4.1

給定 g^{α}, g^{β}, 可算 $g^{\alpha\beta}$ ______________(1)

給定 g^{α}, 可算 g^{α^2} __________________(2)

給定 g^{α} 和 $\alpha \neq 0$, 可算 $g^{\frac{1}{\alpha}}$ __________(3)

給定 g^{α}, g^{β}, 且 $\beta \neq 0$, 可算 $g^{\frac{\alpha}{\beta}}$ ____(4)

當 $g^{\alpha} = g^{\beta}$, $g^{\alpha\beta} = g^{\alpha^2}$，所以(1)等價於(2) -> 此作法需作一次，因此為 poly-time

設存在 γ 使得 $g^{\gamma\alpha} = g^{\frac{1}{\alpha}}$,

則 $g^{\gamma\alpha^2} = g$, $g^{\gamma\alpha^4} = g^{\gamma\alpha^4}$

由(2)知，給定 g^{α^2}, 可算 g^{α^4}

因此當 $(g^{\alpha^4})^{\gamma} = g^{\alpha^2}$, 則 $g^{\gamma\alpha} = g^{\frac{1}{\alpha}}$ -> 此作法需作 γ 次，且 $\gamma <$ q，因此為 poly-time

設 $(g^{\frac{1}{\alpha}})^{k}$ 可解得 g，則 $(g^{\frac{1}{\alpha}})k^2$ 可解得 g^{α}，可得 k -> 此作法需作 k 次，且 k < q，因此為 poly-time

由(4)可得 $g^{\frac{\alpha}{\beta}}$

設存在 k 使得 $(g^{\frac{\alpha}{\beta}})^{k} = g^{\alpha}$，則 $(g^{\frac{\alpha}{\beta}})k^2 = g^{\alpha^2}$ -> 此作法需作 k 次，且 k < q，因此為 poly-time

4.2

1. 2-out-of-2

選擇一數值作為 sk1

並設 $sk2 = s \oplus sk1 \rightarrow s = sk2 \oplus sk1$

此時即可將 s 拆成(sk1, sk2)，並分別放在兩個 key server 中

在解密時，需先將 c 分別交給兩個 server

得到

$D(c, sk1) \rightarrow c'_1$

$D(c, sk2) \rightarrow c'_2$

$C(c, c'_1, c'_2)$:

在取得 c'_1 和 c'_2 後，根據 ElGamal，透過 (c, c'_1) 與 (c, c'_2) 可反推 sk1 與 sk2 的結果

再由一開始的設計 $s = sk2 \oplus sk1$， 即可找到 s，最終透過 D(c, s)即可找出明文 m

因此 $C(c, c'_1, c'_2) \rightarrow m$

此處拆解 secret key 的作法使用 One time pad 的概念，因此根據 one time pad，在沒有重複使用 sk1 的情況下，不會透露任何部份的 key。

2. 2-out-of-t

設計一方程式 $y = ax + b$，其中 $b = s$，隨機產生一個數字 a

在此方程式中，我們可以產生 t 組座標(x_i, y_i)

設 $sk_1 = (x_1, y_1)$, $sk_2 = (x_2, y_2)$, ……, $sk_{t-1} = (x_{t-1}, y_{t-1})$, $sk_t = (x_t, y_t)$

$D(c, sk_i) \rightarrow c'_i$

$D(c, sk_j) \rightarrow c'_j$

從任 2 個 sk 中取得兩點座標 $c'_i = (x_i, y_i)$, $c'_j = (x_j, y_j)$

$C(c, c'_i, c'_j)$:

由兩點座標帶入 $y = ax + b$ 可求出此方程式的係數，即可找到 s，有 S 後即可解開 c，取得明文 m

參考:

https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing

http://www.cs.columbia.edu/~tal/4261/F16/ss_slides_luke.pdf

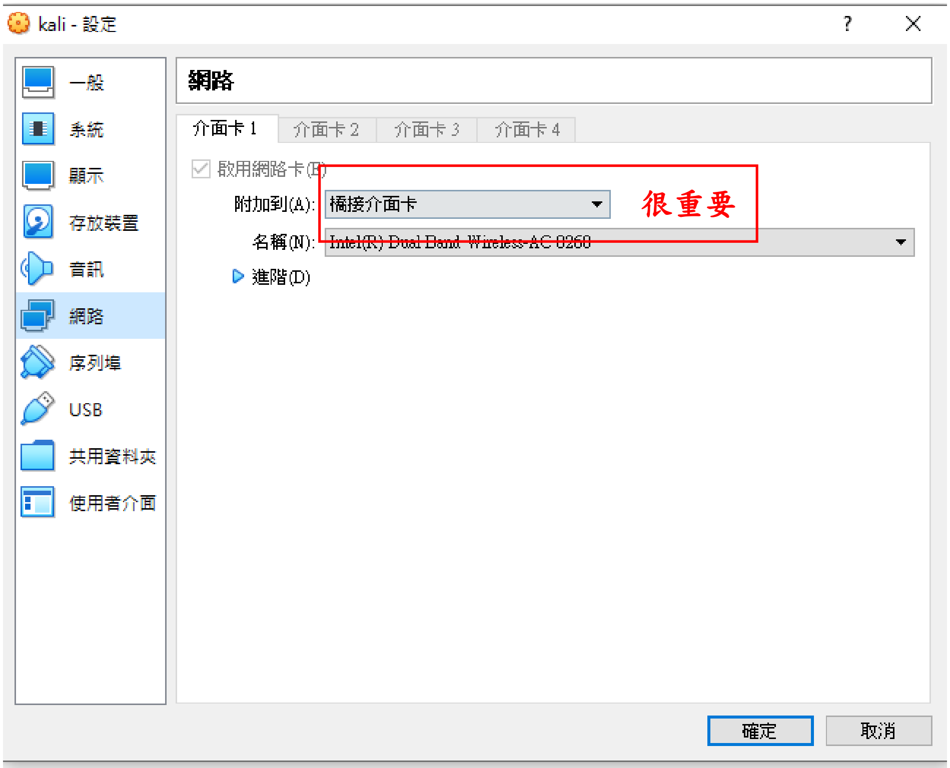
4.3

作法說明:

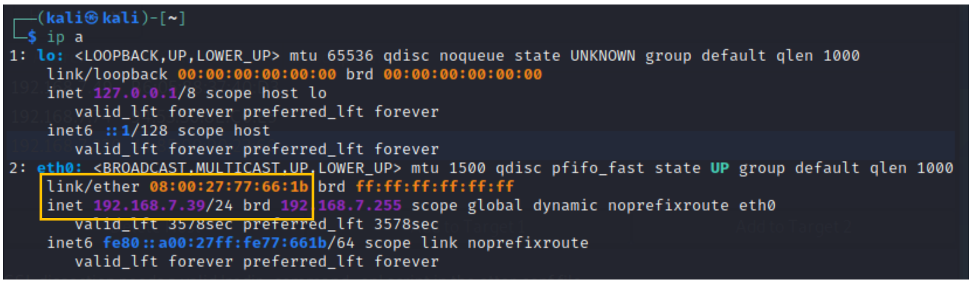
這次的中間人攻擊實作主要是使用 ARP 攻擊，ARP 協定透過廣播功能，獲取 IP 位址和實體位置的對應關係(建表)，而 ARP 攻擊主要是讓發送端誤以為 hacker 端是目標，讓發送端取得錯誤的 MAC 位址，從而監聽發送端的封包資訊

前處理:

