NASA HW12

B11901164 陳秉緯

Security Part II

1. Linux 大小事

ref: 1, 2, 3, 4, 5

(a) 在現行 Linux 系統下，使用者密碼的 hash 儲存在 `/etc/shadow` 檔案中，而非以明文形式儲存。基本上是 shadow 以 `:` 作為分隔符號，總共有九個欄位：`帳號名稱:密碼:最近更動密碼的日期:密碼不可被更動的天數:密碼需要重新變更的天數:密碼需要變更期限前的警告天數:密碼過期後的帳號寬限時間(密碼失效日):帳號失效日期:保留` 其中密碼的格式 `$id$salt$hashed`：
- `$id`：代表使用的 hash 演算法
- `salt`：加 salt value，防止 rainbow table attack
- `hashed`：經過 hash 後的密碼

(b)

- `passwd` 是一個 setuid 程式，也就是該執行檔的擁有者是 root，且擁有 `setuid` 權限欄位：

```
ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 64152 May 30  2024 /usr/bin/passwd*
```

- `s` 表示 setuid，執行此程式的使用者會暫時擁有 root 權限，以便修改 `/etc/shadow` 中該使用者自己的密碼的 hash value
- 安全性透過 PAM (Pluggable Authentication Modules) 和內建限制（只能改自己的密碼）來維護

(c) ssh 登入失敗的紀錄儲存在 `/var/log/auth.log` 內，檔案包含：ssh 登入成功與失敗的紀錄、sudo 使用紀錄、PAM 驗證資訊

(d) 可能的防範方法：

1. 限制登入嘗試次數：透過設定客戶端連續登入失敗的次數上限，超過該次數後封鎖對方的 IP 一段時間，來防止短時間內暴力破解。另外可以用 Fail2Ban 來實做，他可以監控登入失敗記錄，如 `/var/log/auth.log`，並自動封鎖對方的 IP
2. 停用密碼登入，改用 ssh 金鑰登入：只要在 `/etc/ssh/sshd_config` 中設定 `PasswordAuthentication no` 跟 `PubkeyAuthentication yes`，這樣就無法以密碼暴力破解，就跟 NASA 工作站一樣

2. 畫中有話

註：此題檔案寫在 `code/p2.py`

ref: 1

(a)

1. 資料編碼：
   - 將要隱藏的訊息中的每個字元轉換為 8 位元的二進制數，使用 `format(ord(data[i]), "08b")`
   - 對於每個字元，程式會使用連續的 3 個像素一共 9 個 RGB 色彩值來儲存這 8 個位元
2. 像素修改機制：
   - 取得每個像素的 RGB 值，總共 9 個色彩分量 (3個像素 × 3個色彩通道)
   - 對於每個二進制位元：
     - 如果位元是 '1'：將對應的色彩值修改為 `2 * (原值 // 2) + 1` 確保 LSB 為 1
     - 如果位元是 '0'：將對應的色彩值修改為 `2 * (原值 // 2)` 確保 LSB 為0
3. 隱藏機制：
   - 只修改每個色彩值的 LSB
   - 視覺上幾乎無法察覺差異，因為色彩值只改變 ±1
   - 每 3 個像素可以隱藏 1 個字元 (8 位元)

(b)

- flag: HW12{S4KiCh4n_sakiCHAN_S4k1ChaN}
- 解題流程：
  1. 模仿 `hide.py` 將 `colors` 提取出來：`colors = list(pixels[i * 3]) + list(pixels[i * 3 + 1]) + list(pixels[i * 3 + 2])`
  2. 提取出每個 color 內的 LSB：

```
binary_str = ""
for j in range(8):  # 8 bits per character
    lsb = colors[j] & 1  # Get the least significant bit
    binary_str += str(lsb)
```

  3. 將 binary 轉成 char:

```
char_code = int(binary_str, 2)

if char_code == 0:
    break
```

```
char = chr(char_code)
extracted_data += char
```

