

**BỘ THÔNG TIN VÀ TRUYỀN THÔNG**  
**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

-----o0o-----



**BÁO CÁO BÀI THỰC HÀNH**  
**PHÂN TÍCH TÌNH WINDOWS VỚI IDA PRO**

Giảng viên hướng dẫn: TS. Nguyễn Ngọc Điệp

Sinh viên thực hiện: Phạm Vũ Minh Hiếu – B20DCAT061

Lớp: D20CQAT01-B

Mã sinh viên: B20DCAT061

**Hà Nội - 2024**

## MỤC LỤC

MỤC LỤC .....	2
DANH MỤC CÁC HÌNH VẼ.....	3
DANH MỤC CÁC BẢNG BIỂU .....	4
DANH MỤC VIẾT TẮT .....	5
1.1. Giới thiệu bài thực hành .....	8
1.2. Nội dung và hướng dẫn bài thực hành .....	8
1.2.1. Mục đích .....	8
1.2.2. Yêu cầu đối với sinh viên .....	9
1.2.3. Nội dung thực hành.....	9
1.3. Phân tích yêu cầu bài thực hành .....	13
1.4. Thiết kế bài thực hành .....	13
1.5. Cài đặt và cấu hình các máy ảo .....	18
1.6. Tích hợp và triển khai.....	20
1.6.1. Docker Hub .....	21
1.6.2. Github .....	22
1.7. Thử nghiệm và đánh giá .....	23
TÀI LIỆU THAM KHẢO.....	47

## DANH MỤC CÁC HÌNH VẼ

Hình 1: Giao diện Labedit .....	18
Hình 2: Cài đặt Result 1 .....	18
Hình 3: Cài đặt phần Result 2 .....	19
Hình 4: Cài đặt Result 3 .....	19
Hình 5: Cài đặt Result 4 .....	19
Hình 6: Cài đặt phần Goals .....	19
Hình 7: Cài đặt phần Parameter .....	20
Hình 8: Dockerfiles .....	20
Hình 9: Add và comit bài lab .....	21
Hình 10: Thêm bài lab muốn lưu trữ.....	21
Hình 11: Quá trình tải bài lab lên dockerhub.....	21
Hình 12: Đẩy thành công .....	22
Hình 13: Tạo file Imodule.tar.....	22
Hình 14: File imodule.tar chứa bài thực hành.....	22
Hình 15: Đẩy lên github thành công .....	23
Hình 16: Remote sang máy win 10 .....	23
Hình 17: Thành phần của mal-stic-ida .....	24
Hình 18: CFF Explorer.....	24
Hình 19: Giao diện IDA Pro sau khi mở file .....	25
Hình 20: Text View DllMain .....	25
Hình 21: Gethostbyname.....	26
Hình 22: Địa chỉ của gethostbyname .....	26
Hình 23: Bảng tham chiếu của gethostbyname.....	27
Hình 24: Jump to address .....	27
Hình 25: DNS được trở tới.....	28
Hình 26: Biến cục bộ và tham số của hàm.....	28
Hình 27: Bảng tìm kiếm Alt + T .....	29
Hình 28: Địa chỉ của \cmd.exe /c .....	29
Hình 29: Chuỗi lệnh thực thi.....	30
Hình 30: aHiMasterDDDDDD .....	30
Hình 31: Nội dung aHiMasterDDDDDD .....	31
Hình 32: So sánh cmd và command.....	31
Hình 33: Bảng tham chiếu của dword_1008E5C4.....	31
Hình 34: Hàm gán dữ liệu cho dword_1008E5C4.....	32

Hình 35: Nội dung hàm sub_10003695 .....	32
Hình 36: Platform của Microsoft .....	33
Hình 37: So sánh chuỗi với chuỗi robowork .....	34
Hình 38: Hàm nếu so sánh thành công.....	34
Hình 39: Registry Key của hàm.....	34
Hình 40: Giả C của sub_100052A2 .....	35
Hình 41: Hàm Sub_100038EE.....	35
Hình 42: Nội dung PSLIST.....	36
Hình 43: Sub_100036C3.....	36
Hình 44: Sub_1000664C.....	37
Hình 45: Sub_10006518.....	37
Hình 46: Hàm sub_10004E79 .....	38
Hình 47: User xrefs chart .....	39
Hình 48: Recursion depth.....	40
Hình 49: Sơ đồ của sub_10004E79.....	41
Hình 50: Sơ đồ của DllMain .....	41
Hình 51: Mã hàm Sleep.....	42
Hình 52: Tham số truyền vào Socket.....	42
Hình 53: Giá trị của 0x1001D988.....	43
Hình 54: Nội dung Lab05-1.py .....	43
Hình 55: Script file chỉ nhận đuôi IDC .....	44
Hình 56: Script chuyển từ Python qua IDC .....	45
Hình 57: Thông điệp đã giải mã.....	45
Hình 58: Nội dung của ATTT.txt.....	46
Hình 59: Kéo thư mục về máy Linux và kiểm tra .....	46

## DANH MỤC CÁC BẢNG BIỂU

Bảng 1. Bảng Results .....	15
Bảng 2. Bảng Goals.....	17
Bảng 3. Bảng Paramater.....	17

## DANH MỤC VIẾT TẮT

Từ viết tắt	Thuật ngữ tiếng Anh / Giải thích	Thuật ngữ tiếng Việt / Giải thích
IDA	Interactive Disassembler	Công cụ giải mã ngược mạnh mẽ, hỗ trợ phân tích mã nhị phân.
DNS	Domain Name System	Hệ thống phân giải tên miền thành địa chỉ IP.
IOC	Indicator of Compromise	Dấu hiệu nhận diện mã độc, bao gồm IP, domain, hash file, hoặc URL độc hại.
MSV	Mã sinh viên	Dữ liệu định danh sinh viên trong hệ thống.
IP	Internet Protocol	Địa chỉ định danh thiết bị trong mạng.
URL	Uniform Resource Locator	Đường dẫn xác định tài nguyên trên Internet.
SCP	Secure Copy Protocol	Giao thức sao chép tệp an toàn qua SSH.
Mal	Malware	Một file có hành vi độc hại
Stic	Static	Hành động phân tích tĩnh một file độc hại
	Functions	Các hàm được phát hiện trong mã nhị phân, hiển thị trong IDA Pro để phân tích.
	Remote Shell	Truy cập điều khiển từ xa qua giao diện dòng lệnh (terminal) trên thiết bị khác.

	Graph View	Chế độ xem luồng điều khiển chương trình dưới dạng đồ thị trong IDA Pro.
	Export	Các hàm hoặc API mà file chia sẻ để các chương trình khác có thể sử dụng.
	Import	Các hàm hoặc API mà file gọi từ các thư viện bên ngoài.
	User Xrefs Chart	Biểu đồ tham chiếu chéo (Cross-references) trong IDA Pro, hiển thị các liên kết đến hàm hoặc biến.
	Create Thread	API tạo luồng thực thi mới trong chương trình.
	Terminal	Giao diện dòng lệnh để tương tác với hệ thống.
	Domain	Tên miền, định danh của một địa chỉ IP trong hệ thống DNS.
	File	Tệp thực thi hoặc dữ liệu đầu vào cần phân tích.
	Payload	Tải trọng, phần mã độc chính thực thi hành vi như đánh cắp dữ liệu hoặc phá hoại.
	Session	Một phiên làm việc hoặc kết nối giữa máy chủ và máy khách.
	Registry	Cơ sở dữ liệu cấu hình của hệ điều hành Windows, nơi mã độc thường lưu trữ để duy trì.



## **1.1. Giới thiệu bài thực hành**

Bài thực hành " **Tìm hiểu về phân tích tĩnh mã độc Windows với IDA Pro** " được thiết kế nhằm giúp sinh viên hiểu rõ hơn về kỹ thuật phân tích tĩnh – một phương pháp quan trọng trong lĩnh vực phân tích mã độc, đặc biệt là trong việc nhận diện và xử lý các mối đe dọa từ phần mềm độc hại. Đây là bước khởi đầu cần thiết để sinh viên làm quen với quy trình phân tích mã độc và nhận thức rõ cách thức hoạt động của các chương trình độc hại.

Trong bài thực hành này, sinh viên sẽ học cách sử dụng IDA Pro, một trong những công cụ mạnh mẽ và phổ biến nhất để phân tích tĩnh mã độc. IDA Pro cho phép sinh viên thực hiện phân tích file thực thi mà không cần chạy nó, giúp đảm bảo an toàn cho môi trường làm việc. Sinh viên sẽ tìm hiểu cách khám phá các đặc điểm của mã độc thông qua việc phân tích mã máy, cấu trúc file, chuỗi ký tự và các thư viện được sử dụng. Bài thực hành này sẽ hướng dẫn sinh viên cách sử dụng các tính năng của IDA Pro để tạo biểu đồ luồng, phân tích các hàm quan trọng, và tìm kiếm các hành vi đáng ngờ trong mã độc.

Việc sử dụng IDA Pro trong phân tích mã độc không chỉ giúp sinh viên nhận diện được các đặc điểm tiềm ẩn nguy hiểm trong mã độc mà còn trang bị cho họ kỹ năng cần thiết để xử lý các tình huống thực tế trong lĩnh vực bảo mật thông tin. Đây là một bước quan trọng để nâng cao kiến thức và kỹ năng về an toàn thông tin, đồng thời làm quen với quy trình phân tích chuyên sâu và các công cụ hỗ trợ hiện đại.

Thông qua bài thực hành này, sinh viên không chỉ hiểu cách phân tích tĩnh mã độc bằng IDA Pro mà còn nhận thức được tầm quan trọng của việc áp dụng các công cụ tiên tiến trong việc điều tra và đánh giá mức độ an toàn hệ thống. Bài học này cũng giúp sinh viên chuẩn bị tốt hơn cho các công việc liên quan đến xử lý mã độc và bảo mật thông tin trong tương lai.

## **1.2. Nội dung và hướng dẫn bài thực hành**

### **1.2.1. Mục đích**

Giúp sinh viên tìm hiểu khái niệm về phân tích mã độc và cách sử dụng công cụ IDA Pro để thực hiện phân tích tĩnh mã độc. Sinh viên sẽ làm quen với các chức năng của IDA Pro để khám phá hành vi đáng ngờ của mã độc như tải payload, thực



thì lệnh PowerShell, và phát hiện các liên kết, chỉ báo độc hại. Qua đó, sinh viên sẽ phát triển kỹ năng phân tích và đánh giá mã độc một cách có hệ thống và an toàn.

### 1.2.2. Yêu cầu đối với sinh viên

Có kiến thức cơ bản về hệ điều hành Window – Linux. Làm quen với giao diện và chức năng của IDA Pro để phân tích cấu trúc mã, luồng điều khiển, và các hành vi của mã độc.

### 1.2.3. Nội dung thực hành

Khởi động bài lab:

Vào terminal, gõ:

```
labtainer -r mal-stic-ida
```

*(chú ý: sinh viên sử dụng mã sinh viên của mình để nhập thông tin email người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)*

Sau khi khởi động xong sẽ có 1 terminal ảo sẽ xuất hiện, có tên là mal-stic. Trên terminal **mal-stic-ida** đã được cài đặt sẵn công cụ để remote tới máy Windows chứa mã độc là Xfreerdp.

```
xfreerdp /u:user /p:password /v:ip
```

(user: tài khoản máy Window, password: mật khẩu của tài khoản, ip: IP của máy Window)

Ví dụ: xfreerdp /v:192.168.18.128 /u:minhh /p:1

Sau khi remote thành công sang máy Window, sẽ thấy 1 folder **mal-stic-ida** trên màn hình, bên trong đó có sẵn 1 file mã độc và 1 file ATTT.txt (điền output mà bài yêu cầu để checkwork)

- Sinh viên sử dụng công cụ CFF để kiểm tra định dạng chuẩn của file và kéo vào phiên bản IDA Pro nào cho hợp lý
- Sinh viên mở IDA Pro lên và kéo thả file dll cho sẵn vào để bắt đầu đánh giá, kiểm tra file đã cho.

- Đầu tiên sinh viên kiểm tra địa chỉ của hàm DllMain bằng cách di chuột vào tên hàm DllMain và ấn xem TextView hoặc là xem ở cửa sổ Functions. Sau đó mở file ATTT.txt và điền địa chỉ chuẩn vào file.

*0x1000D02E*

- Sinh viên mở cửa sổ import và tìm xem hàm gethostbyname ở đâu. Sau đó điền vào trong file ATTT.txt với định dạng như bên dưới.

*.idata:100163CC*

- Sinh viên dùng bấm chuột vào tên hàm gethostbyname và ấn phím X để xem có bao nhiêu tham chiếu tới hàm gethostbyname. Sau đó điền vào trong file ATTT.txt.

*18*

- Sinh viên phím G và nhập vào địa chỉ 0x10001757 và đọc xem tại địa chỉ đây hàm gethostbyname đang làm gì, đang trở vào DNS nào. Sau đó điền vào trong file ATTT.txt.

*pics.practicalmalwareanalysis.com*

- Sinh viên dùng phím G để đi tới địa chỉ 0x10001656, và xem hàm tại địa chỉ đây có bao nhiêu tham số và biến cục bộ. Mỗi một giá trị viết tại một dòng riêng biệt tại file ATTT.txt, biến cục bộ thì điền số biến cục bộ có trong hàm, tham số thì viết tên tham số như bên dưới.

*23*

*lpThreadParameter*

- Sinh viên dùng Alt + T để tìm kiếm xem chuỗi **\\cmd.exe /c** được nằm ở biến nào ở vị trí nào, sau đó viết viết địa chỉ tìm được vào file ATTT.txt theo định dạng bên dưới.

*xdoors\_d:10095B34*

- Sinh viên tiến hành kiểm tra xung quanh chuỗi đã cho bên trên để xác định xem xung quanh chuỗi đã cho **\\cmd.exe /c** có chuyện gì đang xảy ra, nó là

shell gì. Sau khi tìm hiểu xong thì viết định dạng shell đây vào trong file ATTT.txt như dưới đây.

### *Remote Shell*

- Sinh viên sử dụng phím G để đi tới địa chỉ **0x100101C8**, sử dụng Grap View để xem thông tin đường đi của hàm 1 cách rõ hơn và chỉ ra trong môi trường Windows 10 mà đang sử dụng để phân tích file độc hại thì đường đi của hàm tại địa chỉ đã cho sẽ như thế nào. Viết câu lệnh mà file thực thi sẽ gọi vào file ATTT.txt

*\\cmd.exe /c*

- Đọc theo lệnh bên trên đi xuống, sinh viên sẽ thấy một đoạn dùng để so sánh một chuỗi với chuỗi là “robowork”. Điều gì sẽ xảy ra nếu so sánh thành công ? Điền vào file ATTT.txt key được thao tác khi mà so sánh chuỗi thành công.

*SOFTWARE\Microsoft\Windows\CurrentVersion*

- Sinh viên mở cửa sổ Export, nhảy tới hàm PSLIST. Đọc đoạn mã và cho biết rằng PSLIST có tác dụng gì ? Dùng để làm gì các chương trình đang chạy. Sau đó điền vào trong file ATTT.txt

*List running processes*

- Ở cửa sổ Fuctions, sinh viên tìm tới hàm **sub\_10004E79** và cho biết hàm này gọi tới những API nào ? Sinh viên xem bằng cách ấn vào View -> Graphs -> User xrefs chart để tùy chọn các yêu cầu. Sau đó điền tên của hàm API được gọi tới vào file ATTT.txt

*GetSystemDefaultLangID*

- Ở cửa sổ Fuctions, sinh viên tìm tới hàm **DllMain** và cho biết hàm này gọi tới những API nào ở độ sâu gọi hàm là 1 ? Sinh viên xem bằng cách ấn vào View -> Graphs -> User xrefs chart và điền vào ô Recursion depth là 1. Sau đó điền tên của hàm API được gọi tới vào file ATTT.txt

*CreateThread*

- Sinh viên bấm G và đi tới địa chỉ **0x10001358** đọc code và tính toán xem thời gian mà Sleep API sẽ thực thi là bao lâu. Sau đó điền vào file ATTT.txt với đơn vị là giây.