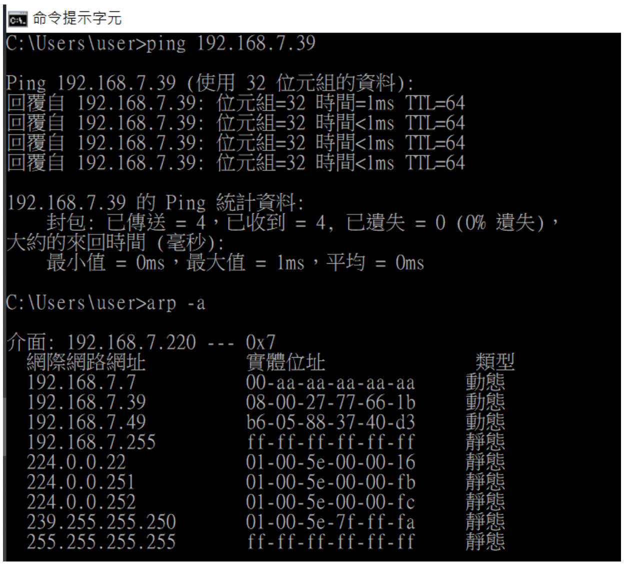
由於我使用 virtualbox 建立 kali linux 環境來對本機進行 ARP 攻擊，因此要記得先將網路設定為橋接介面卡，否則兩邊根本連 ping 都 ping 不到



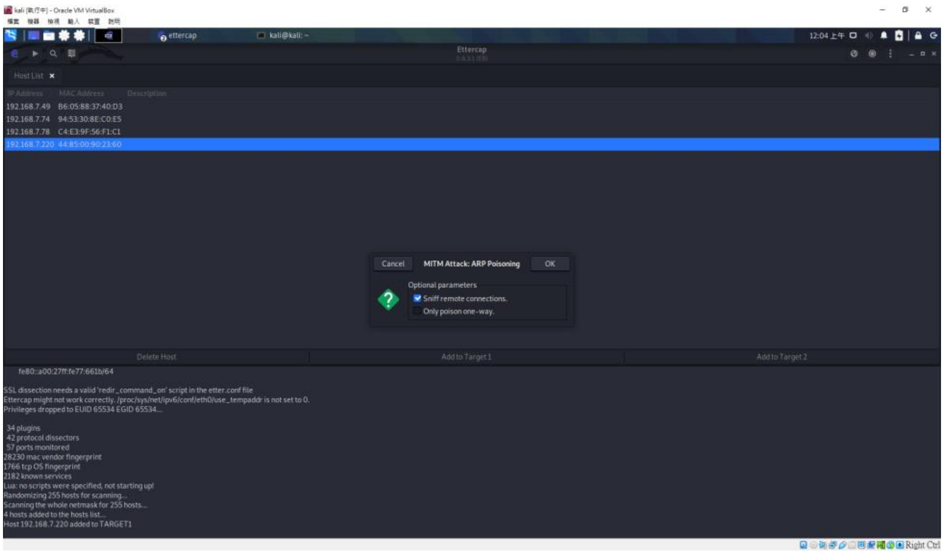
確認 hacker 端 ip(千萬不要亂改，一開始想改個好辨認的，害得自己 ping 的到卻找不到...)



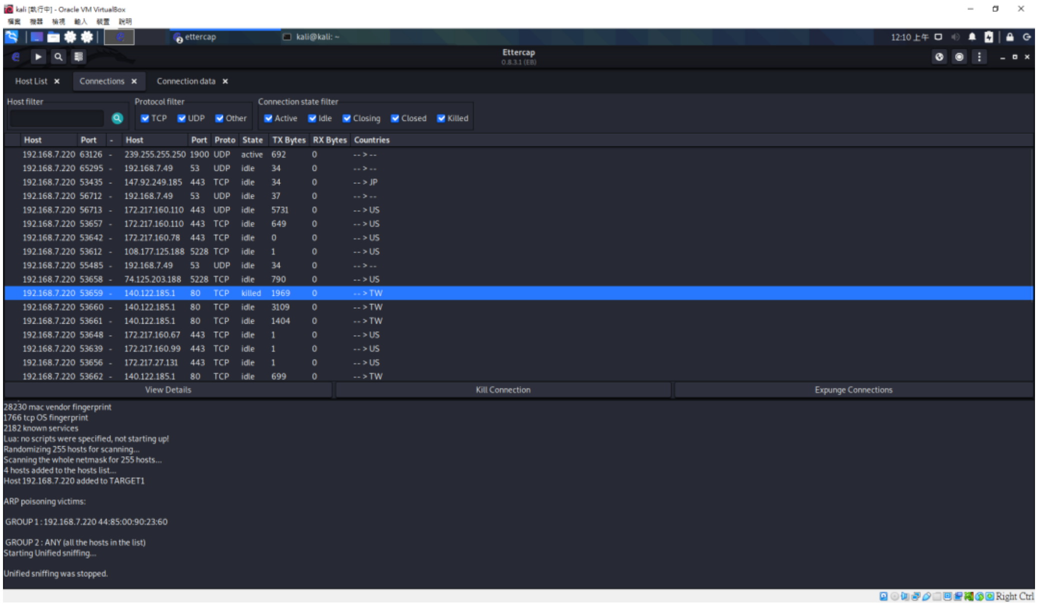
確認本地端能否 ping 到 hacker 端，並且查看 arp 表格，以確保在同一個區網中(否則在 ettercap 中根本抓不到)



