

University of Nevada, Reno

**College of Engineering, Department of Computer Science and
Engineering**

Dynamic Protection Through IPTables With An Intrusion Detection System

Kimberly Meza Martinez - Eric Valdez
CS 445 Instructor - Dr. Shamik Sengupta
May 8, 2023

Table of Contents

Table of Contents	1
Introduction	2
System Architecture	2
Installation and Set-Up	3
Threat Model	8
<i>Scripted Attacks</i>	9
<i>Manual Attacks</i>	10
Defense Model	11
<i>Communication</i>	11
<i>Parsing System/IPTables Creation</i>	11
Conclusion	12

Introduction

In a world where cybersecurity needs to be at the forefront of everyone's minds, it can be difficult to learn the various intricacies of the discipline. While many tools exist out in the world to monitor and defend your network against unwanted actions, the team wanted to create an automatic defense system that could coexist with one such tool. The primary goal of this project is to create autonomous updates to the defense systems within one's network through the use of any alerts generated by an intrusion detection system. As such, the team set out to create a fairly simple network infrastructure that would be able to automatically generate new iptables rules simply from reading the alerts generated by Snort. Through the use of several python scripts, the team created a method with which a honeypot, or other device, could run the IDS Snort and easily communicate with other devices on the network in order to block any suspicious activity. The following paper details how the simulation was conducted as well as the setup required to ensure everything works properly.

System Architecture

The simulated environment that the team used can be viewed in Figure 1 below. For the duration of this project, three virtual machines were created to signify the defender machines, attacker machines, and the honeypot respectively.

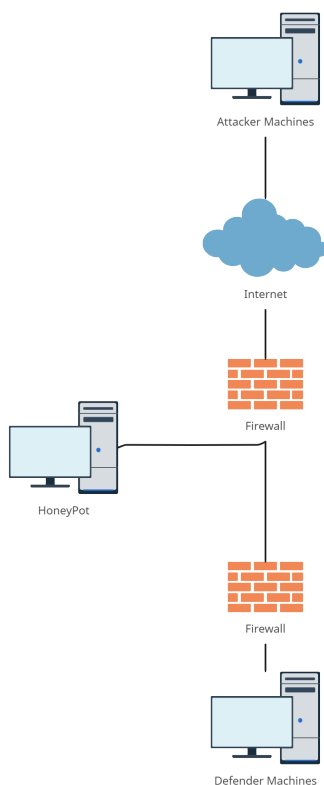


Fig. 1: Network Architecture

Each machine is running Kali Linux as the operating system which came equipped with several of the tools that were used for the creation of this project. The honeypot used was running Snort which would then generate and send all the information to the defender machines so that those firewalls would be bolstered. The external firewall remained untouched by these autonomous updates as the team wanted to ensure that all traffic being directed towards the honeypot could be intercepted by the IDS properly. This would ensure that the defenders would be able to gather as much information as possible from any malicious actors. All machines come equipped with Python 3 which was used for the creation of any and all scripts written by the team. There are two scripts that belong on the defender machine and one that needs to be running from the honeypot itself. These scripts will be specified more in the next section under Installation and Set-Up.

Installation and Set-Up

As described previously in the system architecture section, we will be using three virtual machines for the purposes of this project. All testing for this project was conducted with the use of Oracle VirtualBox which can be downloaded from the following link: <https://www.virtualbox.org/wiki/Downloads>. Kali Linux downloads can be found at here: <https://www.kali.org/get-kali/#kali-platforms>. Once VirtualBox has been installed, create a linux virtual machine from the virtual hard disk and VDI options. For each Kali machine, make sure that once they have been properly installed that the “Enable PAE/NX” option is selected under Settings and System as seen in the figure below.

