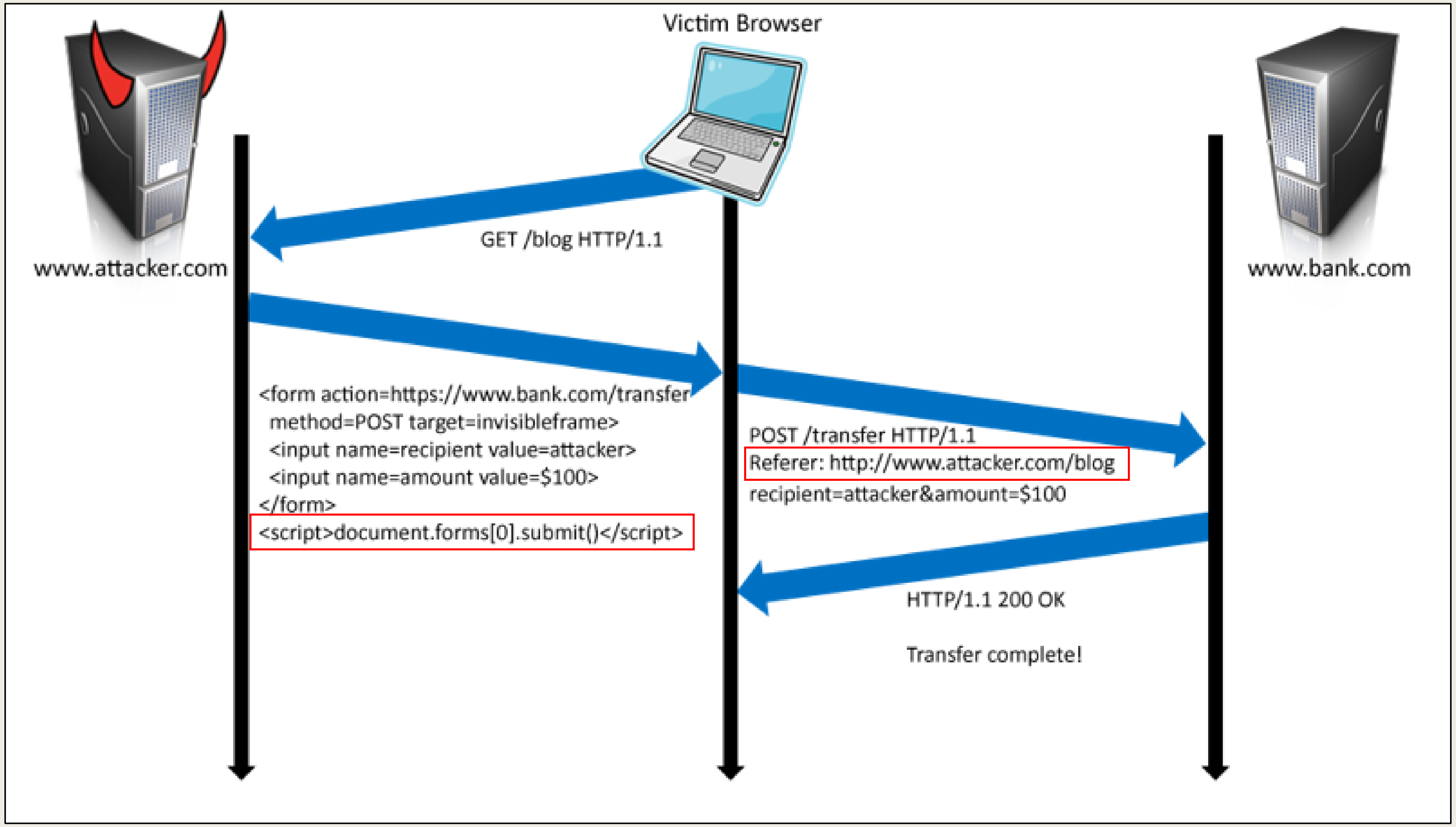
- 當 A.com 的頁面向 A.com 發送請求時，瀏覽器會將 A.com 的 cookies 一併送出
- 當 A.com 的頁面向 B.com 發送請求時，瀏覽器也會將 B.com 的 cookies 送出

Problems of Cross-Site Requests

- 因為 cross-site cookie 的特性， server 端無法分辨一個 request 是 same-site request 還是 cross-site request
- 第三方網站可以偽造一個請求，以跨站請求的方式送出，若瀏覽器附上 cross-site cookie (~~SameSite=Strict~~)，受害網站便會認為這是使用者發出的請求
- 因此這種攻擊手法稱為**跨站請求偽造**



CSRF Attack Flow



```
<form action=https://www.bank.com/transfer
  method=POST target=invisibleframe>
  <input name=recipient value=attacker>
  <input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
```

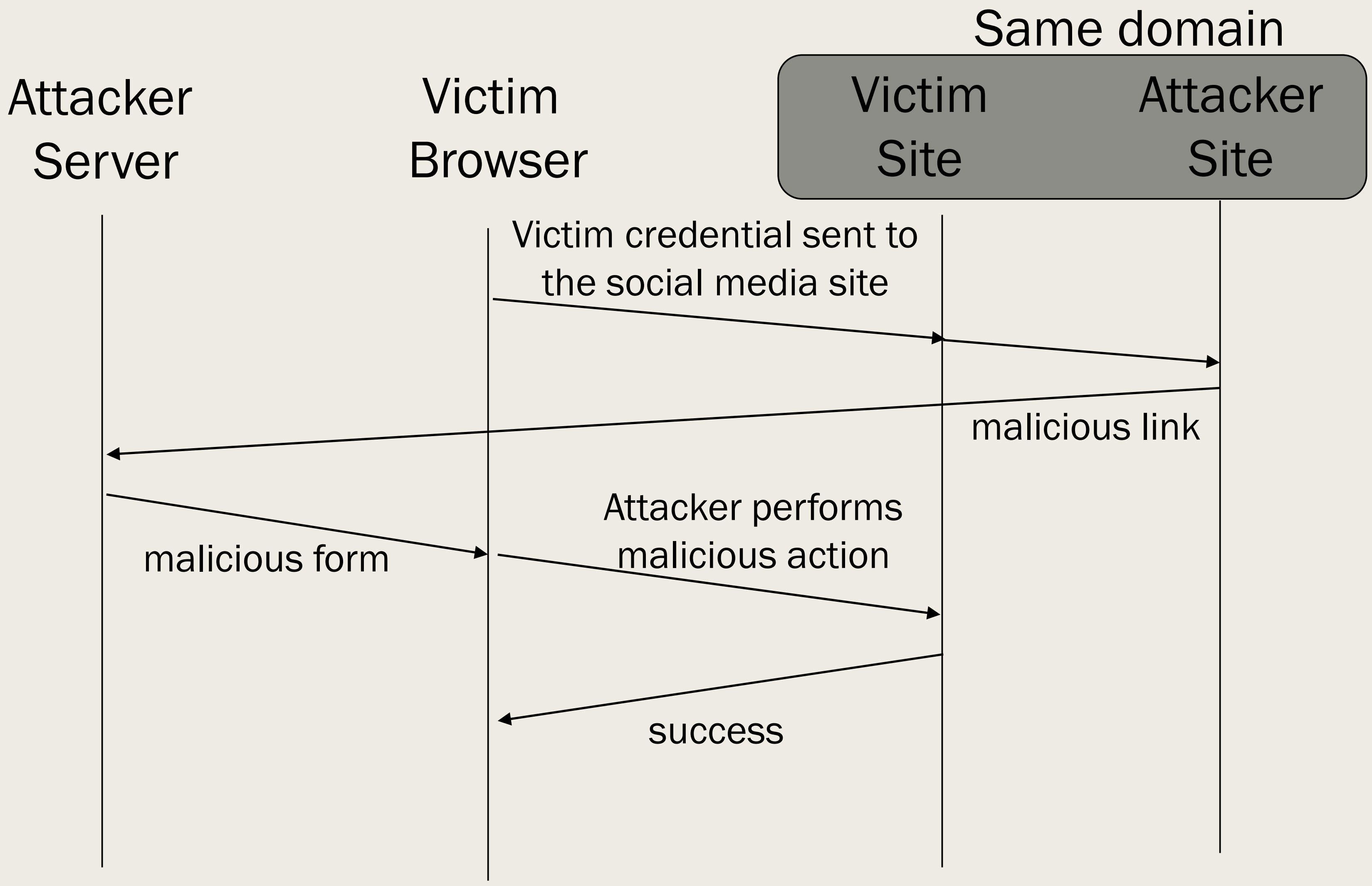
Attacking Scenario

- 受害者必須已經在受害網站上取得授權
  (已登入並取得 cookie)

- 攻擊者要想辦法讓受害者造訪惡意網站
  - *a discussion forum*
  - *HTML email body or attachment*
  - *or any location that a victim is likely to click while logged into the target site*

- 這個惡意網頁包含了偽造的請求，並自動發送給受害網站

CSRF Attack Flow

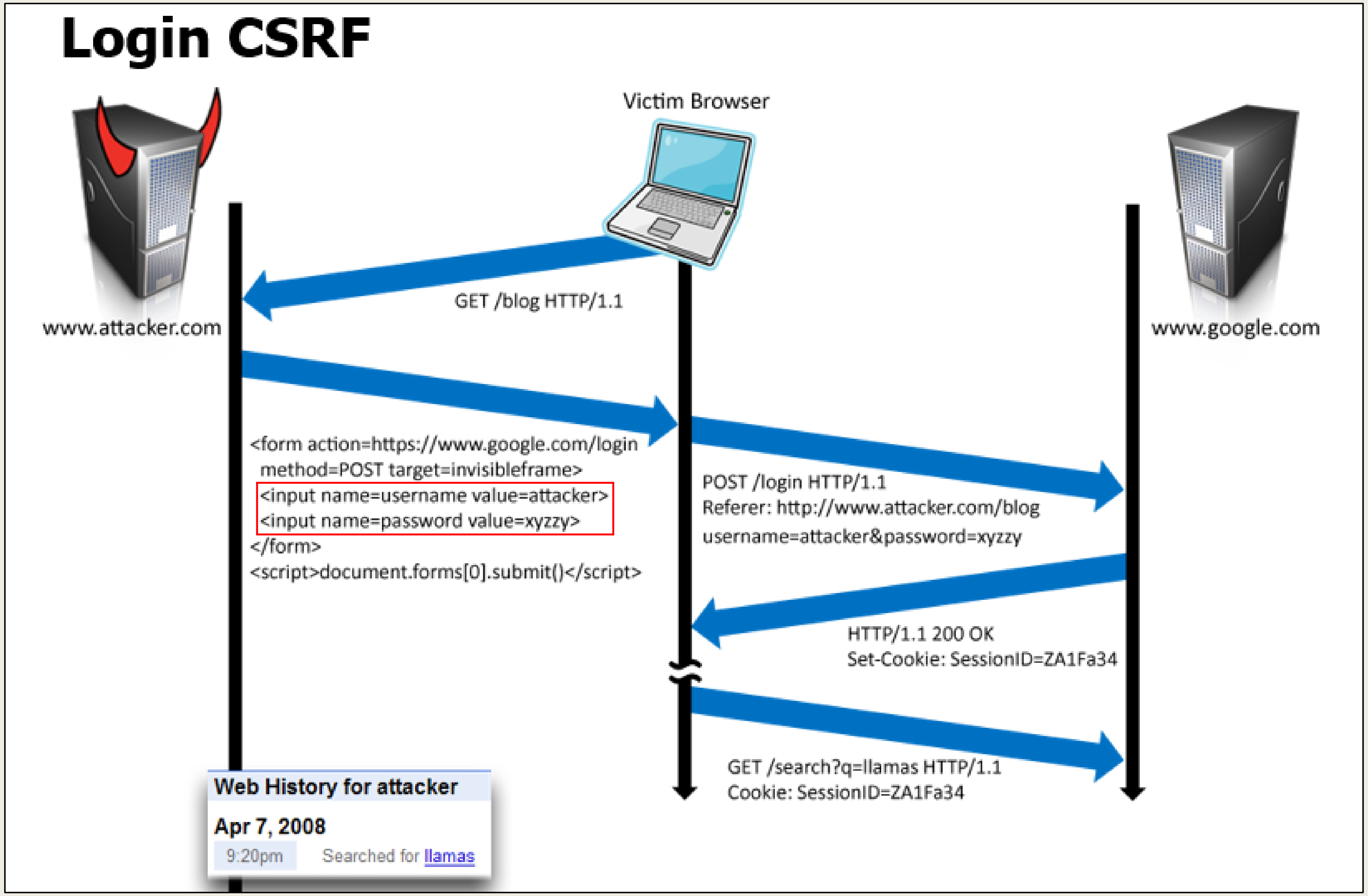


CSRF Login Attack

- 攻擊者偽造一個登入請求，使用的是攻擊者自己的帳號密碼

- 目的：攻擊者之後可以登入自己的帳號，查看歷史紀錄，得知受害者做了那些操作或留下哪些資訊

CSRF Login Attack



```
<form action=https://www.google.com/login
 method=POST target=invisibleframe>
 <input name=username value=attacker>
 <input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy
```

```
HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34
```

```
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
```

Web History for attacker

Apr 7, 2008

9:20pm Searched for llamas

Characteristics of CSRF

- 這個偽造的請求必須取得 Server 的信任 (透過取得 User 的身分)
  - e.g., User 的身分存在 cookie 中
  - 或其他任何 Server 用來辨識 User 的方法

- 使受害者的瀏覽器送出 HTTP requests 到受害網站

- 這個 HTTP request 必須能夠造成影響
  - e.g., 一個線上轉帳的 HTTP Post

CSRF Attack Methods (trigger request)

- JavaScript：
  攻擊者可以用 JavaScript 送出請求

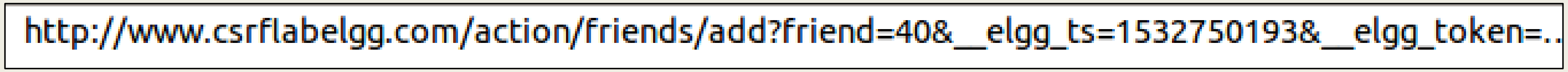
```
document.getElementsByName("myForm")[0].submit();
```