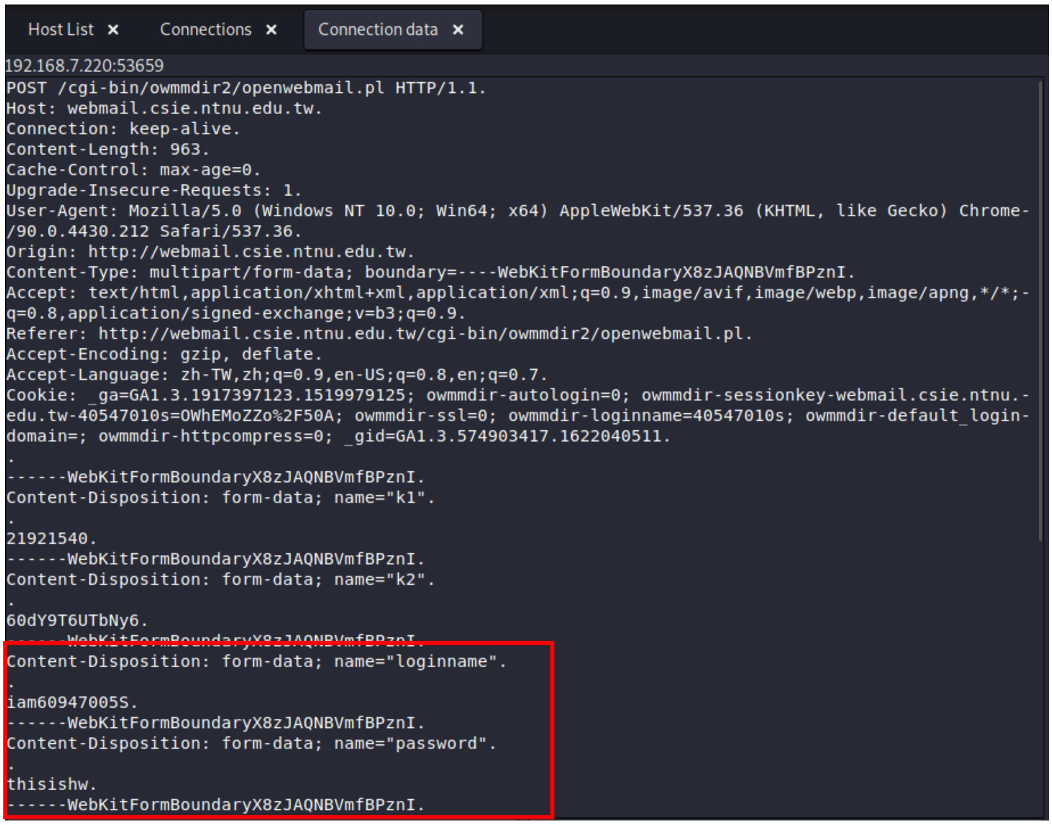
接著使用 ettercap 對本地端電腦進行 ARP 攻擊



透過 ARP 毒化，即可監聽此電腦的所有封包，找到師大類的 IP(140.122)，開始看封包



由於師大資工 webmail 網站是 http，傳輸的過程都是以明文顯示，
因此打開封包後，便可成功獲取師大資工 webmail 的輸入資訊
(此處為作業使用，所以沒有使用真實的帳號密碼)



參考：

https://www.youtube.com/watch?v=5AhB6rLgPuA&ab_channel=%E8%95%AD-%E5%BF%97%E6%98%8E

https://www.youtube.com/watch?v=ogtWS6MfiWM&ab_channel=ProfessorAndrew

https://www.pcnet.idv.tw/pcnet/network/network_ip_arp.htm

4.4

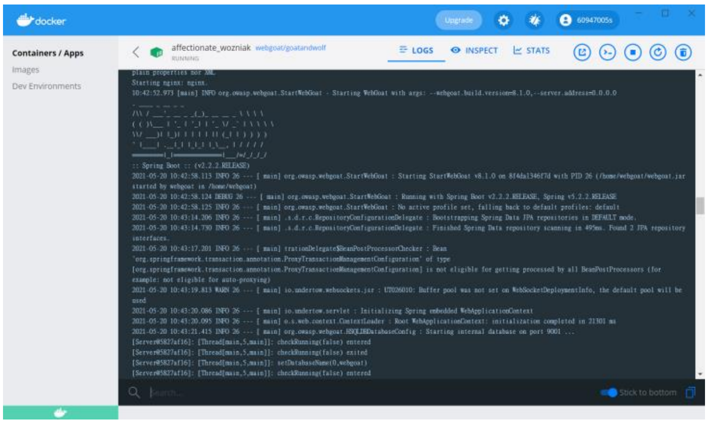
這個作者的攻擊手法，是透過 github，找到一些開源的 source code，並從裡面的一些 pattern(package.json)，來推測有哪些人會使這類 API，藉此抓取目標族群，並針對這目標族群，仿造他們所需的 API，並在內部加入惡意程式碼，讓使用者以為這是該 API 的最新版本，當有人使用此 API 時，就會遭受 DNS 滲透攻擊，讓使用者在封包傳出去的時候，傳到作者的 server 中，藉此從這些封包分析，可能可以獲得該企業所使用的更多 API 以及版本資訊，就可以用同樣的方式偽造更多的 API，來竊取更多的訊息，甚至是在內部加入其他想做的事情。

我的防禦方式是規範工程師在每次使用新版本的 API 時，必須先查看要使用的 API 更新了甚麼內容，像是透過 git 版本控制功能，針對會用到的函式查看內文有沒有出現一些不該出現的內容，另外一點就是不要把一些 pattern 以明文表示，可透過企業內部建立表格用對應的代號表示。

4.5

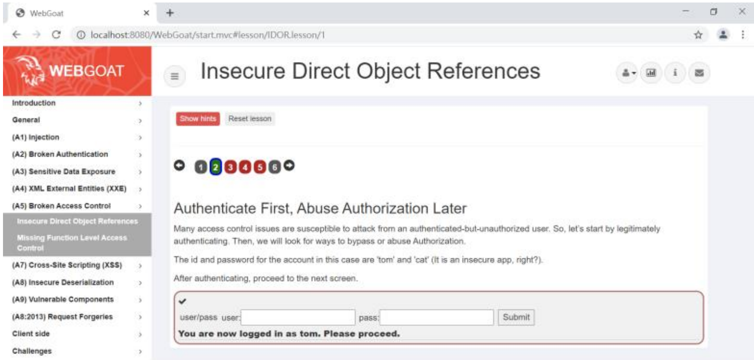
這題光是前置作業就花費我超久時間!

