How efficient AntiViruses are: An empirical investigation of AntiViruses with RATs

By: Dr Mahdi Aiash

AntiViruses (AVs) have always formed the main line of defence to secure our systems. However, despite the huge number of AV programs and the immense effort to improve their detection techniques, the past few years witnessed a dramatic increase of Cyber incidents. Unfortunately, many of the recent malware attacks have been able to bypass AV software and infect systems. Mainly this could be due to the fact that AVs are very efficient at catching and blocking low-level malware but not at catching the unknown, such as non-malware or file-less attacks, for example. AVs in general try to detect malware by examining a file or process to determine if it's bad on the basis of its resemblance to previously seen malware.

This paper conducts an empirical investigation of the efficiency of Antivirus programs in detecting malware. In order to draw a more realistic conclusion, the investigation considers the case of known malware deploying various evasion techniques as well as the case of a newly developed Remote Access Trojan. To analyse the efficiency of AVs, various evasion techniques are deployed by a known malware and checked using 35 different AV available through No Distribute online AV scanner. Additionally, a new Remote Access Trojan (RAT) is developed and checked through the same AVs of the No Distribute scanner. This latter investigation will give an indication on how efficient AVs are in detecting new malware.

The investigation highlights the need for a second line of defence which was implemented using Snort, a well-known Intrusion Detection System.

1. The Testbed

In order to simulate real-life scenarios while testing AVs along with any proposed enhancements, a testbed is created using Oracle VM VirtualBox, which is a hypervisor for x86 computers developed by Oracle Corporation. As shown in Fig 1, using VirtualBox we create two Virtual machines. Both virtual machines are able to communicate with each other as in a Local Area Network (LAN) and to access the Internet through the Network Address Translation (NAT) interface.

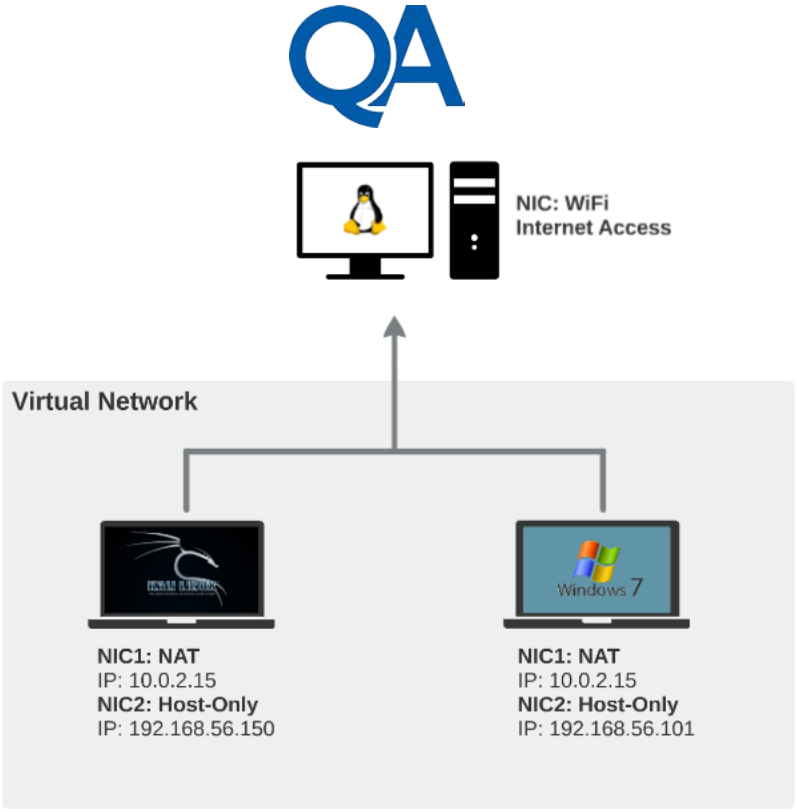

Figure 1: The TestBed

To test the detection engines of different AVs, we use the No Distribute (NoDistribute) online scanner which compares 35 Antivirus solutions at the time of writing. To double check the results of the online scan, the tested, sample malware will also be scanned on the victim's virtual machine using AVG 2016, which is in the Top 10 rated AntiVirus solutions for Microsoft Windows (tested by AV-Test).

The detection test is divided into two main parts:

- To apply a number of the most common evasion techniques to a known malware and test with the online scanner. This will give an idea on how efficient AVs are against evasion techniques.
- To develop a new RAT as a malware and to test it using the online scanner. This helps evaluating AVs against new, unknown malware.