30

- Sau đó sinh viên đi tới địa chỉ **0x10001701** đọc code và viết vào file ATTT.txt giá trị được truyền vào các tham số cho hàm **socket** là gì theo mẫu bên dưới.

*protocol: 6, type: 1, af: 2*

- Sinh viên đi tới địa chỉ **0x1001D988** xem thông tin được lưu tại đây, tiến hành kiểm tra và giải mã thông tin với code python đã cho tại thư mục **mal-stic-ida**, sau đó điền thông tin vào vào file ATTT.txt.

*xdoor is this backdoor, string decoded for Practical Malware Analysis Lab :)1234*

- Sau khi xong hết các nhiệm vụ, sinh viên lưu lại file.
- Về lại terminal mal-stic-ida dùng scp để tiến hành kéo thư mục **mal-stic-ida** từ máy Windows về ( có thể dùng cách khác để kéo file về mà sinh viên biết ).

*scp -r minhh@192.168.18.128:"C:/Users/minhh/Desktop/mal-stic-ida" ~/*

- Lệnh sẽ tự tạo 1 folder share trên máy Linux, tiến hành cd sang folder **mal-stic-ida** và cat file ATTT.txt

*cd mal-stic-ida*

*cat ATTT.txt*

- Cuối cùng ta sẽ checkwork
- Kết thúc bài lab:

Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:

*stoplab mal-stic-ida*

Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.

Khởi động lại bài lab:

Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

*startlab -r mal-stic-ida*

### 1.3. Phân tích yêu cầu bài thực hành

Bài thực hành yêu cầu sinh viên làm quen với công cụ IDA Pro, một trong những công cụ mạnh mẽ nhất để phân tích tĩnh mã độc. Sinh viên sẽ sử dụng IDA Pro để phân tích một file độc hại, thu thập thông tin ban đầu về cấu trúc file, hành vi đáng ngờ, giải mã thông tin và các chỉ số nguy cơ (IOC) nhằm nhận diện và đánh giá mức độ nguy hiểm của mã độc. Để hoàn thành bài thực hành, sinh viên sẽ phân tích tĩnh một file độc hại đã được cài sẵn trong môi trường làm việc.

Sinh viên sẽ bắt đầu bài thực hành bằng lệnh khởi tạo labtainer <tên bài lab> và nhập MSV của mình. Sau đó, sinh viên sẽ sử dụng IDA Pro để mở file độc hại và phân tích các đặc điểm của file, bao gồm xác định các chuỗi ký tự quan trọng (Strings) như URL, IP, domain, hoặc tên file tải xuống. Sinh viên cũng sẽ kiểm tra các thư viện liên kết và hàm API được sử dụng bởi mã độc, phân tích cấu trúc file và các thành phần đáng nghi trong mã nguồn, như các đoạn mã thực thi, trình tải payload, hoặc các dấu hiệu mã hóa. Sinh viên sẽ viết các thông tin vào 1 file ATTT.txt ở trên máy Windows 10, những thông tin được tìm ra, giải mã trong lúc phân tích file độc hại với IDA mà nhiệm vụ yêu cầu.

Cuối cùng, sinh viên sẽ tải file kết quả về máy Linux bằng lệnh scp và dừng bài lab bằng lệnh stoplab <tên bài lab>. Hệ thống yêu cầu file kết quả của sinh viên phải bao gồm các thông tin được thu thập từ các nhiệm vụ như địa chỉ hàm, IOC, API, thông tin bị mã hoá, key, các lệnh thực thi,...

### 1.4. Thiết kế bài thực hành

Trên môi trường máy ảo Ubuntu được cung cấp, sử dụng docker tạo ra container mang tên “mal-stic-ida”.

Cấu hình docker gồm có:

- mal-stic: Lưu cấu hình cho máy thực hành, trong đó gồm có:
  - Tên máy: mal-stic-ida
- Config: Lưu cấu hình hoạt động của hệ thống.
- Dockerfile: Mô tả cấu hình của container mal-stic-ida, trong đó:
  - mal-stic-ida: Sử dụng các thư viện mặc định của hệ thống và tích hợp sẵn công cụ xfreerdp và dcp
  - xfreerdp: Hỗ trợ remote sang máy tính Windows
  - scp: Dùng để truyền file từ máy Windows về Linux

Trên môi trường máy Windows được cài:

- Mở tính năng remote cho phép máy ubuntu remote tới
- SSH server
- Thư mục gồm file độc hại để phân tích, file code python và một file để điền kết quả.
- Công cụ dùng để phân tích IDA Pro

Các nhiệm vụ cần phải thực hiện để thành công trong bài thực hành:

- Lấy địa chỉ của DllMain
- Tìm địa chỉ của gethostbyname, hàm được import vào.
- Thu thập thông tin về số các hàm gọi tới gethostbyname.
- Thu thập thông tin về DNS mà hàm gethostbyname trả về.
- Thu thập thông tin về tham số và biến cục bộ của hàm.
- Tìm kiếm địa chỉ chuỗi.
- Xác định shell.

- Xác định câu lệnh sẽ được dùng.
- Kết quả so sánh chuỗi.
- Nội dung hàm Export PSLIST.
- Dùng Xrefs để xem API của hàm.
- Tính toán thời gian Sleep.
- Tham số của Socket.
- Nội dung hàm Export PSLIST.

Kết thúc bài lab sau khi hoàn thành tất cả các nhiệm vụ và kiểm tra file kết quả ATTT.txt.

Kết thúc bài lab và đóng gói kết quả.

Để đánh giá được sinh viên đã hoàn thành bài thực hành hay chưa, cần chia bài thực hành thành các nhiệm vụ nhỏ, mỗi nhiệm vụ cần phải chỉ rõ kết quả để có thể dựa vào đó đánh giá, chấm điểm. Do vậy, trong bài thực hành này hệ thống cần ghi nhận các thao tác, sự kiện được mô tả và cấu hình như bảng 1,2,3:

*Bảng 1. Bảng Results*

Result Tag	Container	File	Field Type	Field ID	Timestamp Type
adr-dllmain	mal-stic-ida	cat.stdout	CONTAINS	0x1000D02E	File
_import	mal-stic-ida	cat.stdout	CONTAINS	.idata:100163CC	File
_functions-call	mal-stic-ida	cat.stdout	CONTAINS	18	File

dns-request	mal-stic-ida	cat.stdout	CONTAINS	pics.practicalmalwareanalysis.com	File
_variables	mal-stic-ida	cat.stdout	CONTAINS	23	File
_parameters	mal-stic-ida	cat.stdout	CONTAINS	lpThreadParameter	File
string-located	mal-stic-ida	cat.stdout	CONTAINS	xdoors_d:10095B34	File
_shell	mal-stic-ida	cat.stdout	CONTAINS	Remote Shell	File
_cmd-command	mal-stic-ida	cat.stdout	CONTAINS	\\cmd.exe /c	File
_regedit	mal-stic-ida	cat.stdout	CONTAINS	SOFTWARE\Microsoft\Windows\CurrentVersion	File
pslist	mal-stic-ida	cat.stdout	CONTAINS	List running processes	File
_call-api	mal-stic-ida	cat.stdout	CONTAINS	GetSystemDefaultLangID	File
_dll-api 1	mal-stic-ida	cat.stdout	CONTAINS	CreateThread	File
sleep-s	mal-stic-ida	cat.stdout	CONTAINS	30	File



socket	mal-stic-ida	cat.stdout	CONTAINS	protocol: 6, type: 1, af: 2	File
final	mal-stic-ida	cat.stdout	CONTAINS	xdoor is this backdoor, string decoded for Practical Malware Analysis Lab :)1234	File

*Bảng 2. Bảng Goals*

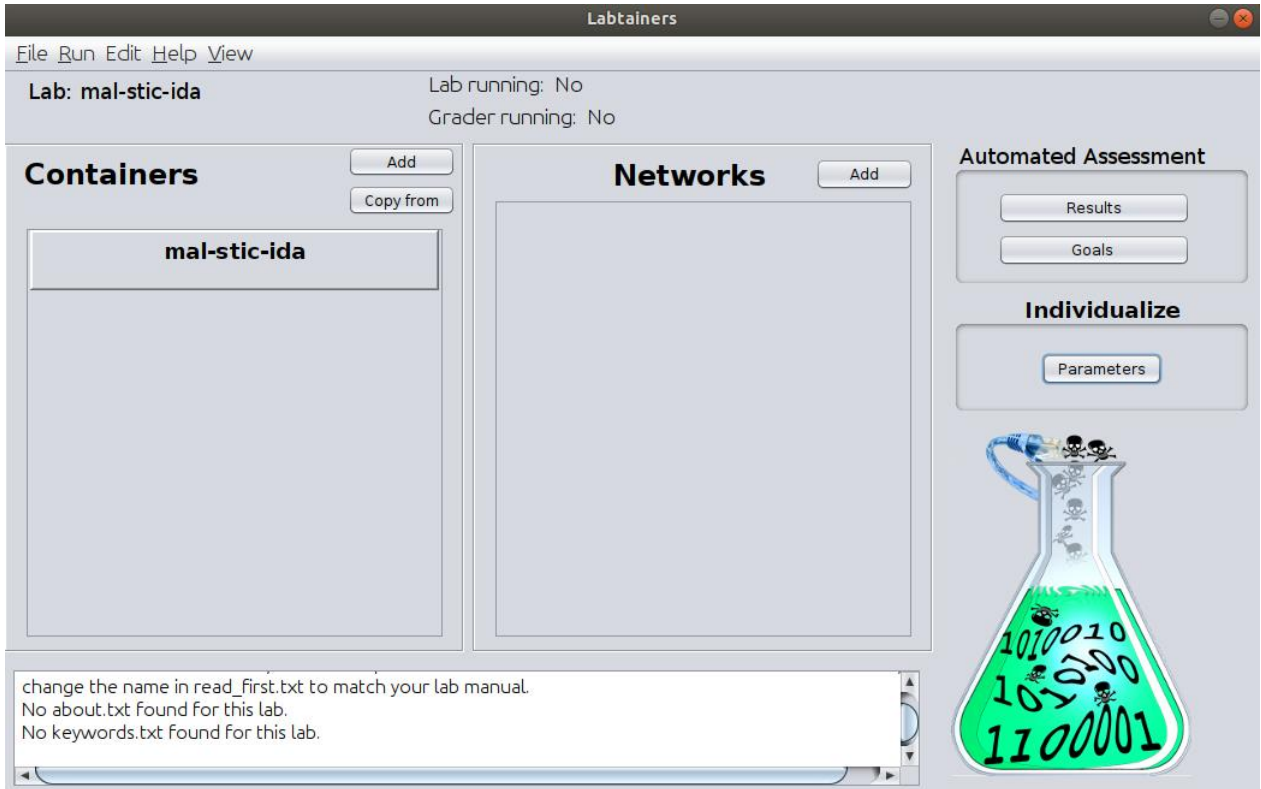
Goal ID	Goal Type	Boolean	Boolean Result Tags
shell-cmd	boolean	_shell and _cmd-command and _regedit	adr-dllmain
variables-para	boolean	_variables and _parameters	adr-dllmain
call-api	boolean	_call-api and _dll-api1	adr-dllmain
gethost	boolean	_functions-call and _import	adr-dllmain

*Bảng 3. Bảng Paramater*

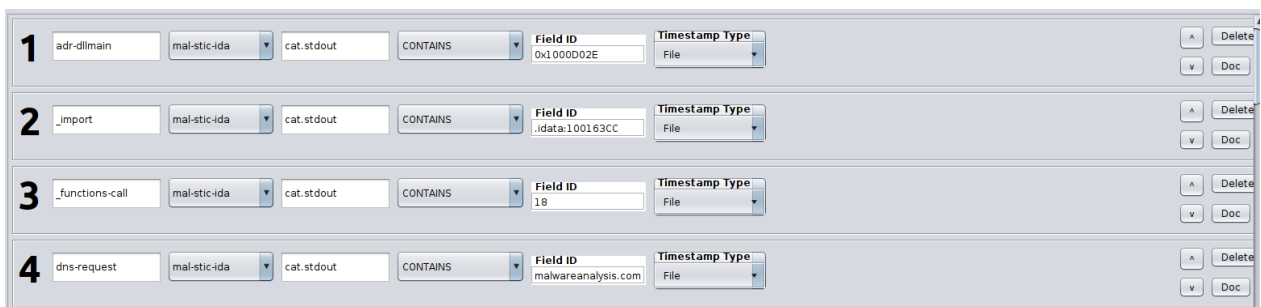
Param ID	Operator	File name	Symbol	Step	Hashstring

--	--	--	--	--	--

## 1.5. Cài đặt và cấu hình các máy ảo



Hình 1: Giao diện Labedit



Hình 2: Cài đặt Result 1

Result Tag	Container	File	Field Type	Field ID	Timestamp Type		
5	_variables	mal-stic-ida	cat.stdout	CONTAINS	23	File	^ Delete v Doc
6	_parameters	mal-stic-ida	cat.stdout	CONTAINS	lpThreadParameter	File	^ Delete v Doc
7	string-located	mal-stic-ida	cat.stdout	CONTAINS	xdoors_d:10095834	File	^ Delete v Doc
8	_shell	mal-stic-ida	cat.stdout	CONTAINS	Remote Shell	File	^ Delete v Doc

Hình 3: Cài đặt phần Result 2

Result Tag	Container	File	Field Type	Field ID	Timestamp Type		
5	_variables	mal-stic-ida	cat.stdout	CONTAINS	23	File	^ Delete v Doc
6	_parameters	mal-stic-ida	cat.stdout	CONTAINS	lpThreadParameter	File	^ Delete v Doc
7	string-located	mal-stic-ida	cat.stdout	CONTAINS	xdoors_d:10095834	File	^ Delete v Doc
8	_shell	mal-stic-ida	cat.stdout	CONTAINS	Remote Shell	File	^ Delete v Doc

Hình 4: Cài đặt Result 3

13	_dll-api1	mal-stic-ida	cat.stdout	CONTAINS	CreateThread	File	^ Delete v Doc
14	sleep-s	mal-stic-ida	cat.stdout	CONTAINS	30	File	^ Delete v Doc
15	socket	mal-stic-ida	cat.stdout	CONTAINS	::ocol: 6, type: 1, af: 2	File	^ Delete v Doc
16	final	mal-stic-ida	cat.stdout	CONTAINS	e Analysis Lab :11234	File	^ Delete v Doc

Hình 5: Cài đặt Result 4

Goal ID	Goal Type	Boolean	Boolean Result Tags	
1	shell-cmd	boolean	_shell and _cmd-command and _regedit	adr-dlmain
2	variables-para	boolean	_variables and _parameters	adr-dlmain
3	call-api	boolean	_call-api and _dll-api1	adr-dlmain
4	gethost	boolean	_functions-call and _import	adr-dlmain

Hình 6: Cài đặt phần Goals



*Hình 7: Cài đặt phần Parameter*

```
ARG lab
ARG labdir
ARG imagedir
ARG user_name
ARG password
ARG apt_source
ARG version
LABEL version=$version
ENV APT_SOURCE $apt_source
RUN /usr/bin/apt-source.sh

RUN apt-get update
RUN apt install freerdp2-x11 -y
#
# put package installation here, e.g.,
#     RUN apt-get update && apt-get install -y --no-install-recommends somepackage
#
#
# Install the system files found in the _system directory
#
ADD $labdir/$imagedir/sys_tar/sys.tar /
ADD $labdir/sys_$lab.tar.gz /
#
RUN useradd -ms /bin/bash $user_name
RUN echo "$user_name:$password" | chpasswd
RUN adduser $user_name sudo
# replace above with below for centos/fedora
#RUN usermod $user_name -a -G wheel

#
# **** Perform all root operations, e.g., ****
# **** "apt-get install" prior to the USER command. ****
#
```

