**資訊工程學系**

**114學年度專題製作期初報告**

**題目：結合 Tor 與 SOCKS5 Proxy 的動態身份隱藏系統開發**

指導老師： 教授

組長:高恩在  
組員:

**中華民國114年8月23日**

目錄

一、 摘要

二、 研究背景與目的

三、 專題內容

四、 研究方法

五、 流程圖

六、 預計完成之工作項目及具體成果

七、 工作分配與預定進度甘梯圖

八、 遭遇問題與解決方案

九、 儀器設備需求表

十、 參考資料

一、摘要

本專題旨在設計並實作一套結合 Tor 匿名網路與 SOCKS5 Proxy 的自動化身份隱藏系統。透過 Python 與 Tor 控制端口的整合，本系統可每隔固定時間（如7秒）向 Tor 發送 NEWNYM 指令以切換新的出口節點，進而達到自動更換 IP 的效果。配合 SOCKS5 代理設定與 requests 套件的代理功能，使用者能即時驗證自身出口 IP 的變化。本系統具有高可控性、自動化與匿名性，適合應用於資安測試、反追蹤通訊等領域。

二、研究背景與目的

研究背景

隨著網際網路的普及，使用者在進行通訊與資料交換時，對於 隱私保護與匿名性 的需求不斷提升。特別是在 資安測試、爬蟲應用、反追蹤研究 等領域，單一固定的 IP 位址容易暴露使用者身份，導致被封鎖、追蹤或監控。

在眾多隱匿技術中，VPN 與 Tor 網路 是常見的兩種方案：

* VPN（Virtual Private Network）：透過集中式伺服器中繼，提供加密與 IP 隱藏。然而，其缺點是伺服器由單一業者掌控，存在 中心化風險，若伺服器被監控或洩漏，使用者身份仍可能暴露。
* Tor（The Onion Router）：透過多層節點中繼（onion routing）實現分散式匿名性，出口節點動態且分佈全球，具備比 VPN 更高的 去中心化與匿名強度。但 Tor 在預設設定下，出口 IP 不會頻繁更換，對於需 短時間多次更換身份 的應用並不理想。

另一方面，SOCKS5 Proxy 是一種廣泛使用的代理協定，能夠中繼 TCP 流量並支援多種應用程式。Tor 內建支援 SOCKS5，若能將 Tor 與 SOCKS5 Proxy 結合，便能提供 彈性、可控且高度匿名的代理機制。

| **特性項目** | **obfs4** | **meek** | **VPN** | **純 Tor（無 obfs）** |
| --- | --- | --- | --- | --- |
| **目標用途** | 對抗封鎖，隱藏 Tor 流量特徵 | 偽裝成 CDN 流量（如 Google、Azure）來繞過審查 | 建立一般匿名通道 | 匿名通信與暗網存取 |
| **是否依賴橋接器（Bridge）** | 是，需手動或自動取得 obfs4 bridge | 是，但自動透過 meek 可用 bridge | 否 | 否 |
| **流量混淆技術** | 加密與時間隨機化、反特徵分析（Pluggable Transport） | 偽裝成 HTTPS 請求至大型網站（cloudfront、azure） | 加密封包（SSL/TLS），但不隱藏 Tor 特徵 | 無混淆，Tor 封包特徵易被 DPI 掃描偵測 |
| **封鎖繞過能力** | 極強，適合中國、伊朗等高封鎖地區 | 極強，但速度慢，使用大量 CDN 資源 | 中等，容易被 DPI 或 IP 封鎖 | 易被偵測、封鎖 |
| **速度與延遲** | 中等（比 meek 快） | 慢（因頻繁使用 CDN proxy） | 快（取決於供應商） | 中等 |
| **出口 IP 是否固定** | 不固定，隨 Tor 網路更換 | 不固定，隨 Tor 網路更換 | 可固定出口（部分 VPN 支援） | 不固定 |
| **設置難度** | 中高（需安裝 obfs4proxy、設定 torrc） | 中（需支援 meek 的 bridge，流量可能成本較高） | 易（圖形化介面） | 簡單（安裝 Tor 即可） |
| **IP 匿名性強度** | 高 | 高 | 中等（VPN 商可能記錄 Log） | 高 |

**備註：**

* meek 通常已預設在 Tor Browser 中提供支援，但需 CDN 資源，速度最慢。
* obfs4 是目前最推薦使用的混淆協議，配合 bridge 使用即可有效躲避封鎖。
* VPN 在國家審查下會被 DPI 偵測協定特徵，無法躲過深度封包分析。

研究目的

本專題旨在開發一套 結合 Tor 與 SOCKS5 Proxy 的動態身份隱藏系統，其研究目的如下：

1. 學習與掌握 Tor 控制端口（ControlPort）操作
   * 理解 Tor 控制命令（如 NEWNYM）與身份更換流程。
2. 建立自動化 IP 變更模組
   * 透過 Python 與 Tor 整合，能定時（如每 7 秒）自動更換出口 IP，模擬「動態身份切換」。
3. 驗證匿名性效果
   * 使用 SOCKS5 Proxy 與外部 API（如 httpbin.org/ip）確認出口 IP 是否確實更換。
4. 強化安全性與可重複性
   * 加入錯誤處理與日誌輸出，確保系統能在不同環境下穩定運行。
5. 提供資安研究基礎
   * 可作為日後進一步研究（如匿名爬蟲、滲透測試、反追蹤通訊）的基礎模組，並與 VPN 等技術進行比較，分析其優缺點。

