

# SecureHub

A collection of write-ups on malware analysis, forensics and more. Arm yourself with knowledge.

≡ MENU



## Malware Analysis – Illusion Bot



Previously, we reverse engineered *Infostealer.Dyre* which was a famous and deadly banking Trojan. In this post, we will try to reverse engineer a malware sample called *Illusion Bot* and develop a YARA rule for it. The sample can be found in theZoo.

I hope you already have a test lab setup. If you don't, you can read the article on *Infostealer.Dyre*.

**Note:** I highly recommend reading my article on *Infostealer.Dyre* before reading this post. I've explained the malware analysis process in detail and that information will not be repeated in every post.

Take note of the malware sample's MD5 hash – `9b9e083a9cf6a1db6251e189e5966a4d`

## Analysis

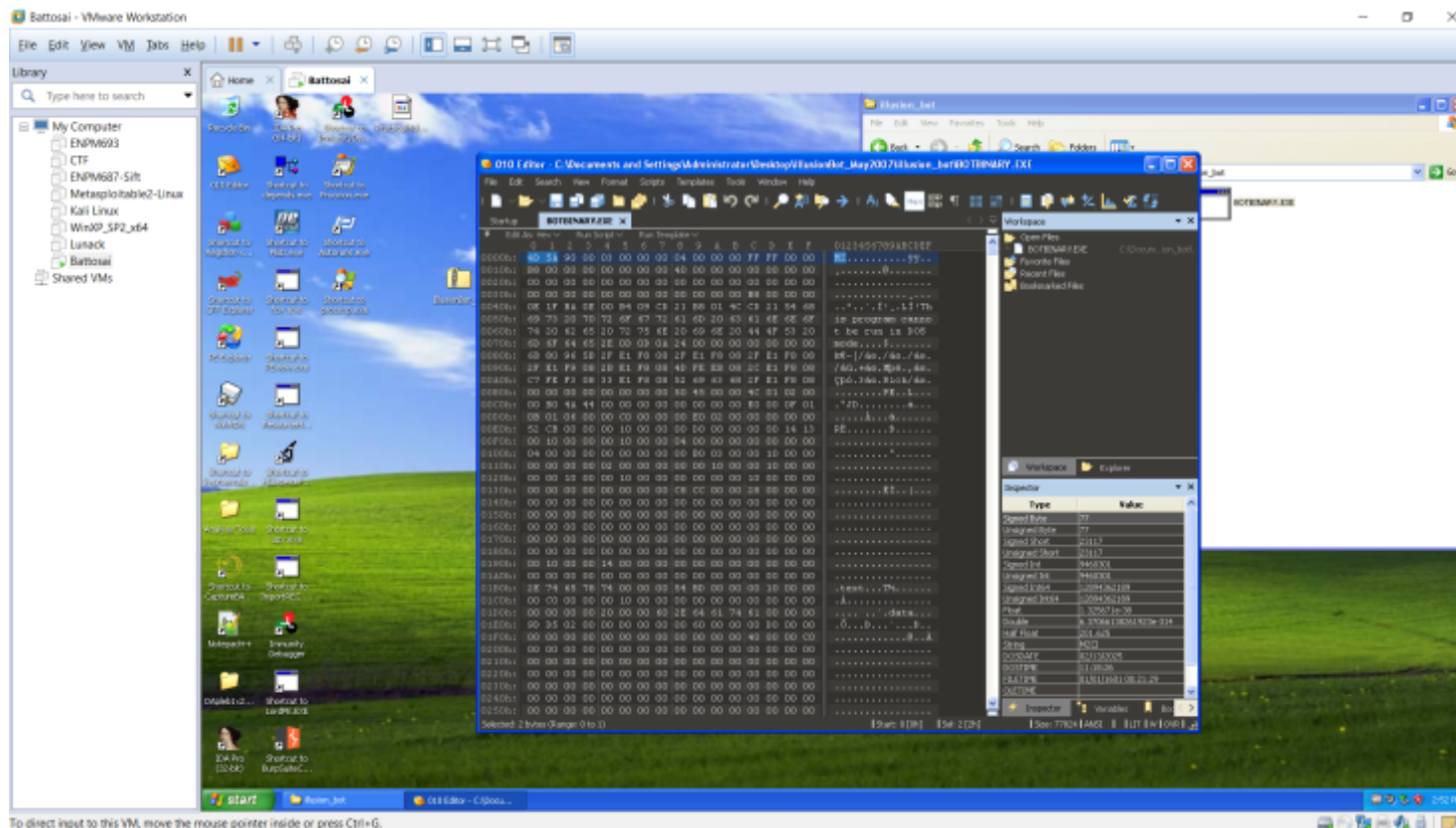
Place the malware sample on the Windows XP VM desktop (drag and drop) and unzip it using 7Zip (password – *infected*). In the *illusion\_bot* sub-folder, there is a file named *BOTBINARY.exe* which is the malware sample that we'll analyze.

## Is the sample executable?

The answer to this seems quite obvious because the malware sample is a *.exe* file. However, we will not believe Windows' ability to identify file type, rather we'll confirm for ourselves by looking at the sample's

magic number.