- <img> 和 <iframe> tag：
  HTML 的 <img> 和 <iframe> tag 會對 src attribute 所指定的 URL 送出 GET request

```
<img src="http://www.bank32.com/transfer.php?to=3220&amount=500">

<iframe
   src="http://www.bank32.com/transfer.php?to=3220&amount=500">
</iframe>
```

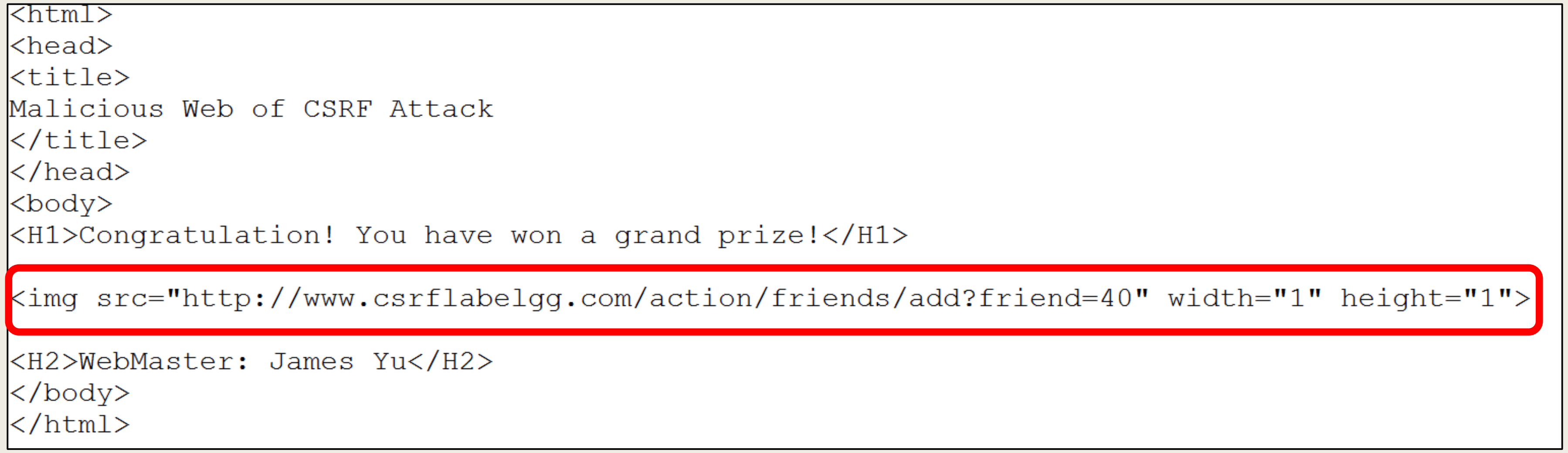
CSRF Attacks on HTTP GET Services

- When you click "Add Friend"



```
http://www.csrflabelgg.com/action/friends/add?friend=40&__elgg_ts=1532750193&__elgg_token=..
```

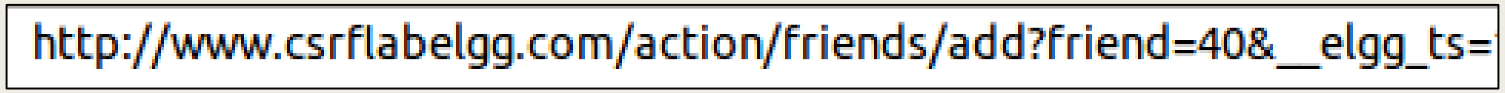
- The img tag will trigger an HTTP GET request.



```
<html>
<head>
<title>
Malicious Web of CSRF Attack
</title>
</head>
<body>
<H1>Congratulation! You have won a grand prize!</H1>

<img src="http://www.csrflabelgg.com/action/friends/add?friend=40" width="1" height="1">

<H2>WebMaster: James Yu</H2>
</body>
</html>
```

Key Elements of CSRF (Attacker’s View)

- 攻擊者必須找到受害網站處理請求的 URL



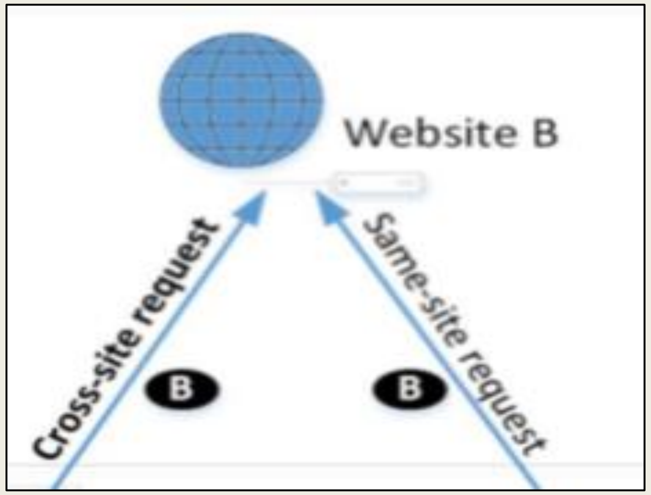
http://www.csrflabelgg.com/action/friends/add?friend=40&__elgg_ts=

- 攻擊者必須找出這個請求所有的必要欄位
(POST: form values, GET: URL inputs)

- 攻擊者必須使受害者造訪自己的惡意網站
(在受害者已登入受害網站的情況下)

Fundamental Causes of CSRF

- server 端無法分辨一個 request 是 same-site request (Trusted) 還是 cross-site request (Not Trusted)



- 然而，瀏覽器知道一個 request 是 same-site 還是 cross-site

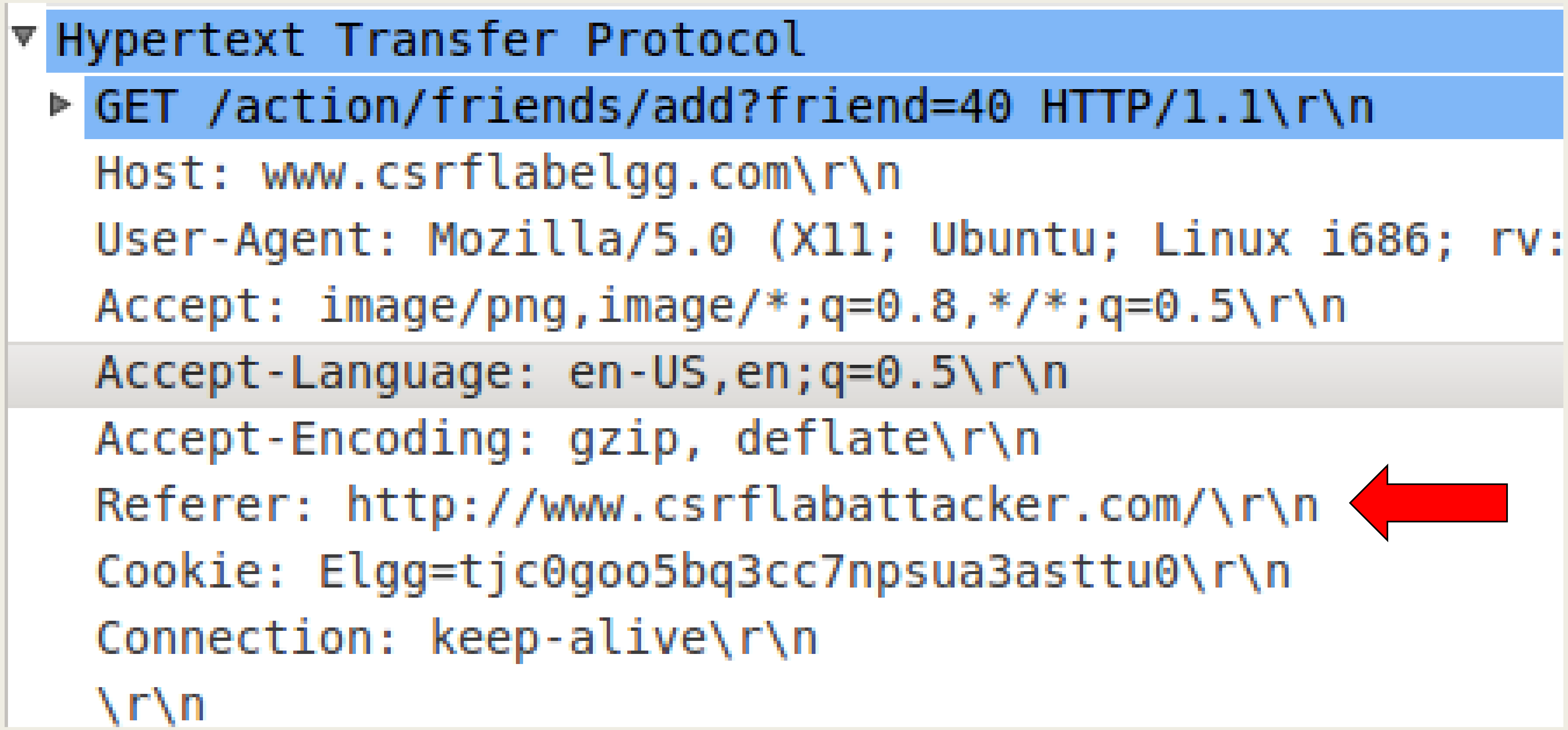
Countermeasures

- 幫助 server 端分辨 same-site/cross-site request

1. *Refer to the header (browser’s help)*
2. *Same-site cookie (browser’s help)*
3. *Secret token (the server helps itself)*

1. HTTP Referer Header

- HTTP header field identifies the address of the web page from which the request is generated.



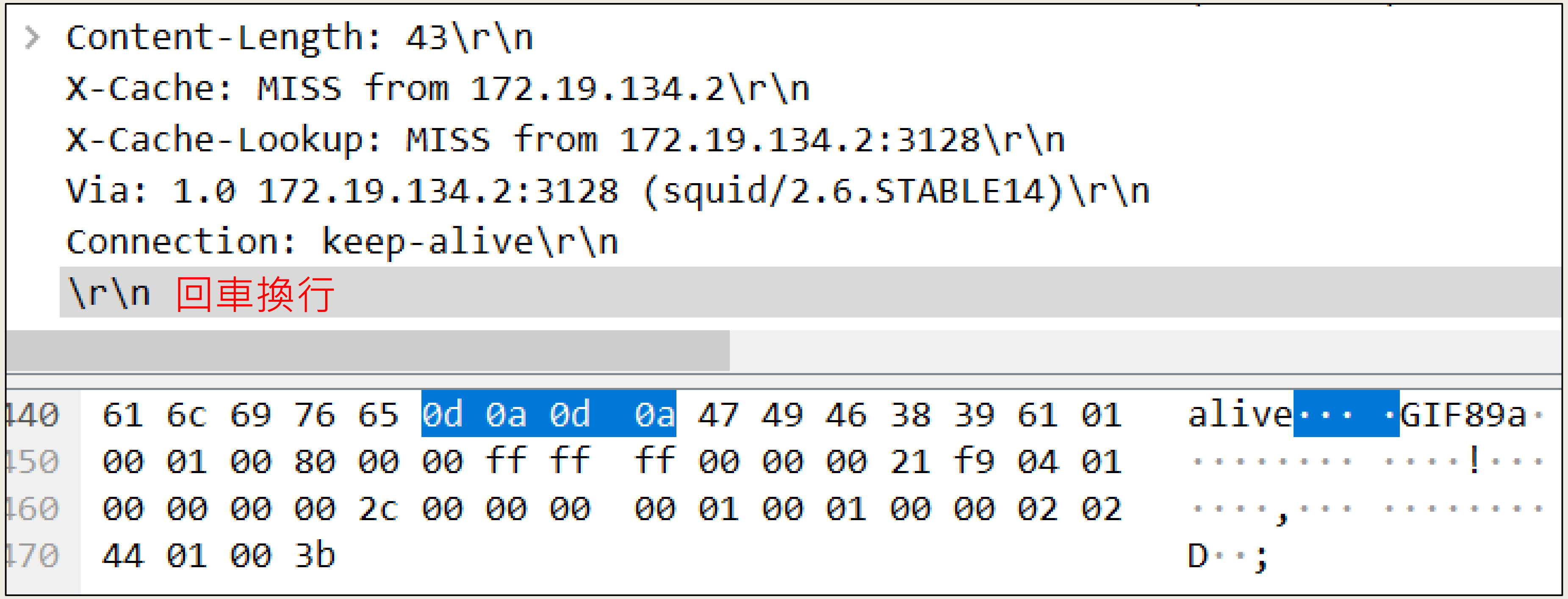
```
▼ Hypertext Transfer Protocol
  ▶ GET /action/friends/add?friend=40 HTTP/1.1\r\n
    Host: www.csrflabelgg.com\r\n
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:
    Accept: image/png,image/*;q=0.8,*/*;q=0.5\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Referer: http://www.csrflabattacker.com/\r\n
    Cookie: Elgg=tjc0goo5bq3cc7npsua3asttu0\r\n
    Connection: keep-alive\r\n
    \r\n
```

1. HTTP Referer Header

- Server 可以確認這個 request 是否源自一個信任的頁面
- 一個 request 若缺少 Referer header， Server 必須拒絕這個請求
- 有些 proxy 會省略 Referer Header (為了隱私)
- 檢查規則容易遺漏或規避，而有些軟體或瀏覽器可以控制傳送或停用 Referer
- 因而這種方式只能用在暫時性防禦，或者是已知使用者環境的情況下。
- 可能存在其他漏洞可以達成 Referer spoofing
  - E.g., CRLF injection
  - 瀏覽器本身的漏洞

Carriage-Return Line-Feed (CRLF) injection a.k.a HTTP Response Splitting

- The header of a HTTP response and its body are separated by two CRLF(\r\n\r\n) characters.