*Hình 8: Dockerfiles*

## 1.6. Tích hợp và triển khai

### 1.6.1. Docker Hub

```
student@ubuntu:~/labtainer/trunk/labs$ git init
Initialized empty Git repository in /home/student/labtainer/trunk/labs/.git/
student@ubuntu:~/labtainer/trunk/labs$ git config --global user.name minhh310
student@ubuntu:~/labtainer/trunk/labs$ git config --global user.email hieupvm310@gmail
student@ubuntu:~/labtainer/trunk/labs$ git commit mal-stic-ida -m "Adding an IModule"
[master (root-commit) f65a73d] Adding an IModule
19 files changed, 516 insertions(+)
```

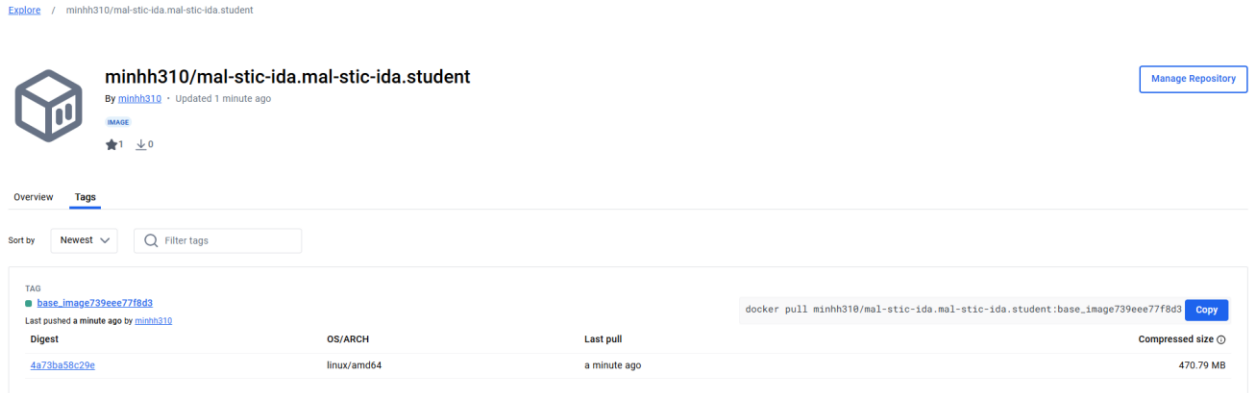
Hình 9: Add và comit bài lab

```
student@ubuntu:~/labtainer/trunk/labs$ git commit mal-stic-ida -m "Adding an IModule"
[master (root-commit) f65a73d] Adding an IModule
19 files changed, 516 insertions(+)
create mode 100644 mal-stic-ida/config/basic-static-home_tar.list
create mode 100644 mal-stic-ida/config/mal-stic-home_tar.list
create mode 100644 mal-stic-ida/config/mal-stic-ida-home_tar.list
create mode 100644 mal-stic-ida/config/parameter.config
create mode 100644 mal-stic-ida/config/start.config
create mode 100644 mal-stic-ida/dockerfiles/Dockerfile.mal-stic-ida.mal-stic-ida.student
create mode 100644 mal-stic-ida/docs/read_first.txt
create mode 100644 mal-stic-ida/instr_config/goals.config
create mode 100755 mal-stic-ida/instr_config/pregrade.sh
create mode 100644 mal-stic-ida/instr_config/results.config
create mode 100644 mal-stic-ida/mal-stic-ida/_bin/.treataslocal.swo
create mode 100755 mal-stic-ida/mal-stic-ida/_bin/fixlocal.sh
create mode 100644 mal-stic-ida/mal-stic-ida/_bin/treataslocal
create mode 100644 mal-stic-ida/mal-stic-ida/_system/etc/login.defs
create mode 100644 mal-stic-ida/mal-stic-ida/_system/etc/securetty
create mode 100644 mal-stic-ida/mal-stic-ida/basic-static.basic-static.student.tar.gz
create mode 100644 mal-stic-ida/mal-stic-ida/home_tar/home.tar
create mode 100644 mal-stic-ida/mal-stic-ida/sys_basic-static.basic-static.student.ta
```

Hình 10: Thêm bài lab muốn lưu trữ

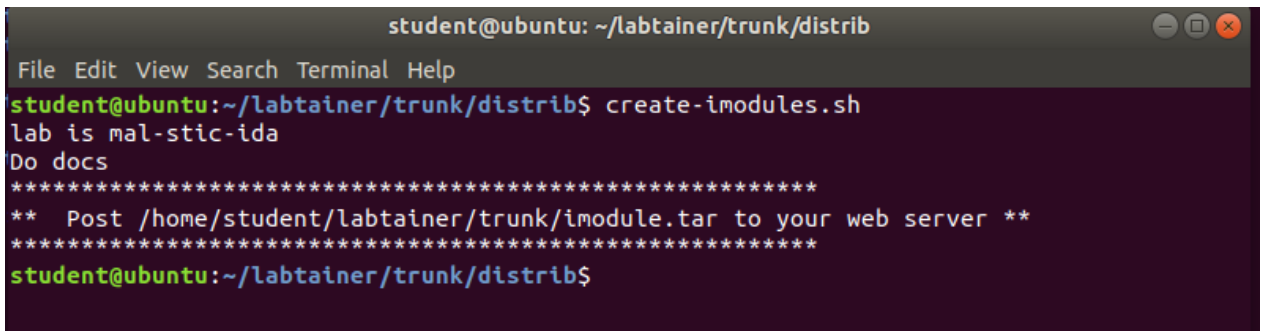
```
student@ubuntu:~/labtainer/trunk/distrib$ ./publish.py -d -l mal-stic-ida
adding [nmaplab]
adding [httplab]
adding [liveforensics]
adding [bind-shell]
adding [tlab]
adding [metasploitable-test]
adding [kali-test]
adding [my-remote-dns]
adding [remote-dns2]
adding [remote-dns]
adding [backups]
adding [centos-log]
adding [dhcp-test]
adding [xlab]
adding [softplc]
adding [iptables]
adding [grfics]
adding [usbtest]
adding [ida]
```

Hình 11: Quá trình tải bài lab lên dockerhub

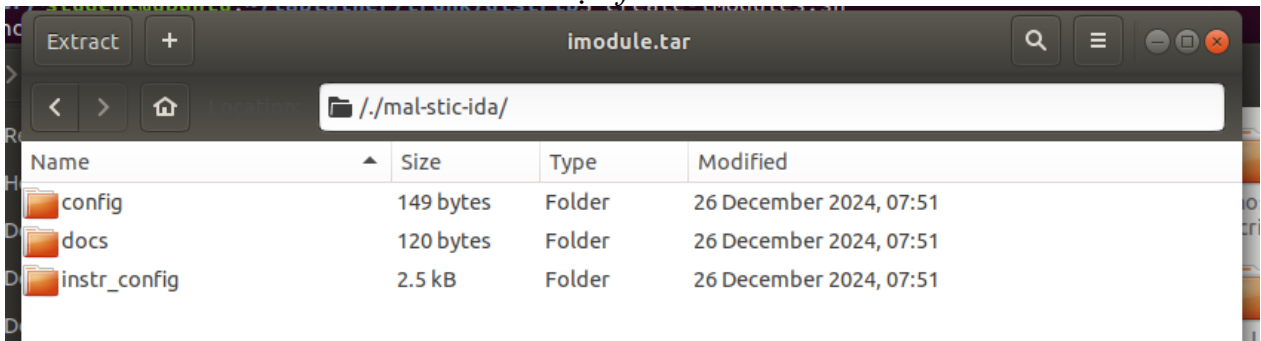


Hình 12: Đẩy thành công

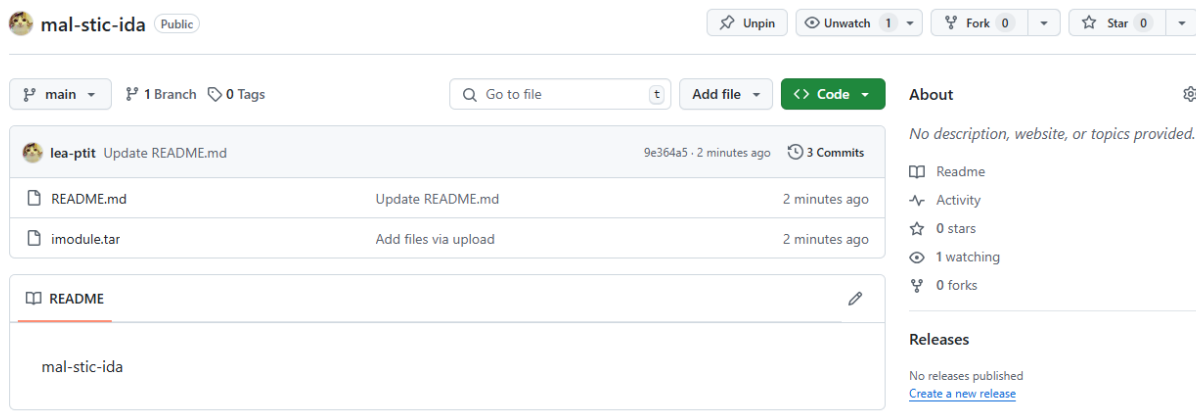
## 1.6.2. Github



Hình 13: Tạo file Imodule.tar



Hình 14: File imodule.tar chứa bài thực hành

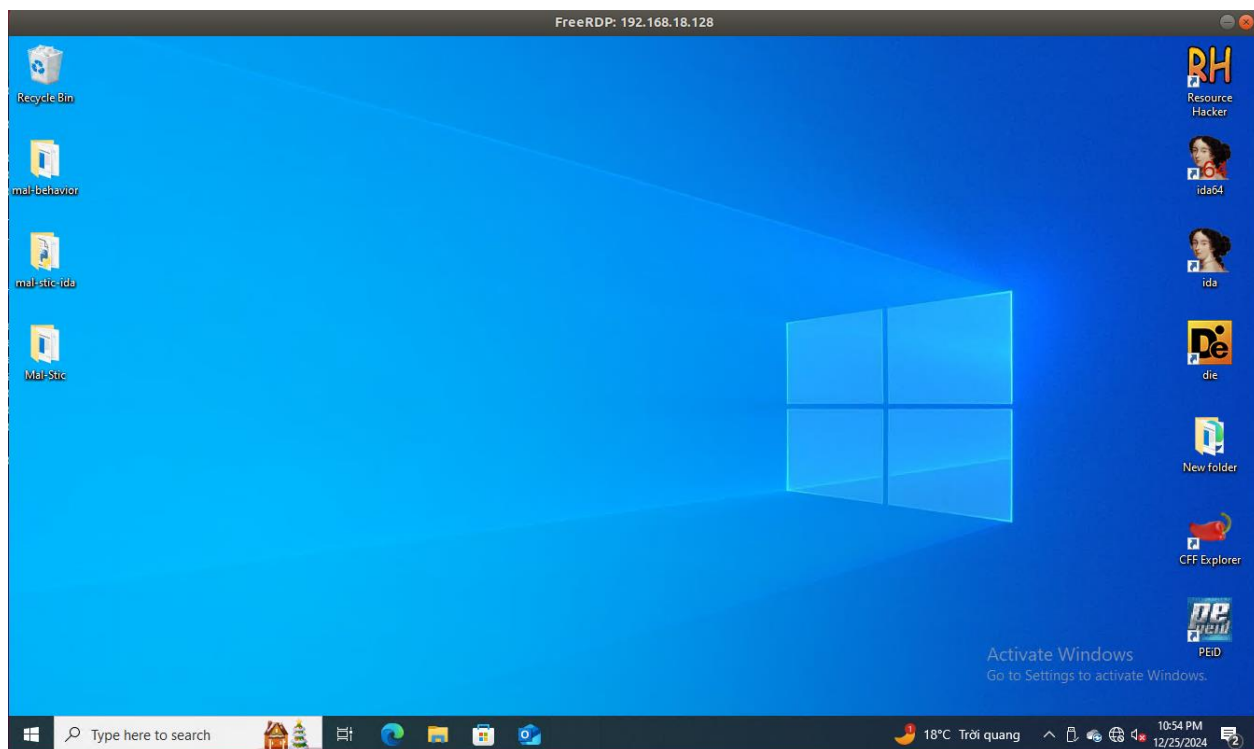


Hình 15: Đẩy lên github thành công

## 1.7. Thử nghiệm và đánh giá

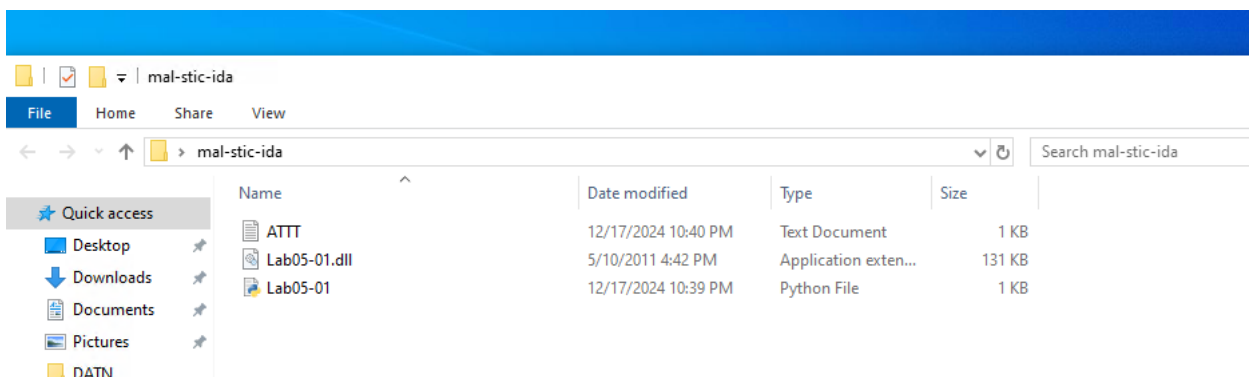
Bài thực hành đã được xây dựng thành công, dưới đây là hình ảnh minh họa về bài thực hành:

Khi bắt đầu labtainer mal-stic-ida, ta tiến hành remote tới máy Windows 10 để thao tác và phân tích tĩnh để giải quyết các nhiệm vụ. Sau khi remote thì có giao diện như sau:



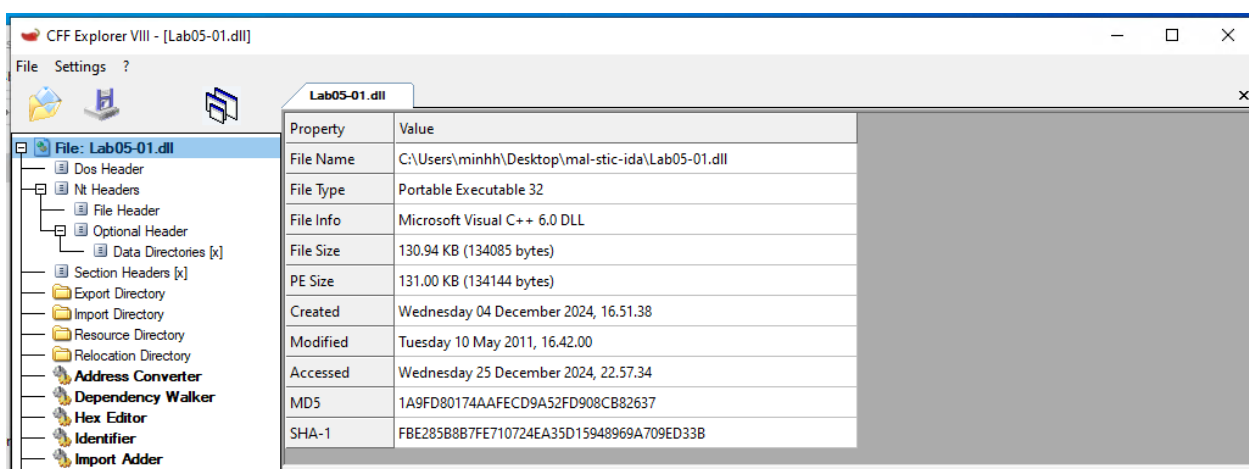
Hình 16: Remote sang máy win 10

Sau khi remote tới được máy chủ Windows 10 thì sinh viên mở thư mục **mal-stic-ida** lên để kiểm tra xem đủ 3 file cần thiết hay chưa. Gồm có ATTT.txt, Lab05-1.dll, Lab05-1.py.