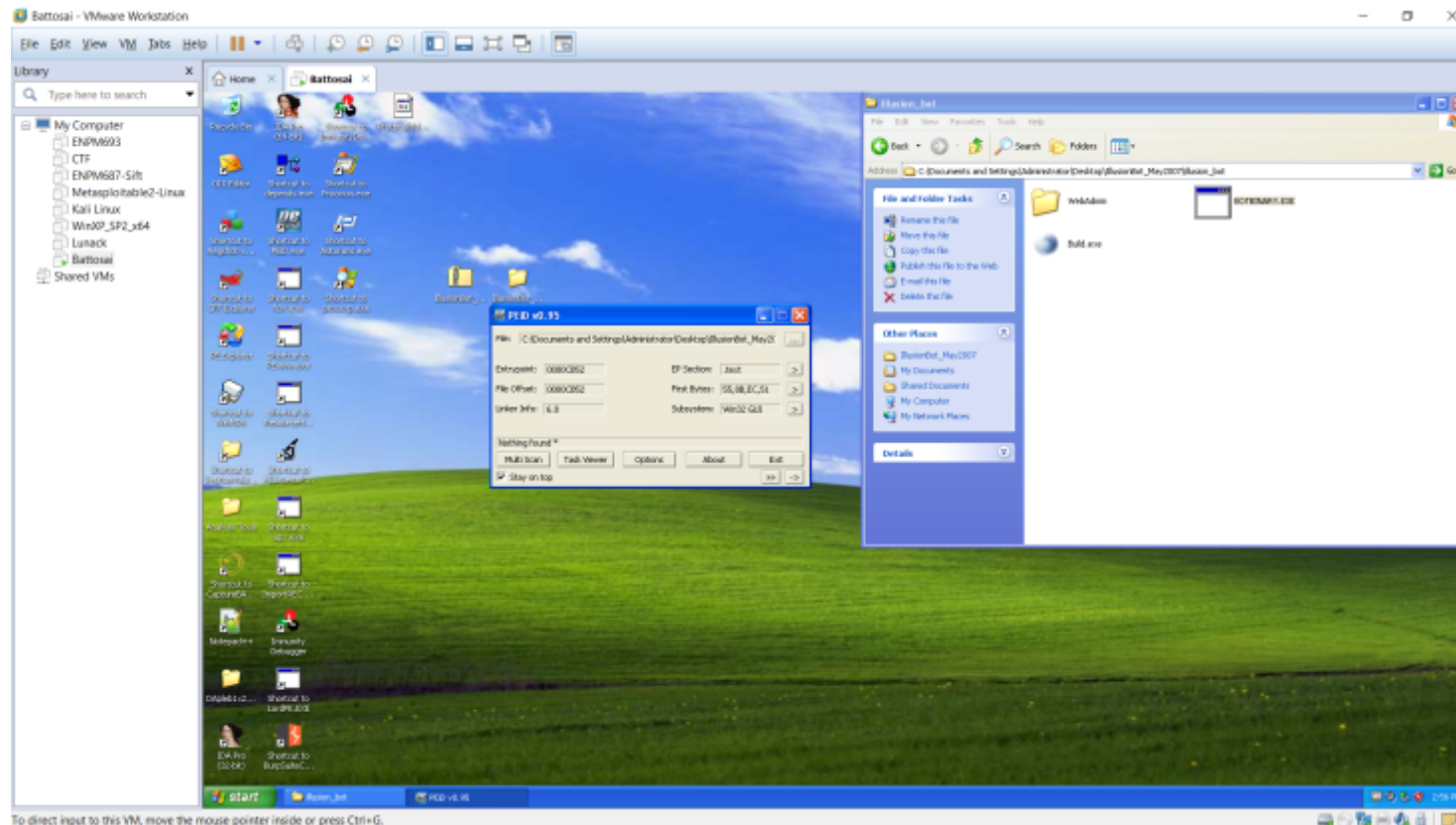
On opening the malware sample in *o10 Editor*, we can see that the first two bytes are 4D5A, which in ASCII is *MZ* and it is the magic number that identifies a Windows executable.



Is the sample packed?

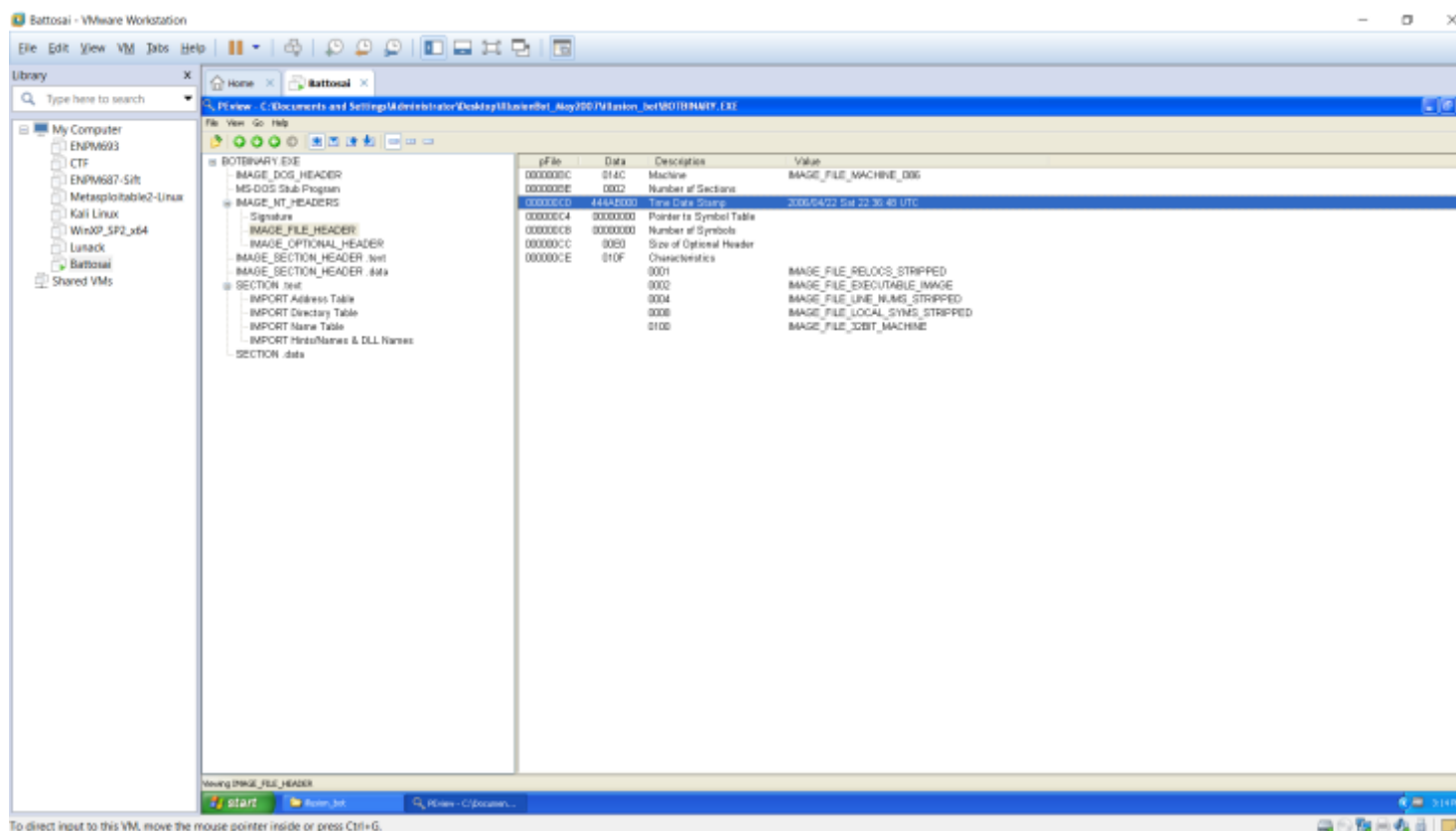


On opening the sample in *PEiD*, we can see that it is not able to identify its compiler. This highly suggests that the sample was packed using a custom packer.