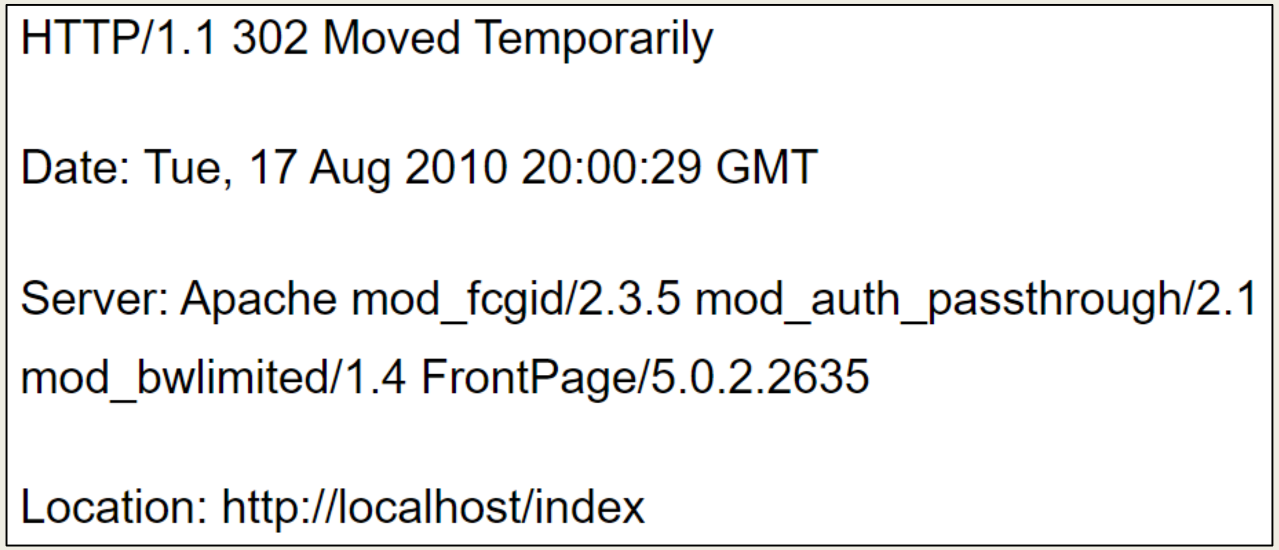


```
> Content-Length: 43\r\n
  X-Cache: MISS from 172.19.134.2\r\n
  X-Cache-Lookup: MISS from 172.19.134.2:3128\r\n
  Via: 1.0 172.19.134.2:3128 (squid/2.6.STABLE14)\r\n
  Connection: keep-alive\r\n
  \r\n 回車換行
```

```
140  61 6c 69 76 65 0d 0a 0d  0a 47 49 46 38 39 61 01   alive··· ·GIF89a·
150  00 01 00 80 00 00 ff ff  ff 00 00 00 21 f9 04 01   ········ ····!···
160  00 00 00 00 2c 00 00 00  00 01 00 01 00 00 02 02   ····,··· ········
170  44 01 00 3b                                        D··;
```

Carriage-Return Line-Feed(CRLF) injection a.k.a HTTP Response Splitting

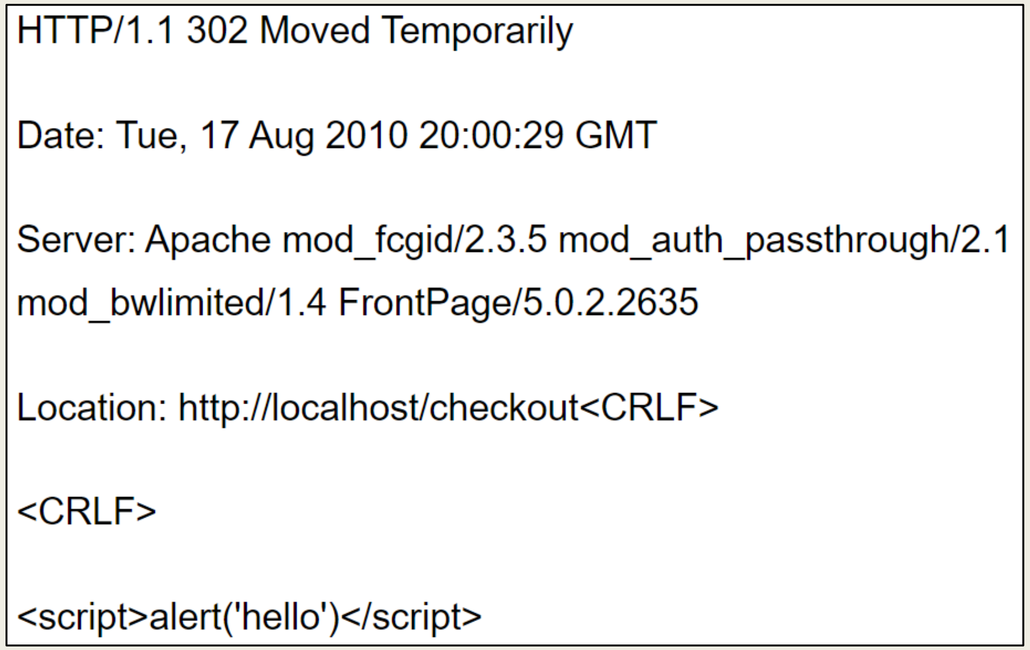
- Redirect users after successful login:
  - *http://localhost/login?page=http%3A%2F%2Flocalhost%2Findex*
  - *HTTP response:*



```
HTTP/1.1 302 Moved Temporarily

Date: Tue, 17 Aug 2010 20:00:29 GMT

Server: Apache mod_fcgid/2.3.5 mod_auth_passthrough/2.1
mod_bwlimited/1.4 FrontPage/5.0.2.2635

Location: http://localhost/index
```

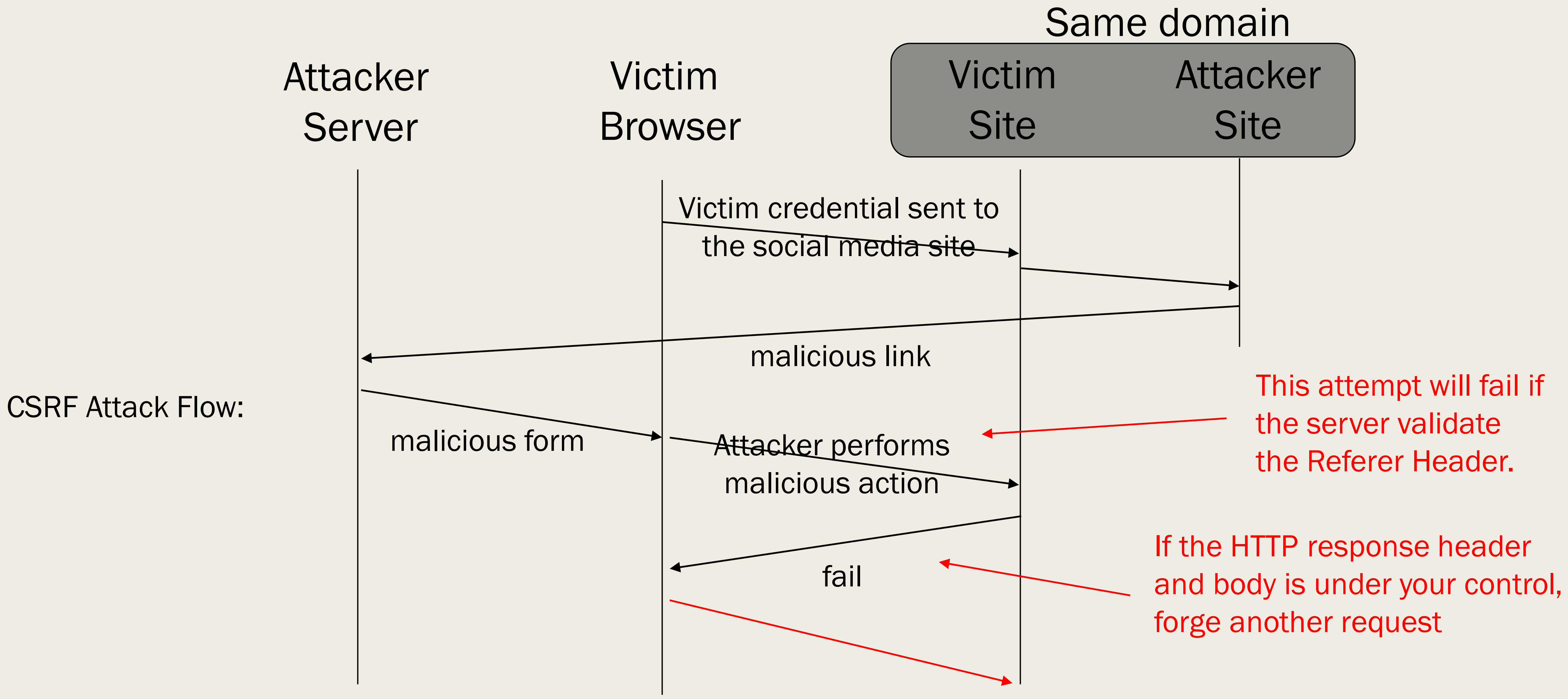
Carriage-Return Line-Feed(CRLF) injection a.k.a HTTP Response Splitting

- Redirect users after successful login:
  - *http://localhost/login?page=http%3A%2F%2Flocalhost%2Fcheckout%0D%0A%0D%0A%3Cscript%3Ealert%28%27hello%27%29%3C%2Fscript%3E*
  - *HTTP response:*



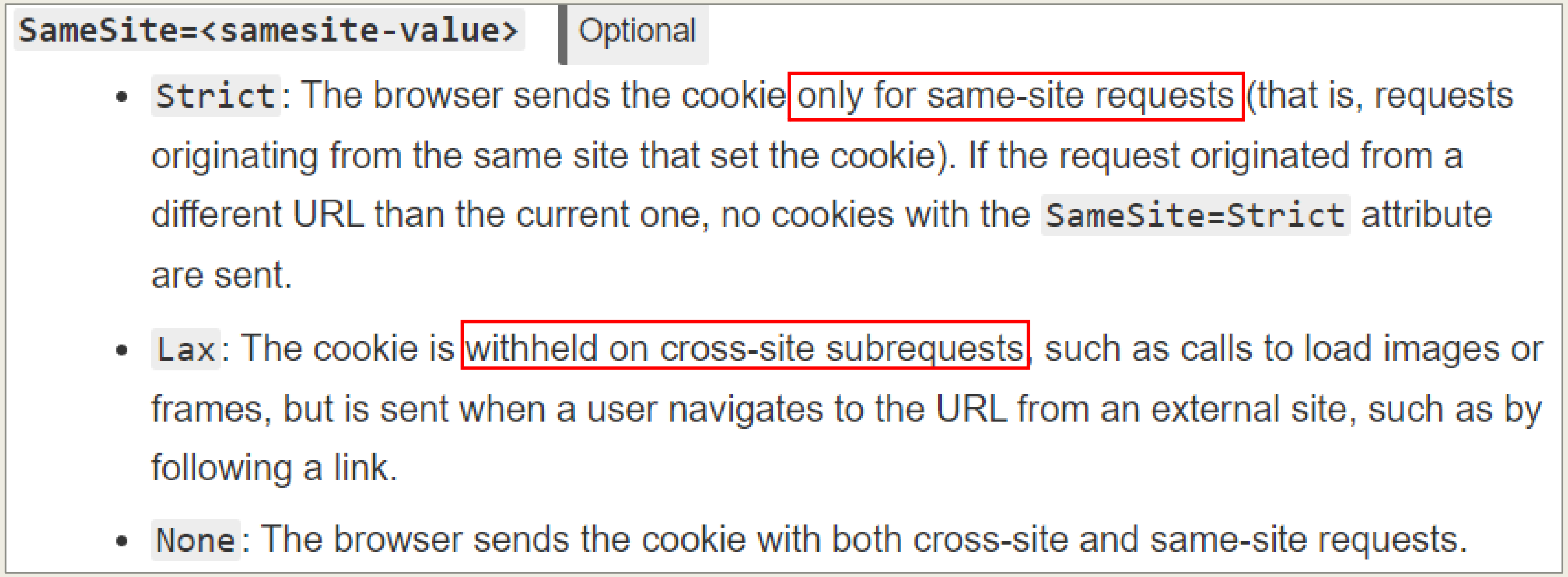
```
HTTP/1.1 302 Moved Temporarily

Date: Tue, 17 Aug 2010 20:00:29 GMT

Server: Apache mod_fcgid/2.3.5 mod_auth_passthrough/2.1
mod_bwlimited/1.4 FrontPage/5.0.2.2635

Location: http://localhost/checkout<CRLF>

<CRLF>

<script>alert('hello')</script>
```

1. HTTP Referer Header (cont.)


Same domain
Attacker Server
Victim Browser
Victim Site
Attacker Site
Victim credential sent to the social media site
malicious link
CSRF Attack Flow:
malicious form
Attacker performs malicious action
fail
This attempt will fail if the server validate the Referer Header.
If the HTTP response header and body is under your control, forge another request

2. Same-Site Cookies

- 透過 "Set-Cookie" headers
  - *Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; Secure; HttpOnly; SameSite=<samesite-value>*



`SameSite=<samesite-value>` Optional

- `Strict`: The browser sends the cookie only for same-site requests (that is, requests originating from the same site that set the cookie). If the request originated from a different URL than the current one, no cookies with the `SameSite=Strict` attribute are sent.
- `Lax`: The cookie is withheld on cross-site subrequests, such as calls to load images or frames, but is sent when a user navigates to the URL from an external site, such as by following a link.
- `None`: The browser sends the cookie with both cross-site and same-site requests.

