

# NASA HWO

---

B11901164 陳秉緯

## Network Administration

---

### 1. Short Answer

1. [ref1](#), [ref2](#), [ref3](#)

- 實體層 (Physical Layer)
  - 功能簡述：負責實際的數據傳輸。
  - 應用例子：電纜、光纖
- 資料連結層 (Data Link Layer)
  - 功能簡述：定義來源/目的的MAC地址，以及所使用的協定為何，負責相鄰節點間的資料傳輸。
  - 應用例子：Ethernet、PPP
- 網路層 (Network Layer)
  - 功能簡述：負責處理將封包由來源電腦傳給目的電腦之間的路徑選擇問題，在此定義該封包需要送往哪一個IP位置。
  - 應用例子：IPv4、IPv6、ICMP
- 傳輸層 (Transport Layer)
  - 功能簡述：負責端到端的數據傳輸，確保數據的可靠性和完整性。
  - 應用例子：TCP、UDP
- 應用層 (Application Layer)
  - 功能簡述：負責應用程式之間的通訊服務，任何的通訊協定、功能都會在此被定義。
  - 應用例子：HTTP、FTP、DNS

2. [ref1](#), [ref2](#), [ref3](#), [ref4](#), [ref5](#)

(a) 虛擬區域網路 (VLAN, Virtual Local Area Network) 是在較大的實體網路的邏輯中相互隔離區域的技術，通常用於將一個大型的區域網路劃分成多個較小的邏輯上獨立的區域，從而隔離流量並提高網路效能

(b)

- Switch（交換器）：連接網路（通常是 LAN）內的裝置，並在這些裝置之間轉寄資料封包，交換器僅將資料傳送到預定的單個裝置（可能是另一個交換器、路由器或使用者的電腦），而不是包含多個裝置的網路。
- Router（路由器）：路由器透過連接不同的網路並在網路之間轉寄資料為資料封包選擇路徑以穿過網路並到達其目的地，這些網路包括 LAN、WAN 或自發系統，構成網際網路的大型網路。
- 比較它們在 TCP/IP 五層架構中扮演的角色和主要差異：
  - Switch：主要運作在資料連結層（Layer 2），部分高階交換器（Layer 3 Switch）則也具備部分網路層的功能，但主要任務還是在局部網路內進行資料封包轉發與處理。
  - Router：主要運作在網路層（Layer 3），負責 IP 層的地址解析與路由選擇，使數據能夠跨越不同網路進行傳輸。

(c)

- Broadcast Storm：當網路中出現大量廣播封包時，導致網路設備（如交換機）不斷轉發這些封包，最終使網路資源耗盡，影響正常通訊。
- Switching Loop：當交換機之間形成環路時，會導致數據包不斷循環轉發，造成網路癱瘓。
- 不同之處：
  - 成因：
    - Broadcast Storm：通常由於過多的廣播流量、病毒攻擊，或是 Switching Loop 導致的廣播封包無限循環。
    - Switching Loop：通常由網路拓撲設計錯誤或未啟用防迴路機制（如STP）引起。
  - 影響：
    - Broadcast Storm：主要消耗網路頻寬和設備資源，導致網路變慢或癱瘓。
    - Switching Loop：除了消耗頻寬外，還可能導致MAC地址表錯亂，導致廣播風暴。
  - 影響範圍：
    - Broadcast Storm：主要影響廣播域內的設備。
    - Switching Loop：可能影響整個交換網路。
- 相似之處
  - 都會導致網路效能嚴重下降，甚至癱瘓。
  - 都會大量消耗網路頻寬和設備資源。
- 如何防範
  - Broadcast Storm：
    - 啟用Broadcast Storm Control，限制廣播封包的數量。
    - 使用VLAN分割廣播域，減少廣播封包的影響範圍。
  - Switching Loop：

- 啟用STP (Spanning Tree Protocol) 來防止迴路形成。
- 確保網路拓撲設計合理，避免不必要的多餘連接。

### 3. [ref1](#), [ref2](#)

1. 主要原因：IPv4 位址不夠用了，IPv4 使用 32 位元，總共約 43 億個 ( $2^{32}$ ) 位址，現在來說已經不夠了，但IPv6 使用 128 位元，可提供  $2^{128}$  個位址，能夠解決目前 IP 位址短缺問題。

2. 有可能：隨著量子網路的發展，現有的網際網路協定（如 IPv6）可能無法充分利用量子通信的特性，因此需要設計新的協定來適應這些變化

3. 差異：

	IPv4	IPv6
位址長度	32 位元	128 位元
位址數量	約 43 億個	$2^{128}$
表示方式	192.168.1.1	2001:0db8:85a3::8a2e:0370:7334
NAT	需要 NAT（因位址不足）	不需要 NAT（每台設備都有唯一 IP）

4. 原因：

- 許多老舊設備、伺服器和網路應用仍依賴 IPv4，全面升級需要時間和成本。
- 許多企業內部網路仍使用 IPv4，因為透過 NAT 就能有效管理內部裝置，而不需要急於轉換到 IPv6。

### 4. [ref1](#), [ref2](#), [ref3](#)

1.   ▪ UDP的運作方式：

- 無連接：UDP 不需要建立連接，發送端直接將資料包發送給接收端。
- 資料傳輸：UDP 只是將資料包傳送出去，並不管資料是否成功送達，也不會做重傳。每個資料包是獨立的，並沒有序列化的過程。
- 無結束過程：傳送資料後，UDP 不會進行任何的結束程序，這樣簡化了流程，減少了延遲。

▪ TCP的運作方式：

- 建立連接：TCP 使用三次握手（3-way handshake）來建立連接。先由客戶端發送一個請求（SYN），伺服器回應一個確認訊息（SYN-ACK），最後客戶端再發送確認訊息（ACK）。
- 資料傳輸：建立連接後，TCP 會將資料分成小段，並確保每一段都有順序和完整性。它會加上序號，接收方會回傳確認訊息（ACK）。如果資料丟失或錯誤，發送方會重傳。
- 結束連接：當資料傳輸完成後，TCP 會使用 4-way handshake 來終止連線，這樣雙方可以安全地關閉連接。

