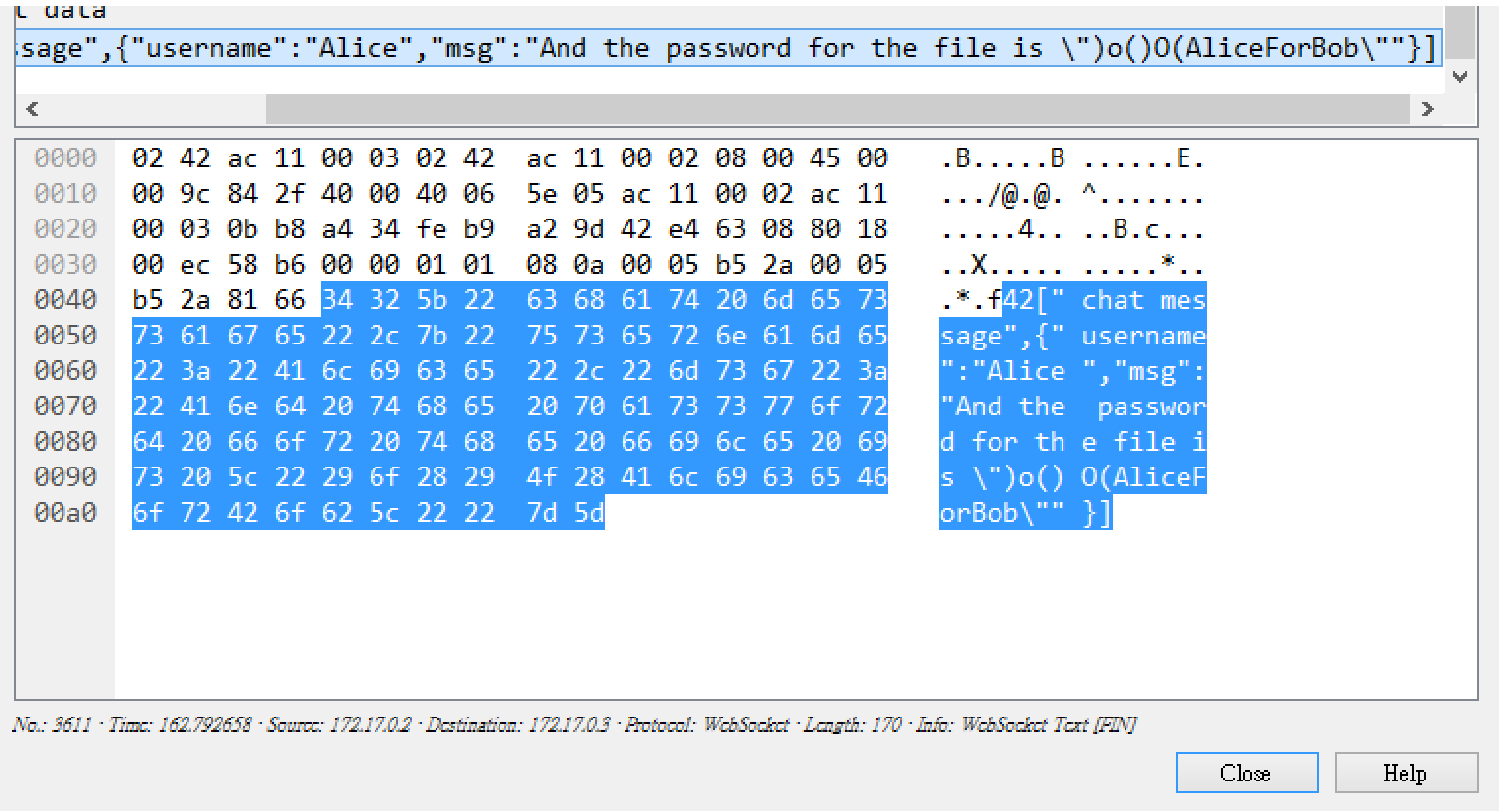
Project2 SSL

0316025 – 賴文揚

Get password from websocket (No encryption)

- Password:)o()O(AliceForBob



Export certifications

- Export certificates as .der file
  - Right click, “Export packet bytes”

```
◢ Secure Sockets Layer
  ▷ TLSv1.1 Record Layer: Handshake Protocol: Server Hello
  ◢ TLSv1.1 Record Layer: Handshake Protocol: Certificate
      Content Type: Handshake (22)
      Version: TLS 1.1 (0x0302)
      Length: 828
    ◢ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 824
        Certificates Length: 821
      ◢ Certificates (821 bytes)
          Certificate Length: 818
        ▷ Certificate: 3082032e30820216020900d1a101a9ad017327300d06092a... (id-at-commonName=NS-HW2-11,id-at-organiza
  ▷ TLSv1.1 Record Layer: Handshake Protocol: Server Hello Done
```

Convert format

- First, convert .der file to pem format file
  - $ openssl x509 -inform der -in in_file.der -outform pem -out out_file.pem
- Second, convert pem format file to public key format
  - $ openssl x506 -pubkey -noout -in in.pem > pubkey.pem

RSA algorithm

- If we can divide the n, we can gain the private key.
  - Do common factorizing.