3. Secret Token

a) Server 端生成的 CSRF token

b) Server 端生成的 Double Submit Cookie

c) Client 端生成的 Double Submit Cookie

3. Secret Token
a) Server 端生成的 CSRF token

- 產生： Server
- 儲存： Server

- 在 form 裡新增一個 name='csrftoken' value='<亂碼>'，比對 Server 端的 <亂碼>

- 漏洞：若 csrftoken 不是亂碼，攻擊者可以先發一個 request 取得 csrftoken

3. Secret Token
b) Server 端生成的 Double Submit Cookie

- 產生： Server
- 儲存： Client

- 一樣在表單放 CSRF token，但這次參照值不是存在 Server 裡，而是存在 cookie 裡
- 這方法利用的是攻擊者無法取得 cookie 的值，因此偽造的表單中的 CSRF token 不會和 cookie 裡的值一樣

- 漏洞： 攻擊者如果掌握了你底下任何一個 subdomain，就可以幫你來寫 cookie

Weak Integrity of Cookie

- Cookies 對 subdomain 並不具有完整性
- 舉例來說，foo.example.com 可以對 example.com 設置 cookie，而這個有可能把 bar.example.com 對 example.com 設置的 cookie 給蓋掉
- 最糟的情況下，當 bar.example.com 收到這個 cookie 時，區分不出是自己設置的還是別人設置的。foo.example.com 就可以利用這個特性來攻擊 bar.example.com。

3. Secret Token
c) Client 端生成的 Double Submit Cookie

- 產生： Client
- 儲存： Client

- 之前提的 Double Submit Cookie，是由 Server 產生、存在 Client 的 Cookie
- 但由於單頁應用 (single-page application) 在拿取 CSRF token 會有困難，可以改成 Client 端生成
- 此 cookie 只是要確保攻擊者無法取得，沒有包含任何敏感資訊，所以不避擔心安全性考量

Summary of CSRF

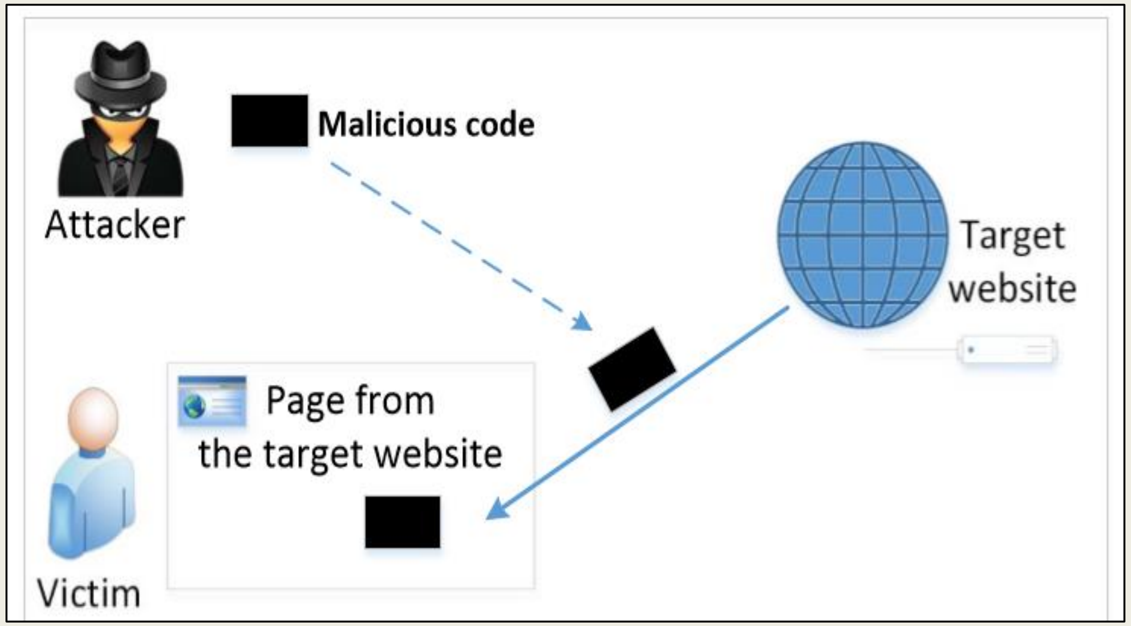
- Cross-site requests vs. same-site requests.
- Why cross-site requests should be treated differently from the same-site requests?
- How does a hacker launch CSRF attack?
- The fundamental cause of the CSRF vulnerability
- Countermeasures against CSRF attack

Cross-Site Scripting (XSS)

- **跨網站指令碼**，屬於代碼注入的一種

Definition: XSS

- 它允許惡意使用者將程式碼注入到網頁上，使其他使用者在觀看網頁時受到影響 (執行程式碼)



What hackers can do with XSS attacks?

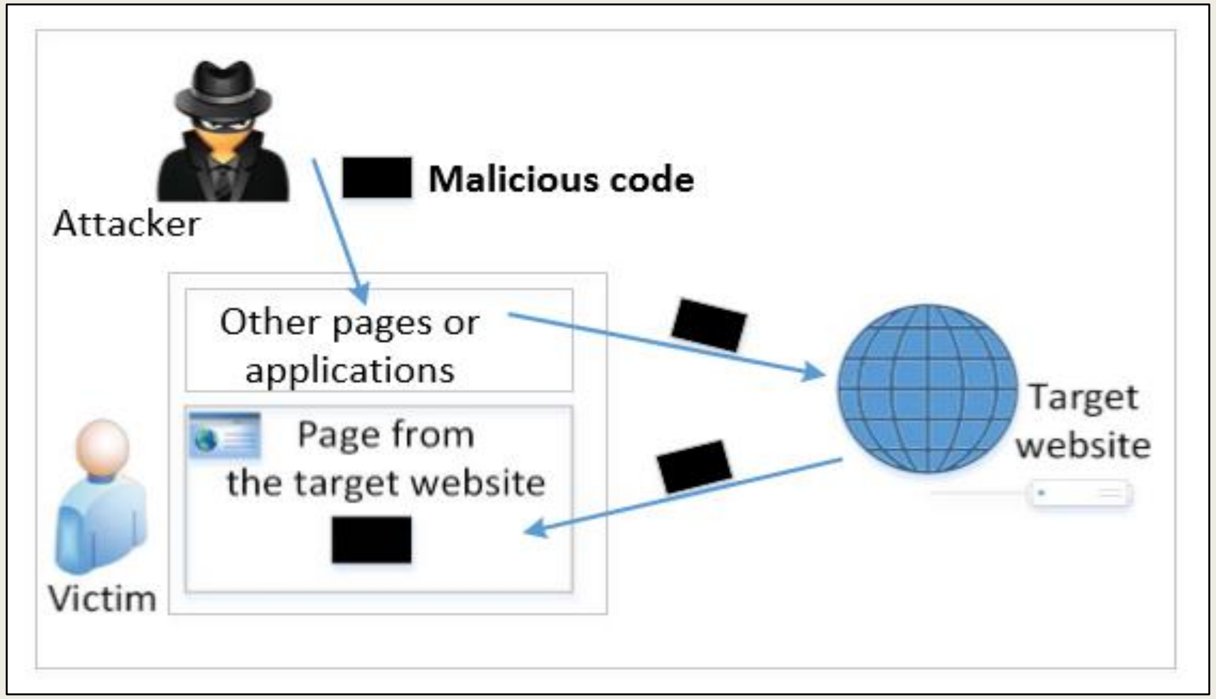
- Forge user requests
- Steal cookies
- Play a sound
- Get user-agent string
- Man-in-the-browser
- Get form values / HTML contents
- Fake notifications
- Tabnabbing (一種釣魚攻擊的手法)

Types of XSS Attacks

- Reflected XSS Attack
  - *Non-persistent*
- Stored XSS Attack
  - *Persistent*

Reflected XSS Attack

- 攻擊者構造出特殊的 URL，其中包含惡意代碼。
- 用戶打開帶有惡意代碼的 URL 時，Server 將惡意代碼從 URL 中取出，拼接在 HTML 中返回給瀏覽器。



Reflected XSS Attack

- Assume a vulnerable service on the website : http://www.example.com/search?input=word , where word is provided by the users.

你搜尋了 **xxx！**

- Now the attacker sends the following URL to the victim and tricks him to click the link: http://www.example.com/search?input=<script>alert(“attack”);</script>

Reflected XSS Attack and E-Mail

- A reflected attack is typically delivered via email or a neutral web site.

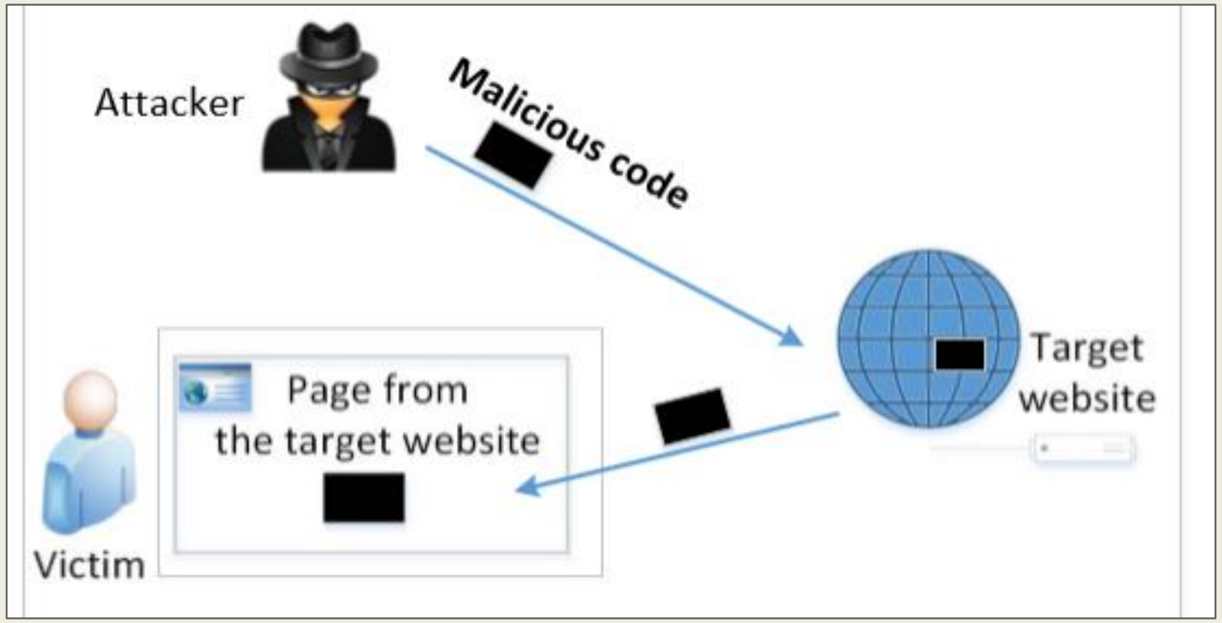
- The bait is an innocent-looking URL, pointing to a trusted site but containing the XSS vector.

  http://www.example.com/search?input=<script>alert("attack");</script>

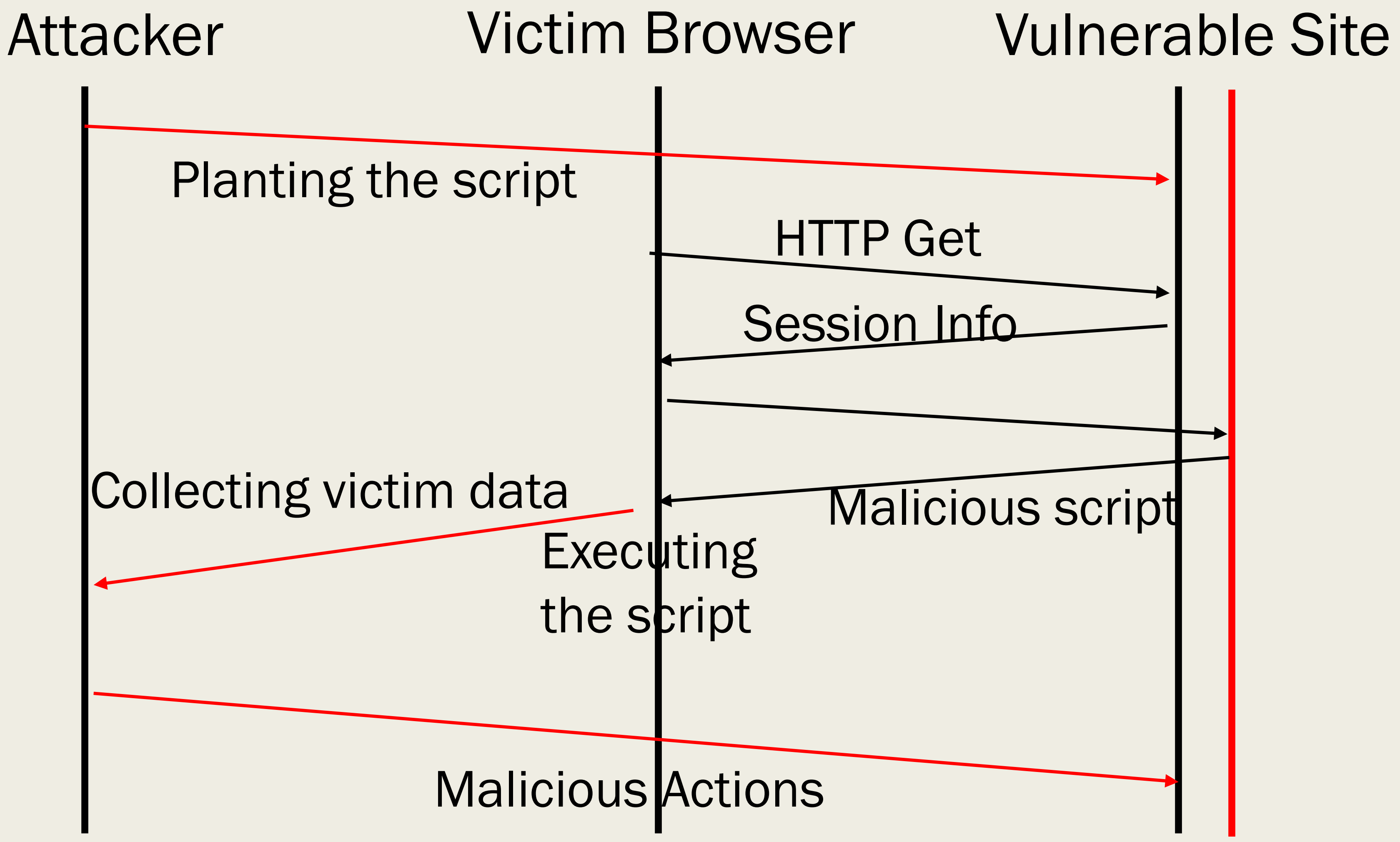
Stored XSS Attack

- 攻擊用戶打開目標網站時，網站服務端將惡意代碼從資料庫取出，拼接在 HTML 中返回給瀏覽器。
- 者將惡意代碼提交到目標網站的資料庫中。

- Example :
  - 社群網站個人頁面
  - 留言



Stored XSS Attack Flow



Fundamental Causes of XSS

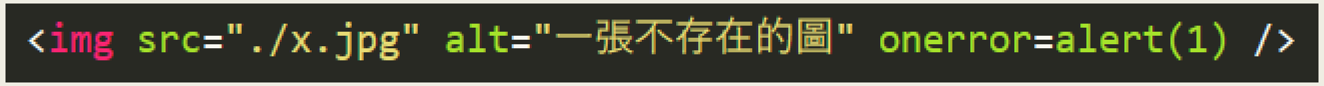
- 使用者提交的 data 包含 HTML tags 和 JavaScript 代碼
- 如果這些 data 沒有經過過濾就傳到 User 的瀏覽器 → XSS Attack
- 瀏覽器無法分辨這些 Script 的來源，認為這就是原網站所希望執行的

