Static Analysis On Linux

Through static analysis, we endeavor to extract pivotal information which includes:

- File type
- File hash
- Strings
- Embedded elements
- Packer information
- Imports
- Exports
- Assembly code

for file type

```
file malware.exe
```

file hash

`md5sum malware.exe`

then virusTotal and all these stuff bro

Import Hashing (IMPHASH)

`IMPHASH`, an abbreviation for "Import Hash", is a cryptographic hash calculated from the import functions of a Windows Portable Executable (PE) file. Its algorithm functions by first converting all imported function names to lowercase. Following this, the DLL names and function names are fused together and arranged in alphabetical order. Finally, an MD5 hash is generated from the resulting string. Therefore, two PE files with identical import functions, in the same sequence, will share an `IMPHASH` value.

```
import sys
import pefile
import peutils

pe_file = sys.argv[1]
pe = pefile.PE(pe_file)
imphash = pe.get_imphash()
```

```
print(imphash)
```

Fuzzy Hashing (SSDEEP)

`Fuzzy Hashing (SSDEEP)`, also referred to as context-triggered piecewise hashing (CTPH), is a hashing technique designed to compute a hash value indicative of content similarity between two files. This technique dissects a file into smaller, fixed-size blocks and calculates a hash for each block. The resulting hash values are then consolidated to generate the final fuzzy hash.

`ssdeep malware.exe`

The command line arguments `-pb` can be used to initiate matching mode in `SSDEEP`

means the matching of two malware samples

```
ssdeep -pb *
```

Section Hashing (Hashing PE Sections)

`Section hashing`, (hashing PE sections) is a powerful technique that allows analysts to identify sections of a Portable Executable (PE) file that have been modified. This can be particularly useful for identifying minor variations in malware samples, a common tactic employed by attackers to evade detection.

```
import sys
import pefile
pe_file = sys.argv[1]
pe = pefile.PE(pe_file)
for section in pe.sections:
    print (section.Name, "MD5 hash:", section.get_hash_md5())
    print (section.Name, "SHA256 hash:", section.get_hash_sha256())
```

String Analysis

`strings malware.exe`

Unpacking UPX-packed Malware

you know, first run strings command then if you see UPX in the top unpack it

`upx -d malware.exe`

Static Analysis On Windows

Get File md5 hash (win)

```
Get-FileHash -Algorithm MD5
C:\Samples\MalwareAnalysis\Ransomware.wannacry.exe
```

Floss

Same as strings

```
floss shell.exe
```

Unpacking UPX-packed Malware (Win)

```
upx -d -o unpacked_credential_stealer.exe
C:\Samples\MalwareAnalysis\packed\credential_stealer.exe
```

Dynamic Analysis

Dynamic Analysis With Noriben

Noriben is a powerful tool in our dynamic analysis toolkit, essentially acting as a Python wrapper for Sysinternals `ProcMon`, a comprehensive system monitoring utility. It orchestrates the operation of `ProcMon`, refines the output, and adds a layer of malware-specific intelligence to the process. Leveraging `Noriben`, we can capture malware behaviors more conveniently and understand them more precisely.