Key Generation Alice

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Common factorizing

- Using python-crypto library to load public key file and calculate gcd from pair of these 12 public files.
  - $ ./common_factorize.py

```python
#!/usr/bin/python

from Crypto.PublicKey import RSA
from fractions import gcd
pems = []
for i in range(1, 11):
    p = open("pems/pubkey_" + str(i) + ".pem").read()
    pems.append(RSA.importKey(p))

for i in range(0, 10):
    for j in range (i+1, 10):
        n = gcd(pems[i].n, pems[j].n)
        if  n != 1:
            print (str(i) + " and " + str(j) +  " have common factor: ")
            print (n)


```

Common factorizing (Cont.)

- We can get the information below.

```
10:22 wylai@rails-env(192.168.220.137)[~/github/NS/NetworkSecurity/proj2] ± [master]
[XD] % ./common_factorize.py
3 and 8 have common factor:
14656665144589336868890576345676445233783803276368267622102594568299164979334002689085447204937159234673
04541912218503714084065814754185790088811115710921735307483316671075826228613097271501609144807818412051
55449584530166428770678446245420268299373990760393892275516496045323891286171163252445865368303271017
10:23 wylai@rails-env(192.168.220.137)[~/github/NS/NetworkSecurity/proj2] ± [master]
[XD] %
```

Retrieve the private key

- Using gmpy library to calculate the multiple modular inverse and export the private key to a file.
  - In this code, I retrieve the private key using in https3.pcapng

```
#!/usr/bin/python

from Crypto.PublicKey import RSA
from fractions import gcd
import gmpy

targets = ['pems/pubkey_3.pem', 'pems/pubkey_8.pe

pems = []

for target in targets:
    p = open(target).read()
    pems.append(RSA.importKey(p))

print ("Get n1:")
print (pems[0].n)

p = gcd(pems[0].n, pems[1].n)
print ("Get p: ")
print (p)

q1 = pems[0].n // p
print ("Get q1: ")
print (q1)

d1 = long(gmpy.invert(pems[0].e, (p-1) * (q1-1)) )
print ("Private key d1")
print (d1)

private_key = RSA.construct((pems[0].n, pems[0].e, d1, p, q1))
open("private_key_3.pem", 'w').write(private_key.exportKey('PEM'))
```

Decryption via wireshark

- Edit -> preferences -> protocol -> SSL
  - Add the private key to here,
  - IP: server ip
  - Port: 443
  - Protocol: http
  - File path: /path/to/private_key.pem

After decryption

2.158319	172.17.0.4	172.17.0.5	TLSv1…	583 Client Hello
2.158329	172.17.0.5	172.17.0.4	TCP	66 443 → 42296 [ACK] Seq=1 Ack=518 W
2.158520	172.17.0.5	172.17.0.4	TLSv1…	981 Server Hello, Certificate, Server
2.158542	172.17.0.4	172.17.0.5	TCP	66 42296 → 443 [ACK] Seq=518 Ack=916
2.169180	172.17.0.4	172.17.0.5	TLSv1…	380 Client Key Exchange, Change Ciphe
2.170931	172.17.0.5	172.17.0.4	TLSv1…	304 New Session Ticket, Change Cipher
2.178063	172.17.0.4	172.17.0.5	HTTP	1070 POST / HTTP/1.1
2.179760	172.17.0.5	172.17.0.4	HTTP	405 HTTP/1.1 200 OK (text/plain)
2.216075	172.17.0.4	172.17.0.5	TCP	66 42296 → 443 [ACK] Seq=1836 Ack=14
3.648947	172.17.0.4	151.101.0.133	TCP	54 45534 → 443 [ACK] Seq=1 Ack=1 Win
3.649132	151.101.0.133	172.17.0.4	TCP	54 [TCP ACKed unseen segment] 443 →
4.180555	172.17.0.5	172.17.0.4	TLSv1…	93 Alert (Level: Warning, Descriptio
4.180585	172.17.0.4	172.17.0.5	TCP	66 42296 → 443 [ACK] Seq=1836 Ack=15
4.180618	172.17.0.5	172.17.0.4	TCP	66 443 → 42296 [FIN, ACK] Seq=1520 A