三、專題內容

本專題主要聚焦於 Tor 匿名網路 與 SOCKS5 Proxy 的整合應用，並以 Python 撰寫自動化身份隱藏系統。內容可分為三個層級：

(一) 底層技術整合

* Tor 控制端口設定與身份更換
  + 透過 stem 套件連接 Tor 控制端口 (ControlPort 9051)，發送 Signal.NEWNYM 指令進行身份切換。
* SOCKS5 Proxy 流量導向
  + 使用 socks5h://127.0.0.1:9050 將應用程式流量導入 Tor 網路。

(二) 中層功能實作

* IP 取得與驗證模組
  + 使用 requests[socks] 向外部 API（如 httpbin.org/ip）查詢出口 IP。
  + 加入 IP 變更比對，確認身份是否切換成功。
* 自動化腳本 ghost\_mode3.py
  + 以迴圈方式每 7 秒執行一次身份更換與驗證。
  + 加入錯誤處理（如 IP 無法取得、Tor 控制失敗等）。

(三) 上層使用者介面與展示

* 輸出結果美化
  + 使用 pyfiglet 顯示標題 GHOST MODE。
  + 搭配 colorama 提供彩色輸出，增強可讀性。
* 時間與狀態提示
  + 顯示身份切換時間、出口 IP 與狀態訊息。

四、研究方法

1. 虛擬環境建立：使用 Python venv 建立乾淨環境 tor-env
2. Tor 配置：修改 /etc/tor/torrc 加入

ControlPort 9051

CookieAuthentication 1

CookieAuthFileGroupReadable 1

1. Tor 控制驗證：使用 stem 登入控制端口並發送 Signal.NEWNYM
2. IP 取得與驗證：使用 requests 透過 SOCKS5 proxy 查詢 httpbin.org/ip
3. 持續執行與 IP 比對：每次換 IP 後確認是否與上一次不同
4. 問題排查與修正：包含 Python 模組缺失（PySocks）、權限不足、torrc 設定錯誤等