一直重複上述步驟直到 `char_code == 0` 就代表到底了，並把每次的 `char` 合併起來，就找到藏在圖中的訊息了

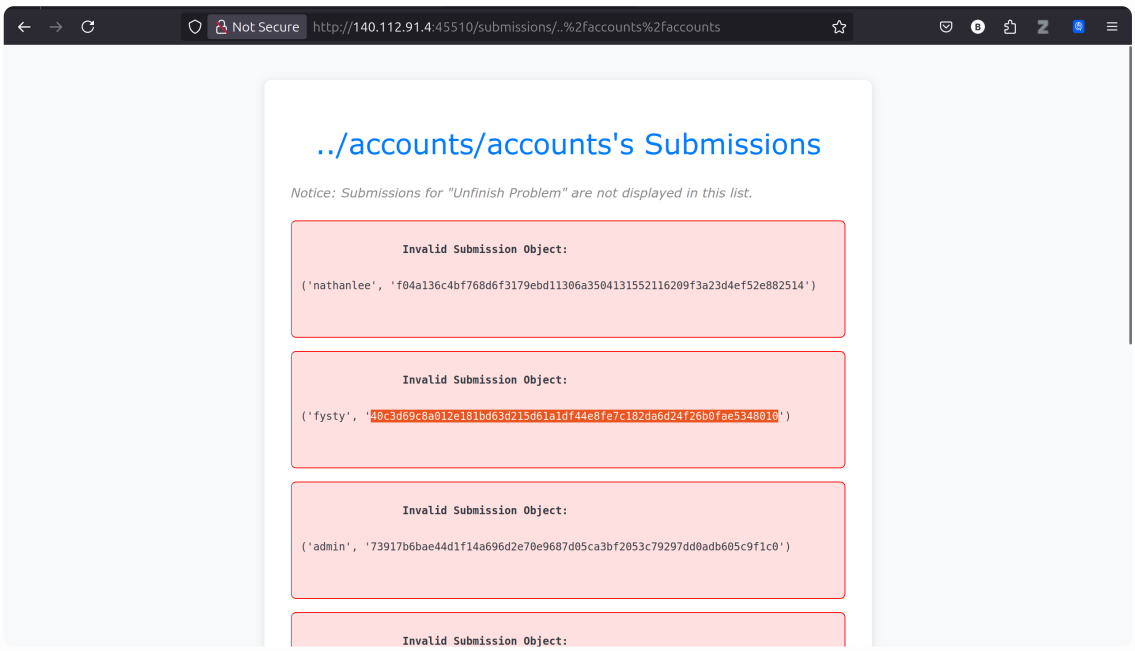
3. Alya Judge

註：(a)，(b) 跟 © 小題檔案分別寫在 `code/p3_a.py` ，`code/p3_b.py` 跟 `code/p3_c.py` 內

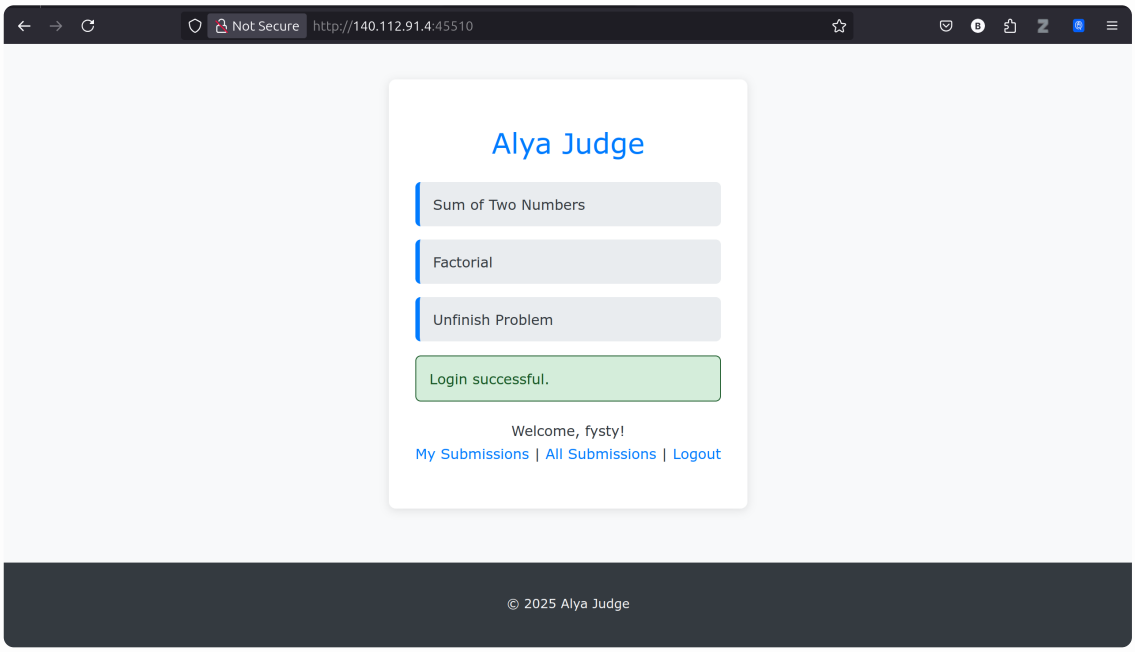
ref: 1, 2, 3

(a)

- flag1: HW12{r3MeM8eR_To_s3t_S7R0Ng_PAS5WOrds}
- 解題流程：
  1. 從 hint1 知道說要找 `accounts.json`，從 `app.py` 中觀察到 line 12 `ACCOUNTS_FILE = 'accounts/accounts.json'` 但是在 route 的定義中沒看到 `/accounts`，所以如果直接訪問 `http://140.112.91.4:45510/accounts/accounts.json` 應該是 404，另外在 line 162-164 看到有定義 route `/submissions/<path:username>` 而且 `load_submissions` 的定義是依據 `"submissions/"` 加上 `username` 再加上 `".json"`，再搭配 hint2 說的 path traversal 攻擊，可以得出 `accounts.json` 的位置在 `/submissions/` + `../` + `accounts/accounts`，組起來是 `/submissions/../accounts/accounts`，就表示要去 `http://140.112.91.4:45510/submissions/../accounts/accounts` ，但是 hint2 也提示說要換掉 `.` 跟 `/`，所以最終要去 `http://140.112.91.4:45510/submissions/%2e%2e%2faccounts%2faccounts` 也就是 `submissions/` 後面的 `.` 跟 `/` 都分別換成 `%2e` 與 `%2f`，最後就可以找到：