## When was the sample compiled?

We can check the compile time (may be faked) on *PEView* and it is 2006/04/22 Sat 22:36:48 UTC. As you can see, this is an old malware and most AV databases will probably have its signature. However, we will move forward with the aim of reverse engineering our sample.



Does the sample have other embedded executables in it?

In this case, the malware sample does not have a *.rsrc* section where another executable could be embedded. So, this sample most likely does not have any other embedded executables.

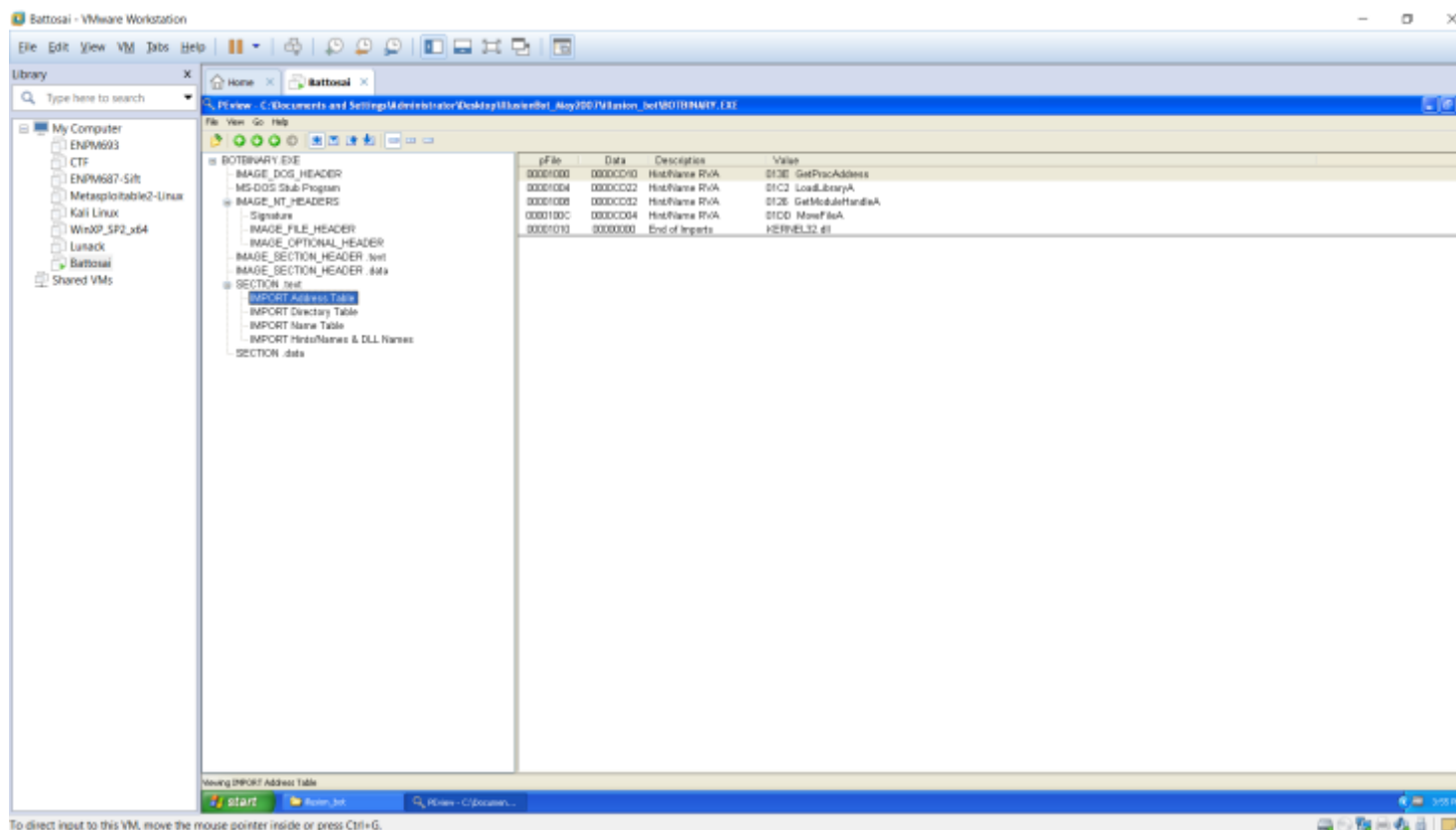
## What sub-system does the sample operate in?

In the *IMAGE\_OPTIONAL\_HEADER* in *PEView*, we can see that the sample has a GUI component to it. We can expect the sample to interact with windows during execution.





Programs can import other functions dynamically as well. In this case, the sample imports functions like *LoadLibraryA()* and *GetModuleHandleA()*, both of which can be used to load other functions dynamically.



What clear text strings does the sample have?

On opening the sample in the *Strings* sub view in *IDA Pro*, we can see hundreds of readable strings.

A string, *Bindport: Couldnot bind main socket* suggests that the sample has a networking component to it.

There are many other strings which are function names and suggest that the sample uses them in some manner.

There are a few very interesting strings such as *SYN Flooder error: resolve failed*, *UDP Flooder error: resolve failed*, *UDP Flooder started* which suggest that the sample may be a DoS (Denial-of-Service) vector.