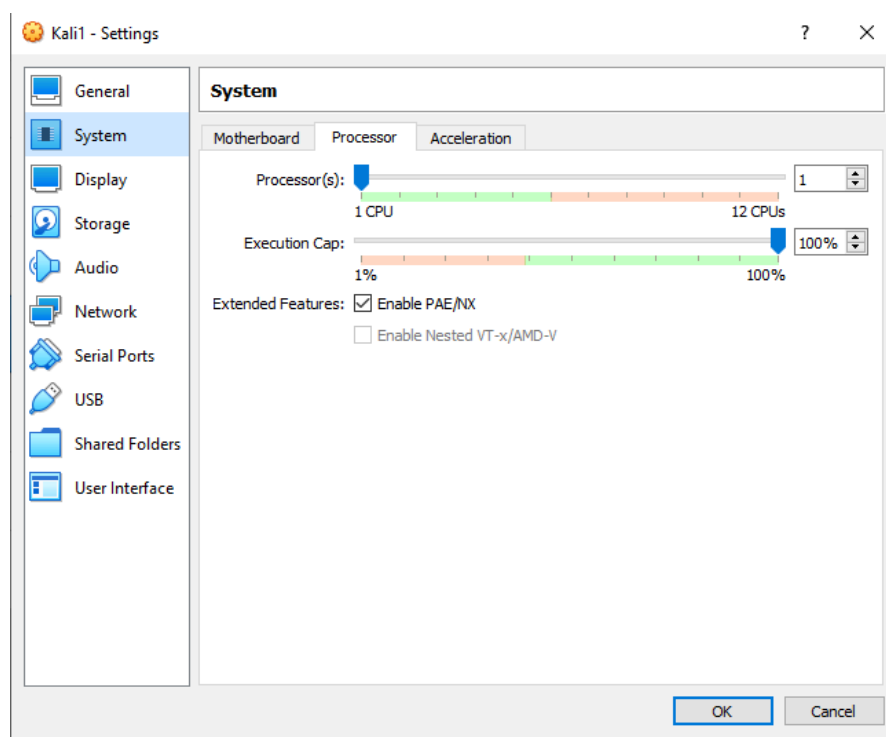


Fig. 2: Enable PAE/NX for each virtual machine

Once these options have been selected, begin the virtual machine and select the Kali installation options you desire. Finally, we need to ensure that the virtual machines are able to properly communicate with each other. Within the VirtualBox application, select “file” and then go into “Host Network Manager” and ensure that “DHCP Enable” is selected as seen in Figure 3 below.