For the first part of the test it is required to have a known malware that is detected by most AV solutions. This can be generated on the Kali Linux VM using msfvenom utility. It is used in this research to create sample malware's payload (sample.exe) that connects back to the attacker machine through a reverse TCP shell, giving the attacker access to the victim system as in Fig 2:

```
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.56.150:4444
[*] Starting the payload handler...
[*] Encoded stage with x86/shikata_ga_nai
[*] Sending encoded stage (267 bytes) to 192.168.56.101
[*] Command shell session 2 opened (192.168.56.150:4444 -> 192.168.56.101:55980)
 at 2016-03-25 13:39:59 +0000

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\IEUser\Desktop>
```

Figure 2: Reverse Shell Connection test

To set the baseline, we submit the sample.exe file to No Distribute, since no evasion has been used, the malware has been detected by most AVs. As shown in Fig 3, only 23 out of 35 AVs were able to detect the threat. Clearly with a common payload like this, the results were below the expectations.

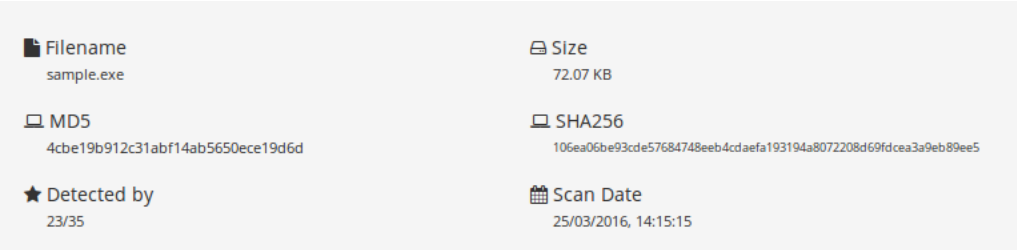

Figure 3: Result of sample.exe scanning

2. Testing Different Evasion

The main evasion techniques will be applied and then an online scan with No Distribute will be executed.

2.1. Footprint Manipulation:

This technique needs to identify the part of the file where the malicious code is written to. In order to do so, the following steps are applied:

- Split the file into chunks of a certain size.
- Scan each chunk to find the one that has the malicious code.
- Repeat the previous steps until the malicious code is isolated.
- Edit the malware with a hexadecimal editor.

The tool of choice to split the file is File Splitter. Following the above steps, we find that the part of the code that triggers AVs is in the first 640 bytes of the file. The next step is to edit the file with a hexadecimal editor such as HxD. As shown in Fig 3, it is possible to change some parameters; for instance, in the sentence starting on the offset 00000040: ”This program cannot be run in DOS mode”, replacing the word ”DOS” with ”ABC” will give the scan results shown in Fig 4. As it is possible to see in the figure, the MD5 for the file has changed and the detection rate is now 20/35 which is 3 AntiVirus less than the original scan.