*Hình 17: Thành phần của mal-stic-ida*

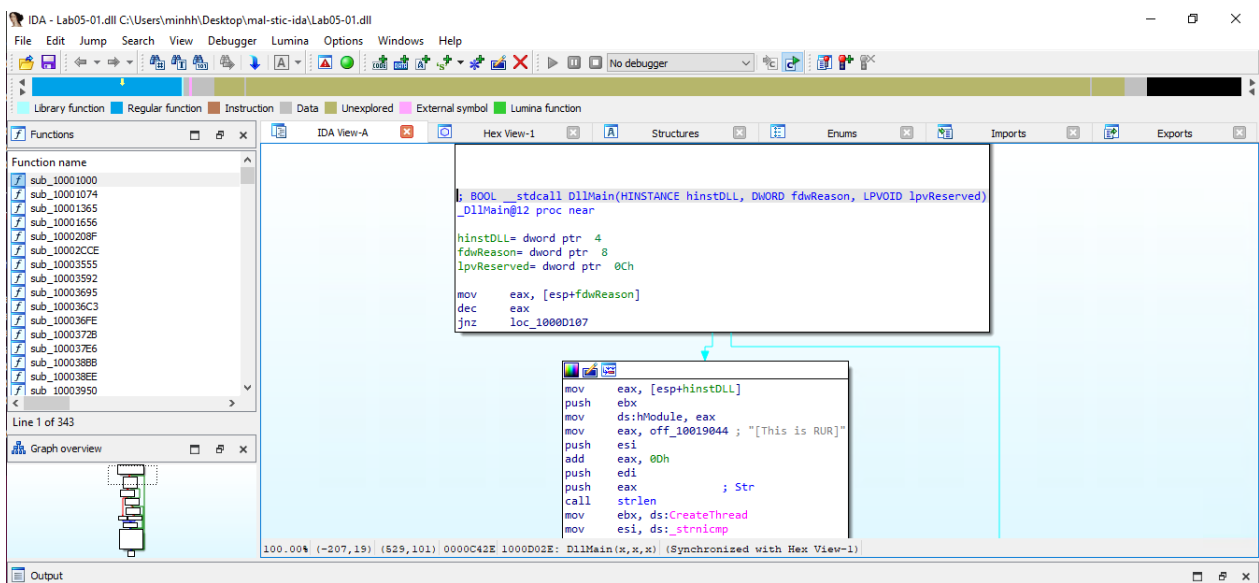
Sinh viên kéo thả file vào công cụ CFF Explorer để xác định định dạng file để chọn IDA 32 hay IDA 64 cho phù hợp



*Hình 18: CFF Explorer*

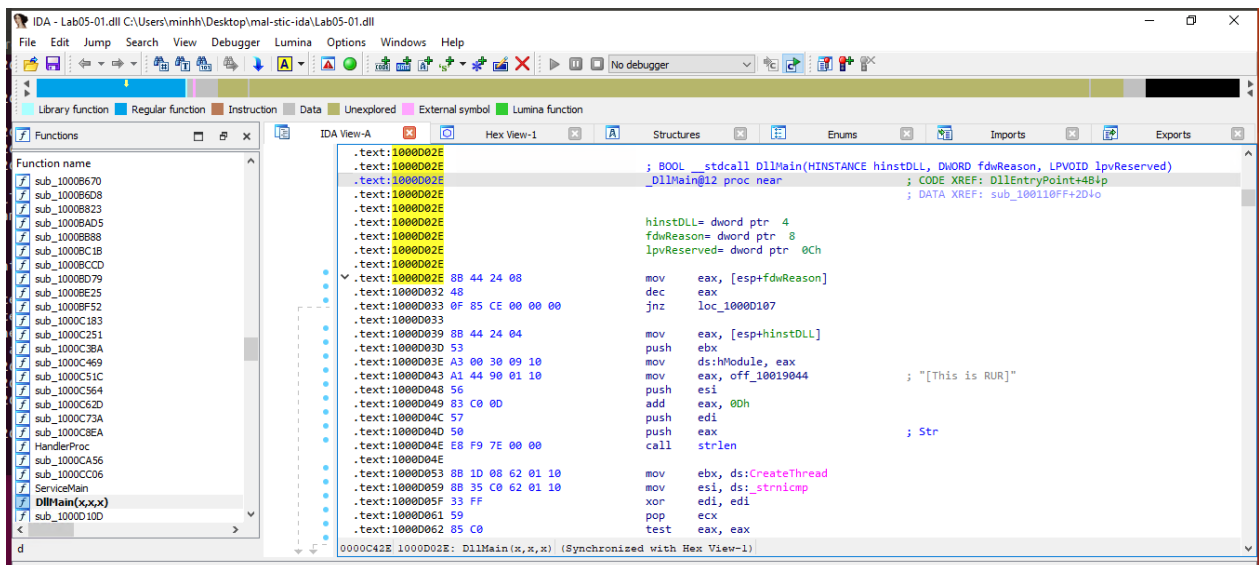
Sau khi xác định được định dạng file thì ta mở file đã cho với công cụ IDA Pro mà đã xác định. Sau khi mở thì sẽ hiển thị như bên dưới.





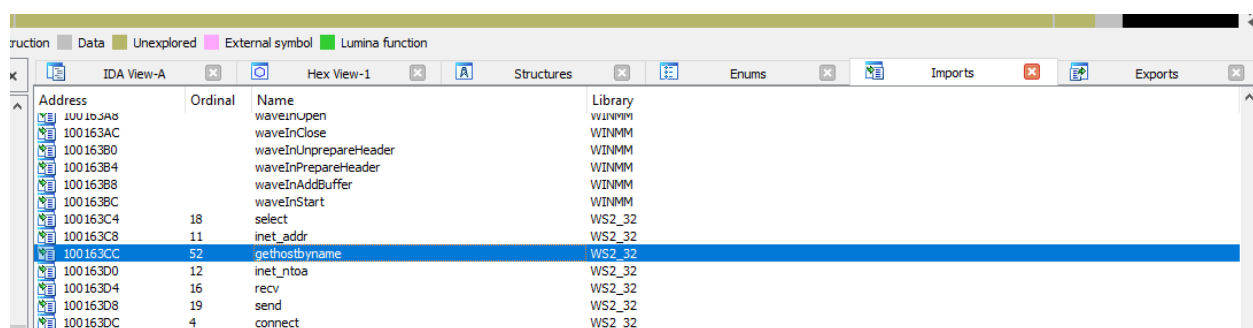
Hình 19: Giao diện IDA Pro sau khi mở file

Chuyển đổi sang Text View và tìm tới địa chỉ của `DllMain` để hoàn thành nhiệm vụ đầu tiên.



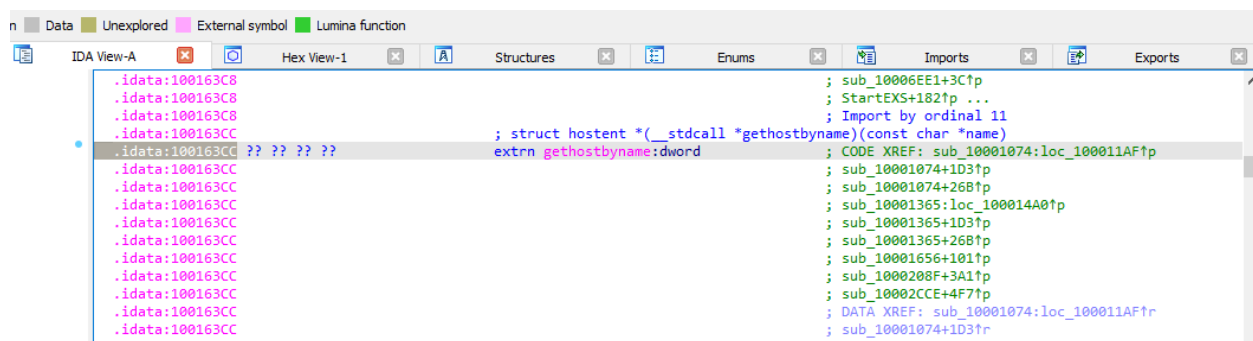
Hình 20: Text View DllMain

Sau đó mở import ra và tìm kiếm hàm **gethostbyname**.



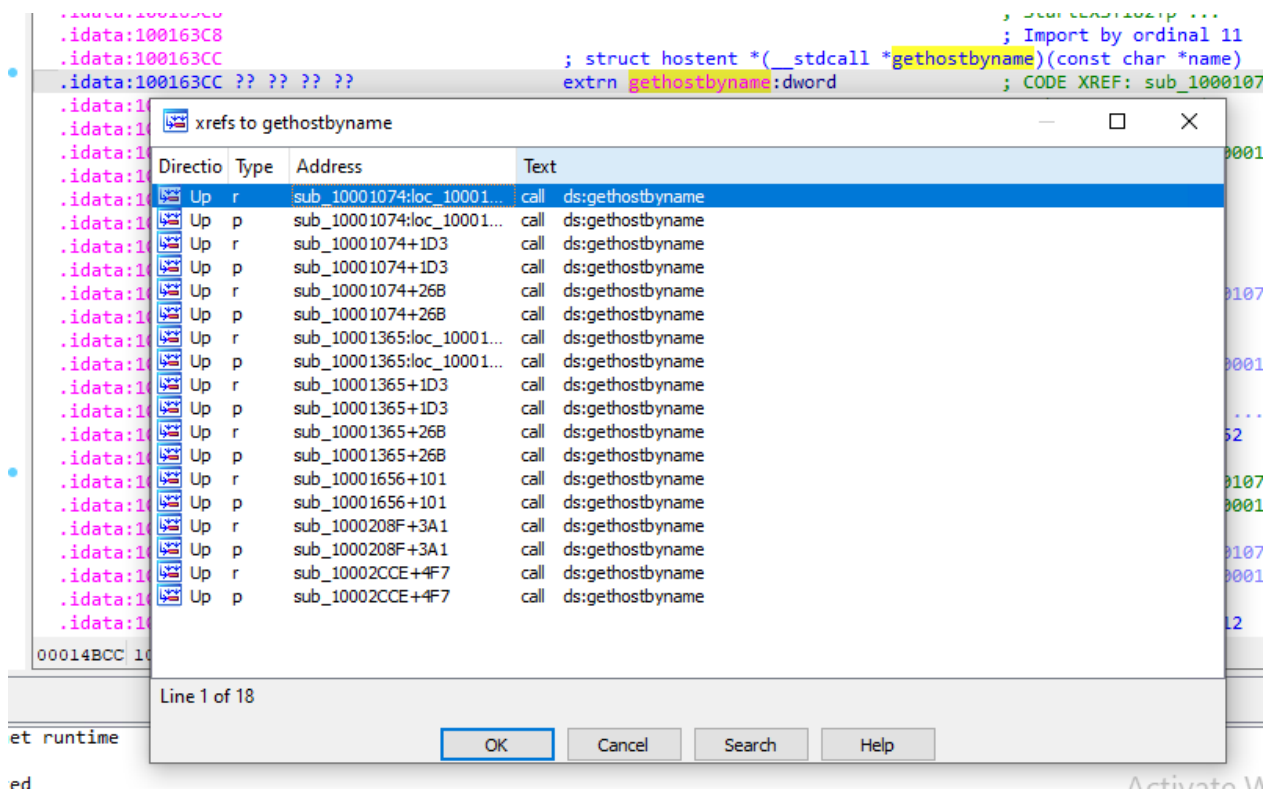
Hình 21: *Gethostbyname*

Nháy đúp vào để nhảy tới vị trí của **gethostbyname**, thu thập địa chỉ theo định dạng mà nhiệm vụ yêu cầu.



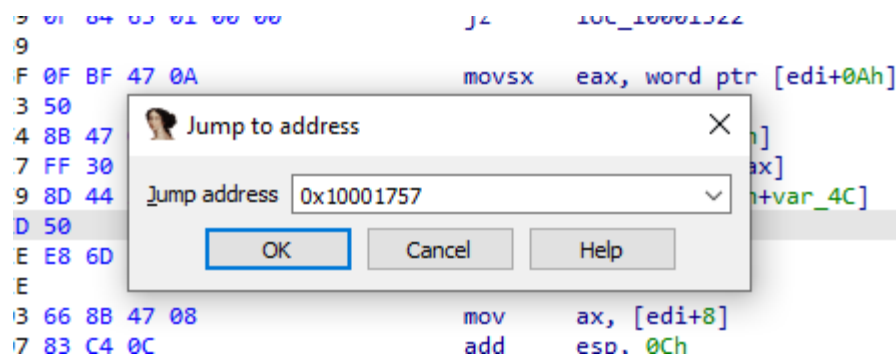
Hình 22: *Địa chỉ của gethostbyname*

Ấn vào chữ **gethostbyname** và ấn phím tắt X để xem bảng tham chiếu của hàm **gethostbyname**



Hình 23: Bảng tham chiếu của gethostbyname

Ấn phím tắt G và nhập địa chỉ cho sẵn là **0x10001757** vào để đi tới hàm cần đọc.