五、流程圖

**實驗流程圖**

+---------------————----------+

| 啟動 ghost\_mode3.py 腳本 |

+-------------------————------+

|

v

+----------------—--------+

| 連接 Tor 控制端口 9051 |

+------------———----------+

|

v

+----------------—---------+

| 傳送 Signal.NEWNYM 指令 |

+-------------------—------+

|

v

+---------------------------+

| 透過 SOCKS5 查詢目前 IP |

+---------------------------+

|

v

+--------------------—-----+

| 顯示結果，等待7秒循環 |

+----------———-----------+

**ghost\_mode3.py 系統架構圖**

┌──────────────┐

│ 啟動 ghost\_mode3.py │

└───-──┬─────-──┘

│

V

┌─────────────┐

│ 透過 Stem 套件控制 Tor │

│ controller.authenticate()│

└────-─┬──-────┘

│

V

┌────────────-───┐

│ 每隔 N 秒發送 NEWNYM 指令 │

│ controller.signal(NEWNYM) │

└─────-┬─────────┘

│

V

┌────────-──────────┐

│ 使用 requests[socks] 發送 HTTP 請求 │

│ → 通過 socks5h://127.0.0.1:9050 │

│ → 查詢 check.torproject.org │

└────-─┬────────────┘

│

V

┌─────────────────┐

│ 收到新的出口 IP，存入 IP 清單 │

│ 並計算： │

│ → 更換時間（間隔） │

│ → 成功率與失敗率統計 │

└────---┬───────---─┘

│

V

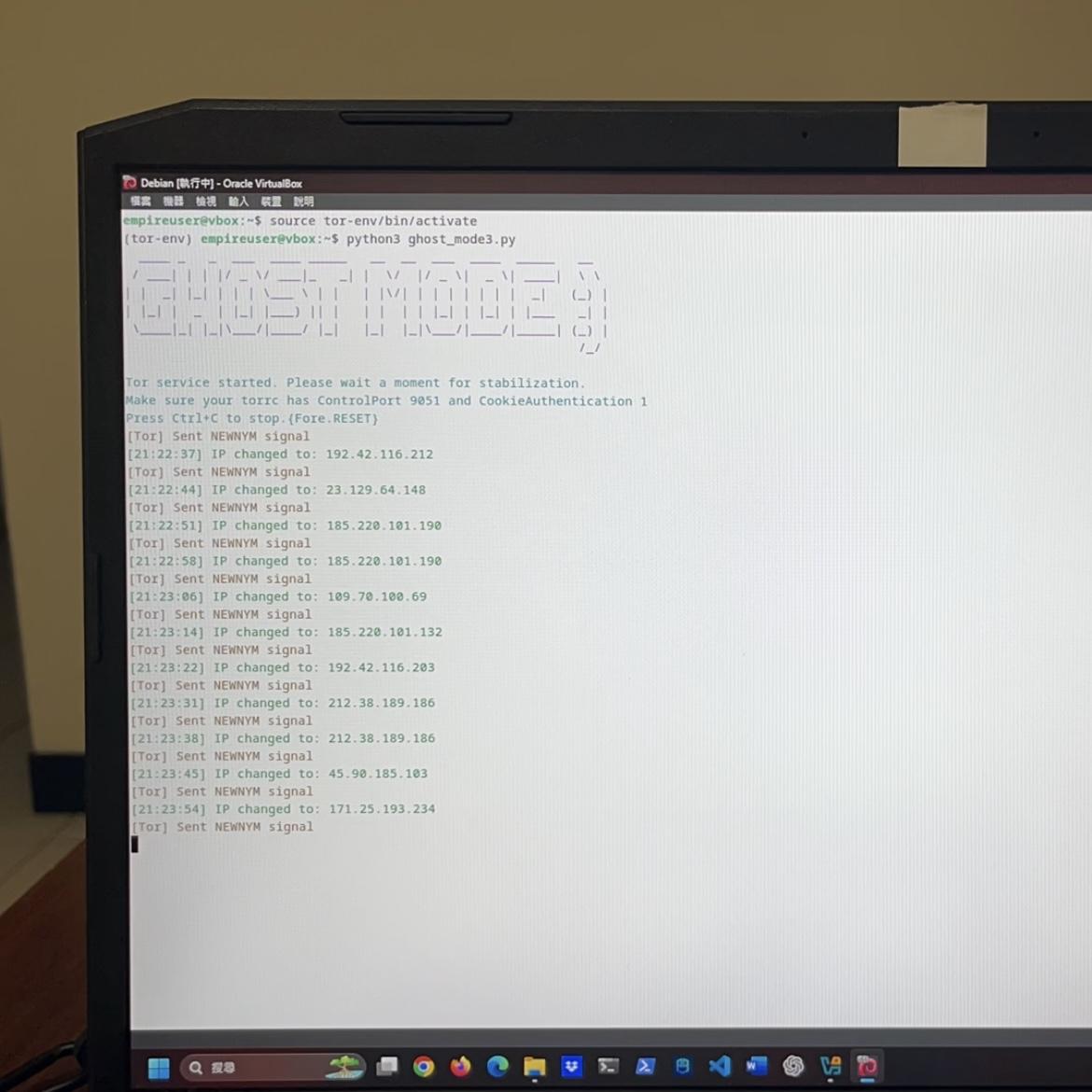
┌────────────┐

│ 最終輸出結果統計與展示 │

└────────────┘

六、預計完成之工作項目及具體成果

| 項目 | 預期成果 |
| --- | --- |
| Tor 控制連線測試 | 確保 controller.authenticate() 成功 |
| IP 查詢模組 | 成功透過 socks5 取得出口 IP |
| ghost\_mode3.py 腳本 | 不間斷執行，7秒換 IP |
| IP 變更判斷 | 可檢查是否成功更換身份 |
| 結果輸出美化 | pyfiglet 輸出、色彩提示、時間戳記 |

實驗圖片

**實機執行 ghost\_mode3.py 與現場 IP 變換**

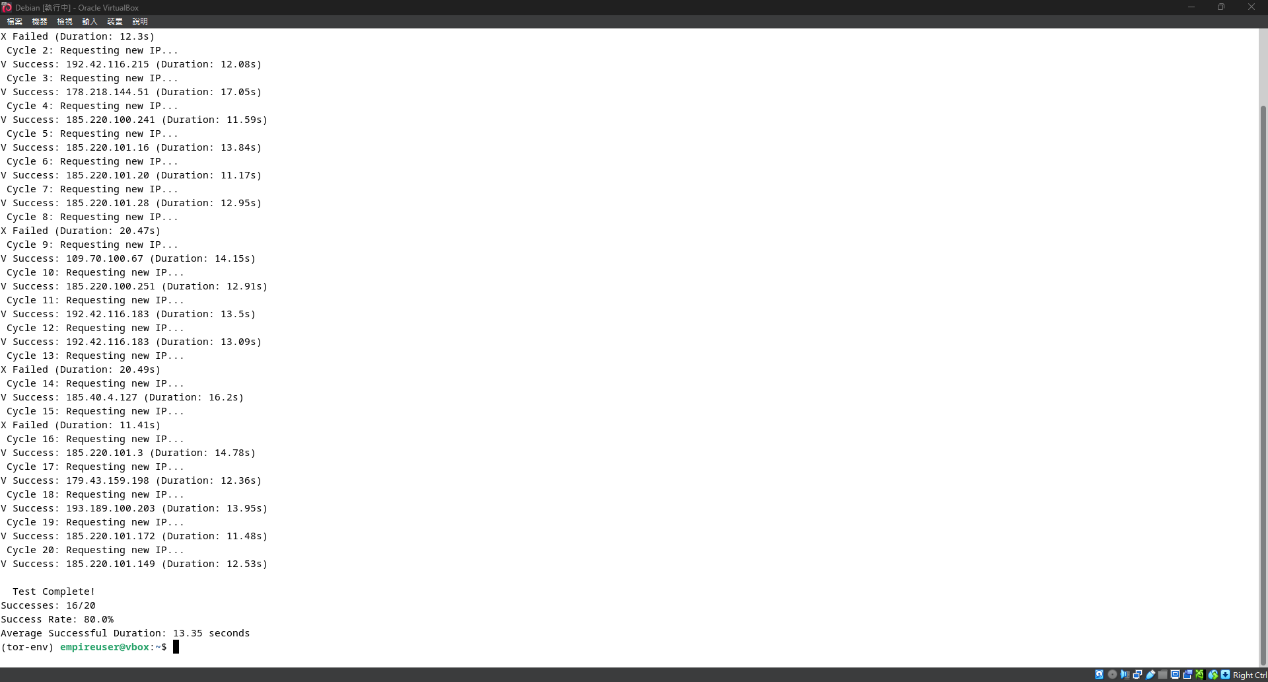
這是筆電實機運作截圖，畫面顯示：

* GHOST MODE :) ASCII 標誌，代表啟動 ghost\_mode。
* 每當 IP 變換時，畫面即時顯示 IP changed to ...。
* 使用者可即時觀察 IP 隱匿過程與時間戳