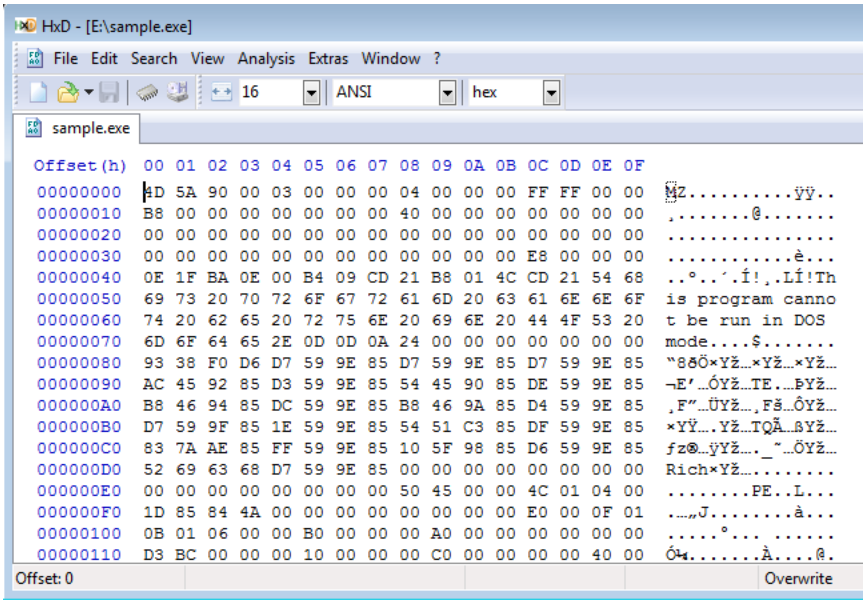

Figure 4: Hexeditor-Sample.exe

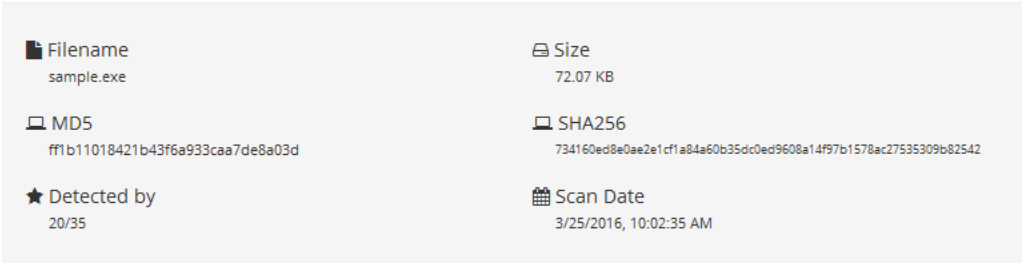

Figure 5: Scan result (editing footprint)

2.2. Use a Binder

A binder is a program that is used to join two or more PE files into a single file. This is used to hide the first bytes of the malicious code, putting another program at the beginning. There are different binders on the market. For this research, we use IExpress binder which is provided free of charge from Microsoft to hide the code of the sample.exe inside the calculator program (calc.exe). The output will be a file that will start the calculator utility on the victim machine and launches the sample.exe. Scanning the resulted output shows more interesting results as in Fig 6. Obviously, the MD5 for the file is completely different from the original one, but the most interesting part is the rate of 11/35. More than half of the AVs that originally flagged the malware as malicious file, now consider it as legit. This is probably due to the fact that for computational reasons, most AVs limit their scan to the first bytes of a file. AntiViruses give up really quickly, and since the first thousands of bytes seemed legitimate, they are not able to determine the threat.

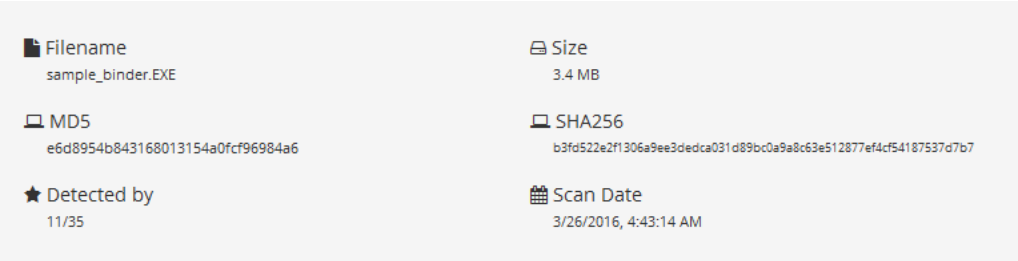

Figure 6: Scan of the sample using binder

2.3. Obfuscating the code

Obfuscation is a technique used to make it more difficult to reverse engineer a program. This can also be useful when it comes to evading signature. In this paper, the tool that is used to test the sample malware is PEScrambler.exe. This program changes the signature by adding junk code or dead callback functions. Once downloaded, it can be launched on the Kali machine using Wine utility as follows:

wine PEScrambler.exe -i sample.exe -o sample obfuscated.exe

The file is still working after the obfuscation, the scanning result of No Distribute is shown in Fig 7. From the scanning result, it is obvious that obfuscation was not very effective as it only helped evading few AVs.