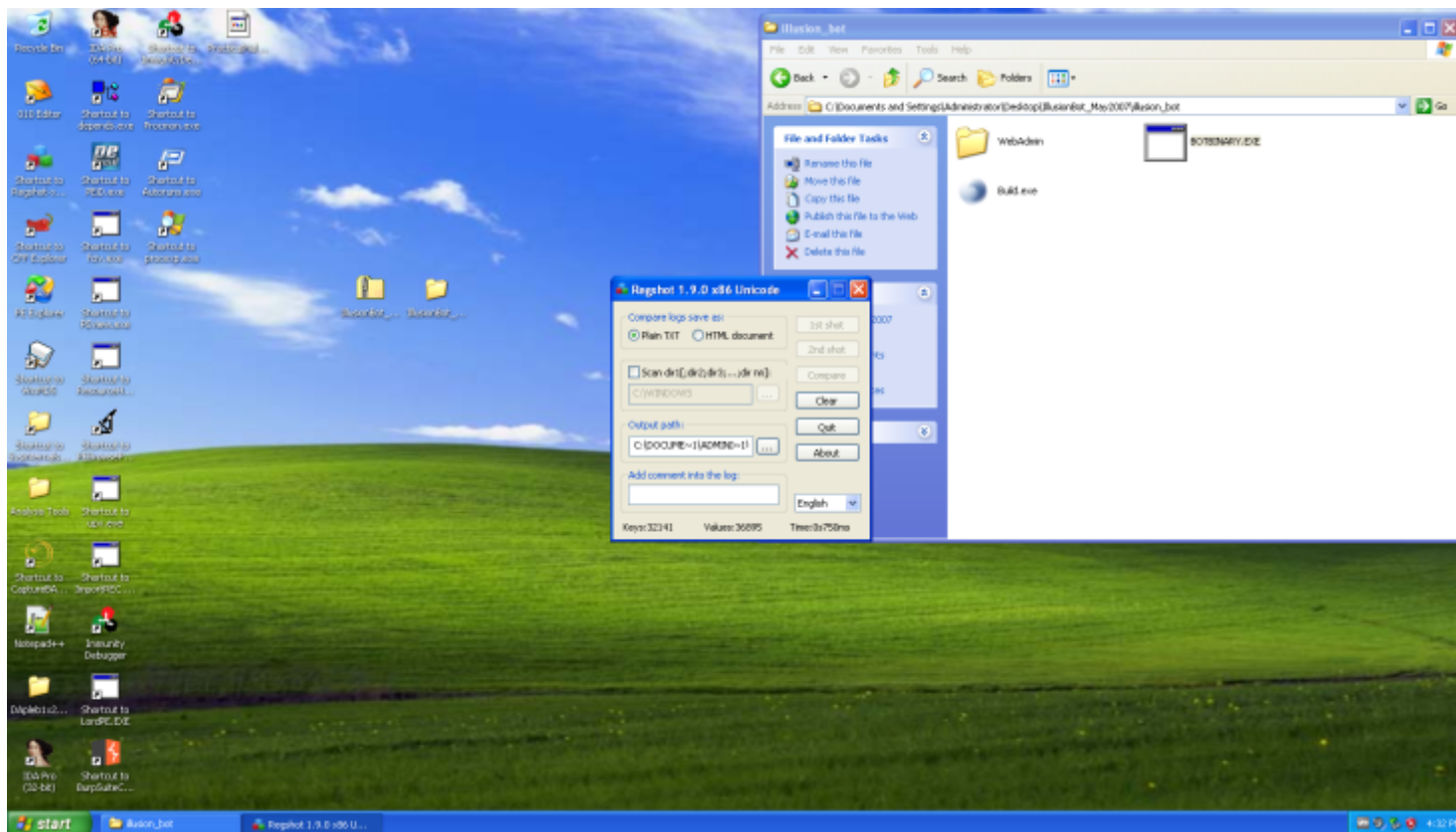
There are also strings like *irc\_pass*, *irc\_port*, *irc\_server*, etc. which suggest that the sample has an IRC networking component to it.

A string, *I need a message on channel instead of private* suggests that the sample may be a backdoor-type malware which waits for commands from a C2 server.



# What changes were made to the Windows Registry?

For some reason, *Regshot* repeatedly hanged when I was trying to snap the registry after malware execution. The window was *Not Responding* for more than 20 minutes, so I guess something went wrong.



We can still track the changes to the Windows Registry with *Process Monitor*. I've filtered the results such that only *BOTBINARY.exe's* *RegSetValue* activity is displayed.





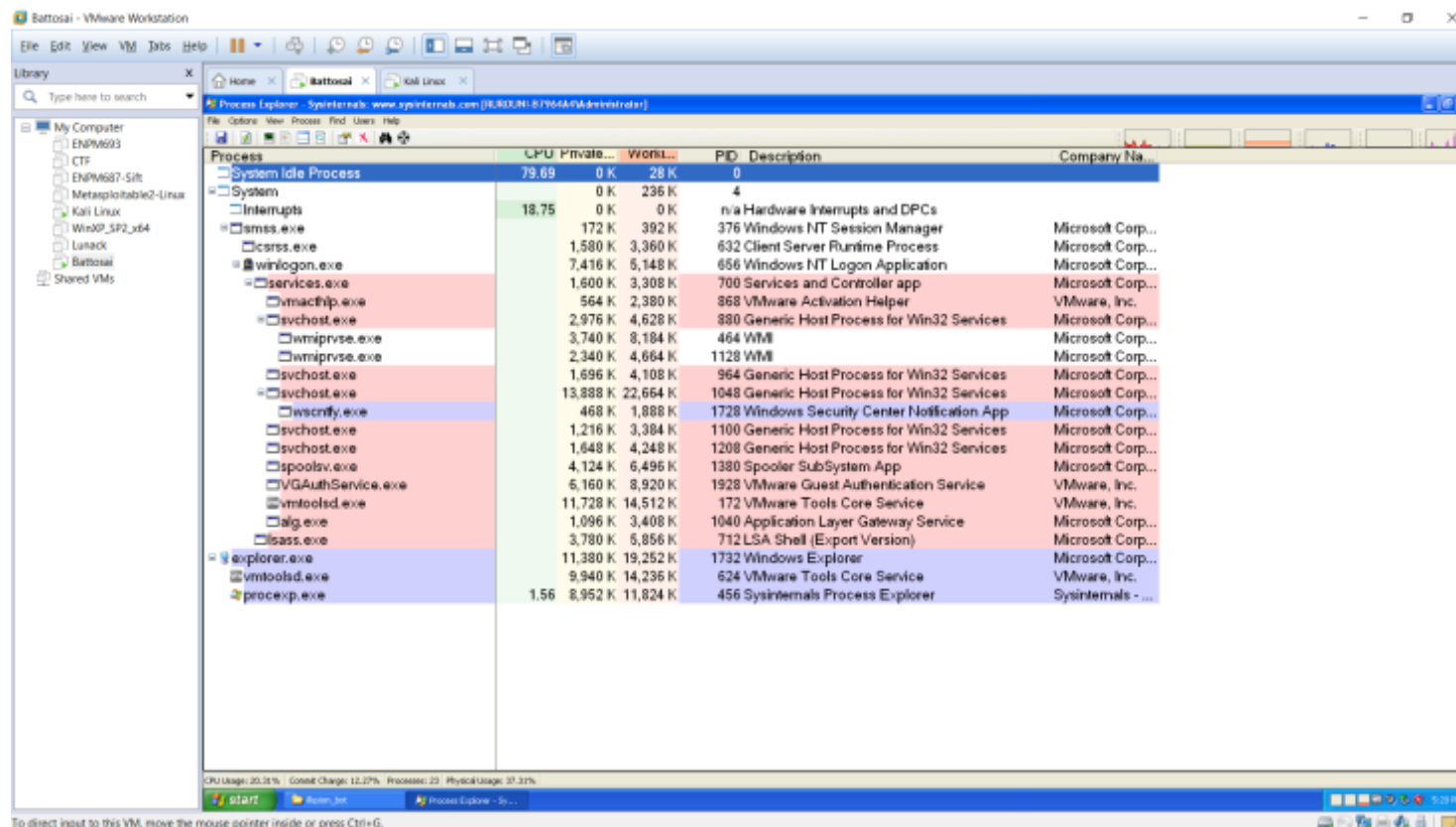
The sample edited the firewall policy registry key to make itself an authorized application. In this way, the firewall would not block any network activity of our sample.

```
"30/1/2019 22:5:49.565","registry","SetValueKey","C:\Documents and  
Settings\Administrator\Desktop\IllusionBot_May2007\illusion_bot\BOTBINARY.EXE","HKLM\SYSTEM\ControlSet001\Serv  
ices\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List\C:\Documents and  
Settings\Administrator\Desktop\IllusionBot_May2007\illusion_bot\BOTBINARY.EXE"
```