2. 接下來要破解 `fysty:40c3d69c8a012e181bd63d215d61a1df44e8fe7c182da6d24f26b0fae5348010` 密碼，這部份的 code 在 `code/p3_a.py` 內，搭配 hint3 把 `xato-net-10-million-passwords-1000000.txt` 下載下來，裡面一行一行的內容做 `sha256` hash 跟 `40c3d69c8a012e181bd63d215d61a1df44e8fe7c182da6d24f26b0fae5348010` 做比較，發現找到 `password` 是 `mortis00`
3. 去 `http://140.112.91.4:45510` ，點 Login，Username 輸入 `fysty` Password 輸入 `mortis00`，就成功登入 fysty 的帳號了



4. 點擊 "My Submissions" 連結，滑到最下面就找到 flag1: HW12{r3MeM8eR_To_s3t_S7R0Ng_PAS5WOrds}

(b)

- flag2: HW12{e5x5Vw2qC}
- 解題流程：
  1. 觀察 `app.py` 中的 `special_judge(code, solution)` 是檢查 `solution` 中的每個字符串是否按順序在 `code` 中出現，有對應到 `cnt += 1` 沒對應到就不加，最後比對位置到 `code` 的最後，跳出 while-loop 並回傳 `score = special_judge(code, problem['solution'][language])`，最後計算 `score`，如果 `score == len(problem['solution'][language])` 長度相等，代表完全 match，"Accepted", 100，但是其他狀況就是 `"Wrong Answer", (score * 100) // len(problem['solution'][language])`
  2. 另外在 `app.py` 也看到：當不是登入自己的帳號的時候，problem 3 也就是 Unfinished problem 會被過濾掉，不會呈現在網站上也 query 不到，現在只有從第 (a) 小題拿到的 fysty 的帳號，沒有 nathanleee 的帳號