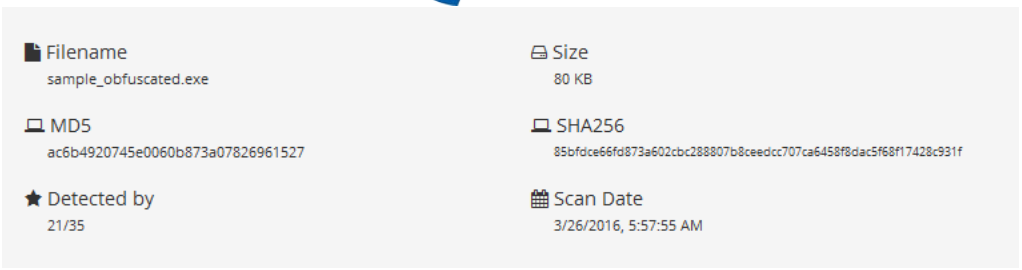

Figure 7: Scan of the sample using obfuscation

2.4. Automated Tools For Malware Encoding:

There are many tools available that automate the task of AntiVirus evasion through encoding. One of the most popular among Penetration Testers is Veil-Evasion. Veil-Evasion is a tool designed to generate metasploit payloads that bypass common AntiVirus solutions. We generated a powershell malware that starts a reverse meterpreter shell to the attacking machine as shown in Fig 8.

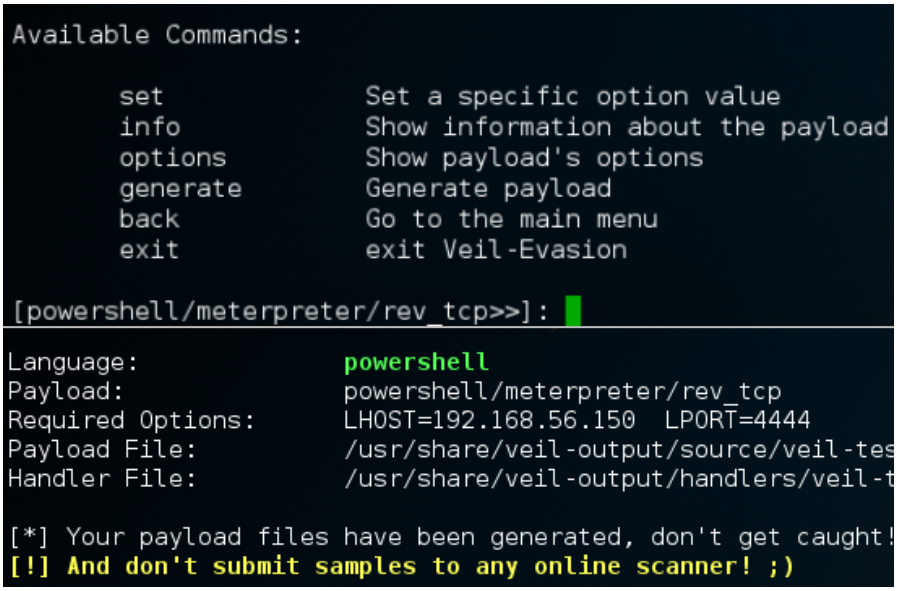

Figure 8: Create a payload with Veil-Evasion

Scanning the generated payload with NoDistribute shows that more AVs have been evaded as shown in Fig 9.

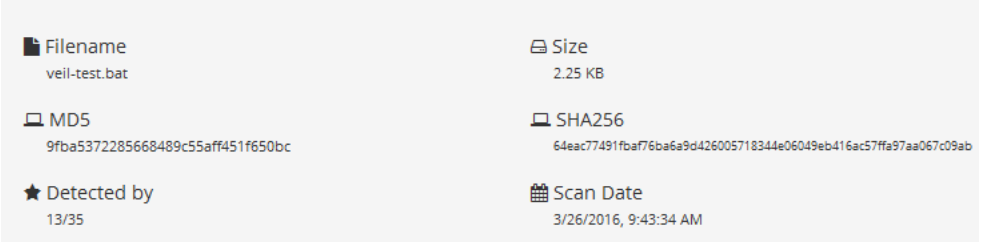

Figure 9: Scan result of a Veil generated file

To improve the evasion of the Veil malware, we combine manual techniques together and try to achieve a better evasion. For example, applying the binder technique after converting the previous payload from .bat to .exe, the online scanner gives back much better results as in Fig 10. At this point

it is possible to say that the payload goes undetected on almost every AntiVirus solutions, exposing end users to risks.