Hình 24: Jump to address

Sau khi nhảy tới địa chỉ đã cho, ta thấy có một offset gồm một chuỗi là *[This is RDO]pics.practicalmalwareanalysis.com*. Đọc tiếp câu lệnh dưới ta thấy add thêm 0DH chính là 13 và 0 trong eax nơi chứa offset mà chứa chuỗi kia. Có nghĩa là con trỏ bây giờ trở tới vị trí thứ 13 của chuỗi, và bỏ qua đoạn *[This is RDO]* từ đây ta thu được DNS mà nhiệm vụ yêu cầu là *pics.practicalmalwareanalysis.com*.

```

.text:1000173L
.text:10001742 39 1D CC E5 08 10      cmp     dword_1008E5CC, ebx
.text:10001748 0F 85 9F 00 00 00      jnz     loc_100017ED
.text:10001748
.text:1000174E A1 40 90 01 10      mov     eax, off_10019040      ; "[This is RDO]pics.practicalmalwareanalys"...
.text:10001753 83 C0 00      add     eax, 13
.text:10001756 50      push    eax                    ; name
.text:10001757 FF 15 CC 63 01 10      call    ds:gethostbyname
.text:10001757
.text:1000175D 8B F0      mov     esi, eax

```

Hình 25: DNS được trở tới

Ta tiếp tục ấn phím G để nhảy tới địa chỉ **0x10001656** được cho bởi nhiệm vụ, và thu thập thông tin về biến cục bộ, tham số của nó.

```

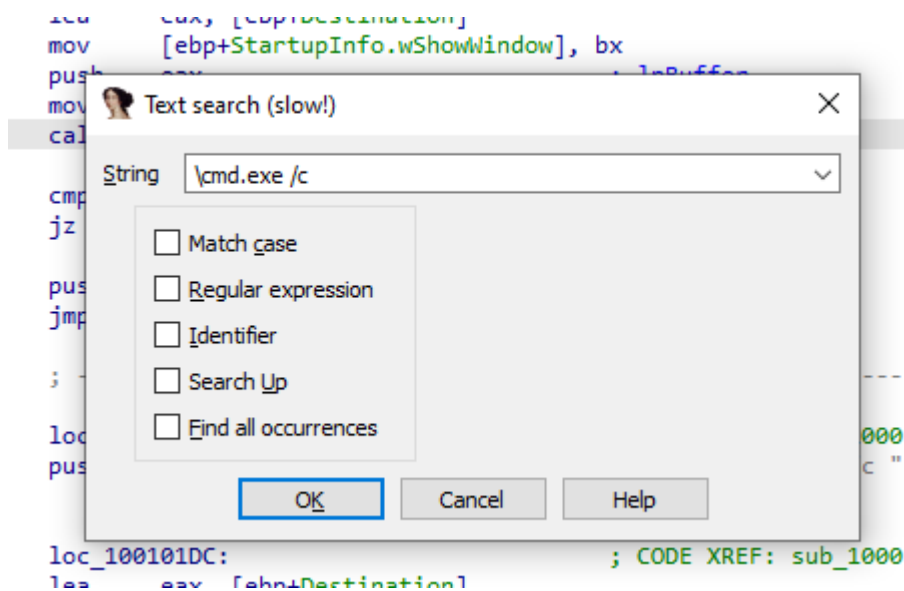
.text:10001656
.text:10001656      sub_10001656 proc near      ; DATA XREF: DllMain(x,x,x)+C840
.text:10001656
.text:10001656      var_675= byte ptr -675h
.text:10001656      var_674= dword ptr -674h
.text:10001656      hModule= dword ptr -670h
.text:10001656      timeout= timeval ptr -66Ch
.text:10001656      name= sockaddr ptr -664h
.text:10001656      var_654= word ptr -654h
.text:10001656      in= in_addr ptr -650h
.text:10001656      Str1= byte ptr -644h
.text:10001656      var_640= byte ptr -640h
.text:10001656      CommandLine= byte ptr -63Fh
.text:10001656      Str= byte ptr -63Dh
.text:10001656      var_638= byte ptr -638h
.text:10001656      var_637= byte ptr -637h
.text:10001656      var_544= byte ptr -544h
.text:10001656      var_50C= dword ptr -50Ch
.text:10001656      var_500= byte ptr -500h
.text:10001656      Buf2= byte ptr -4FCh
.text:10001656      readfds= fd_set ptr -48Ch
.text:10001656      buf= byte ptr -388h
.text:10001656      var_380= dword ptr -380h
.text:10001656      var_1A4= dword ptr -1A4h
.text:10001656      var_194= dword ptr -194h
.text:10001656      WSADATA= WSADATA ptr -190h
.text:10001656      lpThreadParameter= dword ptr 4
.text:10001656

```

Hình 26: Biến cục bộ và tham số của hàm

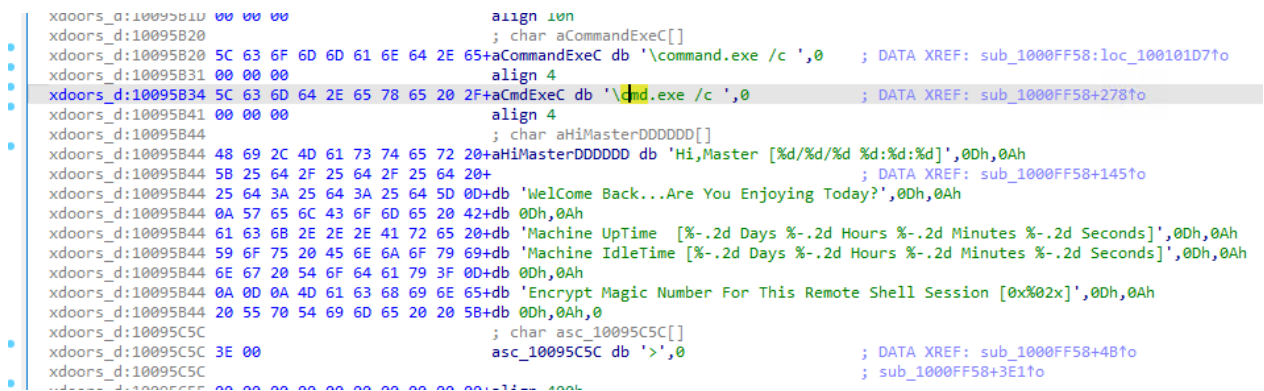
Từ ảnh ta đếm ra được là có 23 biến cục bộ và chỉ có duy nhất 1 tham số là lpThreadParameter.

Để làm nhiệm vụ tiếp theo thì ấn Alt + T để mở bảng tìm kiếm, sau đó nhập chuỗi **\cmd.exe /c** để tìm được địa chỉ của nó.



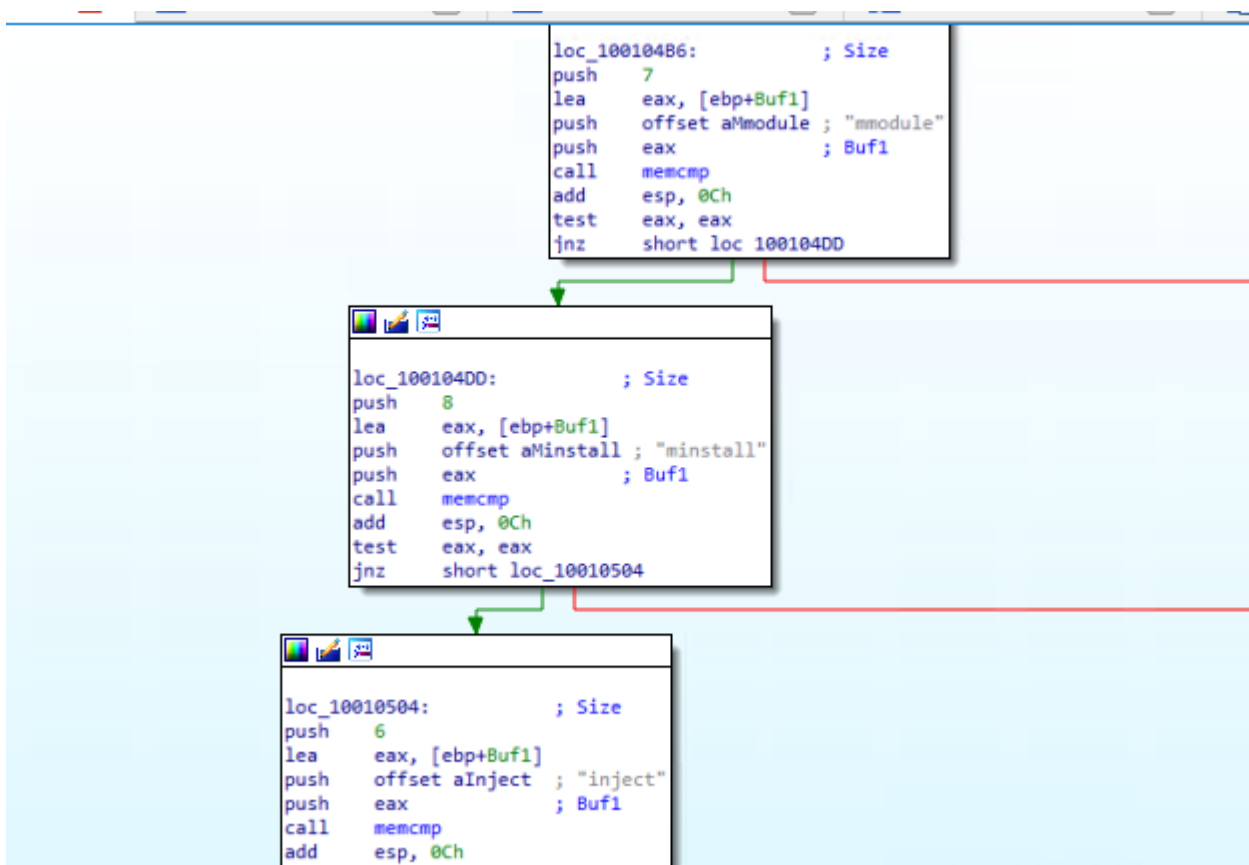
Hình 27: Bảng tìm kiếm Alt + T

Sau khi mở lên ta thu được thông tin về địa chỉ của biến chứa nội dung mà nhiệm vụ đã cho.



Hình 28: Địa chỉ của \\cmd.exe /c

Ta chuyển từ Text View về lại Graph View để lướt xem nội dung đoạn mã code làm gì sau khi có chuỗi kia.



Hình 29: Chuỗi lệnh thực thi

Ta thu được một vài gợi ý về việc thực thi từ xa như quit, exit, cd, inject,... Từ đó có thể đoán được đây là 1 Remote Shell, hoặc chúng ta lướt chuột lên phía trên thì có thấy 1 offset là aHiMasterDDDDDD có chứa cụm Welcome. Từ đây phán đoán được đây là đoạn chào giới thiệu, bắt đầu của shell. Ấn vào để thu thêm thông tin.

```

movzx   eax, [ebp+SystemTime.wDay]
push    eax
movzx   eax, [ebp+SystemTime.wMonth]
push    eax
movzx   eax, [ebp+SystemTime.wYear]
push    eax
lea     eax, [ebp+Str]
push    offset aHiMasterDDDDDD ; "Hi,Master [%d/%d/%d %d:%d:%d]\r\nWelCom"...
push    eax                ; Buffer
call    ds:sprintf
add     esp, 44h
xor     ebx, ebx
lea     eax, [ebp+Str]
push    ebx
push    eax                ; Str

```

Hình 30: aHiMasterDDDDDD

Ấn vào thì thấy được 1 chuỗi các ký tự và trong đó có cụm từ gây chú ý là This Remote Shell Section. Từ đây khẳng định đây là Remote Shell

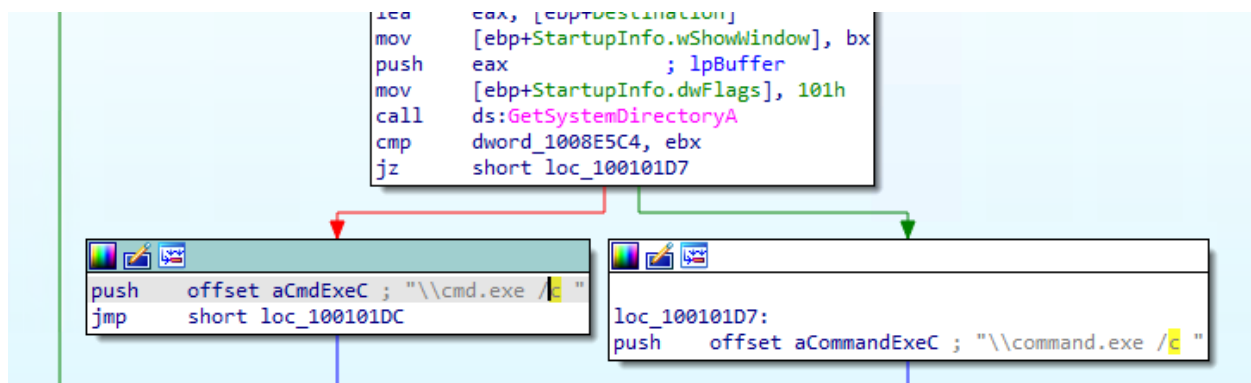
```

• xdoors_d:10095841 00 00 00 align 4
• xdoors_d:10095844 48 69 2C 4D 61 73 74 65 72 20 aHiMasterDDDDDD db 'Hi,Master [%d/%d/%d %d:%d:%d]',0Dh,0Ah
; char aHiMasterDDDDDD[]
; DATA XREF: sub_1000FF58+145to
xdoors_d:10095844 58 25 64 2F 25 64 2F 25 64 20+
xdoors_d:10095844 25 64 3A 25 64 3A 25 64 5D 0D+db 'WelCome Back...Are You Enjoying Today?',0Dh,0Ah
; DATA XREF: sub_1000FF58+145to
xdoors_d:10095844 0A 57 65 6C 43 6F 6D 65 20 42+db 0Dh,0Ah
xdoors_d:10095844 61 63 68 2E 2E 2E 41 72 65 20+db 'Machine UpTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seconds]',0Dh,0Ah
xdoors_d:10095844 59 6F 75 20 45 6E 6A 6F 79 69+db 'Machine IdleTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seconds]',0Dh,0Ah
xdoors_d:10095844 6E 67 20 54 6F 64 61 79 3F 0D+db 0Dh,0Ah
xdoors_d:10095844 0A 0D 0A 4D 61 63 68 69 6E 65+db 'Encrypt Magic Number For This Remote Shell Session [0x%02x]',0Dh,0Ah
xdoors_d:10095844 20 55 70 54 69 6D 65 20 20 5B+db 0Dh,0Ah,0
; char asc_10095C5C[]
xdoors_d:10095C5C 3E 00 asc_10095C5C db '>',0
; DATA XREF: sub_1000FF58+4Bto
; sub_1000FF58+3E1to
xdoors_d:10095C5C

```

Hình 31: Nội dung aHiMasterDDDDDD

Quay trở về chỗ có câu lệnh `\cmd.exe /c` ta thấy được bên trên có 1 đoạn dùng để so sánh, để xem sẽ dùng `cmd.exe` hay `command.exe`.



Hình 32: So sánh cmd và command

Ta thấy chương trình so sánh `dword_1008E5C4` và `ebx`, dùng phím tắt X cho `dword_1008E5C4` để xem bảng tham chiếu, xem hàm nào đã gán giá trị vào cho nó.

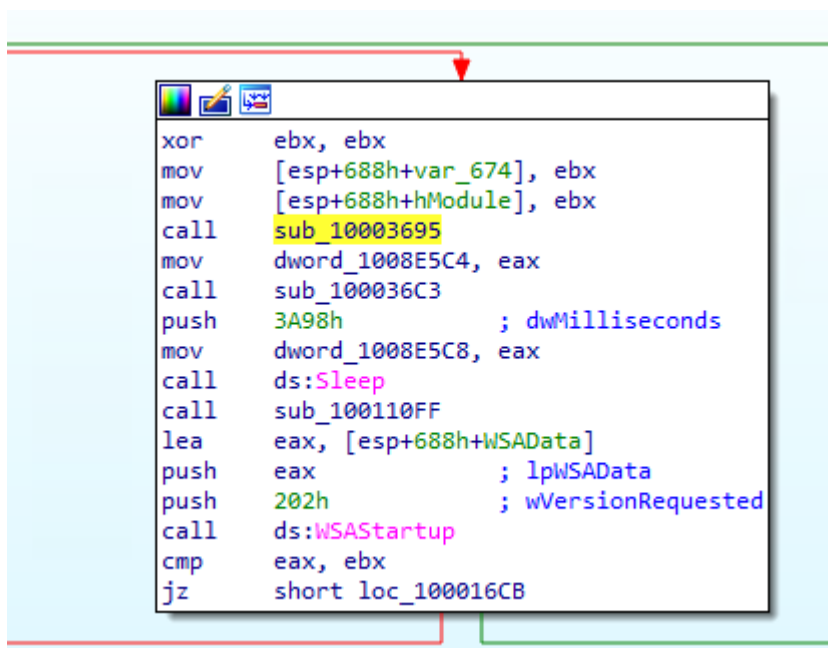
xrefs to dword_1008E5C4			
Direction	Type	Address	Text
Up	w	sub_10001656+22	mov dword_1008E5C4, eax
Up	r	sub_10007312+E	cmp dword_1008E5C4, edi
Up	r	sub_1000FF58+270	cmp dword_1008E5C4, ebx

Line 1 of 3

OK Cancel Search Help

Hình 33: Bảng tham chiếu của dword\_1008E5C4

Ta thấy trong bảng chỉ có 1 chữ W, nghĩ là chỉ có 1 hàm ghi dữ liệu vào cho dword\_1008E5C4, ấn đúp vào thì ta thấy được hàm gán giá trị cho nó là sub\_10003695.