Countermeasures

1. Input Filtering Approach
2. Output Filtering Approach
3. Encoding Approach
4. Web Application Firewall

1. Input Filtering Approach

- 移除所有 user input 的 script/code
- 過濾方法可能存在漏洞，除了用 <script> tag 還有別種方法注入代碼



```
<img src="./x.jpg" alt="一張不存在的圖" onerror=alert(1) />
```

- 有 open-source libraries 可以過濾 JavaScript code
  - *Example: jsoup*

2. Output Filtering Approach

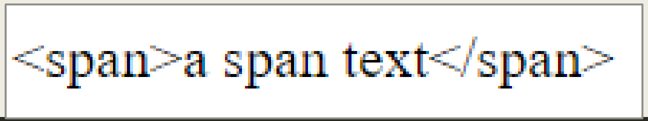
- 移除所有 user input 的 script/code
- 資料庫中保存原始的 user input，輸出到網頁時才做過濾

Notes about Filtering Approach

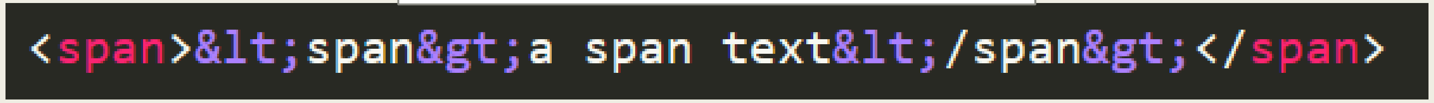
- 這種方法要開發者處理所有 user input 來源
  (query parameters, body parameters of POST request, HTTP headers)

3. Encoding Approach

- 將 HTML Markups 換成 HTML Entities



```
<span>a span text</span>
```



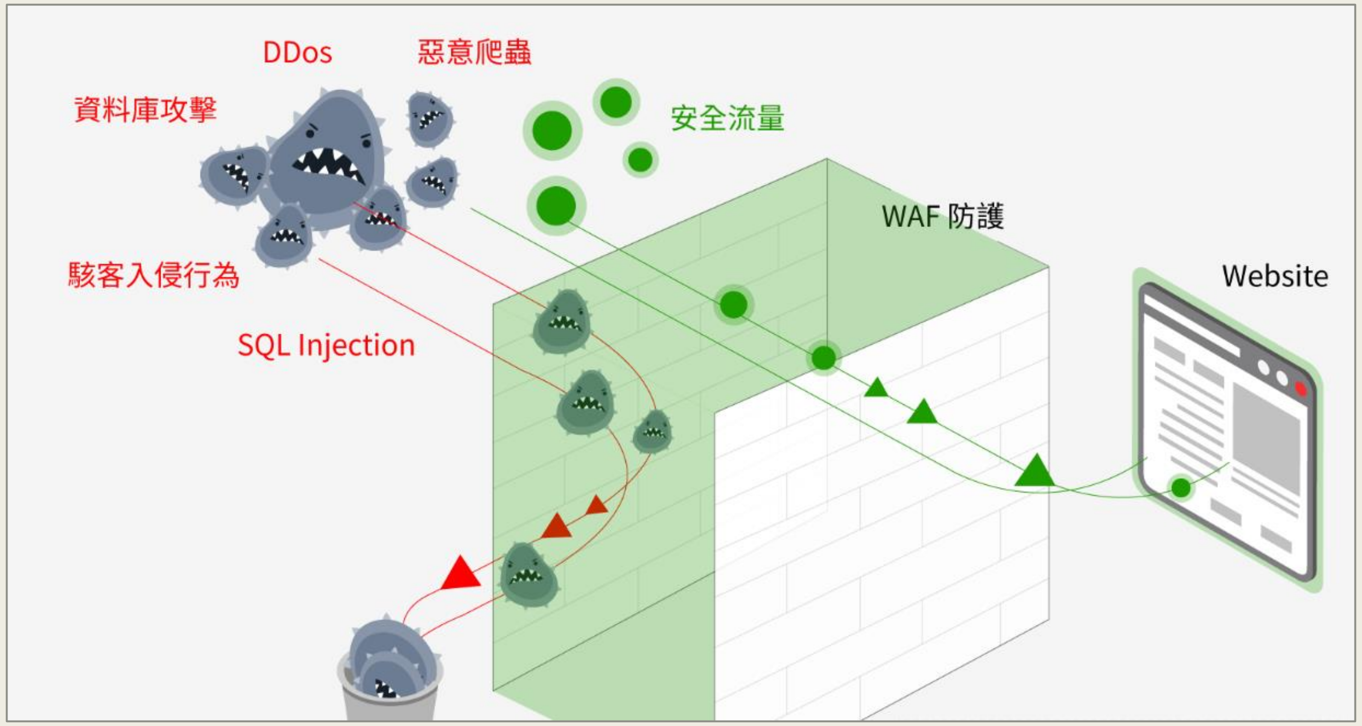
```
<span>&lt;span&gt;a span text&lt;/span&gt;</span>
```

- <script> alert(‘XSS’) </script> → &lt;script&gt;alert(‘XSS’)

- 這些 JavaScript 被 encode 過後，就會被瀏覽器顯示而不是執行

4. Web Application Firewall (WAF)

- 網站應用防火牆 (WAF) 會自動幫你過濾惡意的 input
(包含 path, HTTP headers, HTML tag patterns 和 Javascript patterns)

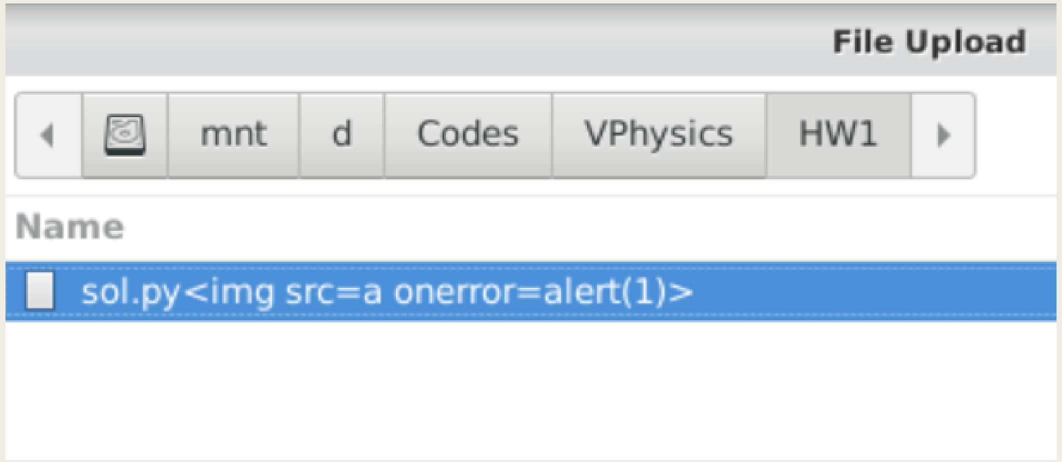


The best defense of XSS?

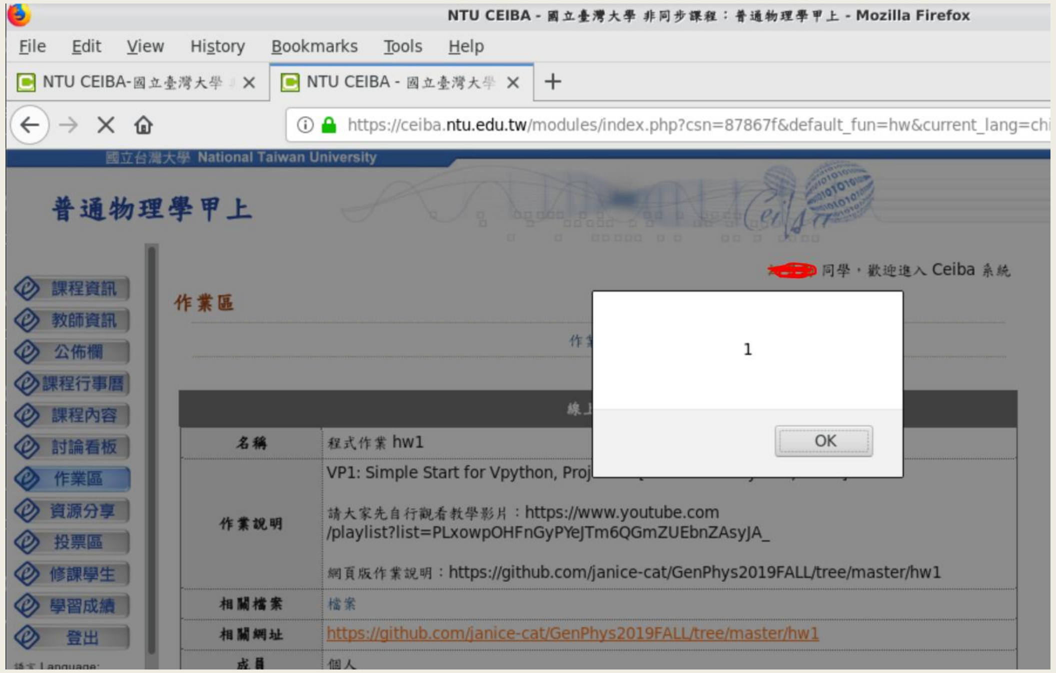
- Never trust the user
- Never trust the user
- Never trust the user
- Never trust the user
- ……

CEIBA 網站 XSS 漏洞

- 可透過上傳檔案之副檔名達成 XSS 攻擊
- 名稱的部分會被重新命名，但是副檔名並不會被過濾



sol.py<img src=a onerror=alert(1)>



https://ceiba.ntu.edu.tw/modules/index.php?csn=87867f&default_fun=hw&current_lang=chi

[redacted] 同學，歡迎進入 Ceiba 系統

作業區

1

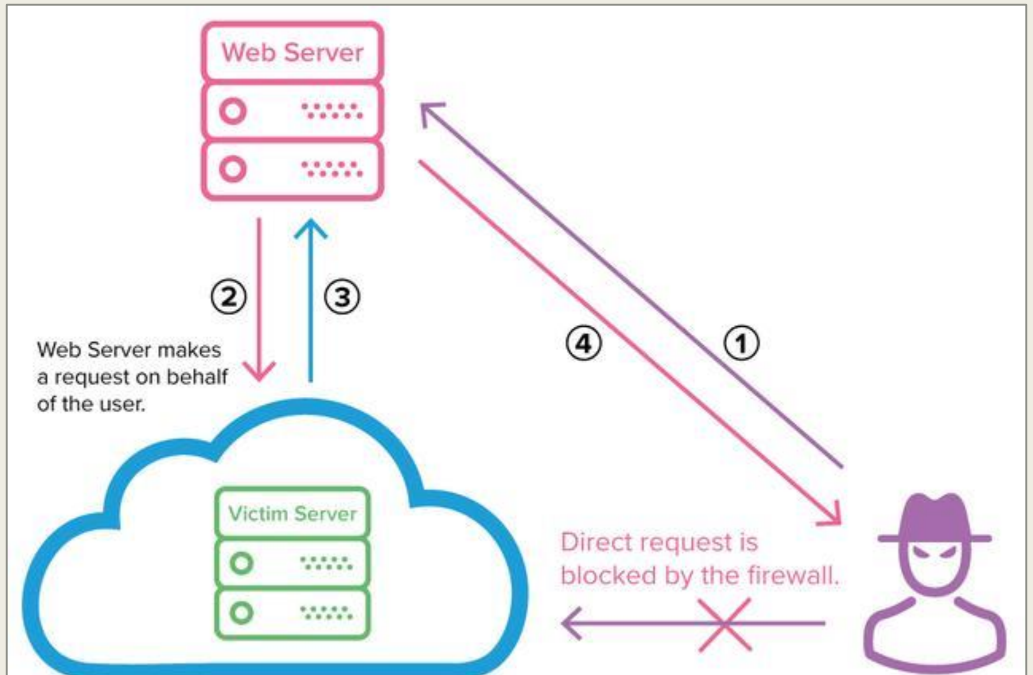
名稱	程式作業 hw1
作業說明	VP1: Simple Start for Vpython, Proj 請大家先自行觀看教學影片：https://www.youtube.com/playlist?list=PLxowpOHFnGyPYeJTm6QGmZUEbnZAsyJA_ 網頁版作業說明：https://github.com/janice-cat/GenPhys2019FALL/tree/master/hw1
相關檔案	檔案
相關網址	https://github.com/janice-cat/GenPhys2019FALL/tree/master/hw1
成員	個人

Summary of XSS

- Two types of XSS attacks
- How to launch XSS attacks
- Countermeasures against XSS attacks