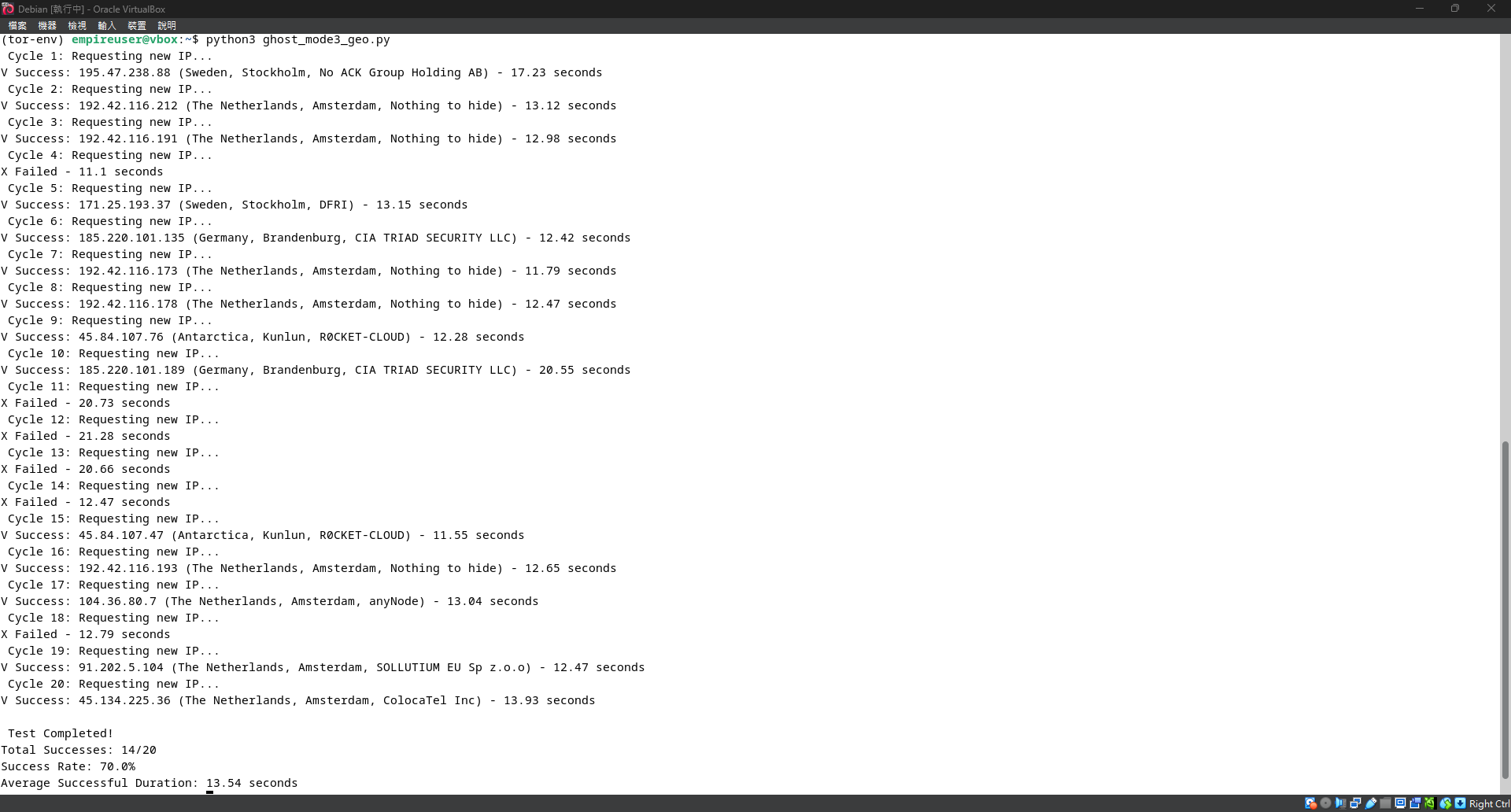
**這張是執行 ghost\_mode3\_data.py 的畫面，與 geo.py 不同，它偏向純統計用途：**