2. ■ 相同處：

- 兩者都屬於 OSI 模型中的傳輸層 (Layer 4)，負責應用程式之間的資料傳輸。
- 兩者都基於 IP (Internet Protocol) 來傳送資料。

■ 相異點：

	TCP	UDP
連接方式	連接導向（需三次握手）	無連接
可靠性	可靠，確保資料完整	不可靠，可能丟失或亂序
傳輸順序	保證順序	不保證順序
適用場景	需要可靠傳輸的應用	需要低延遲、高效能的應用

3. ■ 傾向使用 UDP 的情況

- 對延遲敏感的應用：比如線上遊戲、即時視訊會議（Zoom、Webex）。
- 高效能需求的應用：比如串流媒體（YouTube、Netflix）。
- 簡單廣播或多播的應用：比如網路廣播、多播應用。

■ 傾向使用 TCP 的情況

- 需要可靠傳輸的應用：比如網頁瀏覽（HTTP/HTTPS）、電子郵件（SMTP/POP3/IMAP）、檔案傳輸（FTP）。
- 需要順序傳輸的應用：比如資料庫操作、遠端登錄（SSH）。

## 5. [ref](#)

- EFK 是一種用來分析和管理的工具組合，由三個開源軟體組成：Elasticsearch、Fluentd 和 Kibana。這三個工具分工合作，幫助收集、儲存、分析和視覺化日誌資料，更方便了解系統運行狀態和性能。

1. Elasticsearch 是一個分散式搜尋引擎，用於儲存和檢索大量資料。
2. Fluentd 數據收集和日誌處理系統，可以將日誌從不同來源集中到 Elasticsearch。
3. Kibana 是一個資料視覺化工具，可以將 Elasticsearch 中的資料以圖表或儀表板呈現。

- 優點:
  - EFK 可以處理大量的日誌資料，適合系上需要分析大量程式運行記錄的環境。
  - EFK可以幫助系統管理員和開發人員了解系統中發生的錯誤和故障，並找到解決問題的方法。
  - Kibana 提供許多不同類型的圖表和圖形，讓日誌資料更容易理解，甚至可以用來做數據分析。
  - 一些課程或研究可能會使用雲端或容器化技術（如 Docker、Kubernetes），EFK 可以幫助監控與除錯。
  - EFK 是免費的開源軟體，對學校或學生來說非常划算。
- 缺點與可能會造成什麼問題：
  - Elasticsearch 需要較多的記憶體和運算資源，如果日誌量太大，可能會影響系統效能。
  - 雖然 EFK 是免費的，但它的設定和維護需要時間和人力，如果沒有專人管理，可能會出現問題。
  - EFK 預設沒有強制的安全機制，如果沒有設定好，可能會被外部攻擊。

## 6. [ref1](#), [ref2](#), [ref3](#), [ref4](#), [ref5](#)

- Multiplexing（多工）是一種在單一通道上同時傳輸多個訊號或資料流的技術。透過這種方式，可以有效提升頻寬利用率，讓多個使用者或設備可以同時共享有限的通訊資源，而不互相干擾。
- 3 種：
  1. 分頻多工 (Frequency Division Multiplexing, FDM)：將可用頻寬劃分為多個頻段，每個頻段承載一個獨立的訊號。這些頻段同時傳輸，但彼此之間不會互相干擾，類似於不同的電視頻道各自佔用不同的頻率。
  2. 分時多工 (Time Division Multiplexing, TDM)：將時間劃分為多個時槽 (time slots)，每個時槽分配給不同的訊號來源。各訊號在各自的時槽內傳輸，達到共享同一傳輸媒介的目的。這種方式適用於數位訊號的傳輸。
  3. 分碼多工 (Code Division Multiplexing)：為每個訊號分配一個獨特的編碼，所有訊號同時在同一頻帶上傳輸。接收端透過識別這些編碼來分離各個訊號。
- 系上的 Wi-Fi 主要採用了正交分頻多工 (Orthogonal Frequency Division Multiplexing, OFDM) 與分時多工 (Time Division Multiplexing, TDM)。
- 為什麼這麼想
  1. 正交分頻多工 (Orthogonal Frequency Division Multiplexing, OFDM)：OFDM 是 FDM 的改進版本，將頻寬劃分為多個相互正交的子載波，每個子載波上承載不同的資料，提升頻寬效率，並且降低多重路徑干擾。
  2. 分時多工 (Time Division Multiplexing, TDM)：在多人使用 Wi-Fi 時，路由器會透過 TDM 分配時槽，讓多個裝置可以輪流傳輸數據，避免碰撞與干擾。

## 2. Command Line Utilities

1. (a) `tracert speed.ntu.edu.tw` :

```
~ took 9s
→ tracert speed.ntu.edu.tw
tracert to speed.ntu.edu.tw (140.112.5.178), 30 hops max, 60 byte packets
 1  10.200.200.200 (10.200.200.200)  2.283 ms  3.452 ms  3.525 ms
 2  ip4-126.vpn.ntu.edu.tw (140.112.4.126)  5.060 ms  5.026 ms  5.058 ms
 3  140.112.5.178 (140.112.5.178)  5.095 ms !X  5.131 ms !X  5.165 ms !X
```

(b) 1. `nslookup speed.ntu.edu.tw` :

```
~
→ nslookup speed.ntu.edu.tw
Server:                140.112.254.4
Address:                140.112.254.4#53

Name:    speed.ntu.edu.tw
Address: 140.112.5.178
```

2. `dig speed.ntu.edu.tw +short` :

```
~
→ dig speed.ntu.edu.tw +short
140.112.5.178
```

(c) private network區間：`10.0.0.0 - 10.255.255.255` , `172.16.0.0 - 172.31.255.255` ,  
`192.168.0.0 - 192.168.255.255`

- private network：節點 1：`10.200.200.200`，因為其範圍在 `10.0.0.0 - 10.255.255.255` 之間。
- public network：節點 2：`140.112.4.126` 與節點 3：`140.112.5.178`，因為其範圍不在 private network區間。

(d)