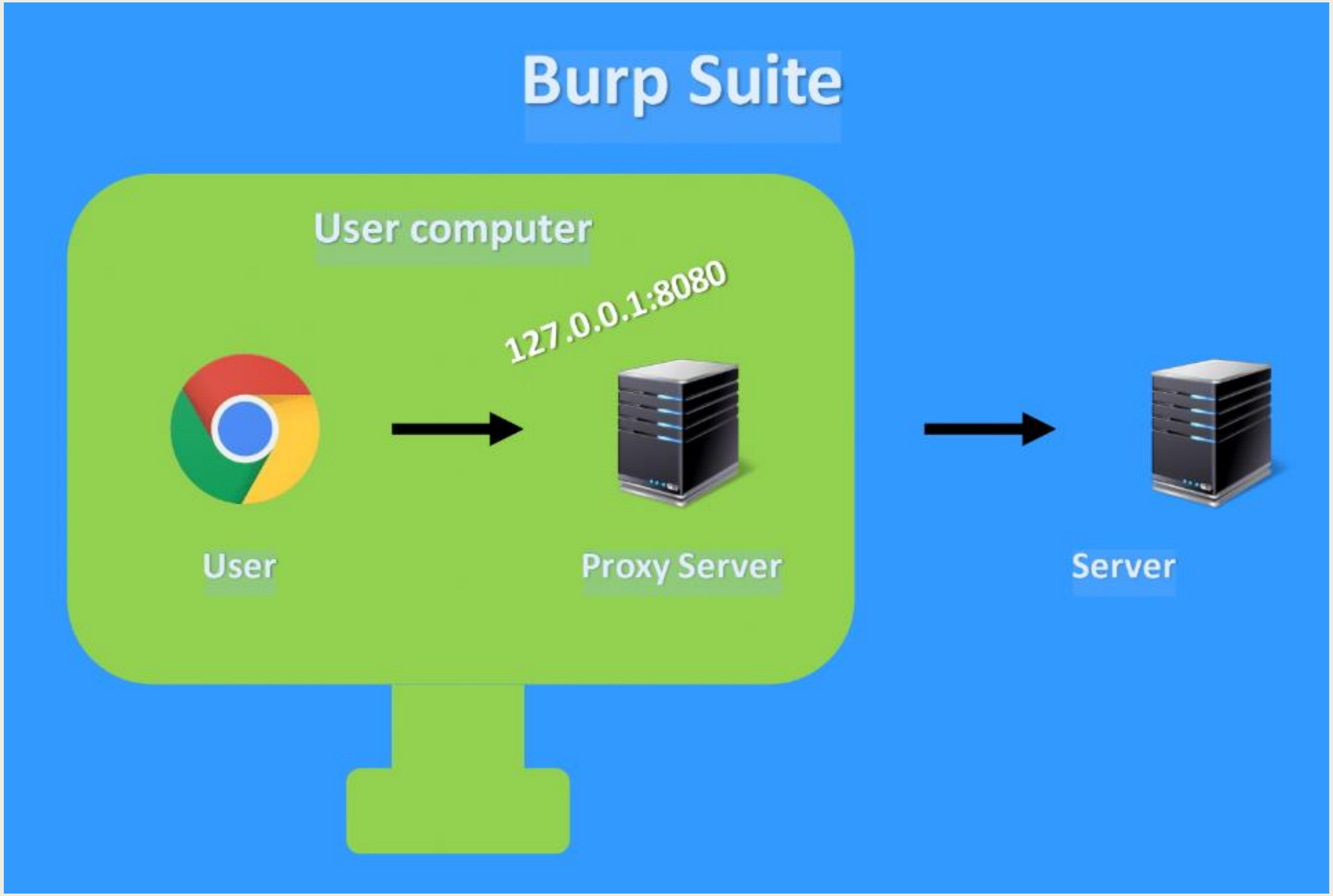
Server-Side Request Forgery (SSRF)

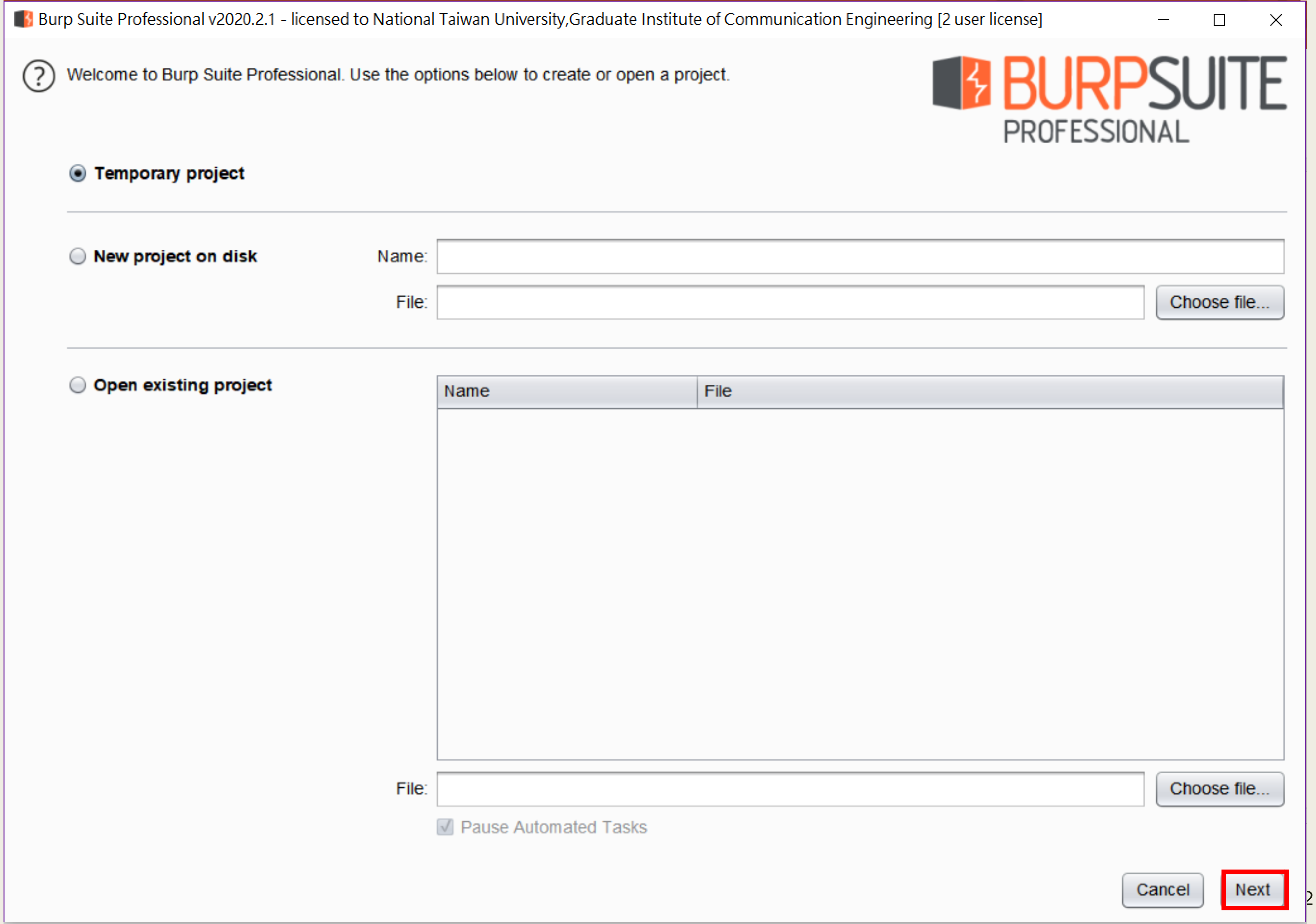
- **服務器端請求偽造**，以伺服器的身份發送一條構造好的請求
- 很多web應用都提供了從其他的伺服器上獲取數據的功能
- SSRF 可讓攻擊者的請求重定向到防火牆後的內部網路或本地主機


Web Server
②
③
Web Server makes a request on behalf of the user.
④
①
Victim Server
Direct request is blocked by the firewall.

SSRF

- https://hw00.zoolab.org/challenges


Burp Suite
User computer
127.0.0.1:8080
User
Proxy Server
Server


Burp Suite Professional v2020.2.1 - licensed to National Taiwan University,Graduate Institute of Communication Engineering [2 user license]

Welcome to Burp Suite Professional. Use the options below to create or open a project.

BURPSUITE PROFESSIONAL

(●) **Temporary project**

( ) **New project on disk** Name: File: Choose file...

( ) **Open existing project**

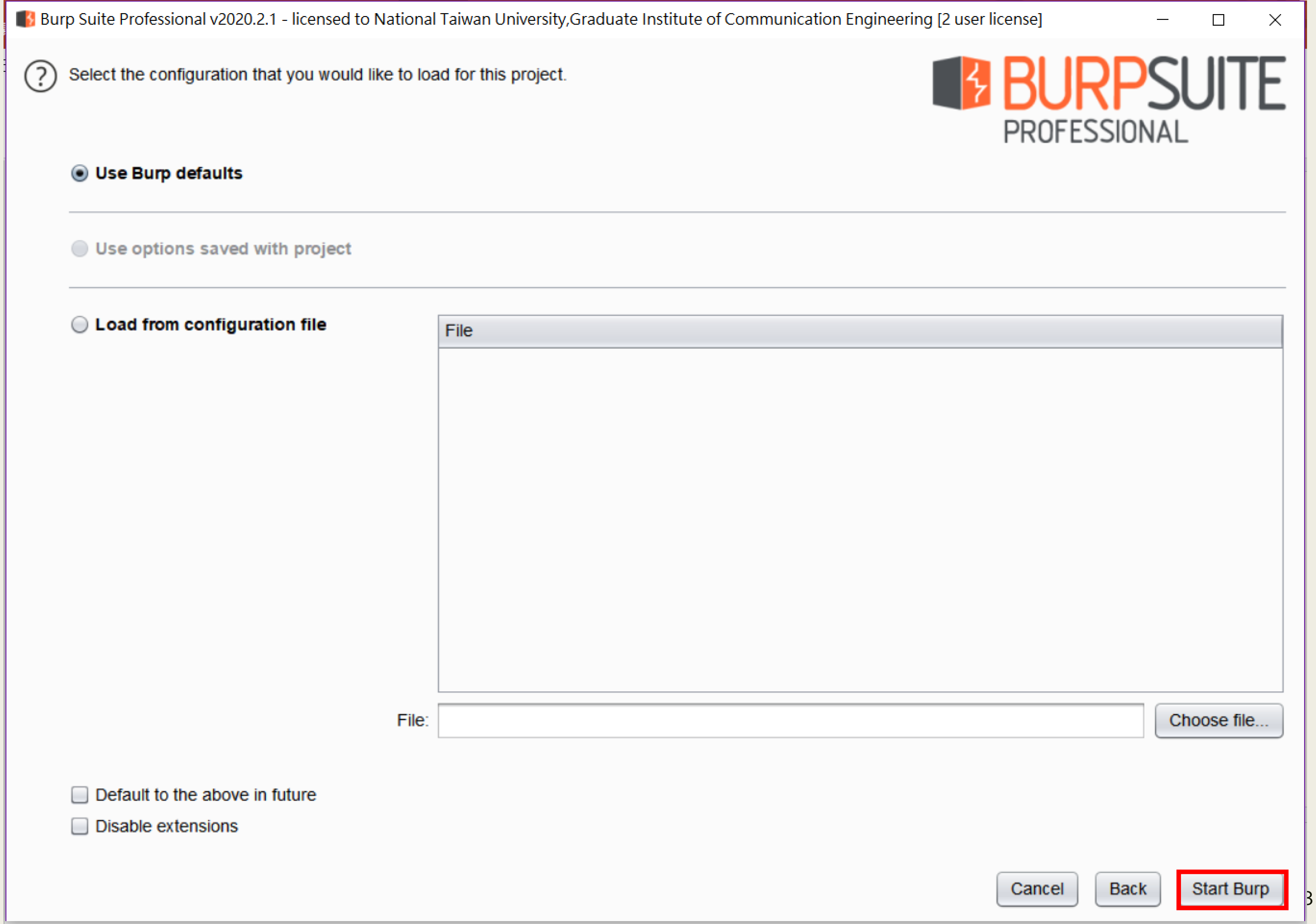
Name	File

File: Choose file...

☑ Pause Automated Tasks

Cancel Next

2



Burp Suite Professional v2020.2.1 - licensed to National Taiwan University,Graduate Institute of Communication Engineering [2 user license]

Select the configuration that you would like to load for this project.