3. 因此先嘗試模仿第 (a) 小題的作法，把 `'nathanlee', 'f04a136c4bf768d6f3179ebd11306a3504131552116209f3a23d4ef52e882514'` 去做字典暴力破解，發現找不到他的密碼
4. 現在用 fysty 帳號看不到 nathanlee 的 problem 3 答案，因為會被跳過，但是我又找不到 nathanlee 的 password，所以只想到用另一種暴力解，用 fysty 帳號瘋狂 Submit prolem 3 的答案做測試，因為此份作業就有說 flag 的格式都是 `HW12{[0-9A-Za-z_]+}` 還有 hint3 也說 flag2（包含 `HW12{...}`）的長度為 15 個字元，這代表我可以一位一位的去試 `0-9`, `A-Z`, `a-z`, 還有 `_{}` ，所以有寫了 python 檔在 `code/p3_b.py` 內，一開使先登入 fysty 帳號，一位一位的 char 併在分數比較高的 string 後面後送出，所以過程是：

```
H: score 6
HW: score 13
HW1: score 20
HW12: score 26
HW12{: score 33
HW12{e: score 40
HW12{e5: score 46
HW12{e5x: score 53
HW12{e5x5: score 60
HW12{e5x5V: score 66
HW12{e5x5Vw: score 73
HW12{e5x5Vw2: score 80
HW12{e5x5Vw2q: score 86
HW12{e5x5Vw2qC: score 93
HW12{e5x5Vw2qC}: score 100
```

最後找到可以讓 score 100 的 flag3: HW12{e5x5Vw2qC} 上面我省略其他在相同長度下拿到的 score 比較低的選項

©

- flag: HW12{i_l1KE_a15CR3am_MoRE_7H4n_Co0Ki3s}
- 解題流程：
  1. 由 hint1 跟 hint2 知道這題一定跟 Cookie 有關，所以要偽造 admin 的 session cookie，另外在 `app.py` 內發現 `app.config['SECRET_KEY'] = 'A_super_SecUrE_$eCR37_keY'` 代表 Flask 使用這個 key 來簽 session cookies
  2. 用 `wget https://raw.githubusercontent.com/noraj/flask-session-cookie-manager/master/flask_session_cookie_manager3.py` 下載 flask-session-cookie-manager
  3. `python3 flask_session_cookie_manager3.py encode -s 'A_super_SecUrE_$eCR37_keY' -t '{"username": "admin"}'` 創建 admin session cookie，拿到 cookie:
     `eyJ1c2VybmFtZSI6ImFkbWluIn0.aD6O7A.2OvliEwHFW4q2Q2hcsa48Coaf9Y`
  4. 寫 code 在 `/code/p3_c.py` 內：

```
session.cookies.set('session', admin_cookie, domain='140.112.91.4')
```

來設定剛剛偽造的 cookie

5. 測試說是不是真的是成功登入 admin 帳號：

```
index_response = session.get("http://140.112.91.4:45510/")
print(f"Index status: {index_response.status_code}")

if "admin" in index_response.text and "Logout" in index_response.text:
    print("Successfully logged in as admin!")
```

回傳的 status 是 200 而且 `"admin"` 在 `index_response.text`，這就代表成功了

6. 最後就去 admin 的 `/my_submissions` route 然後去找含有 `HW12{...}` 的字串，最後就找到 flag3: HW12{i_l1KE_a15CR3am_MoRE_7H4n_Co0Ki3s}