- 每個節點的三個時間值代表了向該節點發送的三個 ICMP Echo 封包的往返延遲（Round Trip Time, RTT）。如果時間差距較大，可能是節點負載較高或網路狀態不穩定。若出現 `*`，表示該節點未回應，可能由於防火牆或過濾規則。

- 否，結果並不一定越下面的延遲時間會比上面大。

- 可能導致下層延遲較大的原因

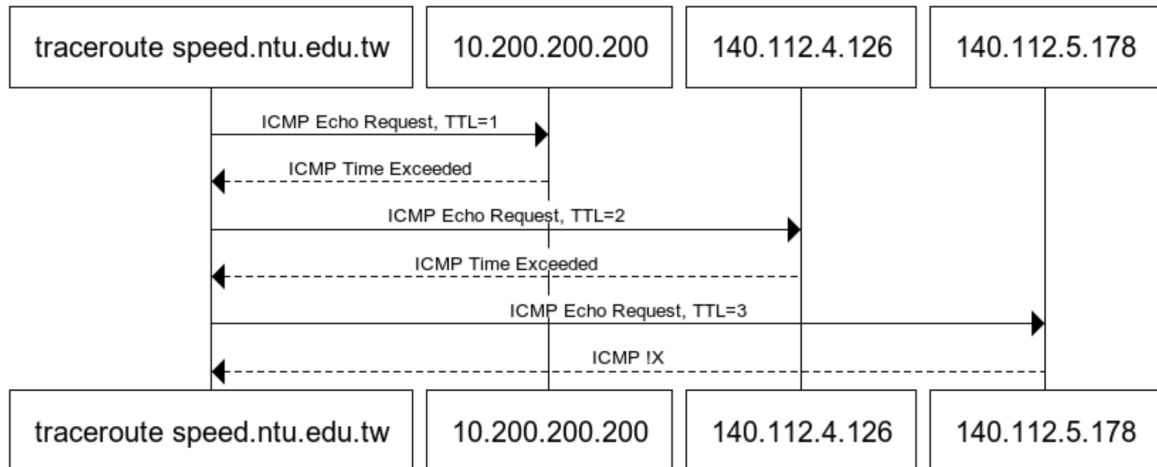
1. 節點越接近目標主機，通常代表其地理位置更遠，傳輸路徑更長，延遲時間可能增加。

2. 某些路由節點可能因負載過高而增加回應延遲時間。

■ 可能導致下層延遲較小的原因

1. 下層節點可能連接更高速或低負載的路徑，因此其延遲反而較小。
2. 下層節點可能使用了快取機制，導致測量結果更快。

(e) [ref1](#), [ref2](#), [tool](#)



ICMP !X (可能是 Administratively Prohibited，拒絕回應)

2. [ref1](#), [ref2](#), [ref3](#), collaborator: 張均豪 B11901016，賴睿廷 B12901194

(a) (1) ICMP Echo Request (ICMP回應要求)，ICMP Echo Reply (ICMP回應回覆)

(2)

```
→ ping 140.112.91.2
PING 140.112.91.2 (140.112.91.2) 56(84) bytes of data.
```

Ctrl+c 中止後 print 出 25 packets transmitted, 0 received, 100% packet loss, time 24616ms，推測似乎該伺服器的防火牆有封鎖 ICMP。

(b) `-sn` 告知 `nmap` 僅進行 `ping scan`，不進行 `port scan`

```
~
→ nmap -sn 140.112.91.2
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-10 00:56 CST
Nmap scan report for nasa2023team02.csie.ntu.edu.tw (140.112.91.2)
Host is up (0.0015s latency).
Nmap done: 1 IP address (1 host up) scanned in 0.01 seconds
```

說明: `ping` 僅發送 ICMP Echo Request，而 `nmap` 的 `-sn` 模式除了發送 ICMP Echo Request 外，還可能發送 TCP SYN 或 ACK 包，從截圖可知該伺服器開放 TCP 連線，但有封鎖 ICMP，所以 `ping` 才沒有顯示任何東西。

(c) (1) 服務名稱：HTTP (HyperText Transfer Protocol) 與服務版本：Nginx 1.26.2。Nginx



是一種高效能的 HTTP 和反向代理伺服器，也可以用作負載平衡器、郵件代理伺服器和通訊協議網關。它通常用於處理高流量網頁的請求，因為其資源使用率低且支援多種應用場景。

(2) `nmap -p 80 -sV 140.112.91.2`

```
~
→ nmap -p 80 -sV 140.112.91.2
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-10 01:00 CST
Nmap scan report for nasa2023team02.csie.ntu.edu.tw (140.112.91.2)
Host is up (0.0019s latency).

PORT      STATE SERVICE VERSION
80/tcp    open  http    nginx 1.26.2

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.17 seconds
```

(d)

```
~
→ telnet 140.112.91.2 80
Trying 140.112.91.2...
Connected to 140.112.91.2.
Escape character is '^]'.
POST / HTTP/1.1
Host: 140.112.91.2
Content-Length: 0

HTTP/1.1 200 OK
Server: nginx/1.26.2
Date: Sun, 09 Feb 2025 17:04:40 GMT
Content-Type: application/octet-stream
Content-Length: 56
Connection: keep-alive

Great, meet me at somewhere between port 48000 and 49000
^CConnection closed by foreign host.

~ took 53s
→ nmap -p 48000-49000 140.112.91.2
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-10 01:07 CST
Nmap scan report for nasa2023team02.csie.ntu.edu.tw (140.112.91.2)
Host is up (0.0015s latency).
Not shown: 1000 closed tcp ports (conn-refused)
PORT      STATE SERVICE
48763/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 1.15 seconds

~
→ telnet 140.112.91.2 48763
Trying 140.112.91.2...
Connected to 140.112.91.2.
Escape character is '^]'.
welcome to nasa!!! (screenshot me as proof for your answer)Connection closed by foreign host.
```



3. (a) 指令：`nslookup Bocchi-Tracker.csie.ntu.edu.tw` 與答案：`140.112.30.131`

```
~
→ nslookup Bocchi-Tracker.csie.ntu.edu.tw
Server:          140.112.254.4
Address:         140.112.254.4#53

Non-authoritative answer:
Name:   Bocchi-Tracker.csie.ntu.edu.tw
Address: 140.112.30.131
```