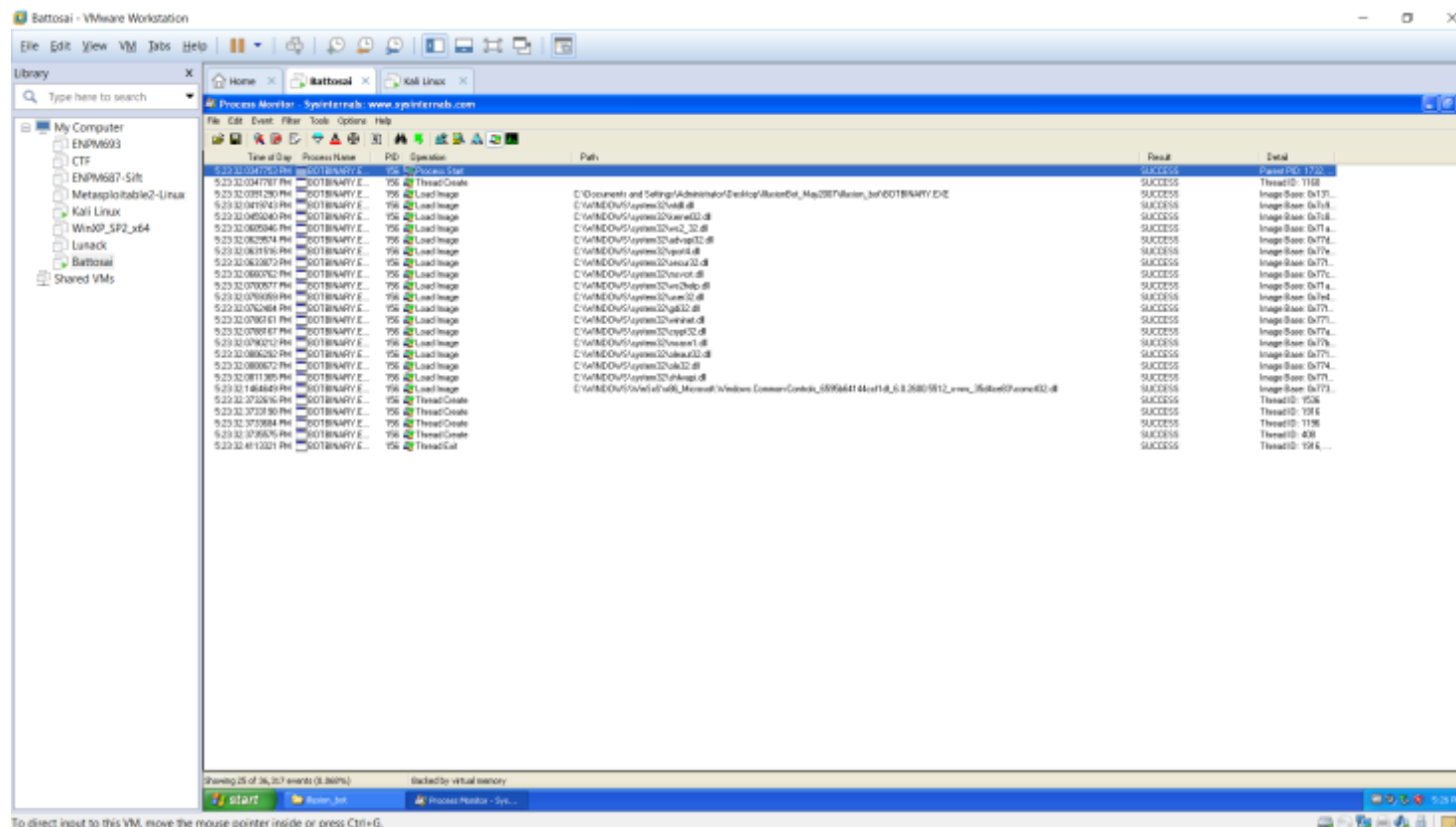
## Were any startup services installed by the sample?

We can check any changes in startup services using *Autoruns*. On executing the sample and examining the services list, we can see that no startup services were installed.