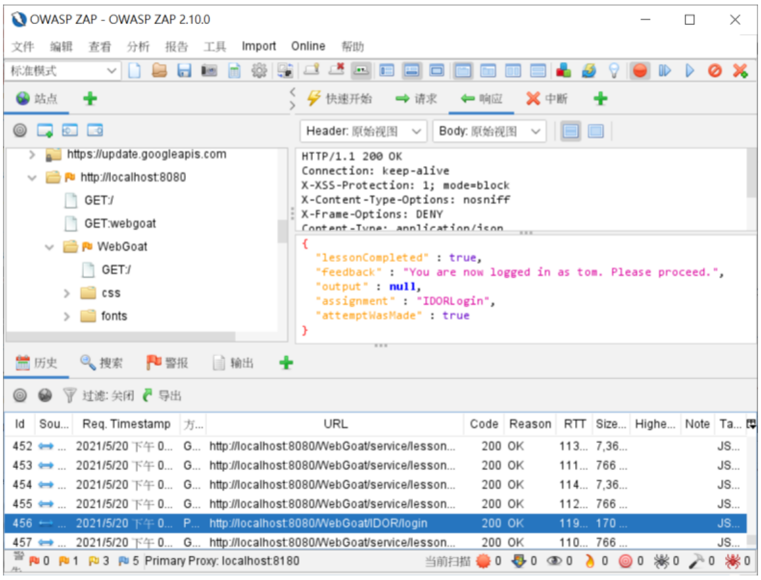
最後裝 docker，才成功連上



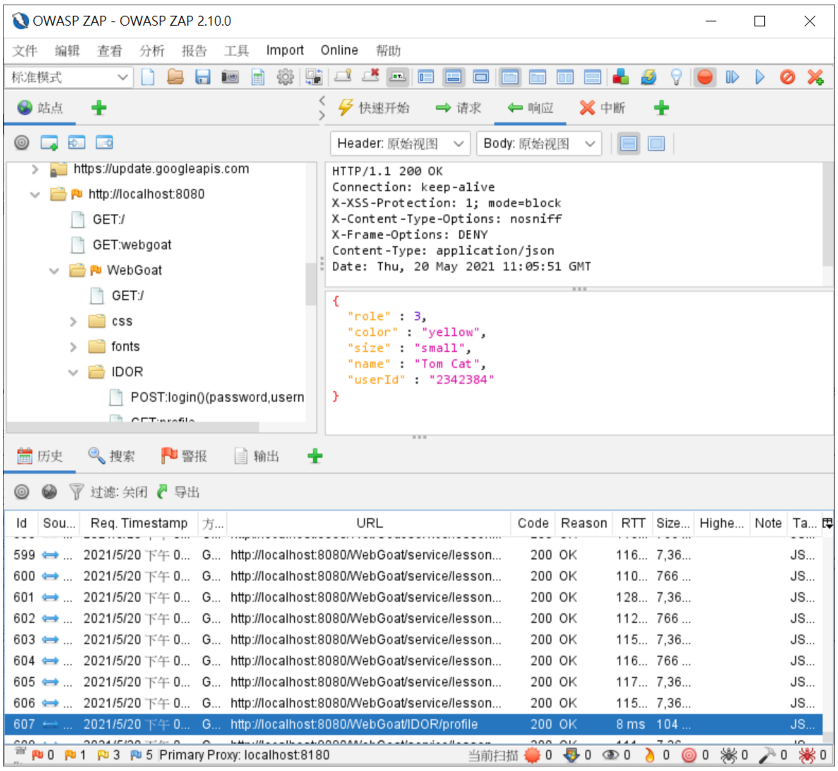
本題使用 OWASP ZAP 來幫忙監控

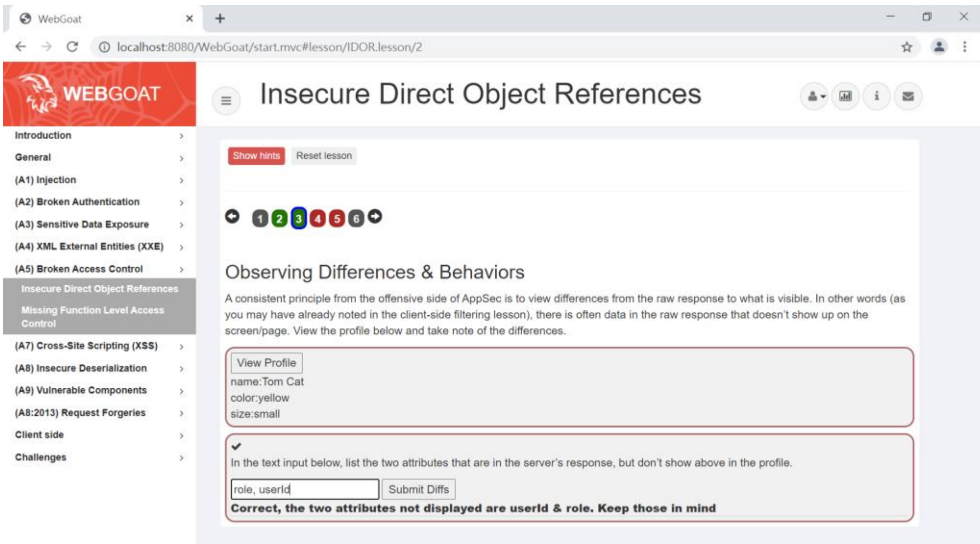
2. 輸入 tom cat，submit 後，會抓取到一個 post 封包



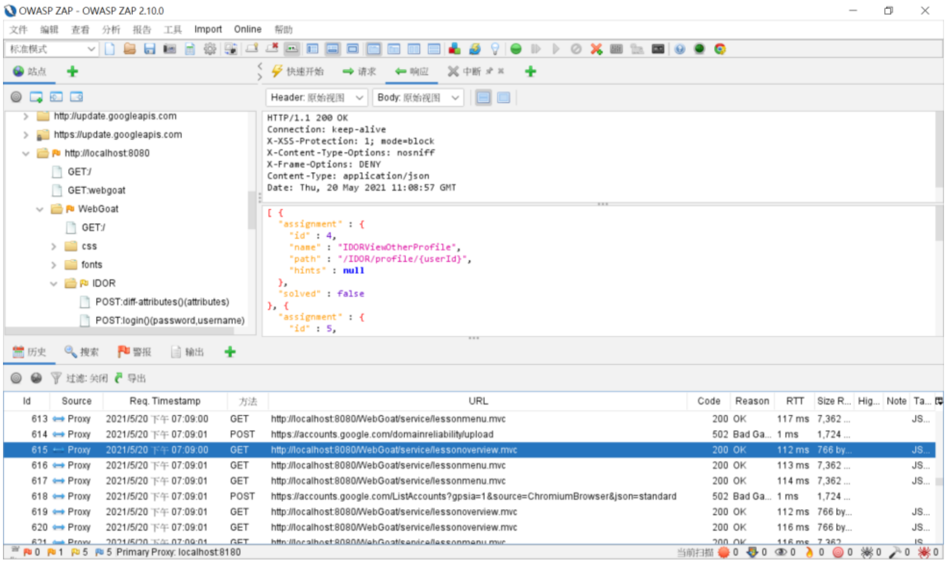


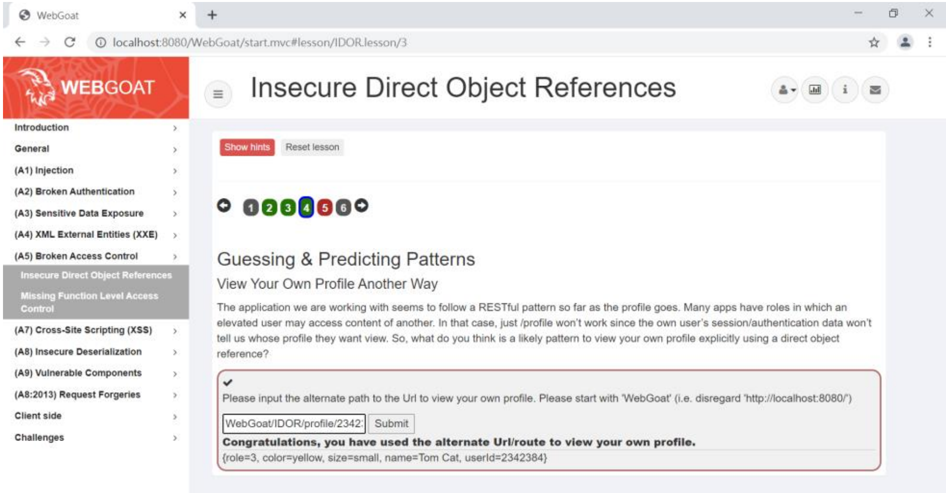
3. 按下 view profile，即可在 zap 中，找到完整的人物資訊





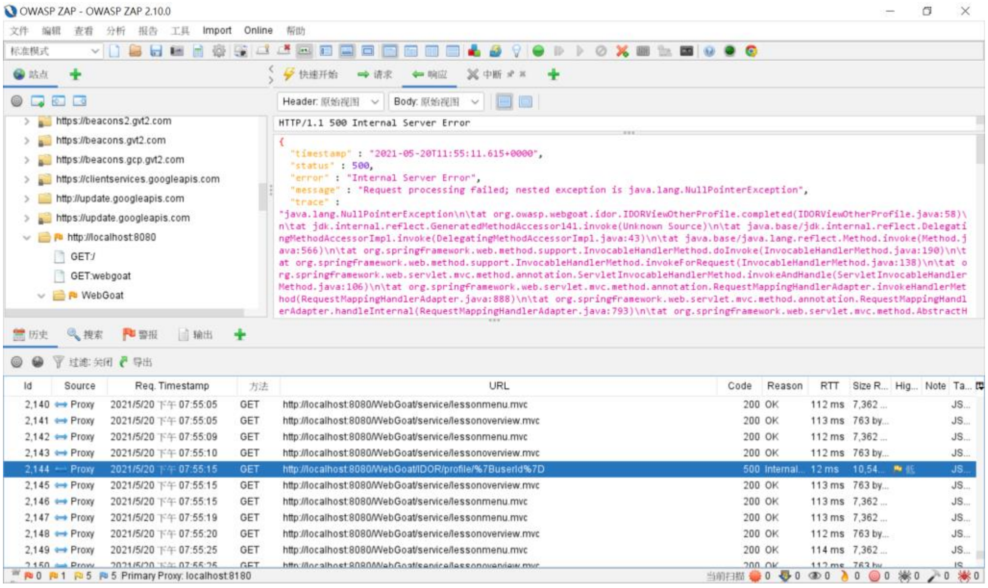
4. 根據上一題的 url，再加上 zap 中發現的一些格式規則(每 5 秒會發一組)，即可找出此題的 pattern