Hình 34: Hàm gán dữ liệu cho dword\_1008E5C4

Mở hàm sub\_10003695 ra thì ta thấy hàm có gọi cho API GetVersionExA để thu thập thông tin về hệ điều hành máy chủ, thu thập về VersionInfoSize và Platform.

```

; Attributes: bp-based frame

sub_10003695 proc near

VersionInformation= _OSVERSIONINFOA ptr -94h

push    ebp
mov     ebp, esp
sub     esp, 94h
lea     eax, [ebp+VersionInformation]
mov     [ebp+VersionInformation.dwOSVersionInfoSize], 94h
push    eax             ; lpVersionInformation
call    ds:GetVersionExA
xor     eax, eax
cmp     [ebp+VersionInformation.dwPlatformId], 2
setz    al
leave
retn
sub_10003695 endp
  
```

Hình 35: Nội dung hàm sub\_10003695



Đọc mã của hàm thì về cơ bản ta thấy và phán đoán được rằng chương trình đang trả về cho dword\_1008E5C4 một giá trị là phiên bản của hệ điều hành máy nạn nhân đang dùng, và tiến hành so sánh xem dùng cmd.exe hay command.exe cho phù hợp.

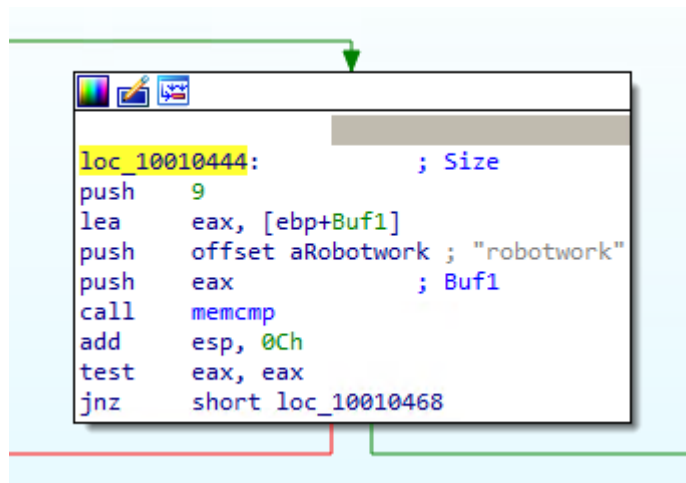
Ta thấy trong mã của hàm có lệnh so sánh với giá trị là 2. Lên trang chủ của Microsoft thì ta thấy được 2 là giá trị thể hiện rằng đây là WIN NT trở lên.

Name	Value	Description
Win32S	0	The operating system is Win32s. This value is no longer in use.
Win32Windows	1	The operating system is Windows 95 or Windows 98. This value is no longer in use.
Win32NT	2	The operating system is Windows NT or later.
WinCE	3	The operating system is Windows CE. This value is no longer in use.
Unix	4	The operating system is Unix.
Xbox	5	The development platform is Xbox 360. This value is no longer in use.
MacOSX	6	The operating system is Macintosh. This value was returned by Silverlight. On .NET Core, its replacement is <code>Unix</code> .
Other	7	Any other operating system. This includes Browser (WASM).

*Hình 36: Flatform của Microsoft*

Kết hợp lại các thông tin có được thì ta thấy rằng đoạn so sánh ban đầu là để xem hệ điều hành mà nạn nhân đang dùng có phải WIN NT trở lên hay không, nếu đúng thì file độc hại sẽ thực thi bằng lệnh của **cmd.exe** còn sai thì sẽ là **command.exe**. Từ đây điền vào file kết quả là **\\cmd.exe /c**.

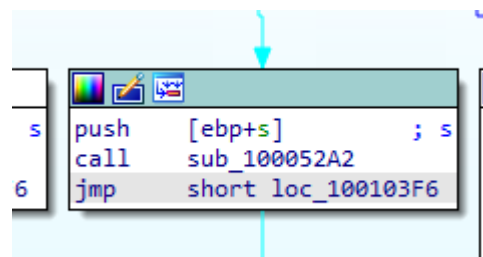
Kéo mã chương trình xuống một đoạn nữa thì ta có thấy 1 hàm dùng để so sánh một chuỗi với lại chuỗi **“robowork”**.



Hình 37: So sánh chuỗi với chuỗi robowork

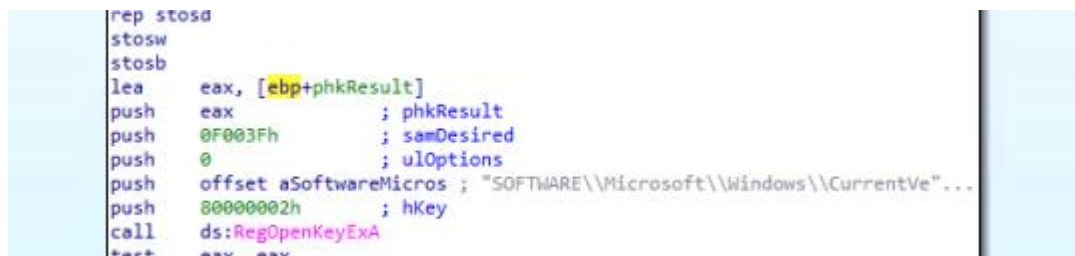
Ta thấy chương trình dùng memcmp để so sánh, hàm này sẽ trả về 0 nếu 2 chuỗi giống hệt nhau. Và nó dùng JNZ để nhảy, sẽ nhảy nếu kết quả không phải là không. Từ đó hiểu được nếu so sánh thành công thì bước nhảy sẽ không được thực hiện và đi theo đường màu đỏ.

Đi xuống theo đường màu đỏ ta thấy đoạn mã sau



Hình 38: Hàm nếu so sánh thành công

Ta thấy có gọi tới hàm sub\_100052A2, ấn đúp vào để xem nội dung hàm. Và ở ngay đoạn mã đầu tiên ta đã thấy có 1 giá trị của registry key. Kết hợp với đề bài thì ta có thể lấy được luôn đáp án là **SOFTWARE\Microsoft\Windows\CurrentVersion**.



Hình 39: Registry Key của hàm

Ấn F5 để chuyển sang ngôn ngữ giả C để nhìn tổng quan hơn mà không cần lướt lên lướt xuống nhiều lần.

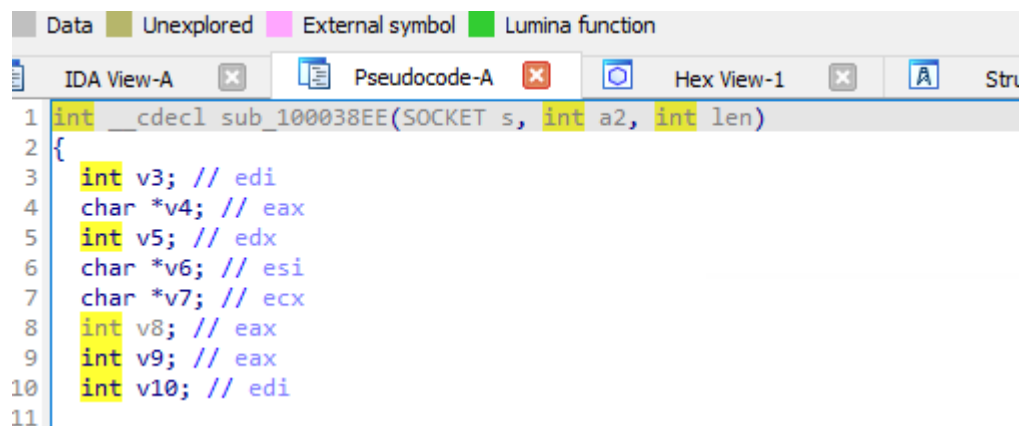
```

4
5  memset(Buffer, 0, sizeof(Buffer));
6  memset(Data, 0, sizeof(Data));
7  v7 = 0;
8  v8 = 0;
9  if ( RegOpenKeyEx(HKEY_LOCAL_MACHINE, aSoftwareMicros, 0, 0xF003Fu, &phkResult) )
0      return RegCloseKey(phkResult);
1  if ( !RegQueryValueEx(phkResult, aWorktime, 0, &Type, Data, &cbData) )
2  {
3      v2 = atoi((const char *)Data);
4      sprintf(Buffer, "\r\n\r\n[Robot_WorkTime :] %d\r\n\r\n", v2);
5      v3 = strlen(Buffer);
6      sub_100038EE(s, (int)Buffer, v3);
7  }
8  memset(Data, 0, sizeof(Data));
9  if ( !RegQueryValueEx(phkResult, aWorktimes, 0, &Type, Data, &cbData) )
0  {
1      v4 = atoi((const char *)Data);
2      sprintf(Buffer, "\r\n\r\n[Robot_WorkTimes:] %d\r\n\r\n", v4);
3      v5 = strlen(Buffer);
4      sub_100038EE(s, (int)Buffer, v5);
5  }
6  return RegCloseKey(phkResult);
7  }

```

Hình 40: Giả C của sub\_100052A2

Từ đoạn mã ta có thể biết được rằng file độc hại đang cố gắng thu thập thông tin về **WorkTime** và **WorkTimes** trong Registry Key **CurrentVersion**.



```

1  int __cdecl sub_100038EE(SOCKET s, int a2, int len)
2  {
3      int v3; // edi
4      char *v4; // eax
5      int v5; // edx
6      char *v6; // esi
7      char *v7; // ecx
8      int v8; // eax
9      int v9; // eax
10     int v10; // edi
11

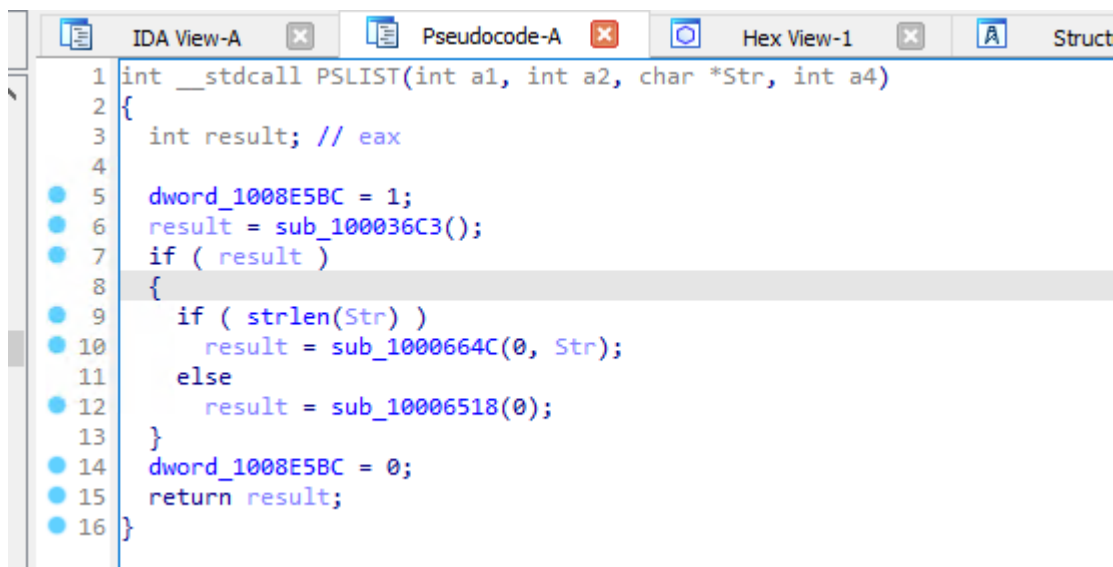
```

Hình 41: Hàm Sub\_100038EE

Ở trong hàm sub\_100052A2 có gọi tới hàm con sub\_100038EE 2 lần sau mỗi lần lấy và in ra thông tin qua lệnh sprintf. Và đầu khai báo của hàm này ta thấy có khai báo

**SOCKET.** Từ đây ta đoán được rằng có thể chương trình đang muốn gửi thông tin WorkTime và WorkTimes về cho kẻ tấn công.

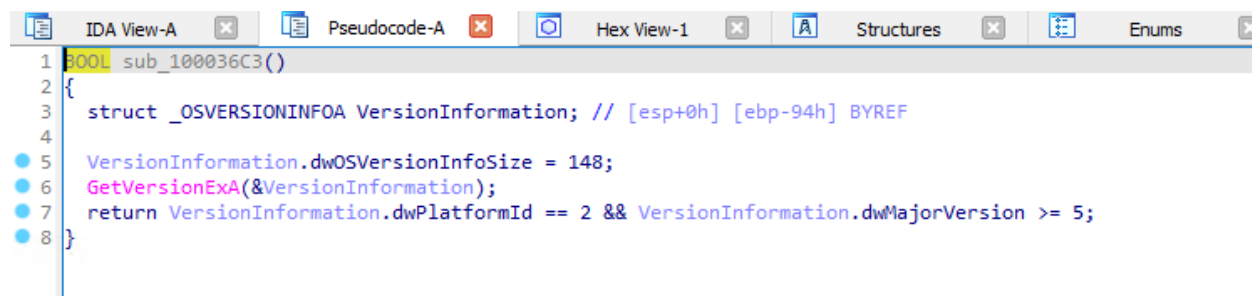
Làm nhiệm vụ tiếp theo thì ta mở bảng Export và ấn đúp vào hàm PSLIST, tiến hành đọc mã hàm PSLIST.



```
1 int __stdcall PSLIST(int a1, int a2, char *Str, int a4)
2 {
3     int result; // eax
4
5     dword_1008E5BC = 1;
6     result = sub_100036C3();
7     if ( result )
8     {
9         if ( strlen(Str) )
10             result = sub_1000664C(0, Str);
11         else
12             result = sub_10006518(0);
13     }
14     dword_1008E5BC = 0;
15     return result;
16 }
```

Hình 42: Nội dung PSLIST

Từ trên ta thấy PSLIST gọi tới 3 hàm nhỏ bên trong.



```
1 BOOL sub_100036C3()
2 {
3     struct _OSVERSIONINFOA VersionInformation; // [esp+0h] [ebp-94h] BYREF
4
5     VersionInformation.dwOSVersionInfoSize = 148;
6     GetVersionExA(&VersionInformation);
7     return VersionInformation.dwPlatformId == 2 && VersionInformation.dwMajorVersion >= 5;
8 }
```

Hình 43: Sub\_100036C3

Hàm trên dùng để kiểm tra lại phiên bản của hệ điều hành 1 lần nữa.

```

2  if ( hSnapshot == (HANDLE)-1 )
3  {
4      SetLastError = GetLastError();
5      sprintf(Buffer, "\r\n\r\nCreateToolhelp32Snapshot Fail:Error%d", GetLastError);
6      sub_1000388B(s, Buffer);
7      return 1;
8  }
9  else
10 {
11     pe.dwSize = 296;
12     sprintf(Buffer, aProcessidProce);
13     sub_1000388B(s, Buffer);
14     if ( dword_1008E5BC )
15         sub_1000620C(aProcessidProce, a1);
16     if ( Process32First(hSnapshot, &pe) )
17     {
18         do
19         {
20             v5 = strlen(Str);
21             if ( !strnicmp(pe.szExeFile, Str, v5) )
22             {
23                 v6 = OpenProcess(0x410u, 0, pe.th32ProcessID);
24                 EnumProcessModules(v6, v8, 4096, v14);
25                 memset(v12, 0, sizeof(v12));
26                 GetModuleFileNameExA(v6, v8[0], v12, 260);
27                 sprintf(Buffer, "\r\n%-16d%-20s%d", pe.th32ProcessID, pe.szExeFile, pe.cntThreads);
28                 sub_1000388B(s, Buffer);
29                 sprintf(Buffer, "\r\n[%s]", v12);