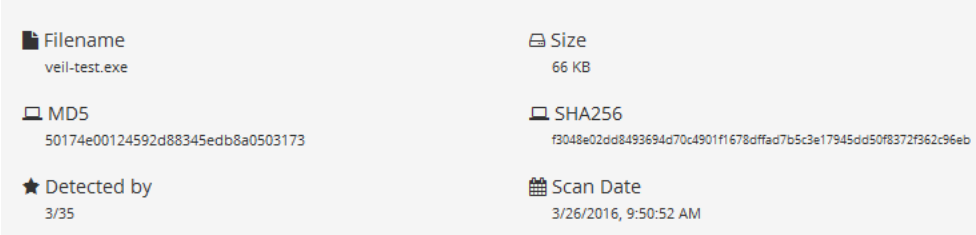

Figure 10: Detection result of a Veil generated file (applying Binder)

Another tool that is relatively new is Shellter. Shellter is a dynamic shellcode injection tool that works with 32bit PE files. The main feature of Shellter is altering the structure of PE files (given in input by the user) using polymorphic junk code and applying advanced code obfuscation techniques. The main screen of Shellter only asks for the PE file. The file used for this test is calc.exe. Once finishing the initial setup and configuration, the original calc.exe file will be a malicious PE. Although this is an automated and widely available tool, the result of the scan shows that the payload goes undetected on every AntiVirus solution registered on No Distribute as shown in Fig 11.

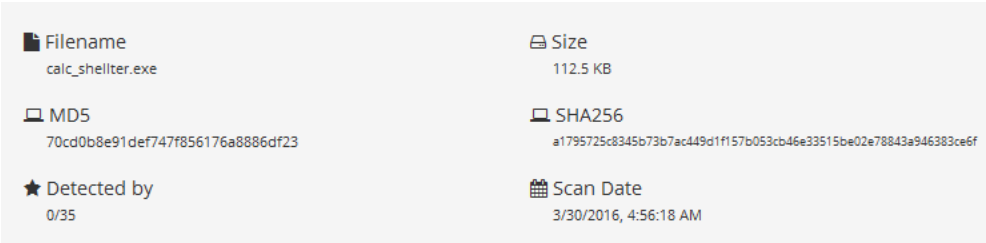

Figure 11: Result of a file injected with Shelter

3. Developing the RAT

The second part of the testing phase involves the use of a RAT (Remote Access Toolkit). This is written using Python scripting language. A RAT is able to inject different payloads to the infected machine. The developed RAT implemented the following payloads:

- Shell: This payload is used to get a reverse-shell on the remote system (the Victim).
- Screenshot: taking screenshots can be useful to figure out some installed applications by checking screenshots of the victim machine's desktop.
- Keylog: this feature will be effective to get information such as typed passwords, credit cards details and for social engineering as well.
- Grab: copy a file from the victim machine to the attacker machine.
- Put: copy a file from the attacker machine to the victim machine.

The RAT is divided into two files; a server.py and a client.py. The server file is the one launched on the Kali machine by the attacker and acts exactly as a server, listening for incoming connections by binding itself to a given IP and port number. The client file is the one that the victim is supposed to open to start the reverse connection. The attacker sends the command to the victim machine, and the client side of the RAT executes the action locally and then sends back the result to the attacker. The interaction with the user is entirely done by command line. Once the connection is established, the main control panel of the RAT will show the commands and the instructions as in Fig 12.

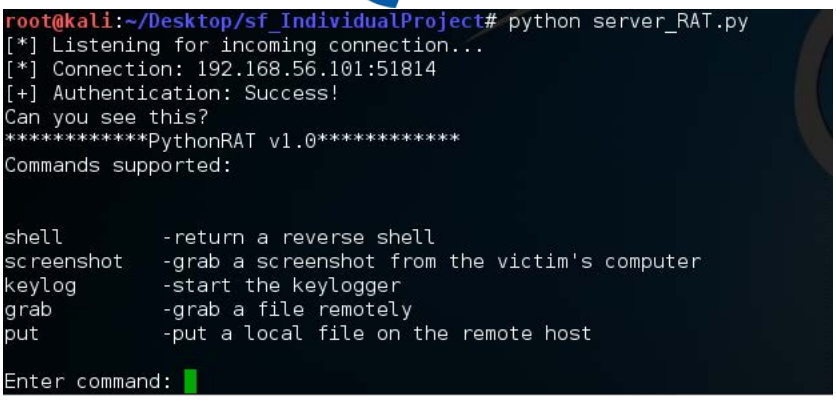

Figure 12: RAT: Main screen

3.1. SSH connection and Paramiko

The main part of the RAT is related to handling the connection between client and server. The RAT has been developed with the aim of evasion, therefore; the connection itself should be able to bypass most AVs. For this reason, the choice was to use Secure Shell (SSH) protocol. The fact that the information with SSH is encrypted gives an advantage in terms of detection, since the AntiVirus software is not able to analyse the messages exchanged in the connection. Furthermore, it is very unlikely for AVs to flag an SSH connection as malicious, due the fact that this protocol is widely used by Network and System Admins. We implemented SSH using Paramiko module of Python.