like ProcccessHacker and other stuff

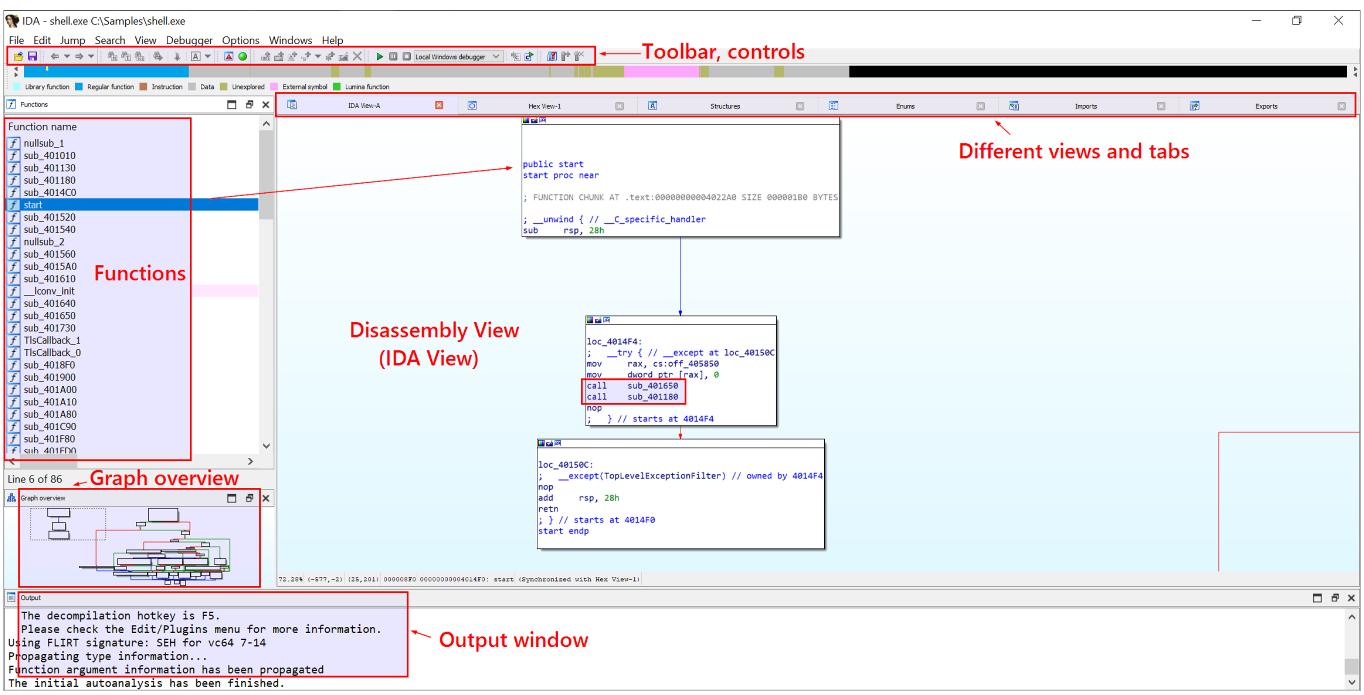
```
python .\Noriben.py
```

Then Active the malware. when you done shit procmon off

```
-=] Sandbox Analysis Report generated by Noriben v1.8.2
-=] Developed by Brian Baskin : brian @@ thebaskins.com  @bbaskin
-=] The latest release can be found at https ://github.com/Rurik/Noriben

-=] Execution time : 18.68 seconds
-=] Processing time : 4.41 seconds
-=] Analysis time : 7.58 seconds

Processes Created :
==================
[CreateProcess] powershell.exe : 2052 > "C:\Samples\shell.exe"[Child PID : 928]
[CreateProcess] shell.exe:928 > "%WinDir%\System32\cmd.exe /k ping [redacted] -n 5"[Child PID : 1636]
[CreateProcess] cmd.exe:1636 > "\??\%WinDir%\system32\conhost.exe 0xffffffff -ForceV1"[Child PID : 5376]
[CreateProcess] cmd.exe:1636 > "ping [redacted] -n 5"[Child PID : 9284]

File Activity :
==================
[CreateFile] svchost.exe : 2320 > % AllUsersProfile % \Microsoft\Windows\AppRepository\StateRepository -
[CreateFile] svchost.exe:2320 > % AllUsersProfile % \Microsoft\Windows\AppRepository\StateRepository - Ma
```