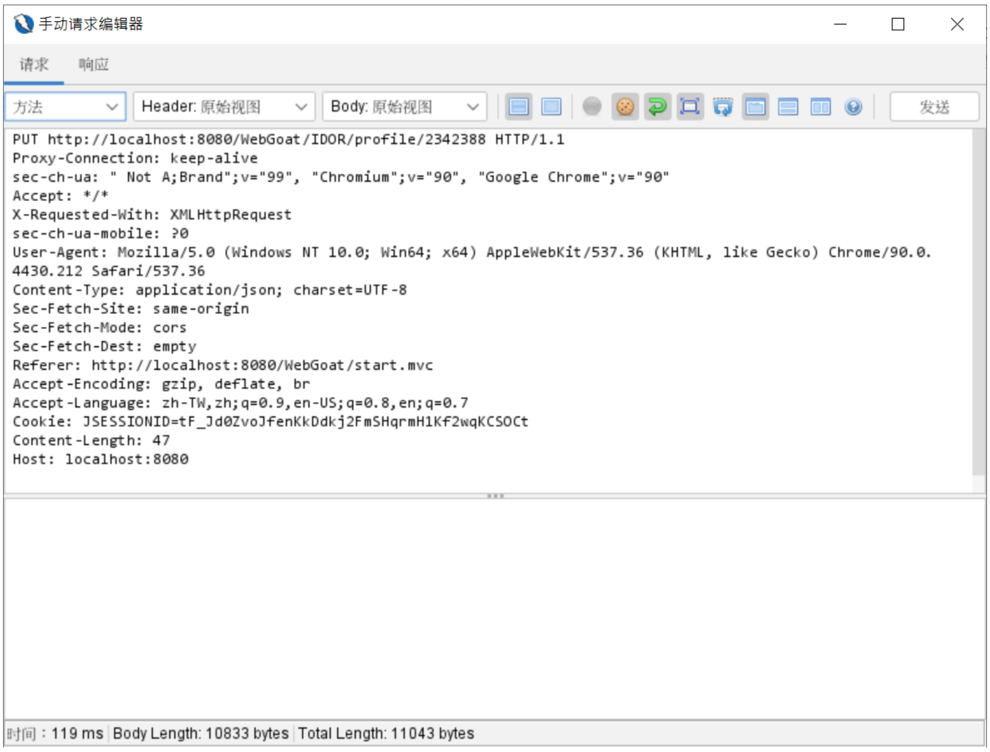


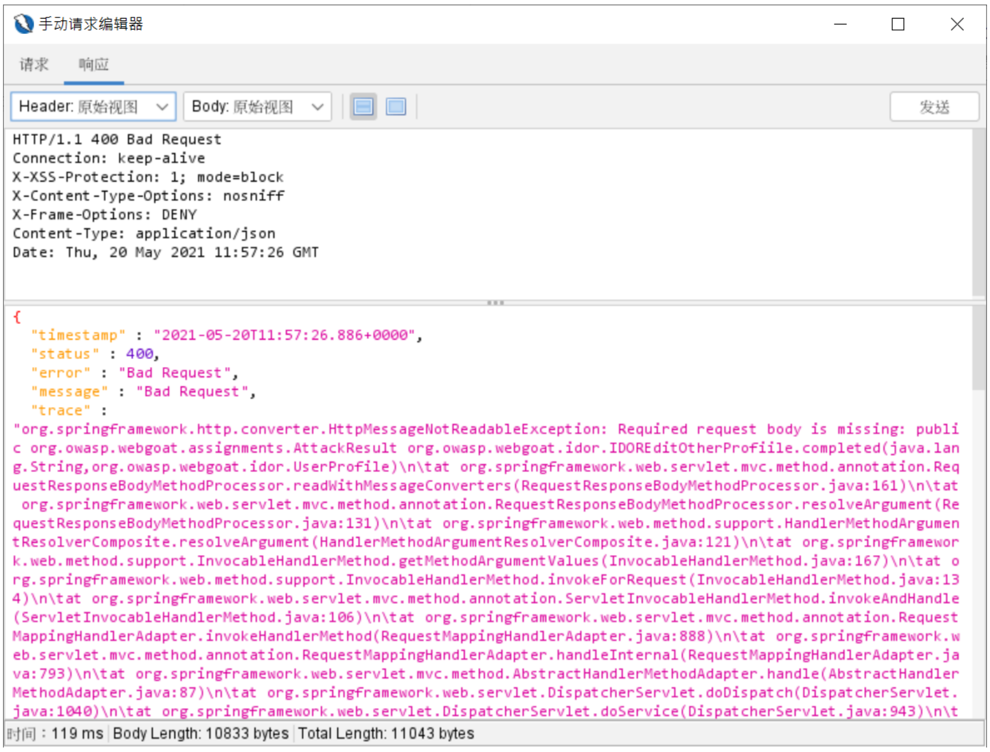
5_1. 老實說我本來完全不知道要幹嘛，按下 view profile 後，在 zap 也抓不到東西，後來另外開一個分頁，並將剛剛的 userId 不斷亂改，最終在 userid=2342388 時，找到另一位使用者