4. Introduction to gnireenignE esreveR

註：此題檔案寫在 `code/p4.py`

ref: 1, 2

- flag: HW12{hW0_8UT_WiTH_r3V3Rse_eN91NE3rinG}
- 解題流程：
  1. 把 `chal.exe` 上傳到 Dogbolt 得到用 Ghidra decompile 的結果
  2. 看起來是用 C 寫，然後最主要的程式是：

```
undefined8 main(int param_1,long param_2)

{
  int iVar1;
  undefined8 uVar2;
  int local_c;

  if (param_1 == 2) {
    for (local_c = 0; local_c < flag_len; local_c = local_c + 1) {
      pattern[local_c] = pattern[local_c] ^ key[local_c % key_len];
    }
    iVar1 = strcmp(*(char **)(param_2 + 8),pattern);
    if (iVar1 == 0) {
      puts("Congratulations! You found the flag!");
    }
    else {
      puts("Haha! wrong >:)!!!!!!");
    }
    uVar2 = 0;
  }
  else {
    puts("Usage: ./chal.exe <flag>");
    uVar2 = 1;
  }
```

```
  return uVar2;
}
```

關鍵的 variables: `flag_len`, `key_len`, `pattern`, `key`

3. 用 `objdump -t chal.exe | grep -E "(flag_len|key_len|pattern|key)"` 找到變數的記憶體位址：

```
0000000000004020 g     O .data  0000000000000009              key
0000000000004068 g     O .data  0000000000000004              flag_len
0000000000004040 g     O .data  0000000000000027              pattern
000000000000406c g     O .data  0000000000000004              key_len
```

   - `key` 在 `0x4020`
   - `flag_len` 在 `0x4068`
   - `pattern` 在 `0x4040`
   - `key_len` 在 `0x406c`

4. 用 `objdump -s -j .data chal.exe` 提取二進制數據：

```
chal.exe:     file format elf64-x86-64

Contents of section .data:
 4000 00000000 00000000 08400000 00000000  .........@......
 4010 00000000 00000000 00000000 00000000  ................
 4020 6e417334 324f3253 00000000 00000000  nAs42O2S........
 4030 00000000 00000000 00000000 00000000  ................
 4040 26164206 49276563 31792660 6d185b07  &.B.I'ec1y&`m.[.
 4050 261e0107 647c6020 0b1e167a 0b7e7c16  &...d|` ...z.~|.
 4060 5d331a5a 75320000 26000000 08000000  ]3.Zu2..&.......
```

   - `key` 在 `0x4020`，對應到後面 `nAs42O2S`，所以 `key = nAs42O2S`
   - `flag_len` 在 `0x4068`：`26000000`，而x86-64 使用 Little Endian：`26000000` -> `0x00000026` = 38，所以 `flag_len = 38`
   - `key_len` 在 `0x406c`：`08000000`，而x86-64 使用 Little Endian：`08000000` -> `0x00000008` = 8，所以 `key_len = 8`，與前面的 `key` 符合
   - `pattern` 在 `0x4040`：

```
 4040 26164206 49276563 31792660 6d185b07  &.B.I'ec1y&`m.[.
 4050 261e0107 647c6020 0b1e167a 0b7e7c16  &...d|` ...z.~|.
 4060 5d331a5a 75320000                    ]3.Zu2..
```

     所以逐字節提取前 38 bytes：

     `encrypted_pattern = [0x26, 0x16, 0x42, 0x06, 0x49, 0x27, 0x65, 0x63, 0x31, 0x79, 0x26, 0x60, 0x6d, 0x18, 0x5b, 0x07, 0x26, 0x1e, 0x01, 0x07, 0x64, 0x7c, 0x60, 0x20, 0x0b, 0x1e, 0x16, 0x7a, 0x0b, 0x7e, 0x7c, 0x16, 0x5d, 0x33, 0x1a, 0x5a, 0x75, 0x32]`

5. 根據 decompile 出來的 C code 邏輯計算 `pattern` 跟 `key` 做 xor：`pattern[i] = pattern[i] ^ key[i % key_len]` 跑過整個 `pattern`，就拿到 flag 了