```

Hình 44: Sub\_1000664C

```

v7 = 0;
hSnapshot = CreateToolhelp32Snapshot(2u, 0);
if ( hSnapshot != (HANDLE)-1 )
{
    pe.dwSize = 296;
    if ( dword_1008E5BC )
        sub_1000620C(aProcessidProce, v3);
    for ( i = Process32First(hSnapshot, &pe); i; i = Process32Next(hSnapshot, &pe) )
    {
        v1 = OpenProcess(0x410u, 0, pe.th32ProcessID);
        EnumProcessModules(v1, v4, 4096, v9);
        memset(v5, 0, 0x104u);
        GetModuleFileNameExA(v1, v4[0], v5, 1024);
        if ( dword_1008E5BC )
            sub_1000620C(a16d20sDS, pe.th32ProcessID);
        CloseHandle(v1);
    }
}
CloseHandle(hSnapshot);
return 0;

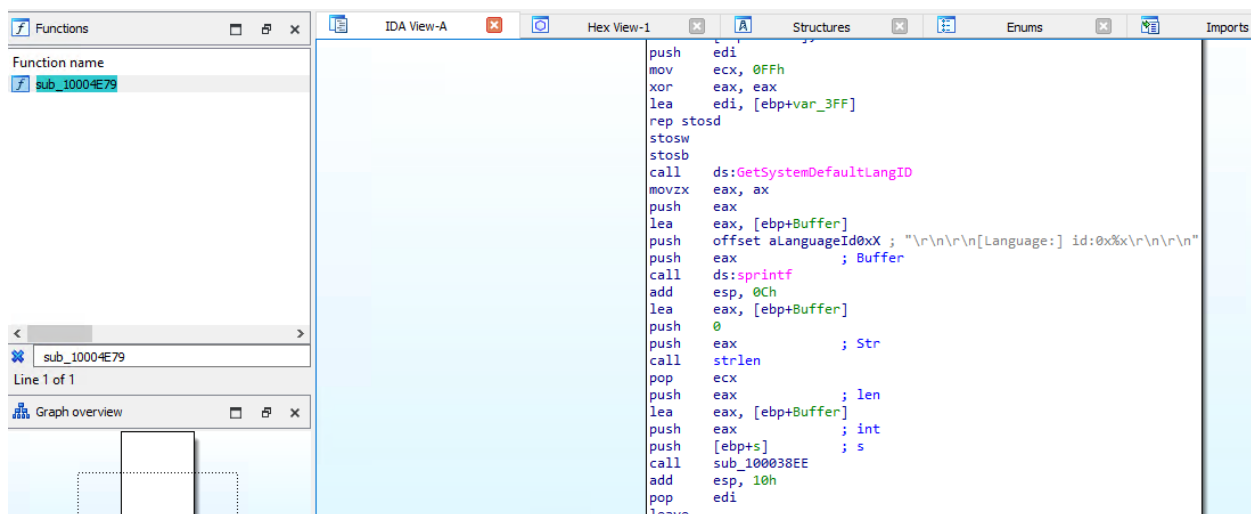
```

Hình 45: Sub\_10006518

Và 2 hàm sub\_1000664C và sub\_10006518 đều là dùng Snapshot để chụp nhanh các tiến trình được chỉ định và các thông tin liên quan, sau đó truy vấn các thông tin như ID, tên và số lượng các luồng của tiến trình đang chạy. Từ trên và tên hàm có thể kết

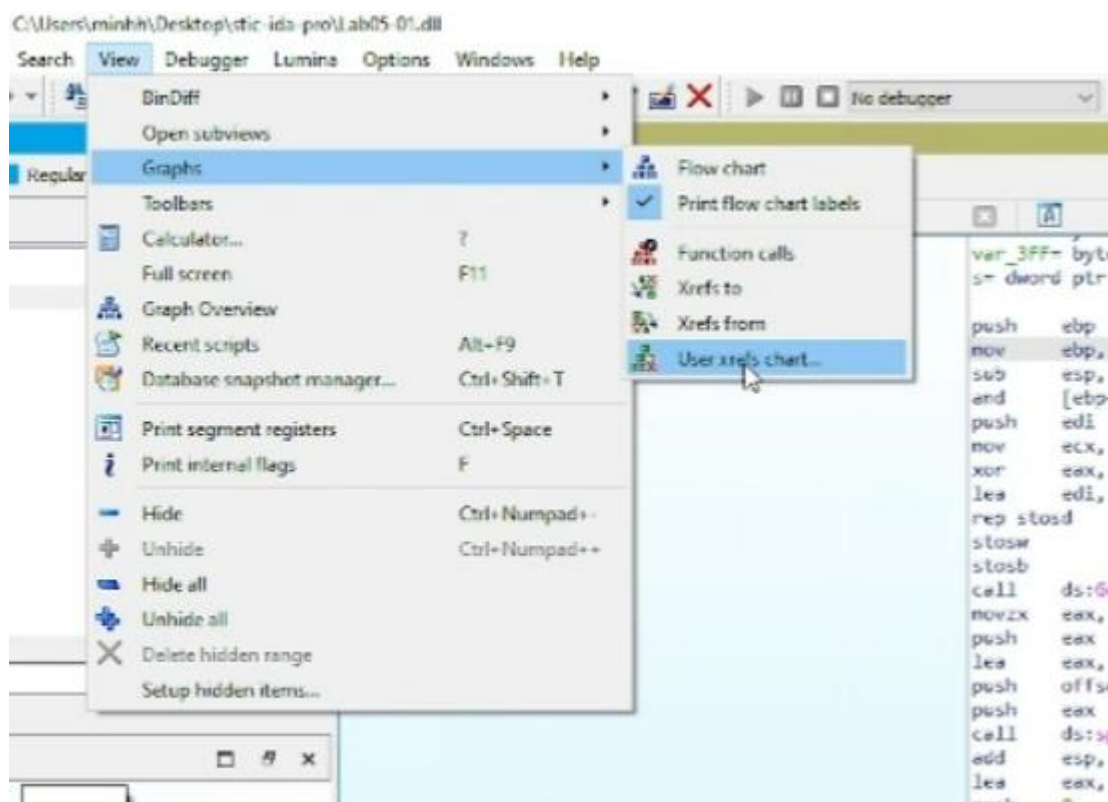
luyện được chức năng của hàm PSLIST là List running process, liệt kê các tiến trình đang chạy của hệ thống.

Để làm nhiệm vụ tiếp theo thì ta tìm hàm **sub\_10004E79** ở cửa sổ Functions.



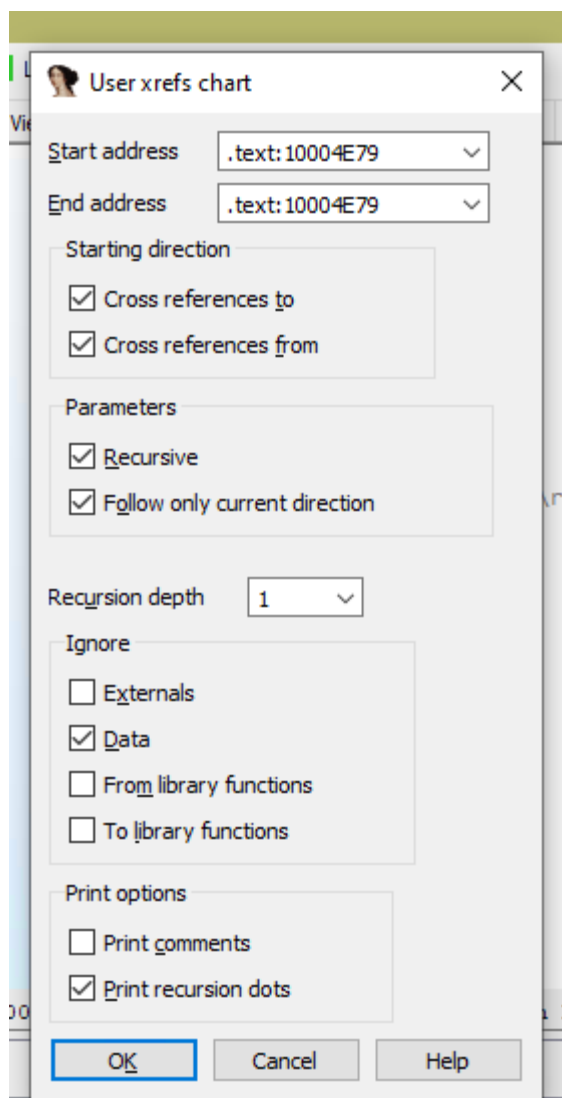
Hình 46: Hàm sub\_10004E79

Để xem sơ đồ mà hàm gọi tới các hàm khác, API khác thì ta dùng User xrefs chart để xem.



Hình 47: User xrefs chart

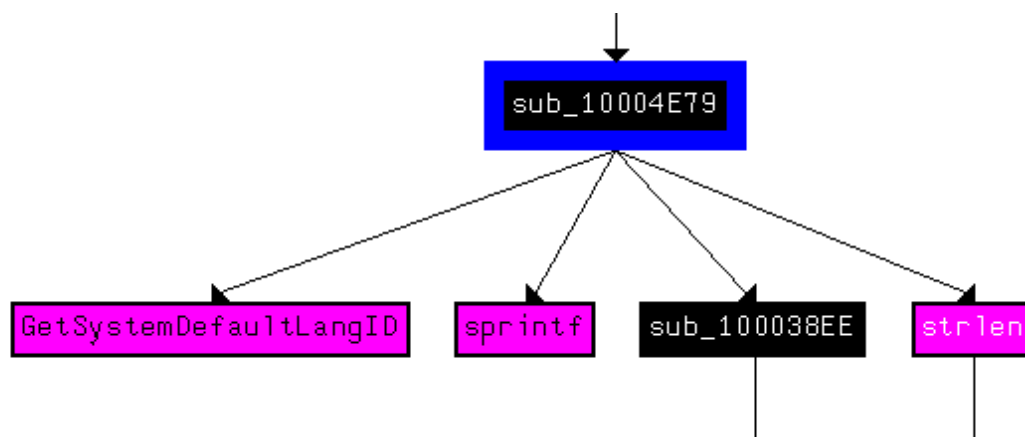
Để xem được các hàm, API mà hàm **sub\_10004E79** trực tiếp gọi thì ta chọn độ sâu là 1.



*Hình 48: Recursion depth*

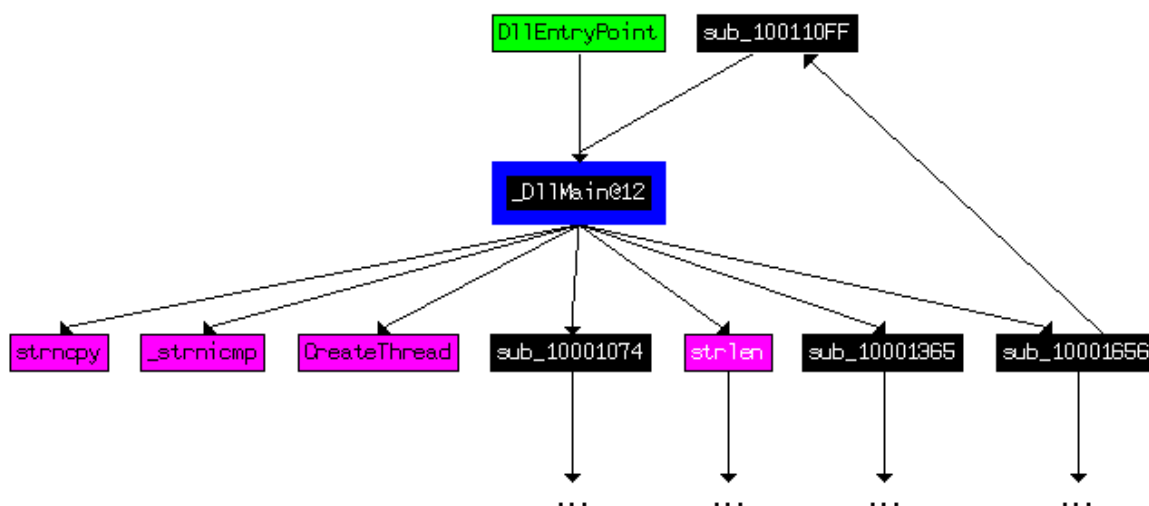
Lúc này IDA Pro sẽ hiển thị lên cho chúng ta xem sơ đồ các API, hàm được chương trình chúng ta muốn xem gọi trực tiếp tới





Hình 49: Sơ đồ của `sub_10004E79`

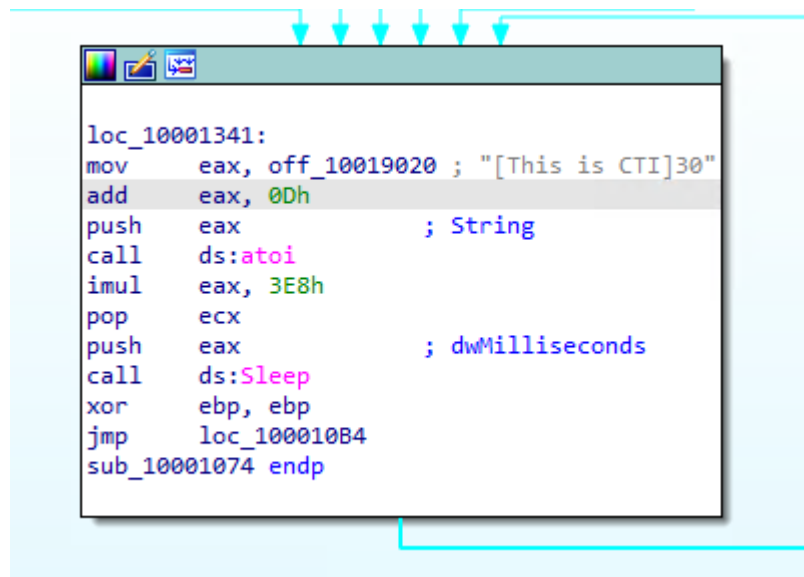
Từ đây ta thấy được tổng cộng chỉ có 1 API được gọi là `GetSystemDefaultLangID`. Tương tự cho hàm `DllMain` ta thu được sơ đồ.



Hình 50: Sơ đồ của `DllMain`

Từ đây ta thấy được `DllMain` cũng chỉ gọi trực tiếp tới 1 API duy nhất là `CreateThread`.

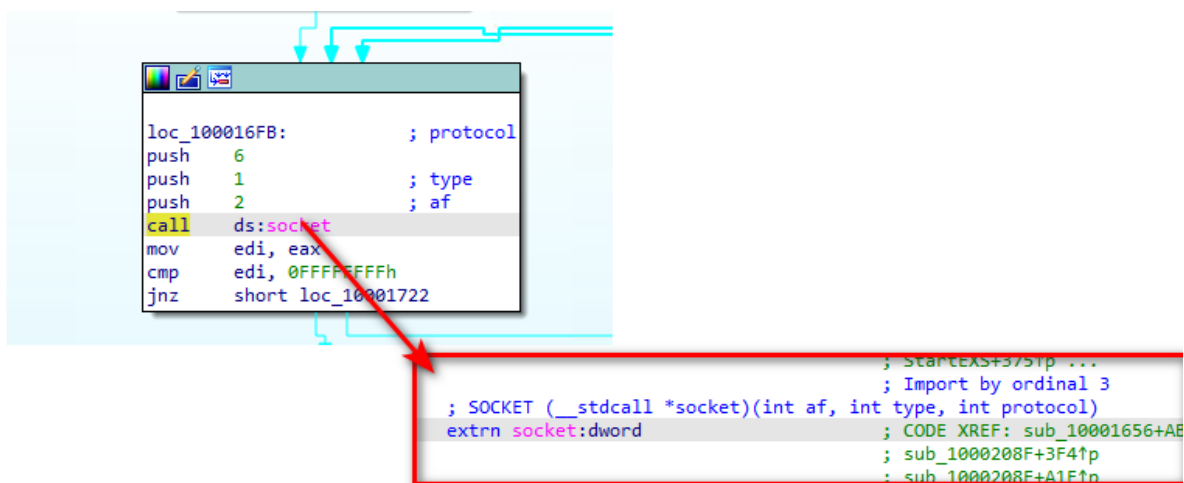
Ta dùng phím tắt G để nhảy tới địa chỉ **0x10001358**, từ đó thu được mã chương trình như sau



Hình 51: Mã hàm Sleep

Ở trên ta lại thấy có 1 offset chứa nội dung là `[This is CTI]30`. Và bên dưới đây cũng mov vào eax 0Dh hay chính là 13 như lúc ta lấy DNS, từ đó ta thấy được eax lúc này chỉ nhận giá trị là 30. Ở bên dưới ta lại thấy `imul`, đây là toán tử nhân, và nó nhân eax cho 3E8h là 1000 ra được 30000, và đơn vị ở đây là ms. Nhiệm vụ yêu cầu đổi ra đơn vị giây từ đó ta thấy là chương trình Sleep 30 giây.