5_2. 按下第二個 view profile 後，在 zap 找到對應的封包，進行手動請
求編輯

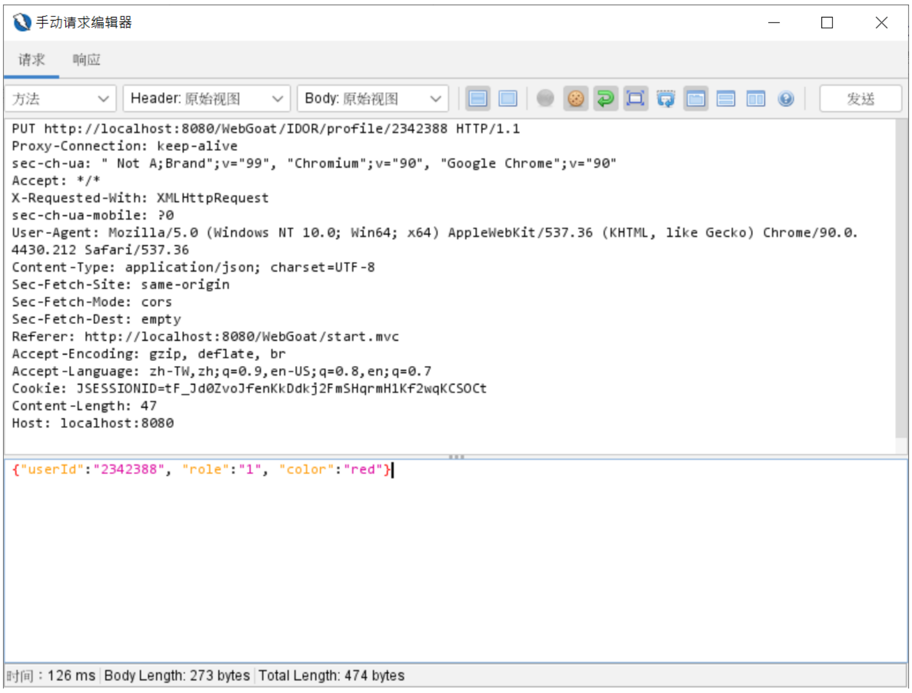


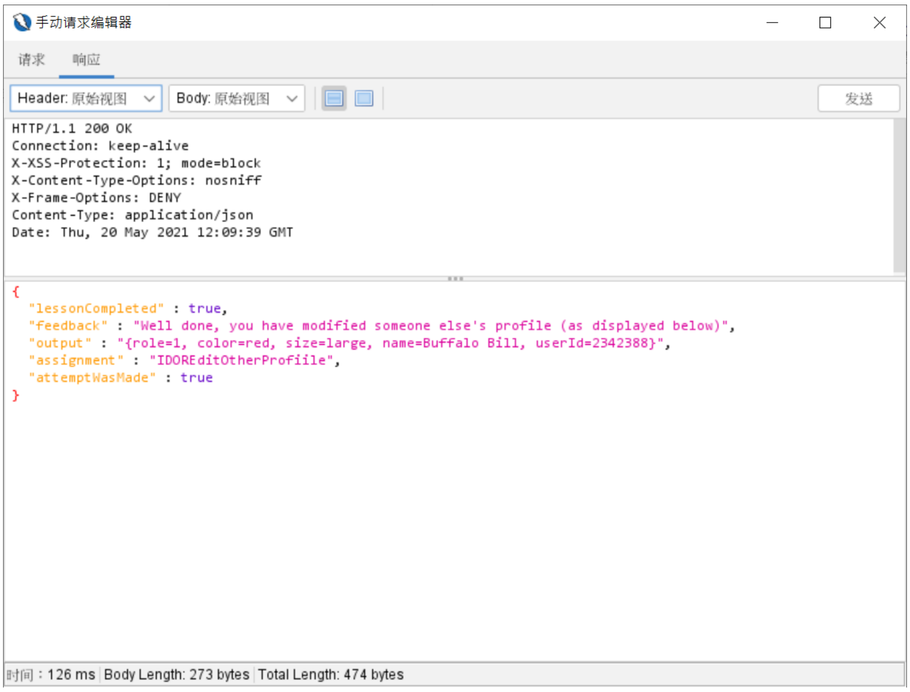
此處先修改 userid，將 content-type 改為 json，嘗試發送

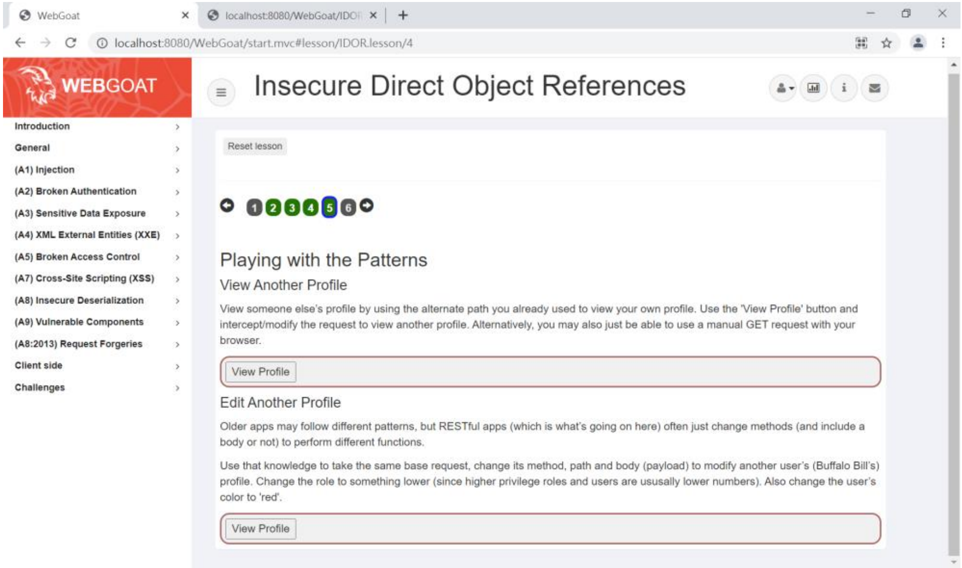




最終再加上 userId 以及要修改的內容，即可完成


手动请求编辑器
请求
响应
方法
Header: 原始视图
Body: 原始视图
发送
PUT http://localhost:8080/WebGoat/IDOR/profile/2342388 HTTP/1.1
Proxy-Connection: keep-alive
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90", "Google Chrome";v="90"
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Content-Type: application/json; charset=UTF-8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: JSESSIONID=tF_Jd0ZvoJfenKkDdkj2FmSHqrmH1Kf2wqKCSOCt
Content-Length: 47
Host: localhost:8080
{"userId":"2342388", "role":"1", "color":"red"}
时间：126 ms Body Length: 273 bytes Total Length: 474 bytes


手动请求编辑器
请求
响应
Header: 原始视图
Body: 原始视图
发送
HTTP/1.1 200 OK
Connection: keep-alive
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Thu, 20 May 2021 12:09:39 GMT
{
"lessonCompleted" : true,
"feedback" : "Well done, you have modified someone else's profile (as displayed below)",
"output" : "{role=1, color=red, size=large, name=Buffalo Bill, userId=2342388}",
"assignment" : "IDOREditOtherProfiile",
"attemptWasMade" : true
}
时间：126 ms Body Length: 273 bytes Total Length: 474 bytes



4.6

Task1

依照題目要求設定 p、q、e，並計算 phi(N) = (p-1)(q-1)，最後計算 ed ≡ 1 (mod phi(N)) 即可找出 d

程式碼如下：

```c
#include<stdio.h>
#include<openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();
    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");
    BIGNUM *n = BN_new();
    BN_CTX *ctx = BN_CTX_new();
    BN_mul(n, p, q, ctx);
    BIGNUM *p_1 = BN_new();
    BIGNUM *q_1 = BN_new();
    BN_sub(p_1, p, BN_value_one());
    BN_sub(q_1, q, BN_value_one());
    BIGNUM *phi = BN_new();
    BN_mul(phi, p_1, q_1, ctx);
    BIGNUM *d = BN_new();
    BN_mod_inverse(d, e, phi, ctx);
    printBN("Private key d =", d);
}
```

輸出結果:

```
[05/21/21]seed@VM:~/.../hw4_seedlab$ gcc task1.c -lcrypto
[05/21/21]seed@VM:~/.../hw4_seedlab$ ./a.out
Private key d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[05/21/21]seed@VM:~/.../hw4_seedlab$
```

Task2

使用 python 將明文轉成 hex string

```
[05/21/21]seed@VM:~/.../hw4_seedlab$ python3 -c 'print("A top secret!".encode("utf-8").hex())'
4120746f702073656372657421
```

程式碼:

```c
#include<stdio.h>
#include<openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
  char *number_str = BN_bn2hex(a);
  printf("%s %s\n", msg, number_str);
  OPENSSL_free(number_str);
}

int main()
{
  BIGNUM *n = BN_new();
  BIGNUM *e = BN_new();
  BIGNUM *m = BN_new();
  BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
  BN_hex2bn(&e, "010001");

  //input the plaintext
  BN_hex2bn(&m, "4120746f702073656372657421");
  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *c = BN_new();
  BN_mod_exp(c, m, e, n, ctx);
  printBN("Encrypt:", c);
  //check
  BIGNUM *d = BN_new();
  BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
  BN_mod_exp(m, c, d, n, ctx);
  printBN("Decrypt:", m);
}
```