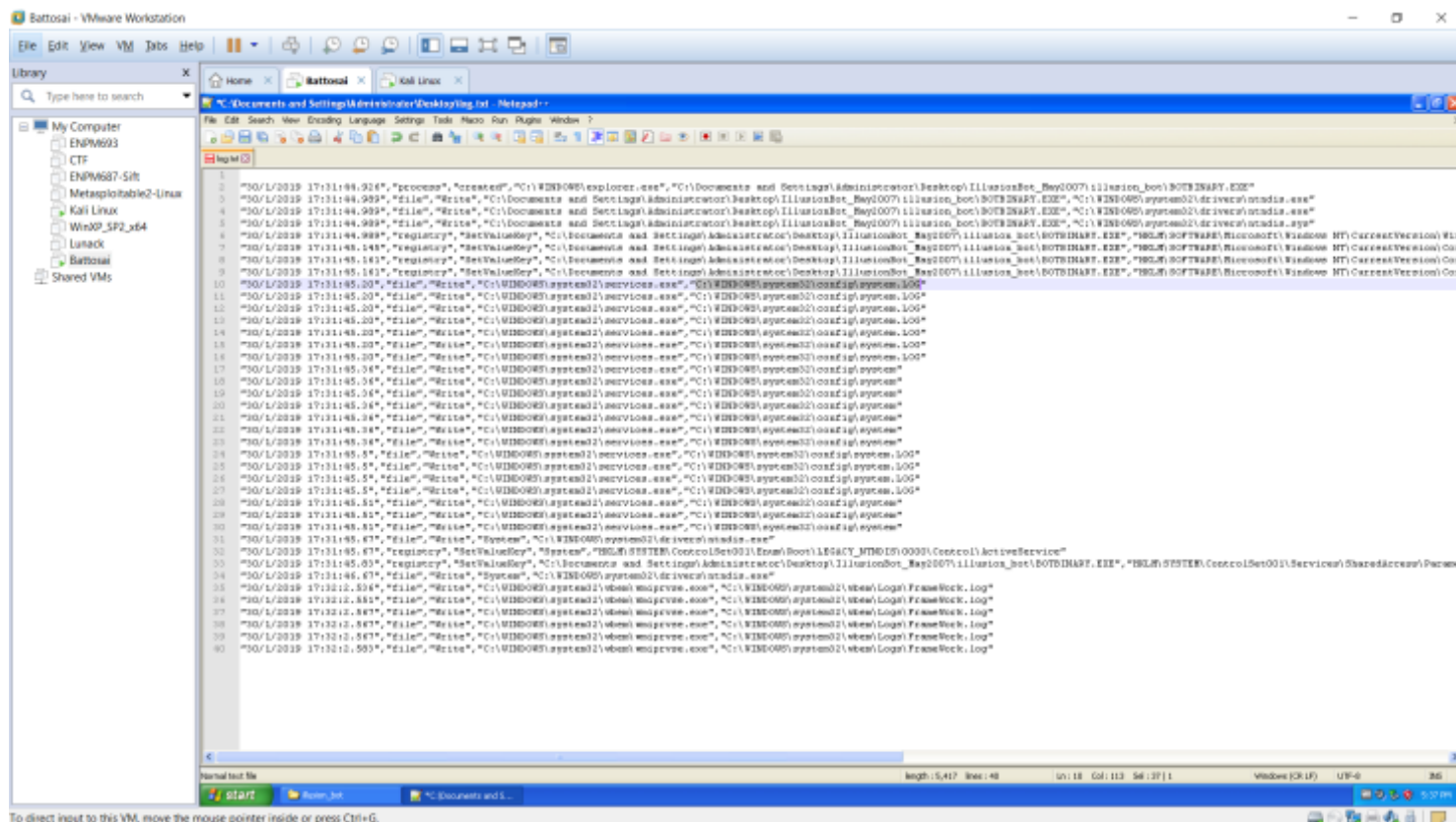


We can use *Process Monitor* to track our sample's process activity. It looks like our sample exited before *Process Explorer* could refresh its list of processes. Anyway, the processes were not of any major significance.



## What changes were made to the file system?

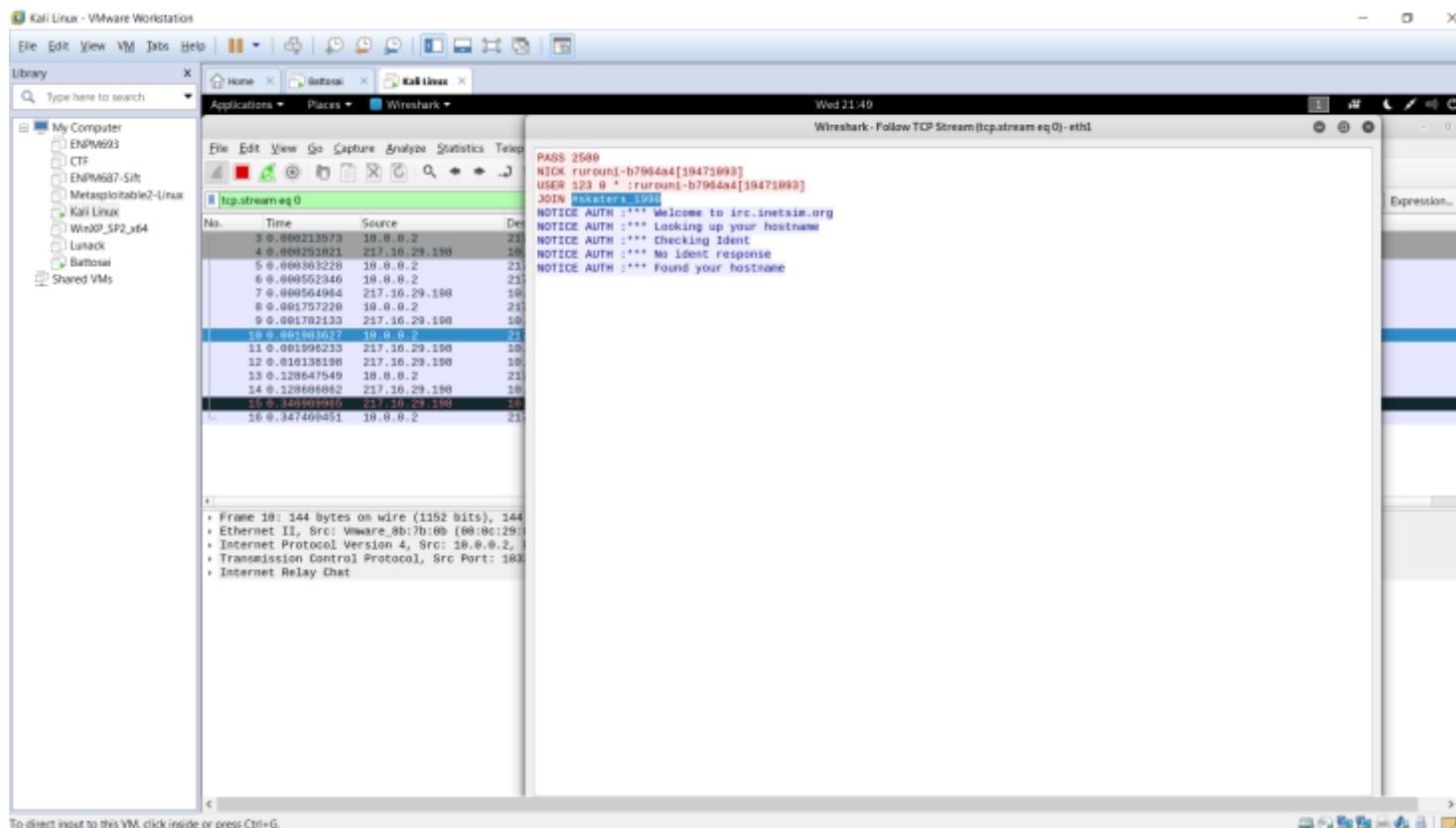
We can track file system changes through *CaptureBAT*. On opening the log, we can see that our sample tried to write to a file, *ntndis.exe* at location, *C:\WINDOWS\system32\drivers*. However, the file does not exist at the location. I'm not sure what the sample was hoping to do there.