(b) 指令：`nslookup 104.112.30.131` 與答案：`Starry.csie.ntu.edu.tw`

```
~
→ nslookup 140.112.30.131
131.30.112.140.in-addr.arpa      name = Starry.csie.ntu.edu.tw.

Authoritative answers can be found from:
30.112.140.in-addr.arpa nameserver = csman.csie.ntu.edu.tw.
30.112.140.in-addr.arpa nameserver = ntuns.ntu.edu.tw.
30.112.140.in-addr.arpa nameserver = csman2.csie.ntu.edu.tw.
csman.csie.ntu.edu.tw    internet address = 140.112.30.13
ntuns.ntu.edu.tw         internet address = 140.112.254.67
ntuns.ntu.edu.tw         internet address = 140.112.254.68
csman2.csie.ntu.edu.tw   internet address = 140.112.30.14
```

(c) 指令：`nslookup -q=TXT Starry.csie.ntu.edu.tw` 與答案：`"Your guitar is in the box"`，代表說她的吉他在箱子裡

```
~
→ nslookup -q=TXT Starry.csie.ntu.edu.tw
Server:          140.112.254.4
Address:         140.112.254.4#53

Non-authoritative answer:
Starry.csie.ntu.edu.tw text = "Your guitar is in the box"

Authoritative answers can be found from:
```

(d) 指令：`nslookup -q=CNAME Bocchi.csie.ntu.edu.tw` 與答案：`GultArHer0.csie.ntu.edu.tw`

```

~
→ nslookup -q=CNAME Bocchi.csie.ntu.edu.tw
Server:      140.112.254.4
Address:     140.112.254.4#53

Non-authoritative answer:
Bocchi.csie.ntu.edu.tw canonical name = GultArHer0.csie.ntu.edu.tw.

Authoritative answers can be found from:

```

### 3. Basic Wireshark

#### 1. [ref1](#), [ref2](#)

(a) 3000

(b) 在Wireshark上顯示 Transmission Control Protocol, Src Port: 58649, Dst Port: 3000, Seq: 0, Len: 0, 代表這是一個 TCP 連線, 來源埠號 (Src Port) 為 58649, 目的埠號 (Dst Port) 為 3000。通常在 TCP 連線中, 用作伺服器的機器會監聽特定的 port, 而客戶端會使用隨機的高位元埠號進行請求, 因此 port 3000 是伺服器使用的埠號, port 58649 是客戶端臨時分配的埠號, 通常用於發起請求。

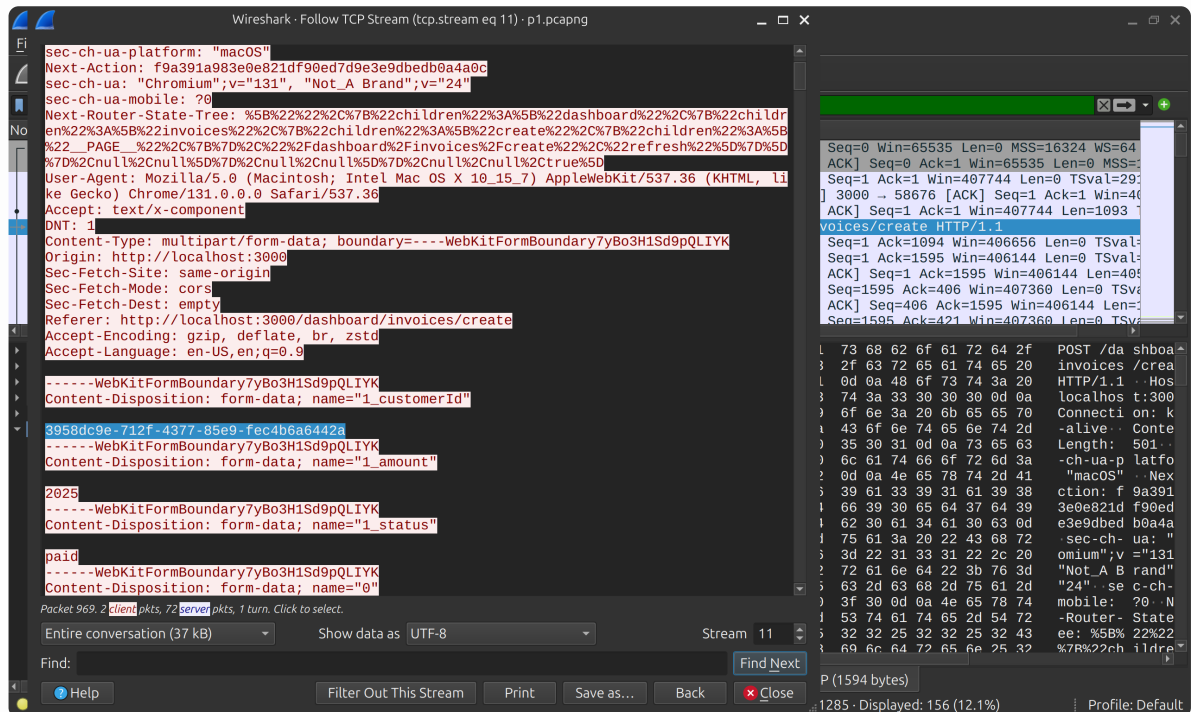
(c)



(d) (1)  $1.268 \times 10^6$  bytes/1 sec (2) 第2秒

(e) (1) 29個 (2) 使用 `http.request.method == "GET"` 篩選出HTTP GET 的 request, 並從右下方的Display讀取request數量。

(f) 3958dc9e-712f-4377-85e9-fec4b6a6442a



2. (a) 說明：


1. 點擊 `p2.cap`，開啟Wireshark

2. 在 Wireshark 中，依序點擊： `Edit -> Preferences... -> Protocols -> TLS`

3. 在 TLS 區域內點擊 `Edit`，然後點擊左下角圓形綠底白色加號"Create a new entry."