輸出結果:

```
[05/21/21]seed@VM:~/.../hw4_seedlab$ gcc task2.c -lcrypto
[05/21/21]seed@VM:~/.../hw4_seedlab$ ./a.out
Encrypt: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
Decrypt: 4120746F702073656372657421
```

Task3

程式碼:

```c
#include<stdio.h>
#include<openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
  char *number_str = BN_bn2hex(a);
  printf("%s %s\n", msg, number_str);
  OPENSSL_free(number_str);
}

int main()
{
  BIGNUM *n = BN_new();
  BIGNUM *d = BN_new();
  BIGNUM *m = BN_new();
  BIGNUM *c = BN_new();
  BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
  BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

  //input the ciphertext
  BN_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

  BN_CTX *ctx = BN_CTX_new();
  BN_mod_exp(m, c, d, n, ctx);
  printf("%s\n", BN_bn2hex(m));
}
```

此處我另外寫了一個 16 進位轉字串的程式(tostring.py):

```
tostring.py
~/Desktop/hw4_seedlab
s = input()
out = ''
for i in range(0, len(s), 2):
  out += chr(int(s[i:i+2], 16))
print(out)
```

輸出結果:

```
[05/21/21]seed@VM:~/.../hw4_seedlab$ gcc task3.c -lcrypto
[05/21/21]seed@VM:~/.../hw4_seedlab$ ./a.out
50617373776F72642069732064656573
[05/21/21]seed@VM:~/.../hw4_seedlab$ python3 tostring.py
50617373776F72642069732064656573
Password is dees
[05/21/21]seed@VM:~/.../hw4_seedlab$
```

Task4

分別將兩串明文轉成 hex string

```
[05/21/21]seed@VM:~/.../hw4_seedlab$ python3 -c 'print("I owe you $2000.".encode("utf-8").hex())'
49206f776520796f752024323030302e
[05/21/21]seed@VM:~/.../hw4_seedlab$ python3 -c 'print("I owe you $3000.".encode("utf-8").hex())'
49206f776520796f752024333030302e
```

簽名程式碼:

```c
#include<stdio.h>
#include<openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
  char *number_str = BN_bn2hex(a);
  printf("%s %s\n", msg, number_str);
  OPENSSL_free(number_str);
}

int main()
{
  BIGNUM *n = BN_new();
  BIGNUM *e = BN_new();
  BIGNUM *d = BN_new();
  BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
  BN_hex2bn(&e, "010001");
  BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

  //input the plaintext
  BIGNUM *m = BN_new();
  BN_hex2bn(&m, "49206f776520796f752024333030302e");
  printBN("M =", m);
  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *s = BN_new();
  BN_mod_exp(s, m, d, n, ctx);
  printBN("Signature:", s);
}
```

輸出結果:

```
[05/21/21]seed@VM:~/.../hw4_seedlab$ python3 -c 'print("I owe you $2000.".encode("utf-8").hex())'
49206f776520796f752024323030302e
[05/21/21]seed@VM:~/.../hw4_seedlab$ python3 -c 'print("I owe you $3000.".encode("utf-8").hex())'
49206f776520796f752024333030302e
[05/21/21]seed@VM:~/.../hw4_seedlab$ gcc task4.c -lcrypto
[05/21/21]seed@VM:~/.../hw4_seedlab$ ./a.out
M = 49206F776520796F752024323030302E
Signature: 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
[05/21/21]seed@VM:~/.../hw4_seedlab$ gcc task4.c -lcrypto
[05/21/21]seed@VM:~/.../hw4_seedlab$ ./a.out
M = 49206F776520796F752024333030302E
Signature: BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
```

Task5

先將明文轉成 hex string，用於對照簽名

```
[05/21/21]seed@VM:~/.../hw4_seedlab$ python3 -c 'print("Launch a missile.".encode("utf-8").hex().upper()
)'
4C61756E63682061206D697373696C652E    _
```

對照簽名程式碼:

```c
#include<stdio.h>
#include<openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
  char *number_str = BN_bn2hex(a);
  printf("%s %s\n", msg, number_str);
  OPENSSL_free(number_str);
}

int main()
{
  BIGNUM *n = BN_new();
  BIGNUM *e = BN_new();
  BIGNUM *s = BN_new();
  BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
  BN_hex2bn(&e, "010001");
  BN_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");

  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *m = BN_new();
  BN_mod_exp(m, s, e, n, ctx);
  printBN("M =", m);
  char *target = "4C61756E63682061206D697373696C652E";
  if (*BN_bn2hex(m) == *target)
    printf("The signature is indeed Alice’s\n");
  else
    printf("Not!!!\n");
}
```

輸出結果:

```
[05/21/21]seed@VM:~/.../hw4_seedlab$ gcc task5.c -lcrypto
[05/21/21]seed@VM:~/.../hw4_seedlab$ ./a.out
M = 4C61756E63682061206D697373696C652E
The signature is indeed Alice’s
```

將 2F 改成 3F 的結果:

```
[05/21/21]seed@VM:~/.../hw4_seedlab$ gcc task5.c -lcrypto
[05/21/21]seed@VM:~/.../hw4_seedlab$ ./a.out
M_2F = 4C61756E63682061206D697373696C652E
M_3F = 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
[05/21/21]seed@VM:~/.../hw4_seedlab$ python3 tostring.py
91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
ñä,O´cè¼rm=fÈ:N¶·¾´�Â
```

Task6

我嘗試的網站是 5278 網站，並將內容存入 task6.txt 中，將兩段憑證分別存入 c0.pem 與 c1.pem

```
[05/25/21]seed@VM:~/.../hw4_seedlab$ openssl s_client -connect www.5278.cc:443 -showcerts > task6.txt
depth=1 C = US, O = Let's Encrypt, CN = R3
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN = 5278.cc
verify return:1
[05/25/21]seed@VM:~/.../hw4_seedlab$ cat task6.txt
CONNECTED(00000003)
---
Certificate chain
 0 s:CN = 5278.cc
   i:C = US, O = Let's Encrypt, CN = R3
-----BEGIN CERTIFICATE-----
MIIFHzCCBAegAwIBAgISBJeW8Mzp6QWNx4aifHVj+el/MA0GCSqGSIb3DQEBCwUA
MDIxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQswCQYDVQQD
EwJSMzAeFw0yMTA0MjcxNTA5MzlaFw0yMTA3MjYxNTA5MzlaMBIxEDAOBgNVBAMT
BzUyNzguY2MwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDYSh94frTs
zCue0o1W2vTE2iS/ccDgTZbqMGQG9igPML+2jpb4rZodE9kyvoBQYyIIuecXJgkm
whfwTO5Xa3ByyTL8cDoqIOPez18In+bwJpiQoQYLhpa8I22LZ/UbZNjOJrPG3lxN
2Gp7pbGindhB6nW1pR7eFAwTOnAxLtjMLffc/m78Ba86ABpVSyqeKpz4CC9Xu8r6
C1vPD2o0UoxFop2zaRhmcydVWnShAoQHmwIhsBmF66B5XEnavxkBA7e2vINxQ/0q
Qn9vxtoZJgHxPdMMD5OkYpvBvuwTGpLK9X76Chh1kJqLmzYnhWZFvBfUg1PXABwJ
X2yjYda16SBnAgMBAAGjggJNMIICSTAOBgNVHQ8BAf8EBAMCBaAwHQYDVR0lBBYw
FAYIKwYBBQUHAwEGCCsGAQUFBwMCMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFEg7
nBwhpyCkR8zfv/PfjNxolEnJMB8GA1UdIwQYMBaAFBQusxe3WFbLrlAJQOYfr52L
FMLGMFUGCCsGAQUFBwEBBEkwRzAhBggrBgEFBQcwAYYVaHR0cDovL3IzLm8ubGVu
Y3Iub3JnMCIGCCsGAQUFBzAChhZodHRwOi8vcjMuaS5sZW5jci5vcmcvMB0GA1Ud
EQQWMBSCCSouNTI3OC5jY4IHNTI3OC5jYzBMBgNVHSAERTBDMAgGBmeBDAECATA3
BgsrBgEFAYLfEwEBATAoMCYGCCsGAQUFBwIBFhpodHRwOi8vY3BzLmxldHNlbmNy
```