## What network activity does the sample exhibit?

We can track network activity of our sample using *inetsim* and *Wireshark*. Ensure that you have *inetsim* and *Wireshark* running on the Kali Linux VM and all IP addresses routing to the Kali Linux VM.





We can see that the sample was reaching out to an IP address, `217.16.29.198` through *IRC* protocol. It seemed to be trying to join an *IRC* channel, `#skaters_1990` with the password, `2580` and username, `rurouni-b7964a4[19471093]`.

Our sample is likely a backdoor program. We could have interacted with it using *netcat*, however the backdoor program crashes if a valid command is not sent to it.

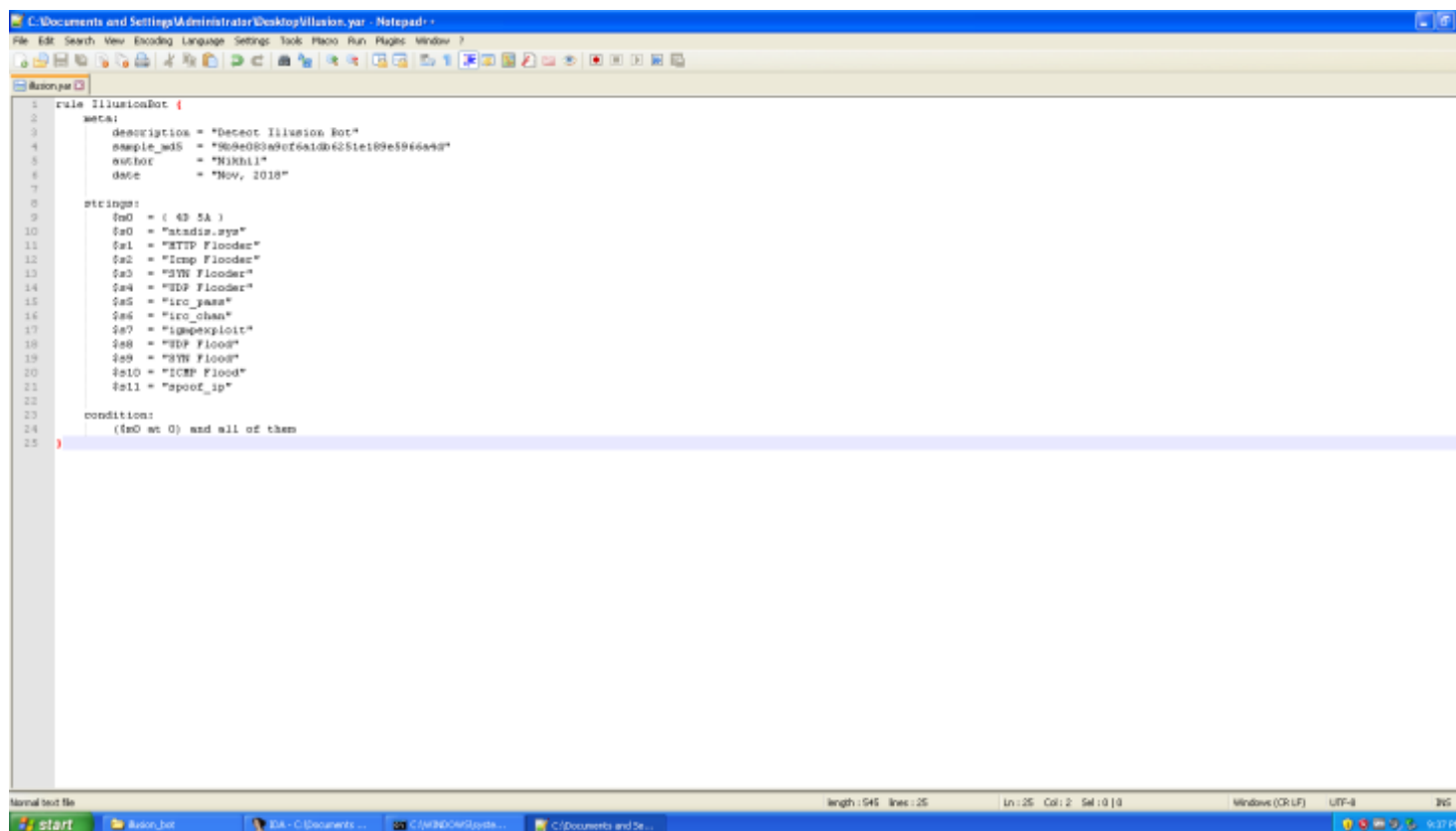
**Note:** At this point we have the following findings:

1. The malware achieves persistence by replacing the default shell program (explorer.exe) with itself.
2. It places itself in the firewall's white list.
3. It tries to join an *IRC* channel and most likely, waits for commands. It may be used for DoS attacks because there were clear text strings that referenced it.
4. It tries to write to a file *ntndis.exe*, and the effect of this is unknown.