`port` 輸入443, `Protocol` 輸入 `http`，`Key File` 選擇 `p2_private.key` 路徑的檔案，其他欄位留空，並按右下角的OK

完成上述步驟後，Wireshark 將顯示解密的 HTTP 資料內容。

(b) 第31個封包 

## 4. Cryptography

ref

flag: `NASA_HWO{1_10V3_r54}`

解題過程：

1. 產生n:

```

1  from Crypto.Util.number import getPrime, inverse
2
3  e = 65537
4  l = 2048
5  p = getPrime(l)
6  q = getPrime(l)
7  n = p * q
8  phi = (p - 1) * (q - 1)
9  d = inverse(e, phi) # private key d
10
11 print(f"n = {n}\ne = {e}\nd = {d}")

```

2. 透過 `nc 140.112.91.1 48763` 來聯繫她並輸入 `n` 與 `e` 得到 encrypted message，也就是 `c`

```

> nc 140.112.91.1 48763
Hi, I'm Misumi Uika, and I have a SECRET mission for you.
To prevent any eavesdropping on this message about the SECRET mission, I'll ensure its confidentiality using RSA encryption.
First and foremost, please generate an RSA key pair consisting of a public key (n, e) and a private key (n, d) such that e=65537, n has 4096 bits, and e*d=1 (mod (p-1)*(q-1)), where n is the product of two 2048-bit prime numbers p and q. Once generated, provide me with your public key (n, e).
Note: d is the private exponent, e is the public exponent, and n is the modulus.
n: 5349595730502454301689830671658834478520854906400621987655925510134210736215039211603340835411554310361081516667450901125506001081
6339626373080567787562115080267222254693473069025380749006704747217857364370387399050907437834882628175299275434082050134798001384921
0386803147203178292022409546568963070208046542179170348753521845963664530514745875364954015639096331865624150928392126077913487380591
2626600427593442704621044532476185689544092175653693729863728953058841346830530111427221392050973715406398896127463134834657617036825
7280902648725046239436032871618788539000868856590830187888019669719063734305179809061018186100010578421655673085826542175100340219774
0055923960667188350049222778957123721982274920573671040060532114535247549430405486555738049680566960065516322854972270332790346962623
4421491066155620110907090735326908826453913767260447593996293294444987628099932675815627832179406394260434219341641442617693924042893
83594337258805197331953264572463307510146362444316118776795802960219102945058624010518743253887790208499195597299097687553951304269
3654356641290176369633736761057417212189471288523310926009022973166814911100988454028114751952800737434302122866566499774875126445501
890712868324206594804987068338243116517
e: 65537
Great! Now I'll tell you the SECRET mission, which is encrypted using the RSA public key you just gave me. You may then uncover it using your private key.
Here is the encrypted message: 315940852693389524484126497519099872270214626919503548087182091403799967081958476983162061597663370626
4709415560005357866779082100013281683170231526827851813421658260957096156384703129475586513236558883479831305501917104844654465754637
5033398183239790104730382296005242040059592011659407511997031669414628779439093586506101466049006432837443816765187740615877323368455
4243738094095426745815626145231924587749159447215826191138463494196220211054604025619652838461029078564247812781665237851877343223986
427002455072697772858434683377704164843845328679257440867956808912746695822749439922500698090551143378255531326108857707281187505273
7566973714102620025867215101797083507649381177633144730867917060929495019936566460075791370370053287892266113830402343206138466188983
3949151270539942847065164569814147190368982294422068100831328432027376412656033002746761126886450990292835784439431117390074675240518
0023037657294179393310954711095969196185603397715528089998137617123320891089150509208280506869439660651418642337374717425875114570350
9161450570064169156085383742815417257131979715736593542430629416637203257355996737373294223318638430178331674948898422043855494777742
1540695983739408318603648404604990560386007746834742927924438440316

```

3. 解密 `c` (已事先把 `c`, `d`, `n` 存到 `secret.py`) :

```

1  from Crypto.Util.number import long_to_bytes
2  from secret import c, d, n
3
4  msg = pow(c, d, n)
5  flag = long_to_bytes(msg).decode()
6
7  print(f"Flag: {flag}")

```

4. 得到結果：

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL ... Python + - [icon] [icon] [icon] [icon] [icon] [icon] [icon]
code/NASA/4_Cryptography via v3.12.2
→ /home/joe/anaconda3/bin/python /home/joe/code/NASA/4_Cryptography/decryption.py
Flag: Someone purposely gave Anon wrong information about RSA, making her use a
bad RSA public key to encrypt her private data! The clues about the identity of
the perp are hidden in Anon's secret diary. To curb the spread of this misinform
ation ASAP, please decrypt Anon's diary and send it to me. I'll figure out who t
he perp is. Oh, by the way, here is the flag for you: NASA_HW0{1_10V3_r54}

```

5. 為什麼簽不了憑證???

[ref1](#), [ref2](#)

1. subnet (子網路) 是網路內的網路，他讓網路更有效率。透過子網路，網路流量傳播距離更短，無需通過不必要的路由器即可到達目的地。
2. gateway (閘道器) 是連接兩種不同協定網路的裝置，以及將 A 協定網路的資料訊息進行運算、轉換成可被 B 協定網路識別的訊號，就需要利用來進行資料傳輸。
3.  $X \rightarrow \text{WAN} \rightarrow A1$  (140.112.28.59/24)
4. 當 A 想送封包到 X 時，查路由表發現 X 不在任何直接連接的網段內，因此會將封包送到 default gateway 172.16.0.1，所以封包會從  $A2 \rightarrow \text{LAN} \rightarrow B2 \rightarrow B \rightarrow B1 \rightarrow \text{WAN} \rightarrow X$ 。
5. Stateful 防火牆會追蹤連線狀態，確保封包符合已建立的連線，而 Stateless 防火牆則單獨檢查每個封包。Stateful 防火牆比較可能擋住 TCP ACK without SYN，因為它會檢查連線是否有正確的 SYN 建立。
6. A 的回應封包從 A1 送出，但 A 的預設 gateway 是 B2 (172.16.0.1)，所以 A2 會接管對外流量，導致 TCP 連線斷裂，B 作為 stateful 防火牆拒絕  $X \rightarrow A1$  的封包，因為它沒看到正確的 SYN。
7.
  - 停止服務並確保所有請求處理完畢並關閉相關進程 (5-10 分鐘)
  - 申請新憑證、安裝至伺服器並測試 HTTPS 是否正常運作 (10-15 分鐘)
  - 重新啟動 Web 伺服器並確認系統正常運行 (5-10 分鐘)
  - 總時間：大約 30-45 分鐘，若中途發現有其他問題則可能更久
8. 修改 A 的 default gateway 為 A1，確保回應封包從同一個介面出去
9. 準備好了

## System Administration

---

### 6. btw I use arch

[ref1](#), [ref2](#), [ref3](#), [ref4](#), [ref5](#)

6-0. 敘述：

1. 下載[安裝檔](#)
2. 用 `sha256sum archlinux-2025.01.01-x86_64.iso` 檢查 `sha256 checksum` 是否為  
`74b109b4b36d20bef8f4203e30b8d223e0ab297a09d1a1213a02894472aa530a`
3. 依自己筆電的os下載[VirtualBox](#)
4. 打開Virtual Box，點擊按鈕New，
  - Name: Arch Linux
  - ISO Image 選擇 `archlinux-2025.01.01-x86_64.iso`
  - Type: Linux
  - SubType: ArchLinux
  - Version: ArchLinux(64-bit)

- Base Memory: 4096 MB
- Processors: 2 CPU
- Hard Disk File Location and Size從預設8.00 GB改成25.00GB

好了之後點Finish

5. 點擊Settings，點擊左側欄位Network，更改Attached to成Bridged Adapter，好了之後右下角按OK
6. 點擊Start開啟Arch linux，選擇Arch Linux install medium (x86\_64, BIOS)按Enter進入安裝環境
7. `ping -c 3 archlinux.org` 確認網路連線