透過以下指令，從中找出(e,n)以及簽名

```
[05/25/21]seed@VM:~/.../hw4_seedlab$ nano c0.pem
[05/25/21]seed@VM:~/.../hw4_seedlab$ nano c1.pem
[05/25/21]seed@VM:~/.../hw4_seedlab$ openssl x509 -in c1.pem -noout -modulus
Modulus=BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F4794145535578C9EA8A23919F582
3C42A94E6EF53BC32EDB8DC0B05CF35938E7EDCF69F05A0B1BBEC094242587FA3771B313E71CACE19BEFDBE43B45524596A9C153CE34C852EEB5AEED8F
DE6070E2A554ABB66D0E97A540346B2BD3BC66EB66347CFA6B8B8F572999F830175DBA726FFB81C5ADD286583D17C7E709BBF12BF786DCC1DA715DD446
E3CCAD25C188BC60677566B3F118F7A25CE653FF3A88B647A5FF1318EA9809773F9D53F9CF01E5F5A6701714AF63A4FF99B3939DDC53A706FE48851DA1
69AE2575BB13CC5203F5ED51A18BDB15
```

```
[05/25/21]seed@VM:~/.../hw4_seedlab$  openssl x509 -in c1.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            40:01:75:04:83:14:a4:c8:21:8c:84:a9:0c:16:cd:df
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: O = Digital Signature Trust Co., CN = DST Root CA X3
        Validity
            Not Before: Oct  7 19:21:40 2020 GMT
            Not After : Sep 29 19:21:40 2021 GMT
        Subject: C = US, O = Let's Encrypt, CN = R3
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:bb:02:15:28:cc:f6:a0:94:d3:0f:12:ec:8d:55:
                    92:c3:f8:82:f1:99:a6:7a:42:88:a7:5d:26:aa:b5:
                    2b:b9:c5:4c:b1:af:8e:6b:f9:75:c8:a3:d7:0f:47:
                    94:14:55:35:57:8c:9e:a8:a2:39:19:f5:82:3c:42:
                    a9:4e:6e:f5:3b:c3:2e:db:8d:c0:b0:5c:f3:59:38:
                    e7:ed:cf:69:f0:5a:0b:1b:be:c0:94:24:25:87:fa:
                    37:71:b3:13:e7:1c:ac:e1:9b:ef:db:e4:3b:45:52:
                    45:96:a9:c1:53:ce:34:c8:52:ee:b5:ae:ed:8f:de:
                    60:70:e2:a5:54:ab:b6:6d:0e:97:a5:40:34:6b:2b:
                    d3:bc:66:eb:66:34:7c:fa:6b:8b:8f:57:29:99:f8:
                    30:17:5d:ba:72:6f:fb:81:c5:ad:d2:86:58:3d:17:
                    c7:e7:09:bb:f1:2b:f7:86:dc:c1:da:71:5d:d4:46:
                    e3:cc:ad:25:c1:88:bc:60:67:75:66:b3:f1:18:f7:
                    a2:5c:e6:53:ff:3a:88:b6:47:a5:ff:13:18:ea:98:
                    09:77:3f:9d:53:f9:cf:01:e5:f5:a6:70:17:14:af:
                    63:a4:ff:99:b3:93:9d:dc:53:a7:06:fe:48:85:1d:
                    a1:69:ae:25:75:bb:13:cc:52:03:f5:ed:51:a1:8b:
                    db:15
                Exponent: 65537 (0x10001)
```

將 c0.pem 簽名內容另存，並去掉所有的空格與冒號

```
[05/25/21]seed@VM:~/.../hw4_seedlab$ nano signature
[05/25/21]seed@VM:~/.../hw4_seedlab$ cat signature | tr -d '[:space:]:'
430ffaf8a1dc514a547d75a1b13a30ffcb625a9bd1c7d55a6a9b7a9dc5161885d7dab93839162621c0ed9f0a2a8091a4fdf107fc50ca3c87f4a8093fef
d25a5e21f24360567d34e488070ac4a385db6bdd472de5859f21dbfdb17450c81cd09151ec85c8a02a1405f98656817515c9fd030680accab91d626858
3ddce632cc4fbeb9f6ffeff91c670ca427b3fa642de8c293dad909510d760c1a2c1f8c0edef2c22fc302d7ee5b703131a816e2b86b6f3b12e4fbde4821
e9c1674620a621c108f53064d754496b0b8097a547851fd85e665ad91c195ae9f3271f6817c533027cfc4593d5af9f11fe11869745ac111e23c78a534e
d6f7aa5f1eb794d4ba59c301[05/25/21]seed@VM:~/.../hw4_seedlab$
```

輸出憑證主體並計算 hash 值，憑證是從 4 開始，所以從 4 開始看

```
[05/25/21]seed@VM:~/.../hw4_seedlab$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
[05/25/21]seed@VM:~/.../hw4_seedlab$ sha256sum c0_body.bin
1c17f0624f5c572e7fa1f1eaf234c2f44e6104f3e76bfd210a05e8c07d2932ec  c0_body.bin
```

利用獲得的所有資訊進行驗證

程式碼:

```c
#include<stdio.h>
#include<openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
  char *number_str = BN_bn2hex(a);
  printf("%s %s\n", msg, number_str);
  OPENSSL_free(number_str);
}

int main()
{
  BIGNUM *n = BN_new();
  BIGNUM *e = BN_new();
  BIGNUM *s = BN_new();
  BIGNUM *hash = BN_new();

  BN_hex2bn(&n,
"BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F4794145535578C9EA8A23919F5823C42A9
  BN_hex2bn(&e, "10001");
  BN_hex2bn(&s,
"430ffaf8a1dc514a547d75a1b13a30ffcb625a9bd1c7d55a6a9b7a9dc5161885d7dab93839162621c0ed9f0a2a8091a4fdf107fc50ca3c87f4a8093f
  BN_hex2bn(&hash, "1c17f0624f5c572e7fa1f1eaf234c2f44e6104f3e76bfd210a05e8c07d2932ec");

  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *c = BN_new();
  BN_mod_exp(c, s, e, n, ctx);
  printBN("Verify result:", c);

}
```

最終輸出結果包含server’s certificate hash即表示驗證成功。

```
[05/25/21]seed@VM:~/.../hw4_seedlab$ gcc task6.c -lcrypto
[05/25/21]seed@VM:~/.../hw4_seedlab$ ./a.out
Verify result: 01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D0609608648016503040201050004201C17F0624F5C572E7FA1F1EAF23
4C2F44E6104F3E76BFD210A05E8C07D2932EC
```

參考:

https://blog.csdn.net/weixin_48392428/article/details/107433009