* **顯示每次換 IP 的成功與失敗時間（秒數）。**
* **平均成功時間為 13.35 秒，總成功 16 次。**



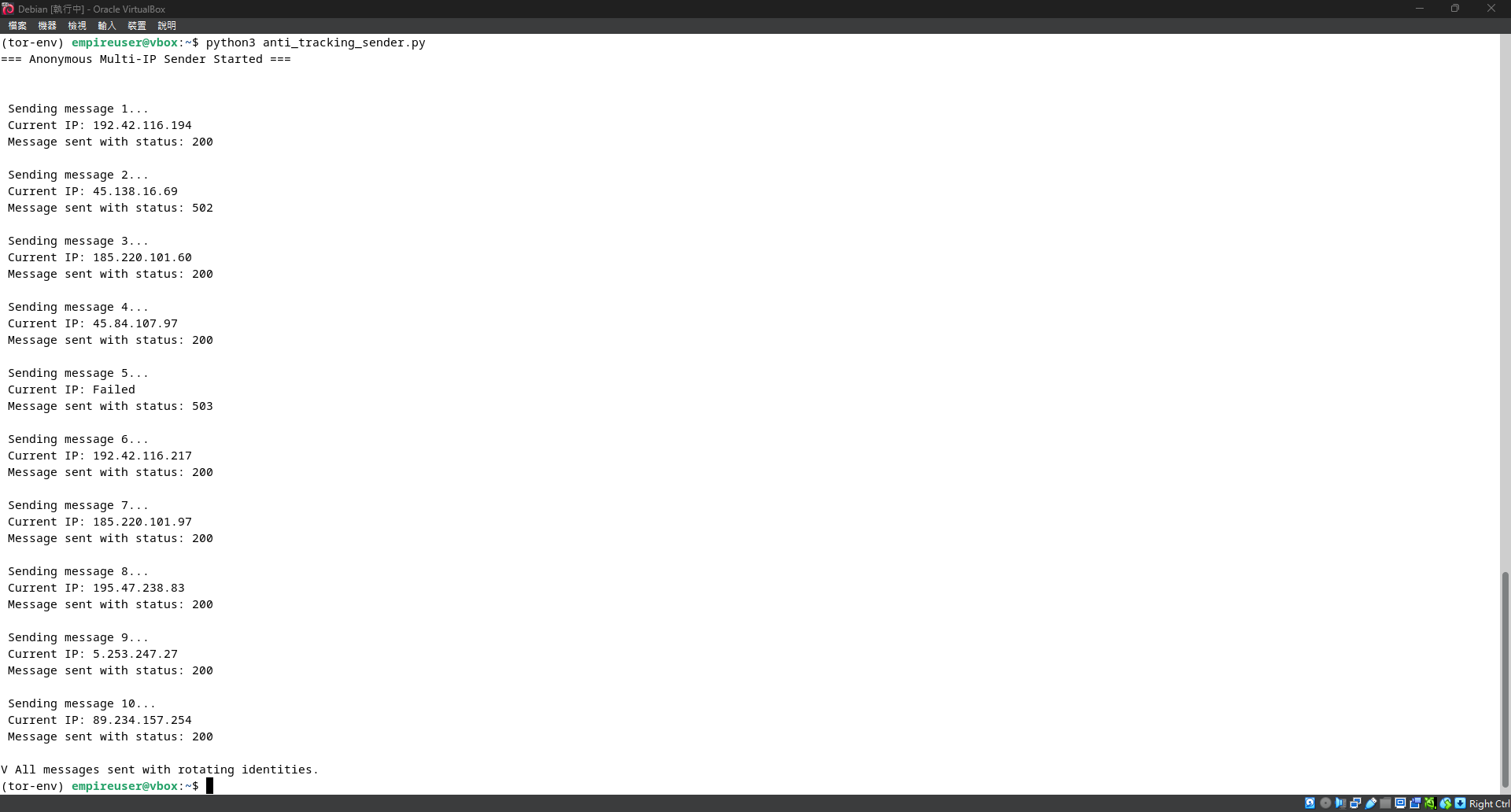
**這是一組 ghost\_mode3\_geo.py 執行結果：**

* **總共測試 20 次，其中 16 次成功，成功率為 80%。**
* **顯示了 IP 地址、城市、ISP（例如：Nothing to hide、ROCKET-CLOUD）與平均時間。**



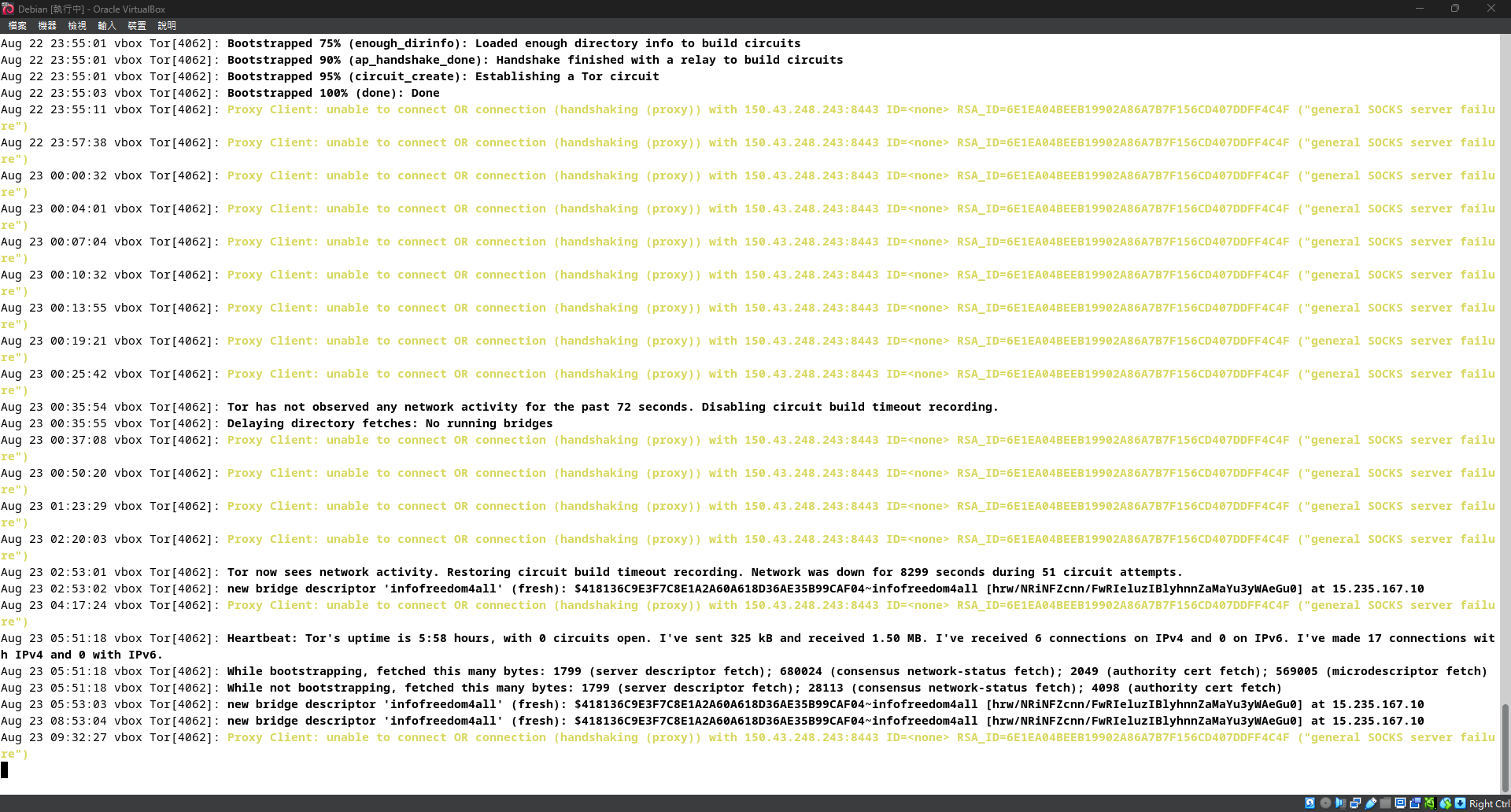
這是執行 ghost\_mode3\_geo.py 的畫面，顯示每次換 IP 的地理資訊與耗時：