[✓ Use Burp defaults]

Use options saved with project

Load from configuration file

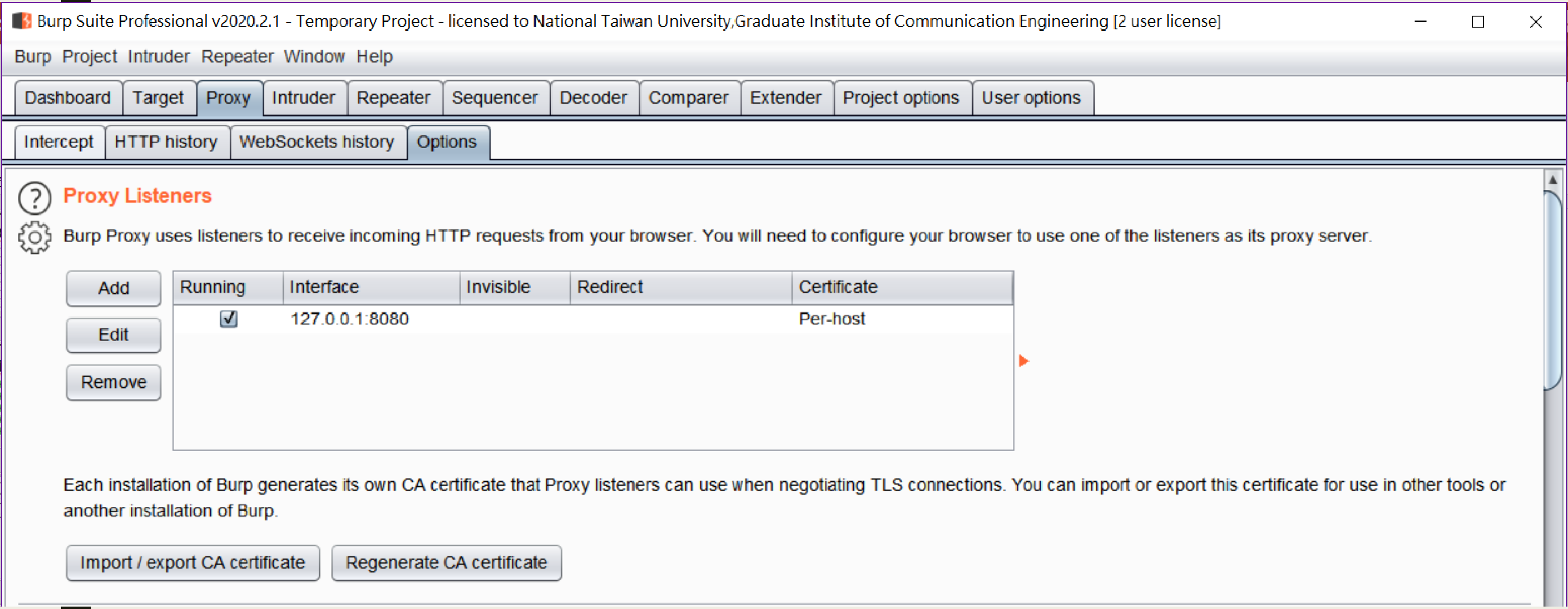
File:

Default to the above in future

Disable extensions

Installing Burp's CA certificate

- https://portswigger.net/burp/documentation/desktop/getting-started/proxy-setup/certificate

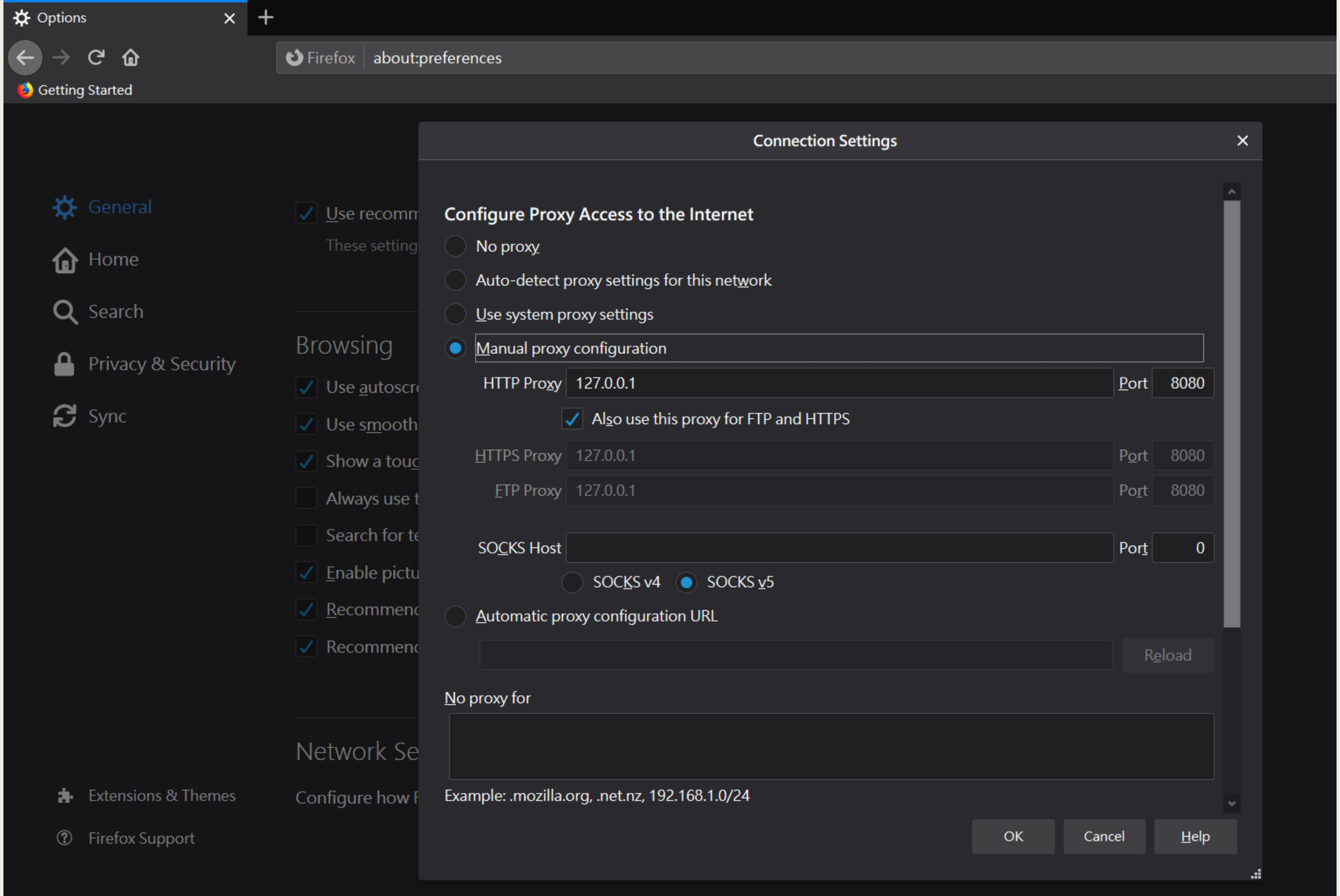

Burp Suite Professional v2020.2.1 - Temporary Project - licensed to National Taiwan University,Graduate Institute of Communication Engineering [2 user license]
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Add
Edit
Remove
Import / export CA certificate
Regenerate CA certificate

Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.

Running	Interface	Invisible	Redirect	Certificate
✓	127.0.0.1:8080			Per-host

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this certificate for use in other tools or another installation of Burp.



Connection Settings

Configure Proxy Access to the Internet

No proxy

Auto-detect proxy settings for this network

Use system proxy settings

[✓ Manual proxy configuration]

HTTP Proxy: 127.0.0.1 Port: 8080

[✓] Also use this proxy for FTP and HTTPS

HTTPS Proxy: 127.0.0.1 Port: 8080

FTP Proxy: 127.0.0.1 Port: 8080

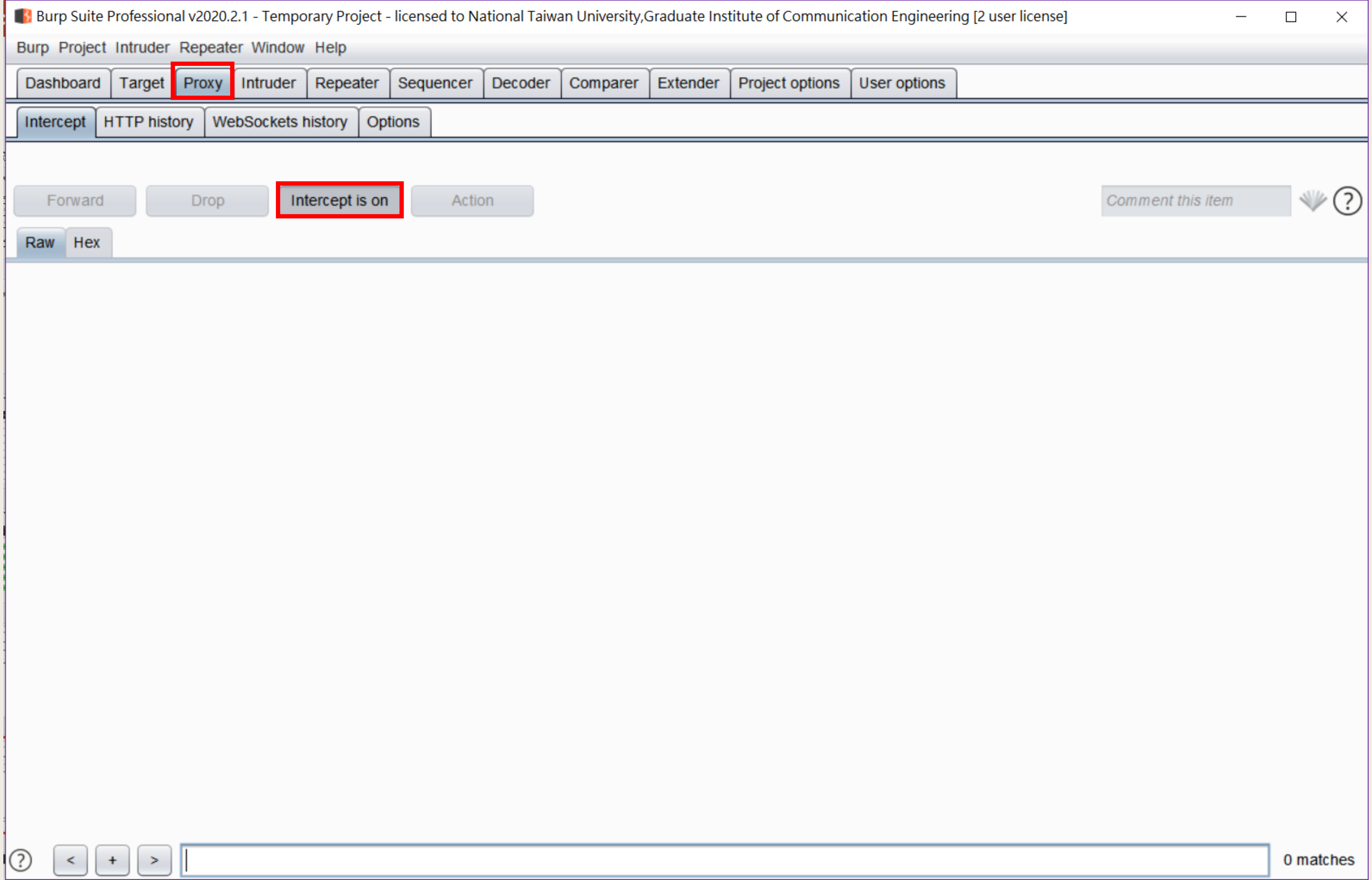
SOCKS Host: Port: 0

SOCKS v4 [✓ SOCKS v5]

Automatic proxy configuration URL

No proxy for

Example: .mozilla.org, .net.nz, 192.168.1.0/24

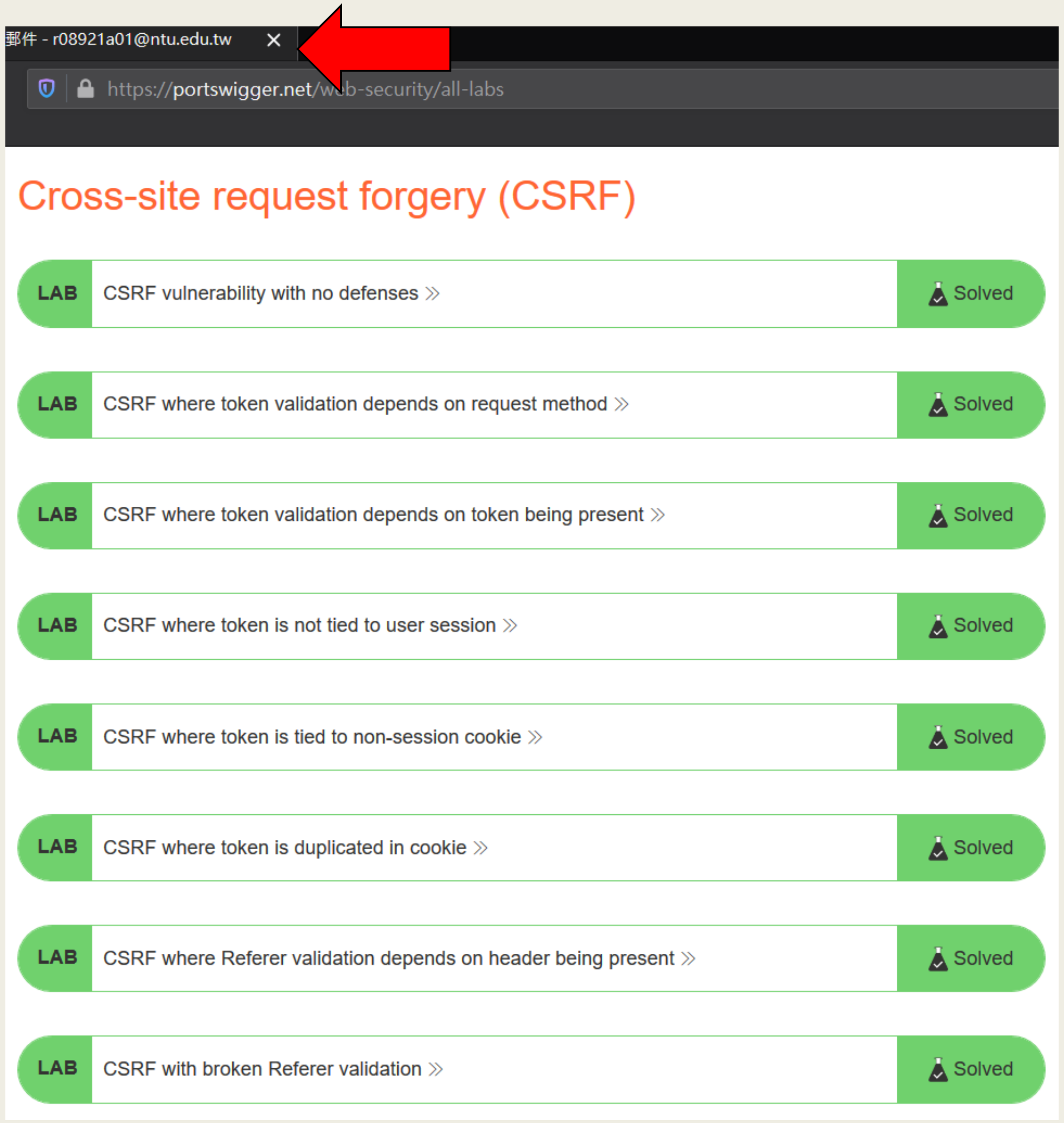

Burp Suite Professional v2020.2.1 - Temporary Project - licensed to National Taiwan University,Graduate Institute of Communication Engineering [2 user license]
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Comment this item
Raw Hex
0 matches