## Developing YARA Rule

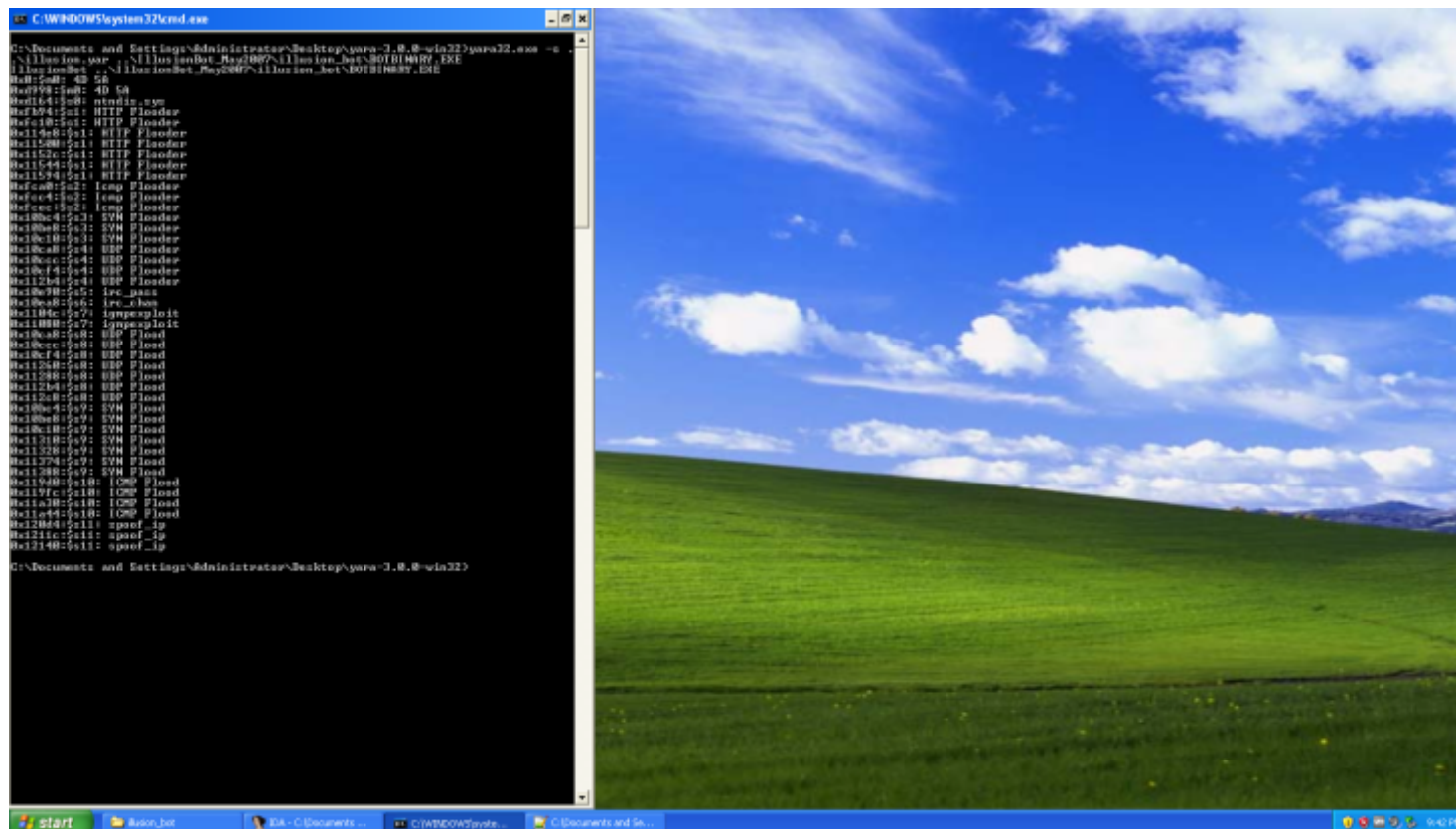
The strings that we'll place in the *.yar* file are the same strings which we determined in the section "*What clear text strings does the sample have?*".

The YARA rule for our malware sample is shown in the snapshot:



```
1 rule IllusionBot {
2   meta:
3     description = "Detect Illusion Bot"
4     sample_md5 = "909e083a9cf6a10d6251e109e5966a9d"
5     author = "Nikhil"
6     date = "Nov, 2018"
7
8   strings:
9     $m0 = { 43 5A }
10    $a0 = "atadix.sys"
11    $a1 = "HTTP Flooder"
12    $a2 = "Icmp Flooder"
13    $a3 = "SYN Flooder"
14    $a4 = "UDP Flooder"
15    $a5 = "irc_pass"
16    $a6 = "irc_chan"
17    $a7 = "imgexpl0it"
18    $a8 = "UDP Flood"
19    $a9 = "SYN Flood"
20    $a10 = "ICMP Flood"
21    $a11 = "spoof_ip"
22
23   condition:
24     ( $m0 at 0 ) and all of them
25 }
```

We can verify the rules using the YARA executable, *yara32.exe* (on 32-bit Windows XP). You can download it from [here](#). If you hit an error, “MSVCR100.dll not found”, download and run *Microsoft Visual C++ 2010 Service Pack 1 Redistributable Package MFC Security Update* from [here](#).



As can be seen in the snapshot, *yara32.exe* was able to detect the strings in the malware sample and thus, positively confirmed that our rules were correct.

# Done!

So far, we've used techniques for both static and dynamic analysis. These basic techniques provided us with considerable information about the malware sample. However, not all questions were answered. We still don't know what commands from the C2 server the malware sample accepts.

To answer such questions, we need to look at the malware's assembly code. Writing in detail about the IDA Pro analysis of our malware sample is probably going to take a lot of time and being a graduate student, I don't have that much time to spare.

Thank you for reading! If you have any questions, leave them in the comments section below and I'll get back to you as soon as I can!

Advertisements

**Earn money from  
your WordPress site**

WordAds

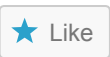
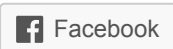
**START EARNING**

REPORT THIS AD

REPORT THIS AD

SHARE THIS:





Be the first to like this.

#### RELATED



Malware Analysis - NanoCore + MITRE  
ATT&CK Mapping  
In "malware analysis"



Malware Analysis - WannaCry  
In "malware analysis"



Malware Analysis - Gozi/Ursnif  
Downloader  
In "malware analysis"



Published by Nikhil

A budding front-line cyber soldier!



View all posts by Nikhil



January 30, 2019



malware analysis



analysis, malware, reverseengineering,  
windows



Malware Analysis – Infostealer.Dyre



## Leave a Reply

Enter your comment here...

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

### FOLLOW SECUREHUB VIA EMAIL

Follow SecureHub to receive notifications by email. One email per week max.

Enter your email address

Follow SecureHub

Search ...



## POSTS BY CATEGORY

- » android (1)
- » cryptography (2)
- » CTF solutions (3)
- » exploits (11)
- » forensics (2)
- » fuzzing (1)
- » git (1)
- » incident response (1)
- » linux (1)
- » malware analysis (8)
- » pentesting (3)
- » playtime (1)
- » programming (4)
- » reverse engineering (3)
- » social engineering (2)
- » splunk (1)
- » stories (2)
- » threat intel (5)

» Uncategorized (1)

CATEGORIES

Select Category ▼

Advertisements

REPORT THIS AD



## ARCHIVES

Select Month ▼

---

## CATEGORIES

Select Category ▼

---



POWERED BY WORDPRESS.COM.

UP ↑