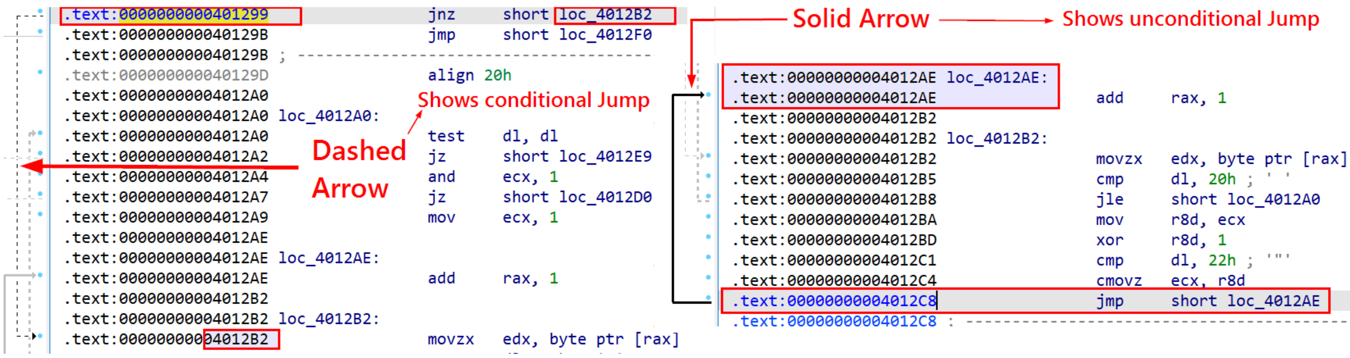
Code Analysis IDA

just Download IDA



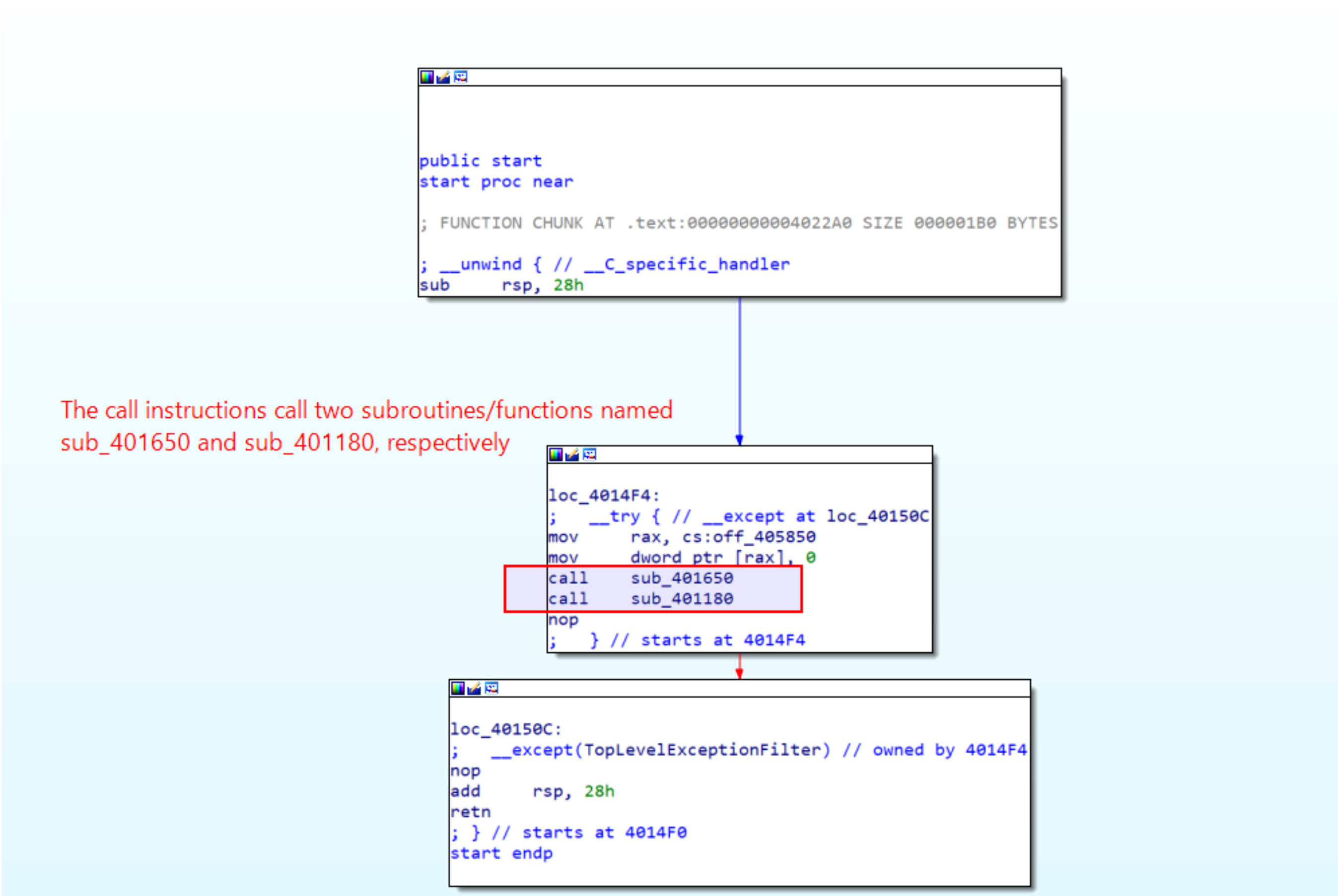
- `Solid Arrow (→)` : A solid arrow denotes a direct jump or branch instruction, indicating an unconditional shift in the program's flow where execution moves from one location to another. This occurs when a jump or branch instruction like `jmp` or `call` is encountered.