```
root@archiso ~ # ping -c 3 archlinux.org
PING archlinux.org (95.217.163.246) 56(84) bytes of data.
64 bytes from archlinux.org (95.217.163.246): icmp_seq=1 ttl=46 time=323 ms
64 bytes from archlinux.org (95.217.163.246): icmp_seq=2 ttl=46 time=262 ms
64 bytes from archlinux.org (95.217.163.246): icmp_seq=3 ttl=46 time=323 ms

--- archlinux.org ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 261.770/302.720/323.213/28.956 ms
```

8. `timedatectl` 確認 System clock synchronized 是否為 yes

```
root@archiso ~ # timedatectl
          Local time: Sun 2025-02-16 06:47:50 UTC
          Universal time: Sun 2025-02-16 06:47:50 UTC
              RTC time: Sun 2025-02-16 06:47:50
              Time zone: UTC (UTC, +0000)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

9. 查看磁碟: `lsblk`，可以看到 `sda` (虛擬硬碟)

```
root@archiso ~ # lsblk
NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
loop0  7:0    0 820.6M  1 loop /run/archiso/airootfs
sda     8:0    0   25G   0 disk
sr0    11:0    1   1.1G   0 rom  /run/archiso/bootmnt
```

10. 進入 `fdisk`: `fdisk /dev/sda`

11. 建立分割區:

- 建立 `/` (15GB)

```
n # 新增分割區
ENTER # 預設 primary partition
ENTER # Partition number (default 1)
ENTER # First sector (default)
+15G # 15GB
```

- 建立 `/home` (5GB)

```
n
ENTER
ENTER
ENTER
+5G
```

- 建立 Swap

```
n
ENTER
ENTER
ENTER
ENTER
```

- 設定 Swap 類型

```
t
3 # 選擇第三個分割區
82 # Linux Swap
```

- 寫入變更並退出 fdisk: `w`

## 12. 格式化分割區

```
mkfs.ext4 /dev/sda1
mkfs.ext4 /dev/sda2
mkswap /dev/sda3
swapon /dev/sda3
```

## 13. 掛載分割區

```
mount /dev/sda1 /mnt
mkdir /mnt/home
mount /dev/sda2 /mnt/home
```

- 14. `sudo reflector --country Taiwan --protocol https --save /etc/pacman.d/mirrorlist` 更新 `/etc/pacman.d/mirrorlist`

- 15. 確保 `pacman` 更新 package lists: `pacman -Syyu`

- 16. `pacstrap -K /mnt base linux linux-firmware` 安裝基本系統

- 17. 產生 `fstab` :

```
genfstab -U /mnt >> /mnt/etc/fstab
cat /mnt/etc/fstab # 確保 UUID 設定正確
```



```

root@archiso ~ # cat /mnt/etc/fstab
# Static information about the filesystems.
# See fstab(5) for details.

# <file system> <dir> <type> <options> <dump> <pass>
# /dev/sda1
UUID=8942a458-7405-44a2-966f-721a3a34c947      /      ext4      rw,relatime    0 1

# /dev/sda2
UUID=364e992c-2bbd-4be1-bf10-314a8c39a982      /home   ext4      rw,relatime    0 2

# /dev/sda3
UUID=9a23103b-768e-4f90-9a27-9742f6527138      none    swap      defaults       0 0

```

18. `arch-chroot /mnt` 切換到新系統

19. 設定時區:

```

ln -sf /usr/share/zoneinfo/Asia/Taipei /etc/localtime
hwclock --systohc

```

20. 設定語言:

```

echo "en_US.UTF-8 UTF-8" >> /etc/locale.gen
locale-gen
echo "LANG=en_US.UTF-8" > /etc/locale.conf

```

21. 設定主機名稱: `echo "b11901164" > /etc/hostname`

22. 設定 root 密碼: `passwd`

23. 建立使用者 nasa:

```

useradd -m -G wheel -s /bin/bash nasa
passwd nasa

```

24. 啟用 `sudo` , 取消 `# %wheel ALL=(ALL:ALL) ALL` 前的 `#`

```

pacman -S vim sudo
EDITOR=vim visudo

```

25. 安裝grub: `pacman -S grub`

26. 安裝 Bootloader:

```

grub-install --target=i386-pc /dev/sda
grub-mkconfig -o /boot/grub/grub.cfg