3.2. Getting the shell:

Probably the most important payload is the shell. This allows attackers to get access to the command line of the victim machine. We use the NetCat utility, to bind of a system shell. Combined with SSH, this shell shall stay undetected. The steps to get a shell are:

- The attacker sends a command to execute through the SSH channel.
- The victim, listening on the allocated port receives the command and executes it locally.
- The outcome of the command is sent back to the attacker through the same channel.

3.3. Files transfer:

Another important feature is the possibility to transfer file both ways between the victim and the attacker. This feature is implemented using SSH File Transfer Protocol (SFTP) which uses the SSH protocol to assure an encrypted file transfer. Since this protocol uses port 22 as well, it was necessary to add another Network Interface Card (NIC) (Host-Only) to the Kali machine, and then bind the SSH server to the new NIC's IP address.

3.4. Windows Keylogger:

Being able to capture the keystroke of a remote user is a good way to obtain sensitive information. Apart from registering keystrokes, the implemented keylogger is also capable of acquiring information on the application where the user is typing (e.g. if he is typing in a browser or in a word document). The keylogger was implemented using a module in python called pyHook.

3.5. Python to Executable:

The whole RAT is written in Python. Therefore, for testing purposes, the Windows 7 machine of the testing scenario has a Python interpreter and compiler already installed to understand and run Python scripts. However, in a real-life scenarios, it is unlikely that a Windows machine comes with a Python interpreter installed. The fastest solution is to convert the Python client

file of the RAT into a PE file. To achieve this, the Py2Exe utility was used to convert the .py script into .exe to work in Windows environment.

The main purpose of creating a new RAT is to test AVs' detection capabilities when analysing unknown threats. The whole RAT has been developed with evasion in mind. Using SSH and SFTP, the malware is able to encrypt the communications and evade detection engines. Furthermore, converting the script into .exe involves using binders which modifies the software signature. We tested the RAT on NoDistribute, none of the AntiVirus solution was able to flag the file as malicious as shown in Fig 13.

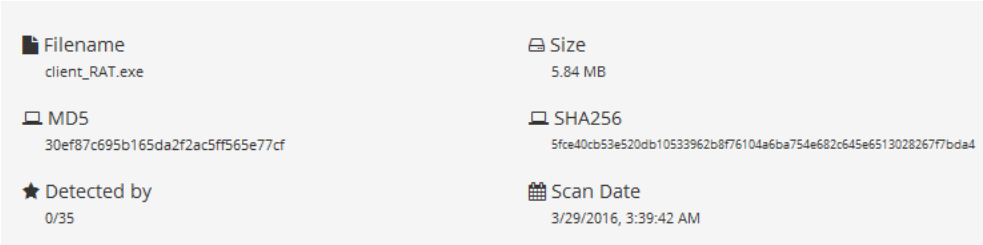

Figure 13: RAT: results by No Distribute

To confirm the result, we tested the RAT using AVG AntiVirus installed on the Windows 7 machine. Even using the Heuristic engine, AVG did not manage to detect any malicious activities as shown in Fig 14.

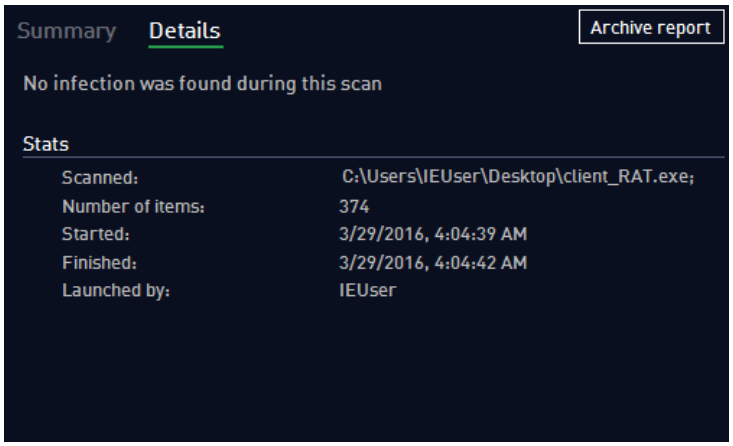

Figure 14: RAT: result by AVG Heuristic Engine

4. Intrusion Detection Systems vs RAT

There are other advanced tools that could be used to detect suspicious files such as Intrusion Detection Systems (IDSs). It is possible to implement a second layer of security using an Intrusion Detection System. An IDS is a software (or a device) that monitors the network for malicious activities or policy violations. One of the most popular IDSs is Snort which has been widely used in both production and research environments. To detect the developed RAT, we installed Snort on the Windows machine (the victim machine) and configured it to monitor and log all traffic started and destined to the attacker's network address.