- `Dashed Arrow (---→)` : A dashed arrow represents a conditional jump or branch instruction, suggesting that the program's flow might change based on a specific condition. The destination of the jump depends on the condition's outcome. For instance, a `jz` (jump if zero) instruction will trigger a jump only if a previous comparison yielded a zero value.

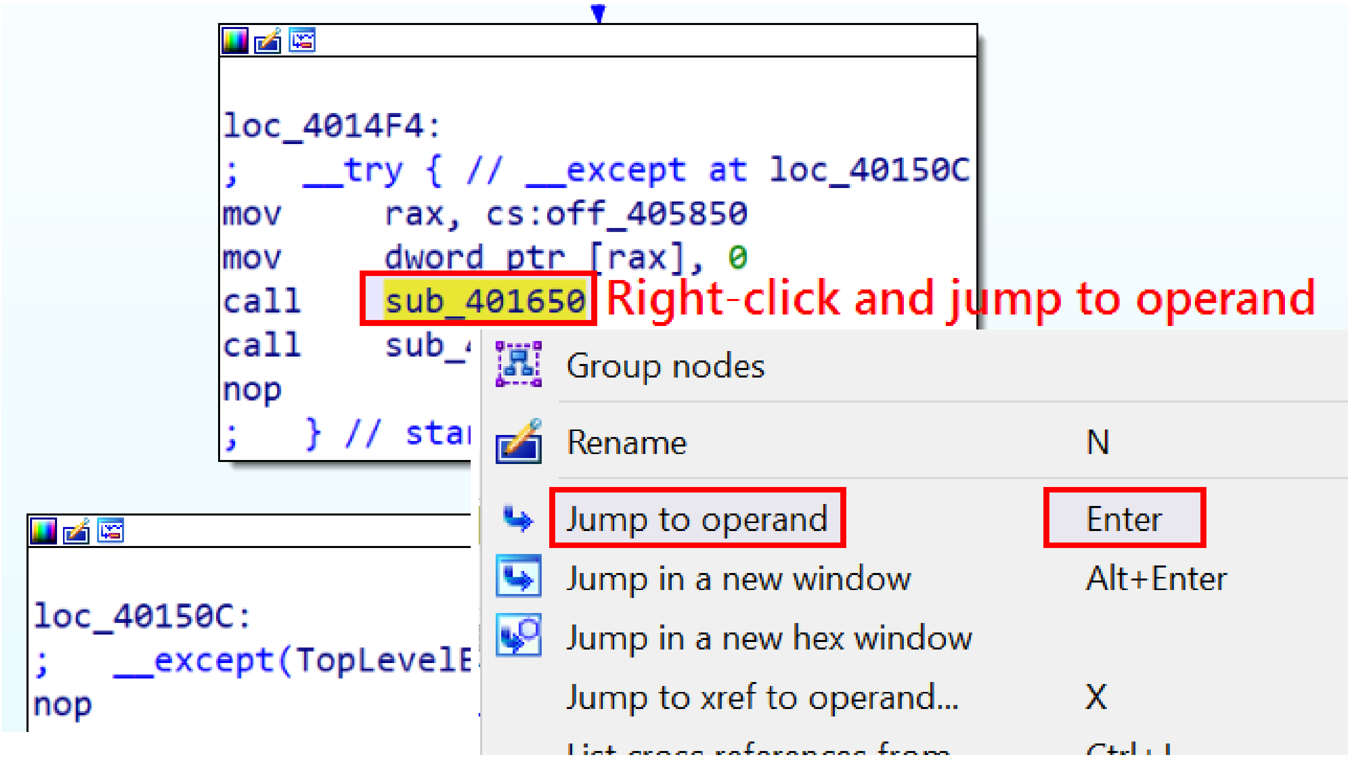


Recognizing the Main Function in IDA

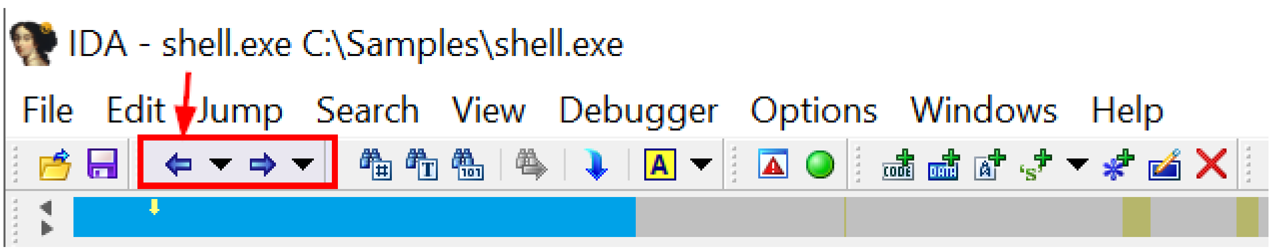
The following screenshot demonstrates the `start` function, which is the program's entry point and is generally responsible for setting up the runtime environment before invoking the actual `main` function. This is the initial `start` function shown by IDA after the executable is loaded.



Simply follow the calls for func then examin the code bro :)