```

27. 退出 `chroot` : `exit`

28. 卸載並重啟:

```

umount -R /mnt
reboot

```

29. 重啟後選擇Boot existing OS進入GNU GRUB並選擇Arch Linux , 登入 nasa 帳號

6-1. 

```
[nasa@b11901164 ~]# cat /etc/hostname
b11901164
```

6-2. 機器中各分割的 uuid:

```
[nasa@b11901164 ~]# sudo blkid
/dev/sr0: BLOCK_SIZE="2048" UUID="2025-01-01-08-45-10-00" LABEL="ARCH_202501" TYPE="iso9660" PTUUID="d736165e" PTTYPE="dos"
/dev/sda2: UUID="364e992c-2bbd-4be1-bf10-314a8c39a982" BLOCK_SIZE="4096" TYPE="ext4" PARTUUID="095f7299-02"
/dev/sda3: UUID="9a23103b-768e-4f90-9a27-9742f6527138" TYPE="swap" PARTUUID="095f7299-03"
/dev/sda1: UUID="8942a458-7405-44a2-966f-721a3a34c947" BLOCK_SIZE="4096" TYPE="ext4" PARTUUID="095f7299-01"
```

磁碟空間分配:

```
[nasa@b11901164 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
dev              2.0G   0    2.0G   0% /dev
run              2.0G  608K   2.0G   1% /run
/dev/sda1        15G   2.0G   12G   14% /
tmpfs            2.0G   0    2.0G   0% /dev/shm
tmpfs            1.0M   0    1.0M   0% /run/credentials/systemd-journald.service
tmpfs            2.0G   0    2.0G   0% /tmp
/dev/sda2        4.9G   1.3M   4.6G   1% /home
tmpfs            1.0M   0    1.0M   0% /run/credentials/getty@tty1.service
tmpfs            392M   4.0K   392M   1% /run/user/1000
```

6-3. Linux Distribution:

```
[nasa@b11901164 ~]# cat /etc/os-release
NAME="Arch Linux"
PRETTY_NAME="Arch Linux"
ID=arch
BUILD_ID=rolling
ANSI_COLOR="38;2;23;147;209"
HOME_URL="https://archlinux.org/"
DOCUMENTATION_URL="https://wiki.archlinux.org/"
SUPPORT_URL="https://bbs.archlinux.org/"
BUG_REPORT_URL="https://gitlab.archlinux.org/groups/archlinux/-/issues"
PRIVACY_POLICY_URL="https://terms.archlinux.org/docs/privacy-policy/"
LOGO=archlinux-logo
```

kernel 版本:

```
[nasa@b11901164 ~]# uname -r
6.13.2-arch1-1
```

## 7. Flag Hunting

[ref](#)

1. (a) `.bash_history`
- (b) `echo $HISTSIZE : 1000`
- (c) `echo $HISTFILESIZE : 2000`

(d) NASA{y0UF1nd+heCoRr3tFL4G}

```
nasa@nasa:~$ cat .bash_history
./gen_flag --line 104 --out new_history_file
exit
nasa@nasa:~$ echo $HISTFILE
/home/nasa/kickstart.nvim/.git/logs/refs/remotes/origin/HEAD
nasa@nasa:~$ sed -n "104p" $HISTFILE
echo NASA{y0UF1nd+heCoRr3tFL4G}
nasa@nasa:~$
```

2. NASA{EZ\_TrEa\$Ur3\_HunT!}

```
nasa@nasa:~/treasure-chest$ ls -ls | tail -n 1
-rw-rw-r-- 1 nasa nasa 1971706 Feb 11 09:52 flag-922
nasa@nasa:~/treasure-chest$ sed -n '1218p' flag-922
NASA{EZ_TrEa$Ur3_HunT!}bPnrAgZzkWRI{z72jap2kYqeKR2RPjQF
E4Iw}Q6RfPM}px}3rt4FmMOC9PKdZALXQV4IqxGRQaApXPH2lOSrtpt
zlewcKdJR5cgjqrPfZI4RU_EW3}U9pcnLFS9y8Gs7EVmy74XSULXMOr
LESYFeHqsWJ}4CiJ2UQGlgigbCXwnJ{EHAiudrWDuFNim8SWQSQn{}}2
X0ui9yqJZEJ6oilCVKHaGv0QIzzQjLBPI{mAtFu9Vm13DkDz8hGSrX}
acR2lQqiGMAbzF78cVhr_r0jfxfrCzb9s_5BYV2XvIDrXWwIqQnr1jH
Tm5whDpqUa0I9beNYw9KYgBj{N8FQFJT5UkopP{ujDmx0}
nasa@nasa:~/treasure-chest$
```