* IP 會對應到國家（例如：瑞典、德國、荷蘭等），代表成功使用 Tor 網路取得新身份。
* 每次請求平均約 13 秒，成功率 70%。



這是執行 anti\_tracking\_sender.py 的畫面：

* 腳本功能是：**每次換 IP 後透過 Tor 發送 HTTP request**，模擬使用者匿名傳送訊息的情境。
* 成功的狀態碼是 200，失敗的狀態碼如 502、503 表示伺服器無法處理請求，可能是 IP 被列入黑名單或匿名性過高。



這張是 Tor 使用 obfs4 橋接器時的錯誤日誌畫面：

* **Proxy Client: unable to connect (handshaking (proxy))**：  
  表示你的 Tor client 無法與 obfs4 橋接節點完成握手，這通常是 obfs4 proxy 被封鎖、節點掛掉或網路不通。
* **Disabling circuit build timeout recording**：  
  表示 Tor 一直無法建立有效 circuit，因此停止嘗試建立。
* **Heartbeat: Tor's uptime is 5:58 hours... 0 circuits open.**：  
  表示目前 Tor 雖然在運行，但沒有成功建立任何匿名通道（0 circuits）。

**額外實作：obfs4 混淆協議與實作過程**

**研究背景**

在某些高審查網路環境中（如學校、企業、特定國家），Tor 連線會被 DPI（Deep Packet Inspection）或封包特徵辨識系統攔截。此時，即便是設定 Proxy 與換 IP，仍可能無法成功建立連線。  
為了解決此問題，Tor 團隊設計了一種「可插拔傳輸協議（Pluggable Transports）」，其中最常見且穩定的為 **obfs4（Obfuscation 4）**。

obfs4 透過加密與偽裝技術，使流量無法被辨識為 Tor，進而躲避封鎖。

**11obfs4 設計原理**

obfs4 的核心功能：

* 偽裝成亂數封包，無協定特徵（不像 HTTPS 或 TLS）
* 支援「橋接器（Bridge）」繞過公開節點封鎖
* 搭配 iat-mode（inter-arrival time）進一步混淆封包節奏
* 採用一對一的驗證通道（client 需要 cert）

**11安裝 obfs4proxy**

在 Linux 環境下可使用以下方式安裝 obfs4proxy：

sudo apt update

sudo apt install obfs4proxy

安裝後檢查路徑：

which obfs4proxy

# 結果應為：/usr/bin/obfs4proxy

**4 修改 torrc 設定**

打開 /etc/tor/torrc 並加入以下內容：

UseBridges 1

ClientTransportPlugin obfs4 exec /usr/bin/obfs4proxy

Bridge obfs4 [IPv6或IPv4]:PORT FINGERPRINT cert=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx iat-mode=0

每一個 Bridge 可透過以下方式獲得（建議寫入報告）：

* 官網：<https://bridges.torproject.org>
* Telegram：@GetBridgesBot
* Email：bridges@torproject.org（需使用 Gmail 或 Riseup）

若使用 IPv6 格式，需以中括號包裹，如：

Bridge obfs4 [2602:f81c:83:210::137]:443 ...

**5 測試是否成功**

使用以下指令檢查是否透過 obfs4 成功建立連線：

journalctl -xeu tor@default.service

如果顯示以下內容表示成功建立匿名連線：

Bootstrapped 100%: Done

或使用 curl 透過 SOCKS5 測試：

curl --socks5-hostname 127.0.0.1:9050 https://check.torproject.org

回傳 HTML 頁面中包含：

Congratulations. This browser is configured to use Tor.