to return

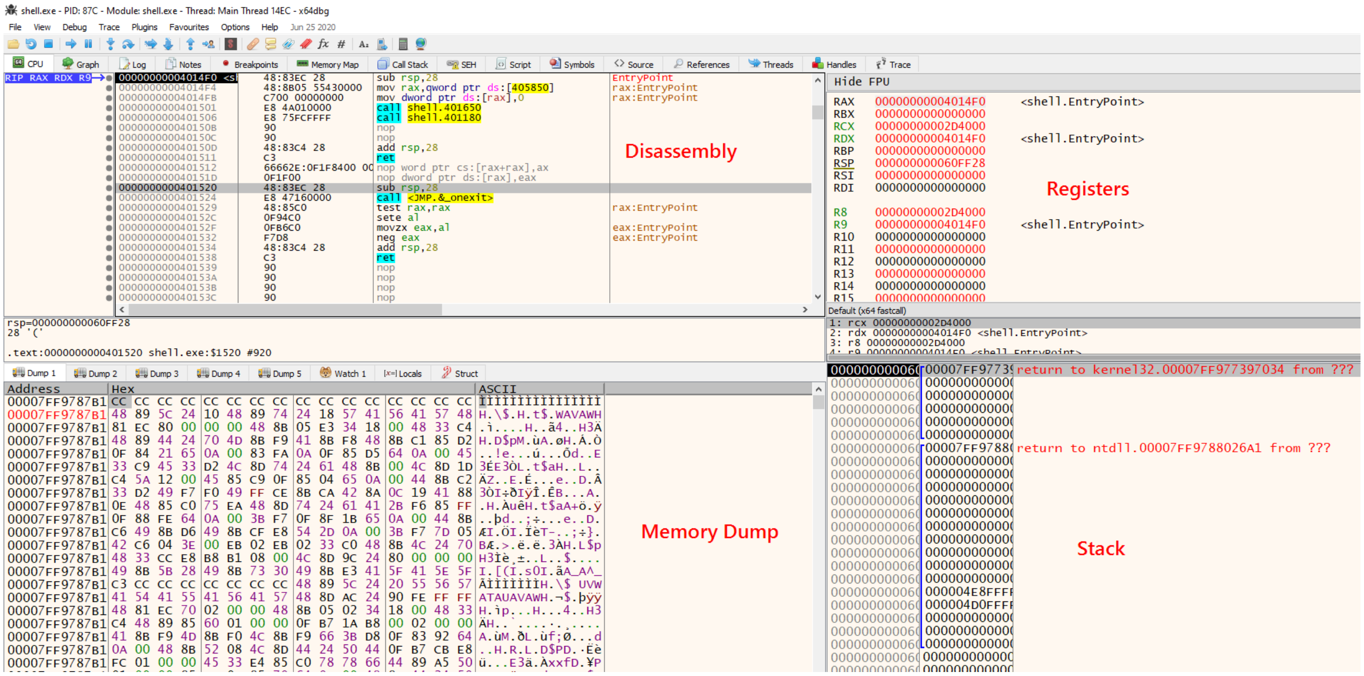


at the end, we can modify the code also like changing the instraction from jz (jump if 0) to jmp (which means jump anyway) in x64dbg

---

Debugging

for debugging `x64dbg`



Simulating Internet Services

The role of `INetSim` in simulating typical internet services in our restricted testing environment is pivotal. It offers support for a multitude of services, encompassing `DNS`, `HTTP`, `FTP`, `SMTP`, among others. We can fine-tune it to reproduce specific responses, thereby enabling a more tailored examination of the malware's behavior. Our approach will involve keeping `InetSim` operational so that it can intercept any `DNS`, `HTTP`, or other requests emanating from the malware sample (`shell.exe`), thereby providing it with controlled, synthetic responses.