Dùng phím tắt G tới địa chỉ **0x10001701** ta thu được code như bên dưới.



Hình 52: Tham số truyền vào Socket

Ta thấy có 3 số lần lượt được đẩy vào stack là 6, 1, 2. Và tương ứng với các giá trị mà Socket nhận là protocol, type, và af. Theo mẫu file ATTT.txt cho sẵn thì kết quả cần điền sẽ là: protocol: 6, type: 1, af: 2

Và đến nhiệm vụ cuối cùng. Chúng ta dùng phím G để nhảy tới địa chỉ **0x1001D988** từ đây thu được một đoạn gồm các giá trị sau.

.data:1001D987 00	uu 0
.data:1001D988 2D	db 2Dh ; -
.data:1001D989 31	db 31h ; 1
.data:1001D98A 3A	db 3Ah ; :
.data:1001D98B 3A	db 3Ah ; :
.data:1001D98C 27	db 27h ; '
.data:1001D98D 75	db 75h ; u
.data:1001D98E 3C	db 3Ch ; <
.data:1001D98F 26	db 26h ; &
.data:1001D990 75	db 75h ; u
.data:1001D991 21	db 21h ; !
.data:1001D992 3D	db 3Dh ; =
.data:1001D993 3C	db 3Ch ; <
.data:1001D994 26	db 26h ; &
.data:1001D995 75	db 75h ; u
.data:1001D996 37	db 37h ; 7
.data:1001D997 34	db 34h ; 4
.data:1001D998 36	db 36h ; 6
.data:1001D999 3E	db 3Eh ; >
.data:1001D99A 31	db 31h ; 1
.data:1001D99B 3A	db 3Ah ; :
.data:1001D99C 3A	db 3Ah ; :
.data:1001D99D 27	db 27h ; '
.data:1001D99E 70	db 70h ; u

Hình 53: Giá trị của 0x1001D988

Ta có thể thấy đây là một chuỗi ký tự đã bị mã hoá. Và cần được giải mã, đề bài đã cho 1 file python để giải mã với nội dung như sau.

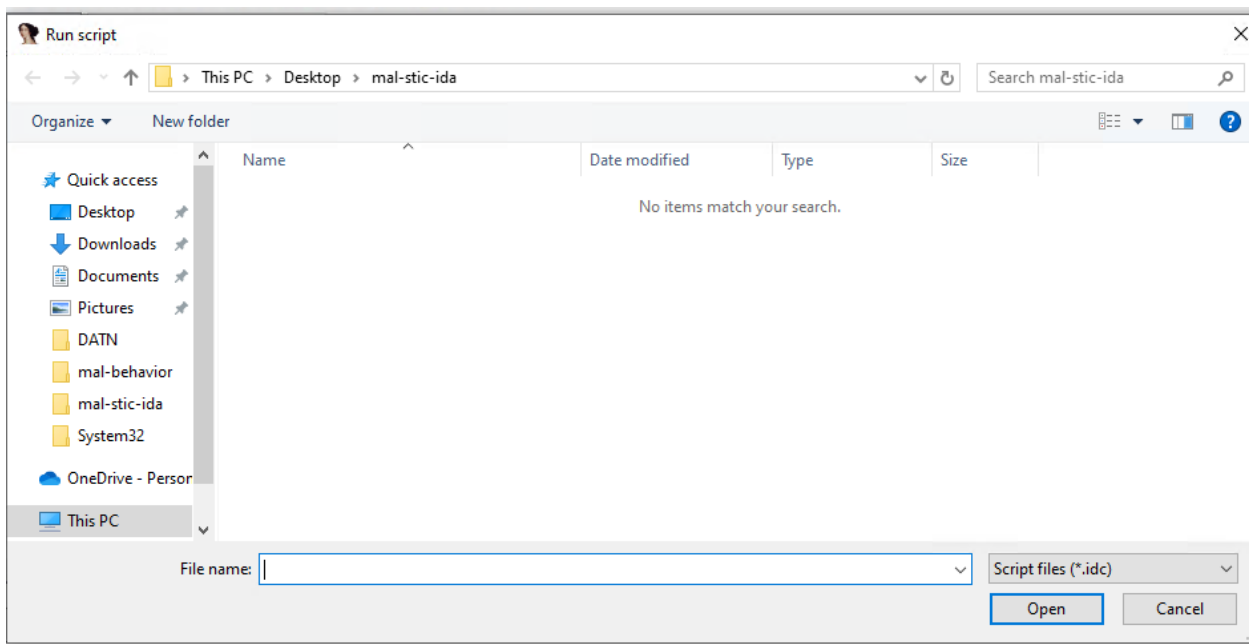
```
File Edit Format View Help
sea = ScreenEA()

for i in range(0x00,0x50):
    b = Byte(sea+i)
    decoded_byte = b ^ 0x55
    PatchByte(sea+i,decoded_byte)
```

Hình 54: Nội dung Lab05-1.py

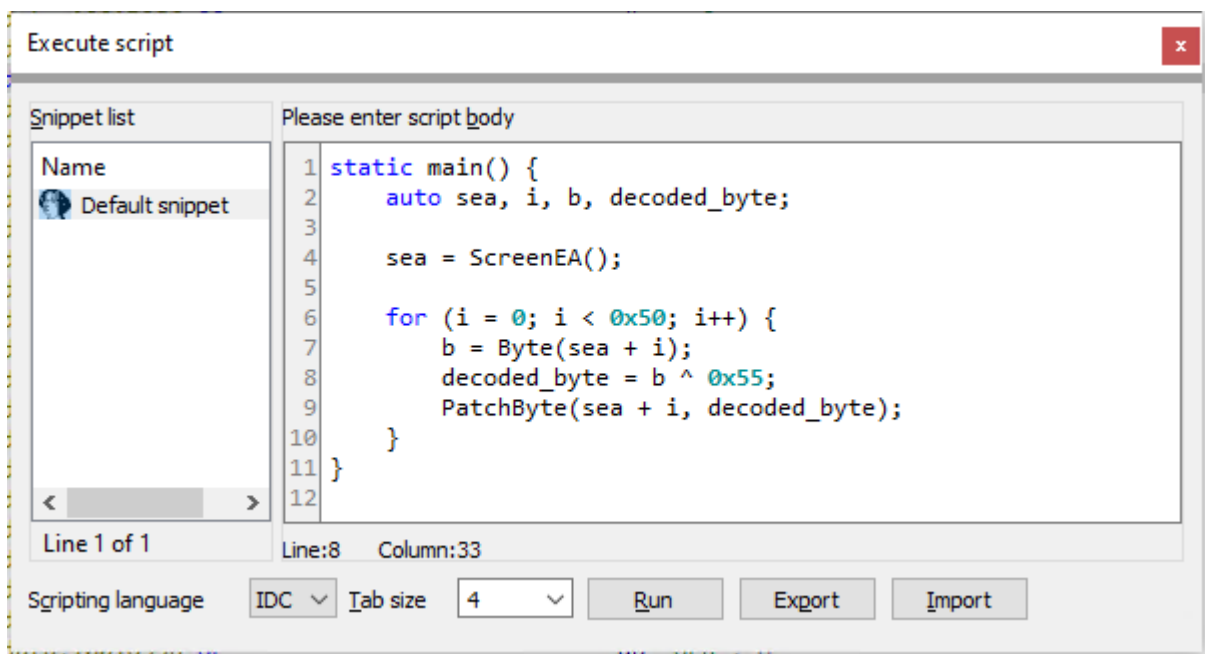
Nội dung chính là lấy 0x50 byte từ con trỏ hiện tại, và thực hiện XOR với 0x55 để thu được kết quả. Ta tính import script đây vào trong IDA nhưng thấy rằng IDA này

chỉ nhận file IDC, không nhận file Python. Ta có 2 cách để làm là cài đặt Python vào cho nó hoặc viết lại từ Python sang IDC. Và ở đây thì ta chọn viết lại sang IDC



*Hình 55: Script file chỉ nhận đuôi IDC*

Sau khi chuyển từ Python sang IDC thành công thì ta mở File chọn Script Command và nhập vào đây đoạn script đã chuyển đổi của mình và ấn run. Nhớ ấn chuột lên đầu đoạn thông tin, bởi vì script sẽ lấy từ vị trí trỏ chuột với 0x50 byte.



Hình 56: Script chuyển từ Python qua IDC

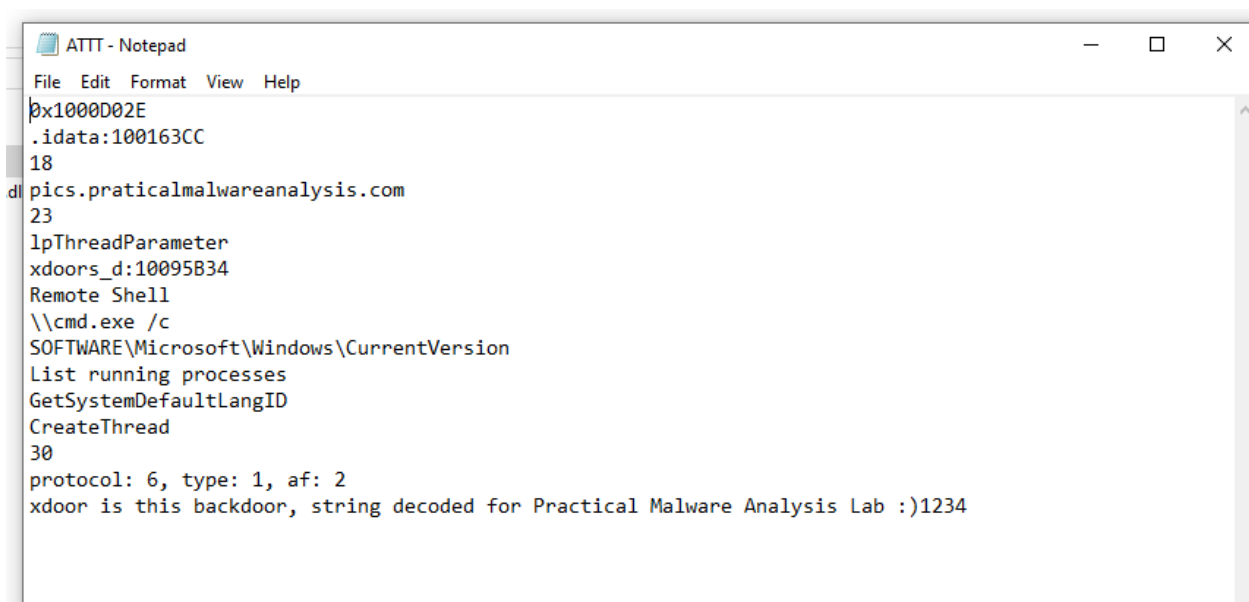
Sau khi chương trình chạy xong, ta ấn lên đầu đoạn văn bản rồi ấn phím A thì ta thu được kết quả như sau.

*xdoor is this backdoor, string decoded for Practical Malware Analysis Lab :)1234*

```
-- ~
db 0
db 0
3 20 74+aXdoorIsThisBac db 'xdoor is this backdoor, string decoded for Practical Malware Analysis Lab :)1234',0
db 0
db 0
"
```

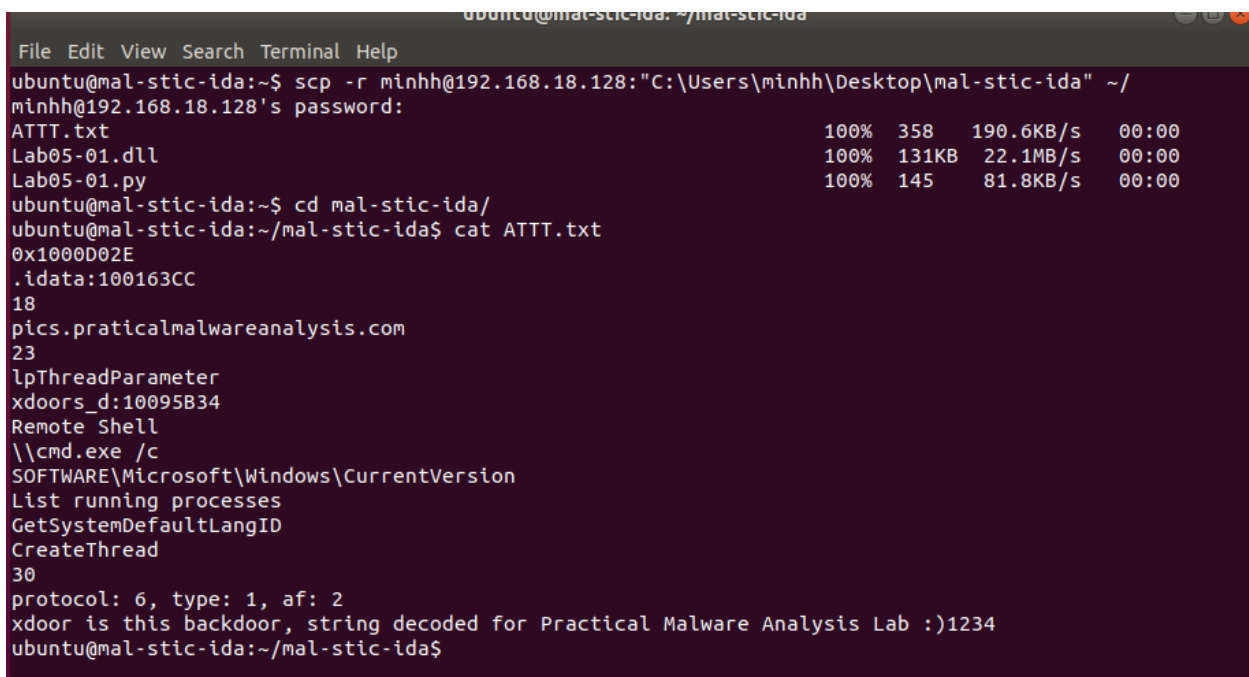
Hình 57: Thông điệp đã giải mã

Sau khi làm xong từng bước thì ta có ghi kết quả vào trong file ATTT.txt



Hình 58: Nội dung của ATTT.txt

Sau khi điền và lưu xong thông tin thì ta tắt cửa sổ của Windows 10 đi và kéo file ATTT.txt về máy Linux để kiểm tra lại lần cuối.



Hình 59: Kéo thư mục về máy Linux và kiểm tra

Tiến hành checkwork

```

student@ubuntu:~/labtainer/labtainer-student$ checkwork mal-stic-ida
Results stored in directory: /home/student/labtainer_xfer/mal-stic-ida
Labname mal-stic-ida

Student      final |      shell-cmd | variables-para |      call-api |      gethost |      adr-dllmain |      dns-request |      string-located |      pslist |      sleep-s |      socket |
===== | ===== | ===== | ===== | ===== | ===== | ===== | ===== | ===== | ===== | ===== |
B20DCAT061  |      Y |      Y |      Y |      Y |      Y |      Y |      Y |      Y |      Y |      Y |
b20dcatt061 |      Y |      Y |      Y |      Y |      Y |      Y |      Y |      Y |      Y |      Y |
What is automatically assessed for this lab:

```

## TÀI LIỆU THAM KHẢO

[ 1 ] Sikorski, M., & Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*.