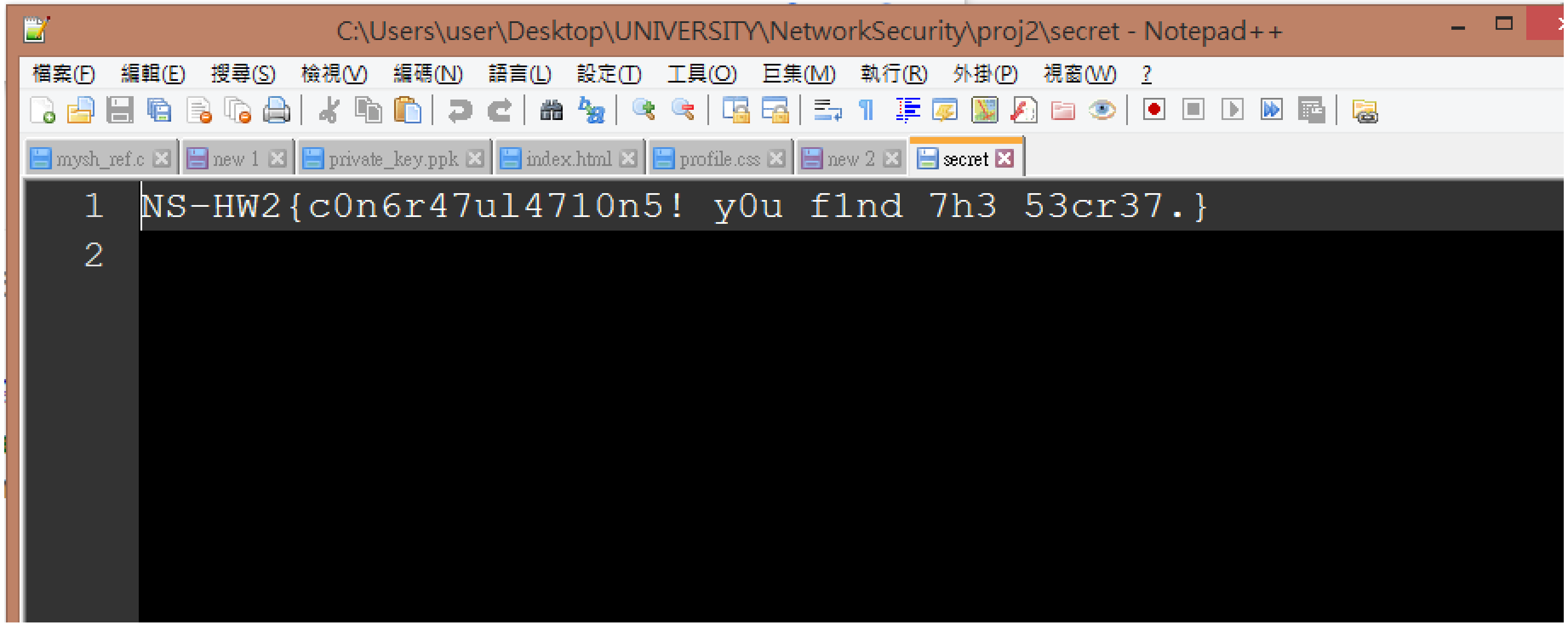
Export secret.zip

- See the data payload section, and export “Data” as “secret.zip” file

```
◢ Encapsulated multipart part:  (application/octet-stream)
    Content-Disposition: form-data; name="file[]"; filename="secret.zip"\r\n
    Content-Type: application/octet-stream\r\n\r\n
  ◢ Data (236 bytes)
      Data: 504b03040a0009000000c47b8a4adf08e58b3a0000002e00...
      [Length: 236]
  Last boundary: \r\n-----------------------------1510992731146319113775032890--\r\n
```

Retrieve the secret

- Decompressed the zip file by the key gain in “websocket”, and get the secret.



Summary – What I learned

- Python-crypto library
  - We can use this library to help us generate the RSA key pair and load RSA keys.
- Wireshark
  - It will help us to snoopy the network traffic and get information.
- RSA common factory vulnerability
  - If the p, q pair chose not randomly enough. Collect enough packets, and the attack will retrieve the private key and crack the secret.