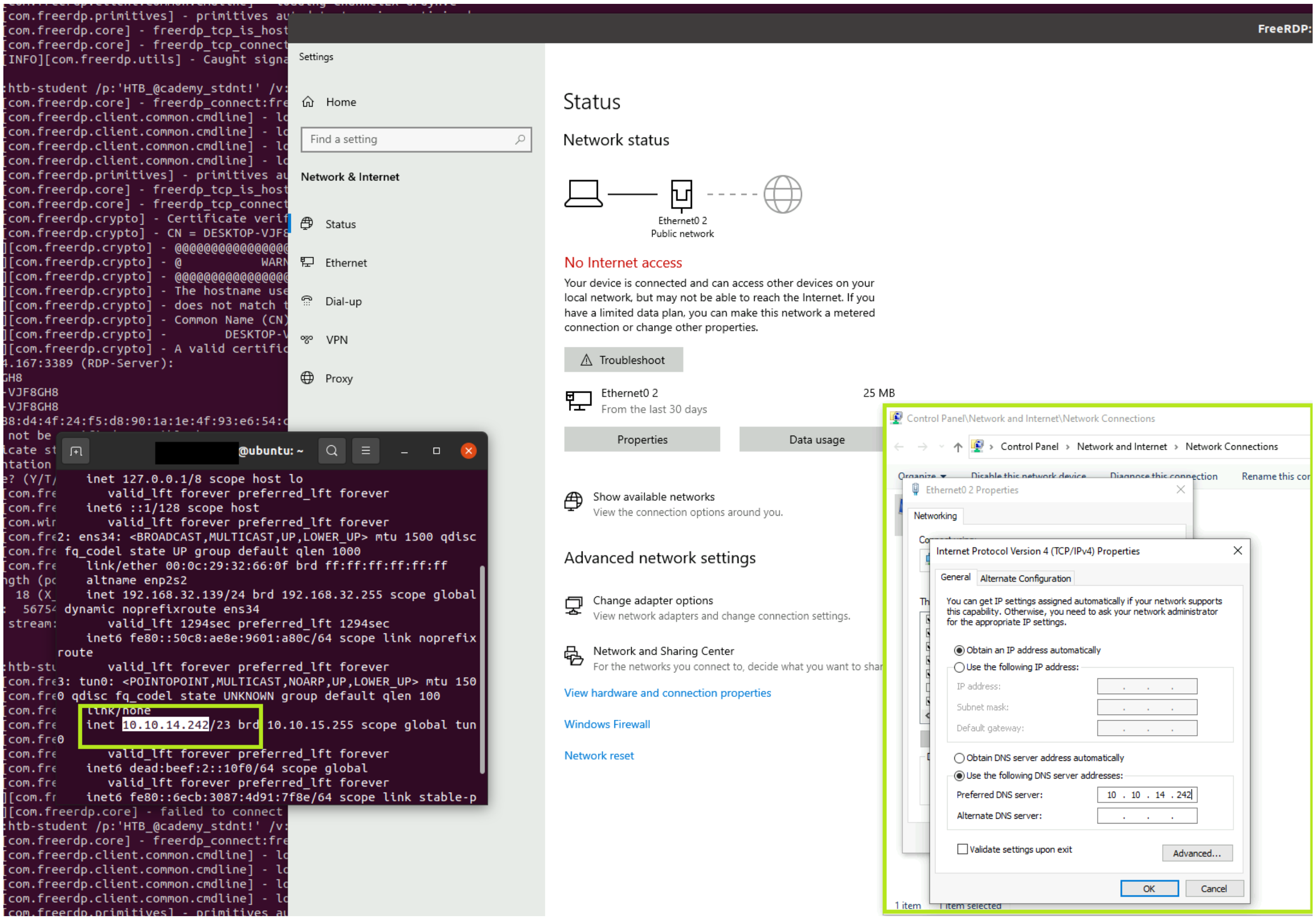
```
ltjax@htb[/htb]$ sudo nano /etc/inetsim/inetsim.conf
```

The below need to be uncommented and specified:

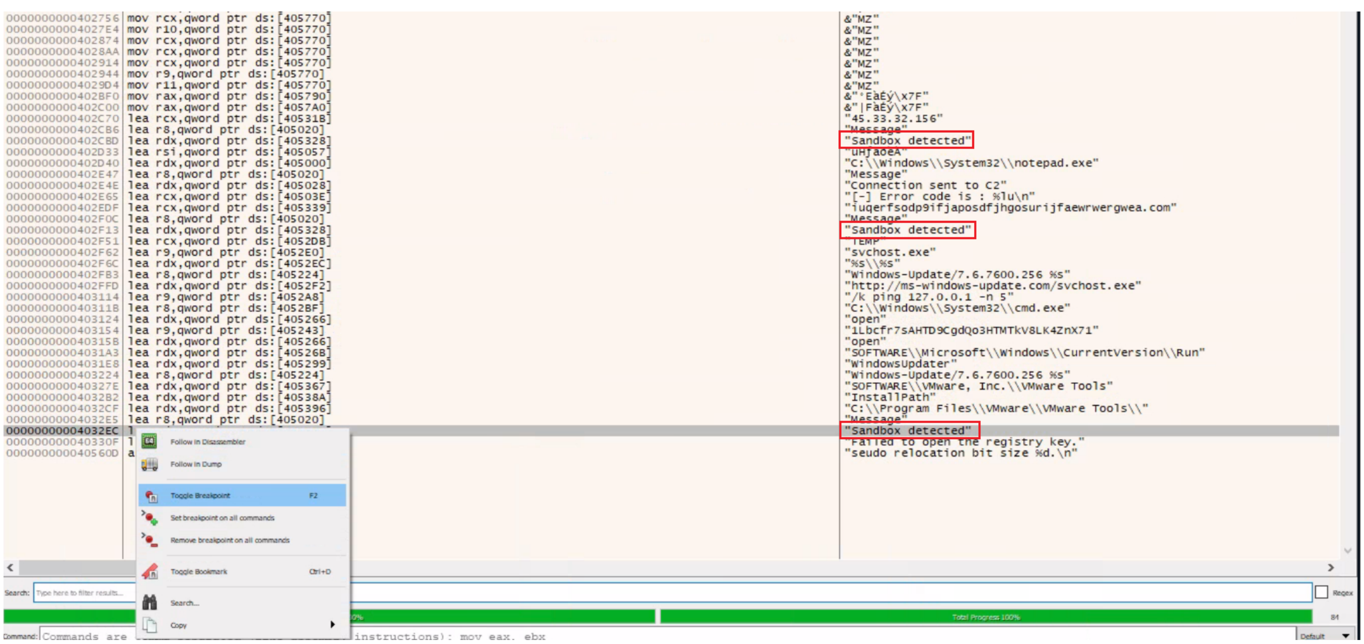
```
service_bind_address <Our machine's/VM's TUN IP>
dns_default_ip <Our machine's/VM's TUN IP>
dns_default_hostname www
dns_default_domainname iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
```

```
sudo inetsim
```