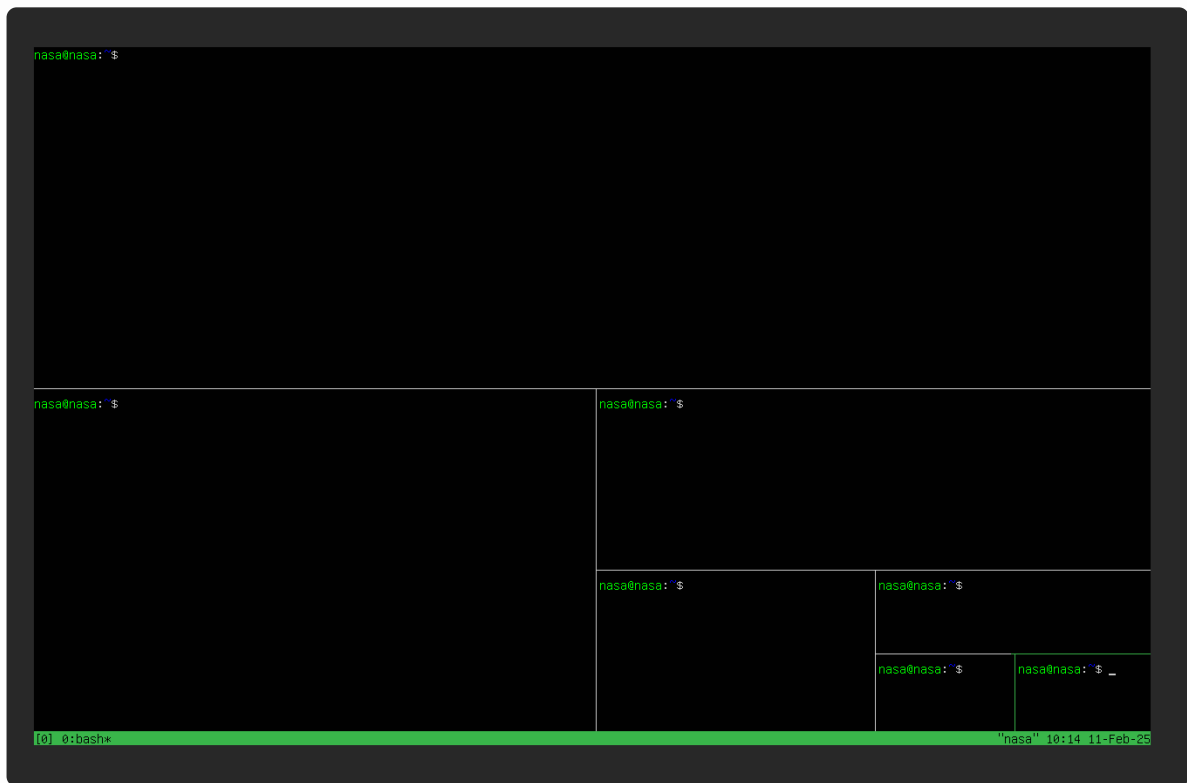
3. NASA{m0dERn\_Pr0B1em\$\_reQU1r3\_m0dERn\_S0luT10N5}

<pre>nasa@nasa:~\$ /home/nasa/boss If you can kill all my subprocesses within 3 seconds, I will show you my secret! Well done! NASA{m0dERn_Pr0B1em\$_reQU1r3_m0dERn_S0luT10N5} nasa@nasa:~\$</pre>	<pre>nasa@nasa:~\$ cat k.sh #!/bin/bash  boss_pid=\$(pgrep -f boss)  if [ -n "\$boss_pid" ]; then     for pid in \$boss_pid; do         pkill -P \$pid     done fi nasa@nasa:~\$ ./k.sh nasa@nasa:~\$</pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4. NASA2025{n4ndeharuh1ka93yatt4n0}

```
nasa@nasa:~$ strings ./chal | grep "486" | grep "re02"
ioewe3h486hu5tnjdsre029y814mmq
nasa@nasa:~$ ./chal
Usage: ./chal <passcode>
nasa@nasa:~$ ./chal ioewe3h486hu5tnjdsre029y814mmq
Here is the flag: NASA2025{n4ndeharuh1ka93yatt4n0}
```

5. 流程 : Ctrl+a " -> Ctrl+a % -> Ctrl+a " -> Ctrl+a % -> Ctrl+a " -> Ctrl+a %



因為在 `.tmux.conf` 內 `unbind C-b` (Ctrl+b), 改成 `C-a` , 也就是Ctrl+a。

## 8. NASA 國的大危機

[ref1](#), [ref2](#), collaborator: 張均豪 B11901016

8-1. 說明 :

1. `FROM` : 指定了基礎映像，所有 Docker 容器都是從某個基礎映像構建的
  2. `RUN` : 用於在構建映像過程中執行命令。
  3. `COPY` : 將本地文件或目錄複製到容器內。
  4. `CMD` : 容器啟動時執行的命令。
- `FROM python:3.9-slim` : 指定基礎映像為 `python:3.9-slim` , 這是一個較精簡版的 `Python` 3.9 映像，適合用於輕量化的容器應用。
  - 安裝必要工具 ( `RUN` ) :

```
RUN apt-get update && apt-get install -y \  
    build-essential \  
    libssl-dev \  
    net-tools \  
    iproute2 \  
    tcpdump \  
    tshark \  
    nano \  
    curl \  
    wget \  
    vim \  
    less \  

```

```
procps \  
lsof \  
iputils-ping \  
&& rm -rf /var/lib/apt/lists/*
```

- 建立 `/usr/libexec/run` 目錄，可能用來存放執行腳本或其他必要檔案：`RUN mkdir -p /usr/libexec/run`

- 複製檔案 (COPY)

```
COPY usr/libexec/run/dist/transfer /usr/libexec/run/transfer  
COPY usr/libexec/run/run.sh /usr/libexec/run/run.sh
```

- COPY 指令將主機內的 `usr/libexec/run/dist/transfer` 和 `usr/libexec/run/run.sh` 複製到容器內的 `/usr/libexec/run/` 目錄。

- 設定檔案執行權限 (RUN)

```
RUN chmod +x /usr/libexec/run/transfer  
RUN chmod +x /usr/libexec/run/run.sh
```

- `chmod +x`：讓 `transfer` 和 `run.sh` 有執行的權限，讓他們可以執行。

- 設定容器啟動時執行的指令 (CMD)

```
CMD ["/usr/libexec/run/run.sh"]
```

- CMD 指定容器啟動時執行 `/usr/libexec/run/run.sh`。

## 8-2. 以下為喚醒步驟：

1. `cat ./mystic-cup/usr/libexec/run/run.sh` 發現，`"$MAGIC_SPELL=" = "hahahaiLoveNASA"`
2. 執行 `docker run -it -e MAGIC_SPELL="hahahaiLoveNASA" my-magic-cup` 發現 `[transfer.py] Sent data from 5000=>6000`

## 8-3. 接續8-2.：

1. `Ctrl + c` 退出，再 `docker run -it -e -d MAGIC_SPELL="hahahaiLoveNASA" my-magic-cup`，讓他執行在背景
2. `docker ps` 找到 CONTAINER ID 為 `eddcef30e0f7`
3. `docker exec -it eddcef30e0f7 /bin/bash` 進入容器內部
4. 執行 `tcpdump -i any udp port 6000 -A` 就出現 `flag[I'll send our killer on 3948/02/22]`

所以殺手入侵的日期為3948/02/22