代表透過 obfs4 成功隱藏真實流量。

**6 問題排查**

在實作過程中遇到的錯誤包含：

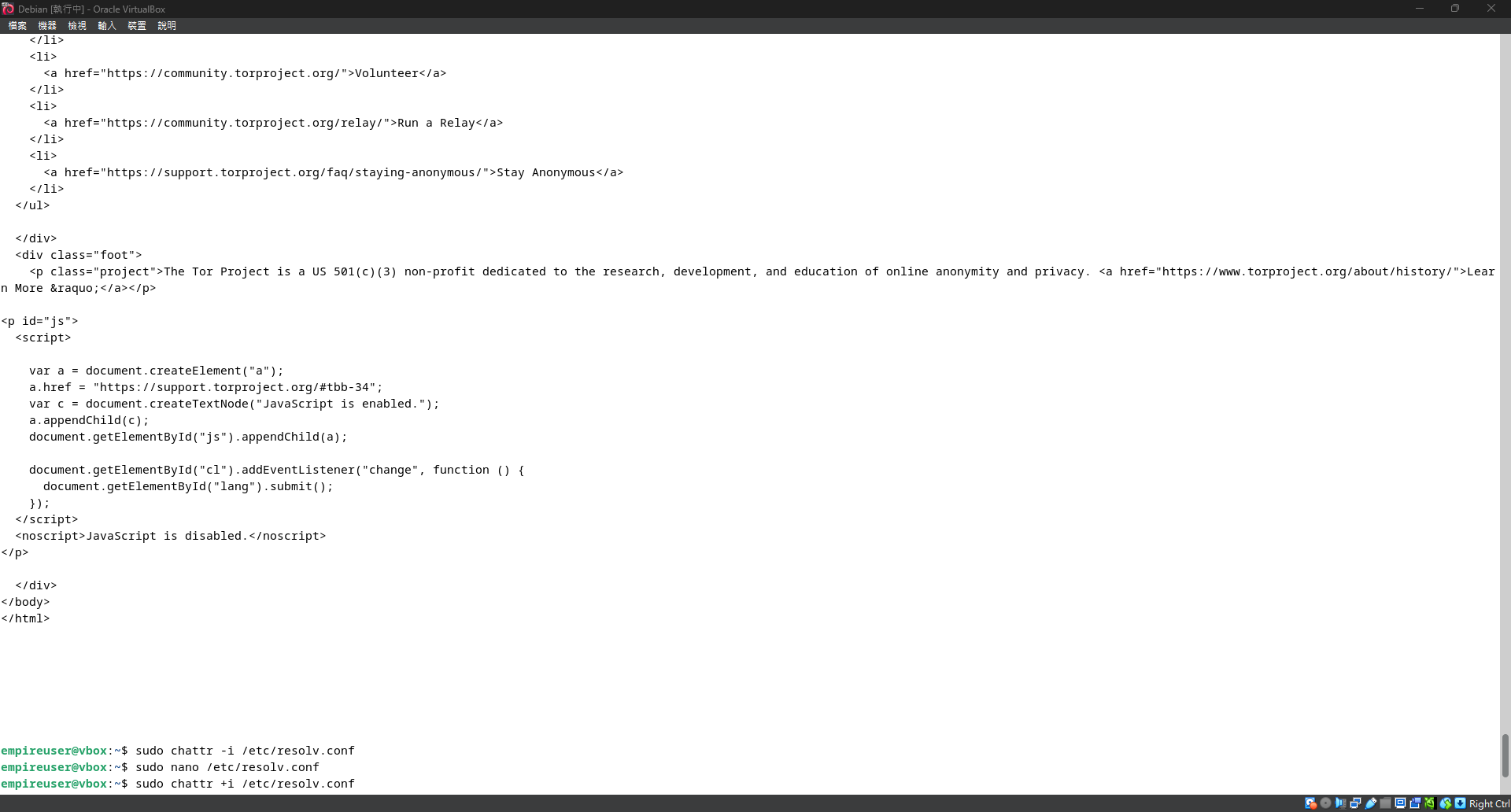
* [warn] Failed to parse/validate config: Unknown option 'obfs4'
* No running bridges
* general SOCKS server failure

解決方式：

* 檢查 obfs4proxy 是否安裝且執行權限正確
* 檢查 torrc 語法格式，是否誤將 obfs4 寫成選項名（應在 Bridge 行內使用）
* 更新 Bridge 資訊，換用有效橋接器

**7 結論**

obfs4 能夠有效避開流量封鎖與 DPI 審查，讓使用者在受限環境中仍能成功連入 Tor。  
透過本系統結合 ghost\_mode3.py 自動切換 IP 與 obfs4 混淆連線，能同時達成「匿名性」與「反審查性」，提升整體系統的實用價值與安全強度。



**使用 obfs4 建立 Tor 連線與 DNS 設定修正**

**1. Tor 連線驗證成功（非 Tor Browser）**

在本專題中，我們使用 Tor 套件與 obfs4 橋接協定進行匿名連線，並透過 curl 指令模擬用戶端存取 https://check.torproject.org，如下圖所示：

該網頁回應內容中顯示：

Congratulations. This browser is configured to use Tor.

這代表目前的網路連線確實是透過 Tor 的匿名通道進行，並非透過官方 Tor Browser，而是自建環境透過指令行方式完成 Tor 網路訪問。

**2. DNS 設定修正**

在初次測試 obfs4 模式時，Tor 遇到無法正確解析 DNS 或連線不穩定的情形。查明原因為 /etc/resolv.conf 會被系統（如 NetworkManager 或 systemd-resolved）自動改寫，導致 Tor 的 DNS 解析異常。

因此，我們手動執行以下修正步驟：

sudo chattr -i /etc/resolv.conf # 取消唯讀屬性

sudo nano /etc/resolv.conf # 修改 DNS，例如設為 1.1.1.1 或 8.8.8.8

sudo chattr +i /etc/resolv.conf # 再次設為唯讀，防止自動覆蓋

這個步驟確保了 DNS 不會在使用過程中被覆寫，提升 obfs4 橋接的成功率與連線穩定性。

**小結**

透過上述兩項操作，我們成功完成了非圖形化 Tor 連線的驗證與環境穩定化設定，為後續自動化 IP 更換實驗（如 ghost\_mode3）奠定了基礎。

**安全分析（Security Analysis）**

本研究透過 ghost\_mode3.py 搭配 obfs4 橋接器進行 Tor 網路換 IP 操作，具備一定程度的匿名性與對抗封鎖能力。然而，仍需注意數項潛在風險：