Finally, the spawned target's DNS should be pointed to the machine/VM where `INetSim` is running.

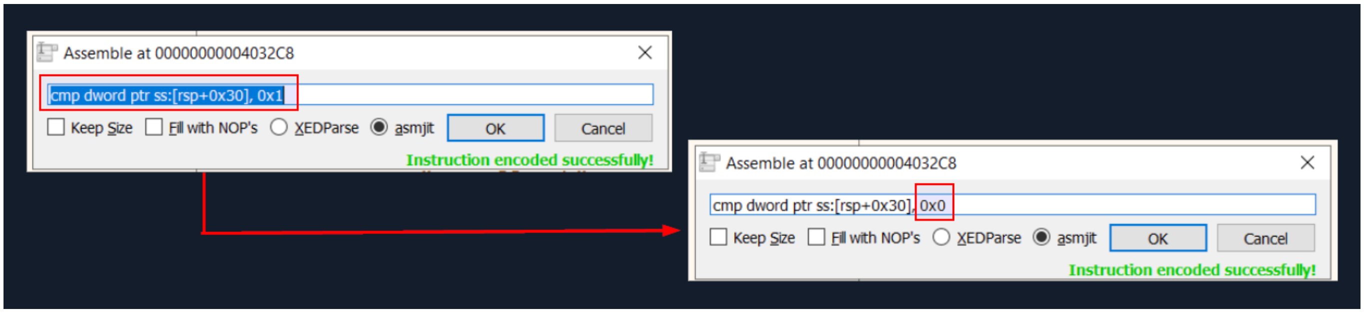


we can add a breakpoint to mark the location, then study the instructions before this Sandbox `MessageBox` *to discern how the jump was made to the instruction printing* `Sandbox detected`.



a `cmp` instruction is present above this `MessageBox` which compares the value with `1` after a registry path comparison has been performed. Let's modify this comparison value to match with

`0` instead. This can be done by placing the cursor on that instruction and pressing `Spacebar` on the keyboard. This allows us to edit the assembly code instructions.



Attaching Another Running Process In x64dbg

In order to delve further, let's open another instance of `x64dbg` and attach it to `notepad.exe`.

- Start a new instance of `x64dbg`.
- Navigate to the `File` menu and select `Attach` or use the `Alt + A` keyboard shortcut.
- In the `Attach` dialog box, a list of running processes will appear. Choose `notepad.exe` from the list.
- Click the `Attach` button to begin the attachment process.****