The next step is to set up the RAT. Once started, a successful connection to the victim's machine is still successful, but the Snort console seemed to detect some activities as shown in Fig 15. Snort was able to see the malicious connection and flag it as "Potentially Bad Traffic".

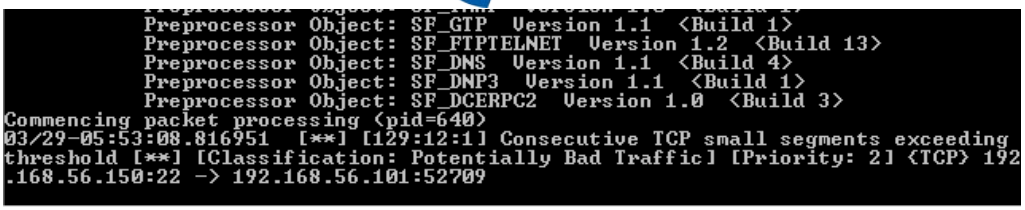

Figure 15: Snort Detection

When Snort detect a malicious packet, it saves a log file in binary PCAP format, which is possible to open with a software like Wireshark. Once opened it is possible to see full information on the detected packet as in Fig 16. The information carried by the packet is encrypted because it uses SSH protocol. However, since SSH is an application layer protocol, the remaining information in the other layers of the OSI model are not encrypted. It is possible to find out the attacker's IP address and potentially blacklist it. Furthermore, it is possible to use Snort as an Intrusion Prevention System (IPS) to block the detected packet before it can establish the SSH connection.

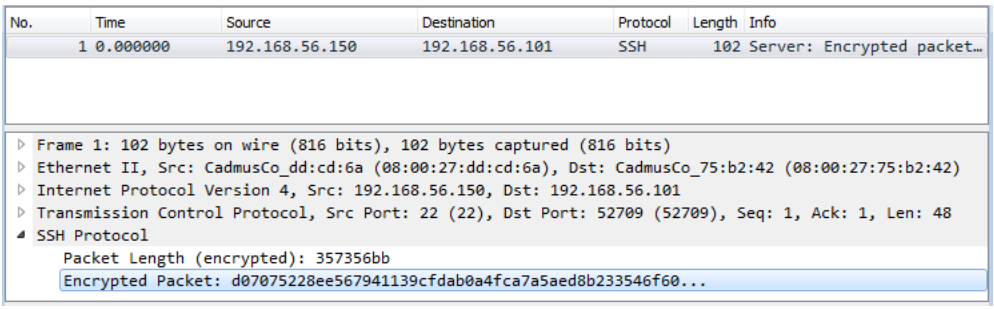

Figure 16: Wireshark Analysis

The analysis in this article is based on an online scanner while applying one (or more) evasion technique. Based on the analysis, it is possible to compare the efficiency of different AVs in detecting malware in the presence of evasion techniques. Since it is infeasible to compare every AV on the market, the presented comparison considered the AVs available in the No Distribute online scanner as well as Microsoft or Emsisoft which will be checked on Windows 7 Virtual machine. Fig 17 shows how AVs' detection ability decreases as the evasion complexity increases. The figure also shows that using Binders as one of the oldest evasion technique was successful to evade a number of AVs. This implies that these AVs analyse the first bytes of the suspected code, but not the entire file. Additionally, it obvious from the figure that a more effective evasion is achieved by combining more than one evasion technique; using a binder with Veil for instance.

Detection Rate

	Sample	Footprint Mod.	Sample +Binder	Code Obfusc.	Veil Sample	Veil +Binder	Shellter Sample	Python RAT
Kaspersky Lab								
ESET								
Bitdefender								
Avast								
AVIRA								
Panda								
Emsisoft								
Microsoft								
F-Secure								
Symantec								
AVG								
MalwareBytes								

= Threat Correctly Detected

= Threat Undetected

Figure 17: Comparison of AntiViruses