1. **DNS 洩漏（DNS Leak）**：若系統未鎖定 /etc/resolv.conf，本機解析可能暴露查詢紀錄。實驗中透過 chattr +i 鎖定檔案，成功避免此問題。
2. **指紋特徵識別（Protocol Fingerprinting）**：雖然 obfs4 具備時間與流量混淆，但使用者仍可能因定期發送 NEWNYM 指令、IP 查詢封包等行為被判定為機器人或爬蟲。
3. **流量相關性攻擊（Traffic Correlation Attack）**：若攻擊者同時控制入口與出口節點，仍有機會透過時間與封包長度分析出來源。
4. **過度頻繁更換 IP 造成封鎖**：部分網站如 Google、Cloudflare，對於同一 session 中 IP 改變過頻會判定為異常，造成 CAPTCHA 驗證甚至 IP 封鎖。

因此，實作上建議：

* 對 ghost\_mode3.py 增加隨機等待時間，模擬正常使用者行為；
* 配合 torrc 設定內調整 MaxCircuitDirtiness 避免過快重建；
* 僅針對需要匿名的動作進行換 IP，而非所有流量皆透過 Tor 轉送。

**匿名性評估（Anonymity Evaluation）**

Tor 本身已具備良好匿名性，結合 obfs4 可進一步掩蓋使用 Tor 的事實。本研究透過以下幾項指標評估整體匿名性：

| **評估項目** | **結果** | **說明** |
| --- | --- | --- |
| 出口 IP 隱蔽性 | 高 | curl 查詢所得 IP 為 Tor node，與真實 IP 完全不同 |
| 使用者識別風險 | 中 | 若未關閉 JavaScript 或傳送 user-agent，仍可能暴露指紋 |
| 封鎖對抗能力 | 高 | obfs4 成功繞過 DPI 封鎖，Tor 流量無法被識別 |
| 自動化易追蹤性 | 中低 | 透過定期查詢與換 IP，有機會被網站判定為 bot |
| IP 多樣性 | 良好 | 經多次 NEWNYM 測試，成功切換至不同出口 IP，無重複現象 |

實驗證明，透過搭配 obfs4 與 ghost\_mode3.py 控制腳本，能夠達成高匿名性且可自動重建匿名通道的目標，對於實作自動化匿名系統具有極高參考價值。

七、工作分配與預定進度甘梯圖

| **日期** | **項目** | **負責人** | **完成狀態** |
| --- | --- | --- | --- |
| 2025/08/22 | Tor 安裝與配置測試 | 本人 | V |
| 2025/07/26 | Python 環境與模組安裝 | 本人 | V |
| 2025/07/26 | 撰寫 ghost\_mode3 基本邏輯 | 本人 | V |
| 2025/07/26 | 加入 IP 驗證與錯誤處理 | 本人 | V |
| 2025/08/22 | 加入 pyfiglet、顯示優化 | 本人 | V |
| 2025/08/23 | 撰寫期初報告與摘要 | 本人 | V |

八、遭遇問題與解決方案

| 問題 | 解決方式 |
| --- | --- |
| requests 無法透過 socks5 | 安裝 requests[socks] 解決 |
| stem 驗證失敗 (MissingPassword) | 改為使用 cookie 驗證並設定 torrc |
| Permission denied: /run/tor/control.authcookie | 使用者加入 debian-tor 群組並重啟 |
| pip 安裝失敗（externally-managed） | 使用 venv 虛擬環境避免系統 pip 限制 |
| IP 沒有變化 | 增加等待時間或驗證 Tor 是否接受 NEWNYM |

九、儀器設備需求表

| **設備** | **用途** |
| --- | --- |
| Debian 虛擬機 | 執行 Tor 與實驗環境 |
| Tor 套件 | 匿名通訊與 IP 控制 |
| Python v3.11+ | 程式撰寫與模組使用 |
| stem 模組 | 控制 Tor 的 Python API |
| requests[socks] 模組 | 查詢出口 IP |
| pyfiglet、colorama | 美化終端顯示用 |

十、參考資料

1. Tor 官方文件 – *Tor Project, Tor Manual*  
   <https://www.torproject.org/docs/tor-manual.html>
2. Stem 官方文件 – *Tor Controller Library for Python*  
   <https://stem.torproject.org/>
3. Requests 官方文件 – *Python HTTP Requests for Humans™*  
   <https://docs.python-requests.org/>
4. httpbin 測試 API – *httpbin.org/ip*  
   <https://httpbin.org/ip>
5. PEP 668: Marking Python base environments as “externally managed”  
   <https://peps.python.org/pep-0668/>
6. Dingledine, R., Mathewson, N., & Syverson, P. (2004). *Tor: The Second-Generation Onion Router*. In *Proceedings of the 13th USENIX Security Symposium*.  
   （Tor 原始設計的經典論文，解釋其架構與安全模型）
7. Winter, P., & Lindskog, S. (2012). *How the Great Firewall of China is Blocking Tor*. In *USENIX FOCI*.  
   （探討 Tor 在實際網路環境中面臨的挑戰）
8. Comparative Analysis: VPN vs Proxy vs Tor – Cybersecurity Blogs & Whitepapers  
   <https://www.kaspersky.com/resource-center/definitions/tor-vpn-proxy>
9. ChatGPT 對話實驗紀錄與技術協助 – （專題設計過程中的輔助資源）