HW

- 10 Labs in XSS (First 10):
  - *https://portswigger.net/web-security/cross-site-scripting*
- 8 Labs in CSRF:
  - *https://portswigger.net/web-security/csrf*
- 5 Labs in SSRF (First 5):
  - *https://portswigger.net/web-security/ssrf*

- Screenshot of XSS, CSRF and SSRF section at:
  - *https://portswigger.net/web-security/all-labs*

With your Email logged in

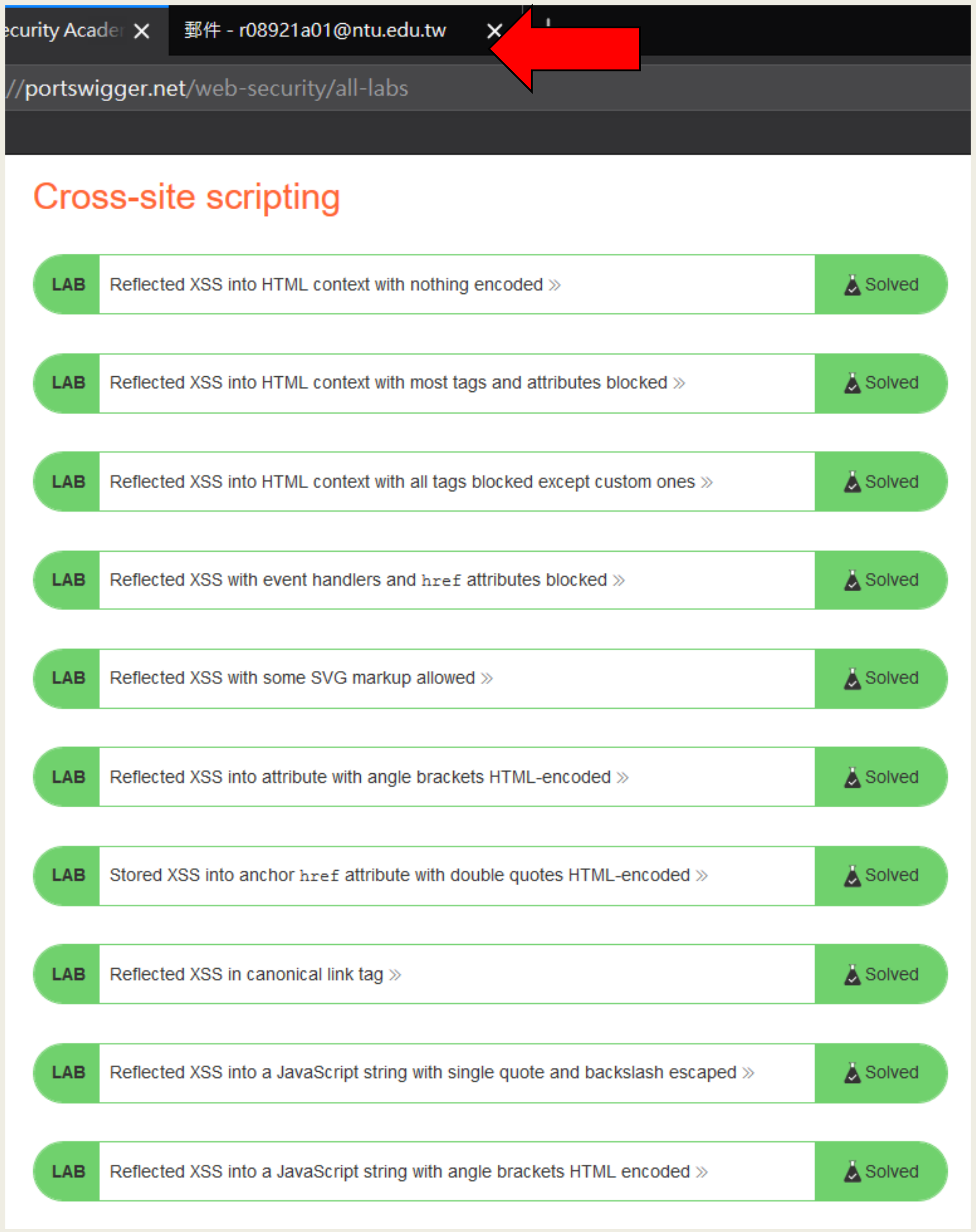


郵件 - r08921a01@ntu.edu.tw

https://portswigger.net/web-security/all-labs

Cross-site request forgery (CSRF)

LAB	CSRF vulnerability with no defenses »	Solved
LAB	CSRF where token validation depends on request method »	Solved
LAB	CSRF where token validation depends on token being present »	Solved
LAB	CSRF where token is not tied to user session »	Solved
LAB	CSRF where token is tied to non-session cookie »	Solved
LAB	CSRF where token is duplicated in cookie »	Solved
LAB	CSRF where Referer validation depends on header being present »	Solved
LAB	CSRF with broken Referer validation »	Solved



ecurity Acade

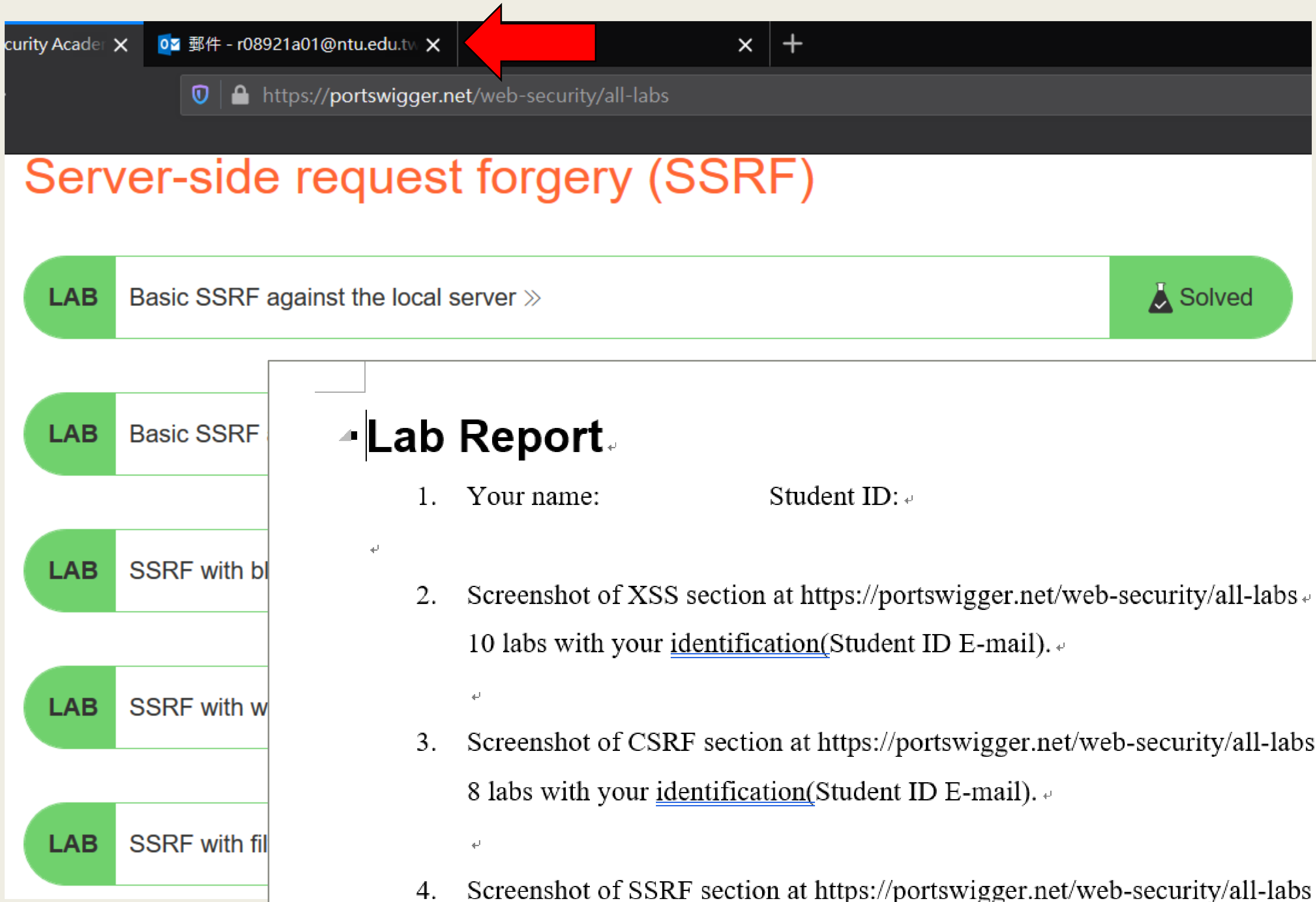
郵件 - r08921a01@ntu.edu.tw

//portswigger.net/web-security/all-labs

Cross-site scripting

LAB	Reflected XSS into HTML context with nothing encoded »	Solved
LAB	Reflected XSS into HTML context with most tags and attributes blocked »	Solved
LAB	Reflected XSS into HTML context with all tags blocked except custom ones »	Solved
LAB	Reflected XSS with event handlers and `href` attributes blocked »	Solved
LAB	Reflected XSS with some SVG markup allowed »	Solved
LAB	Reflected XSS into attribute with angle brackets HTML-encoded »	Solved
LAB	Stored XSS into anchor `href` attribute with double quotes HTML-encoded »	Solved
LAB	Reflected XSS in canonical link tag »	Solved
LAB	Reflected XSS into a JavaScript string with single quote and backslash escaped »	Solved
LAB	Reflected XSS into a JavaScript string with angle brackets HTML encoded »	Solved

With your Email logged in


curity Acade
郵件 - r08921a01@ntu.edu.tw
https://portswigger.net/web-security/all-labs

Server-side request forgery (SSRF)

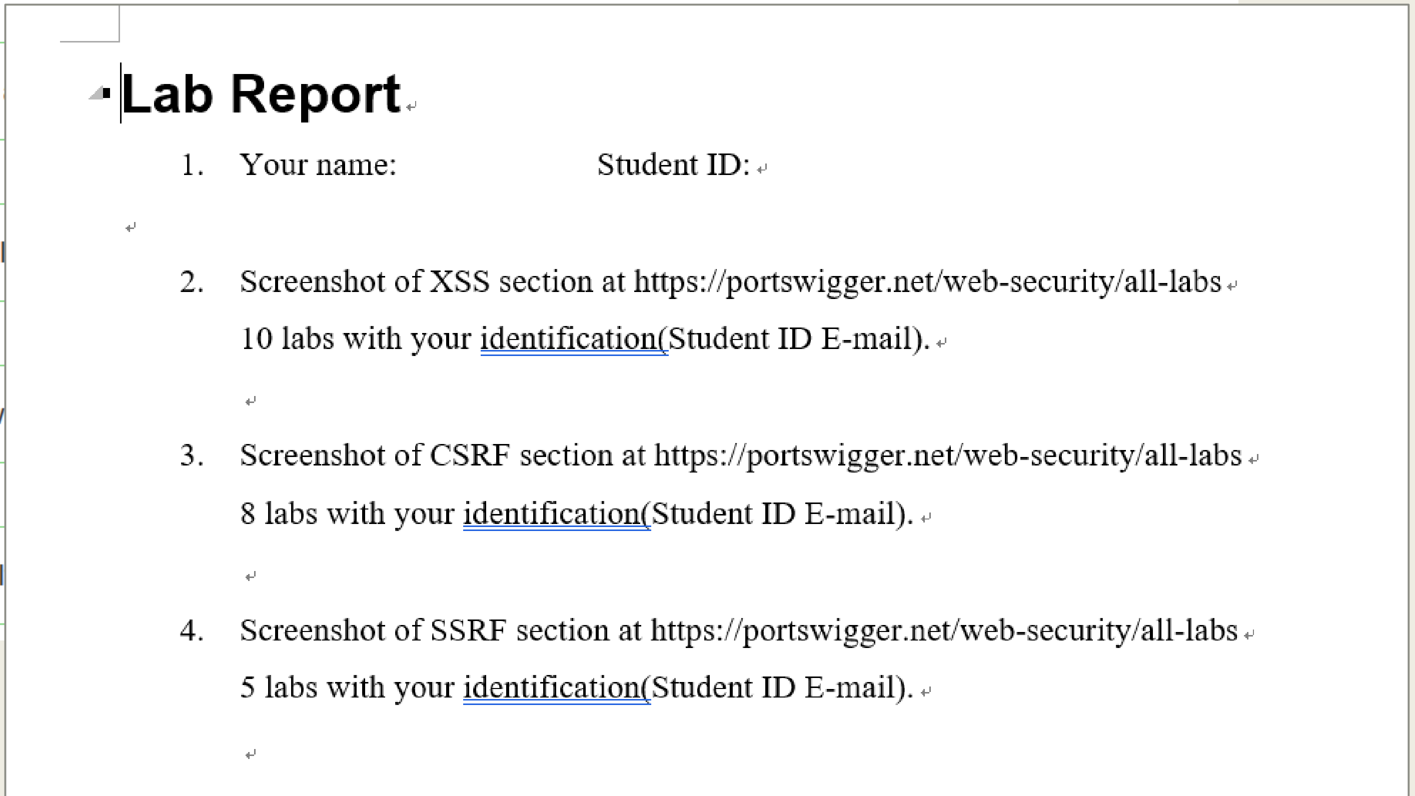
LAB Basic SSRF against the local server » Solved

LAB Basic SSRF

LAB SSRF with bl

LAB SSRF with w

LAB SSRF with fil



Lab Report

1. Your name: Student ID:

2. Screenshot of XSS section at https://portswigger.net/web-security/all-labs
   10 labs with your identification(Student ID E-mail).

3. Screenshot of CSRF section at https://portswigger.net/web-security/all-labs
   8 labs with your identification(Student ID E-mail).

4. Screenshot of SSRF section at https://portswigger.net/web-security/all-labs
   5 labs with your identification(Student ID E-mail).