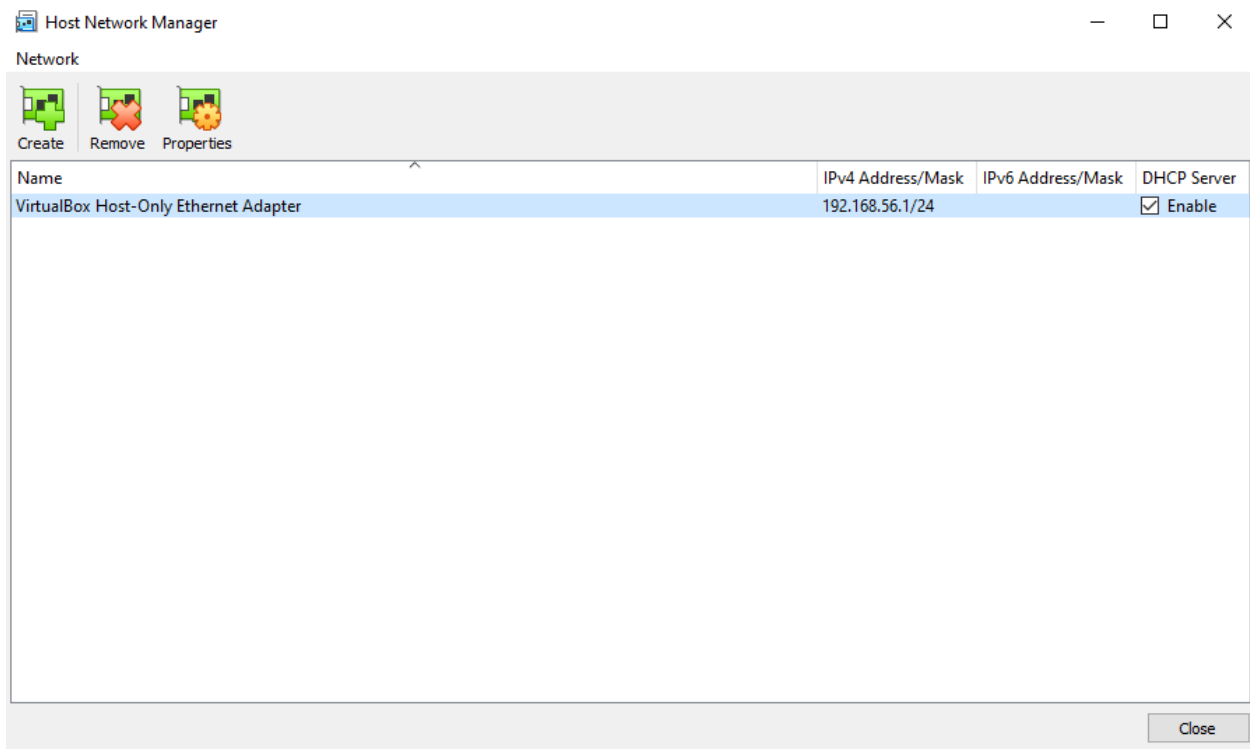


Fig. 3: Enable DHCP for each virtual machine

The last step for setting up these machines is to ensure that the network adapter is set to the “Host-only Adapter” setting. These settings should ensure that the virtual machines are able to communicate with one another, and this can be tested simply by pinging one of the machines from another.

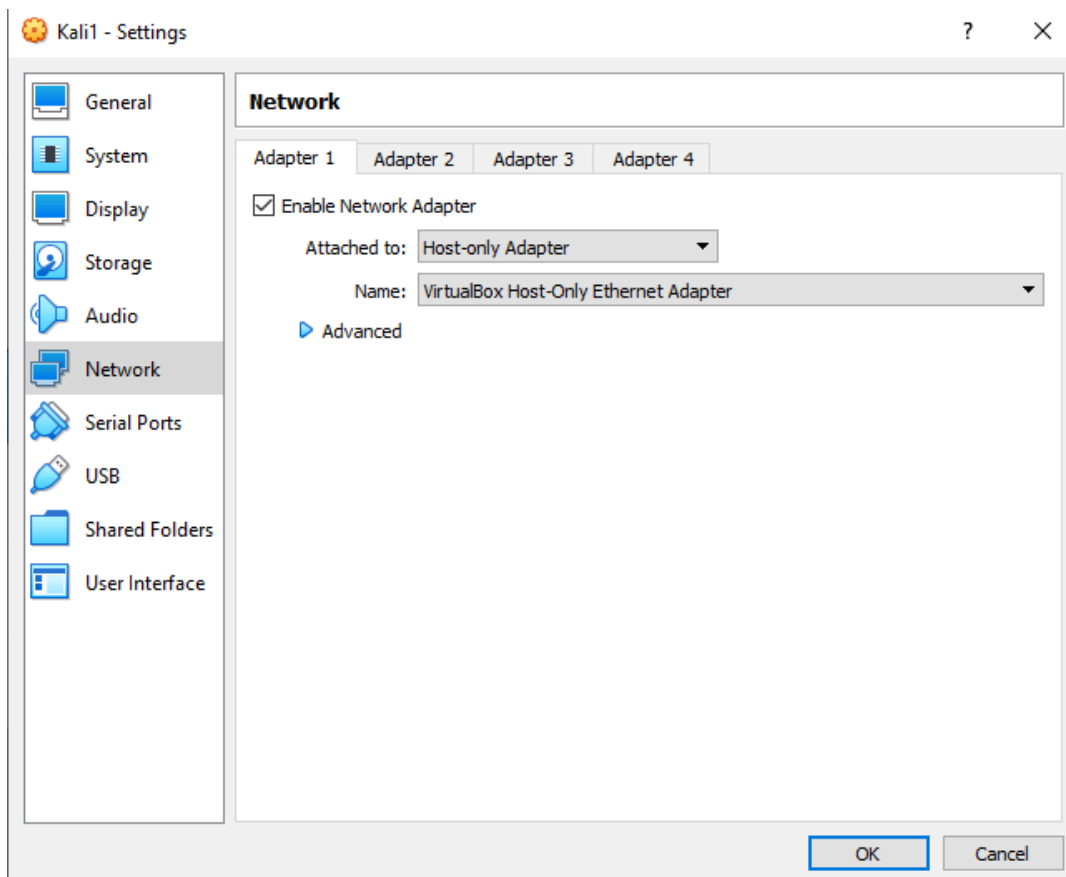


Fig. 4: Proper network adapter settings

Now that the machines have been properly configured, we are able to install all options required for the project. Kali Linux comes equipped with many of the tools already needed by the project, however several may need to be installed depending on the version of Kali. On whichever machine you deem as the attacker machine, ensure that Scapy is installed as this will be part of the manual threat model. If this is not already present, simply run the command “sudo apt install scapy.” On the same machine, download the “attacker.py” script that was provided as part of this project. This file can be placed anywhere, though for the simplicity the team kept this file on the desktop for ease of reference. At the top of this file is a macro where the user can place the target IP address

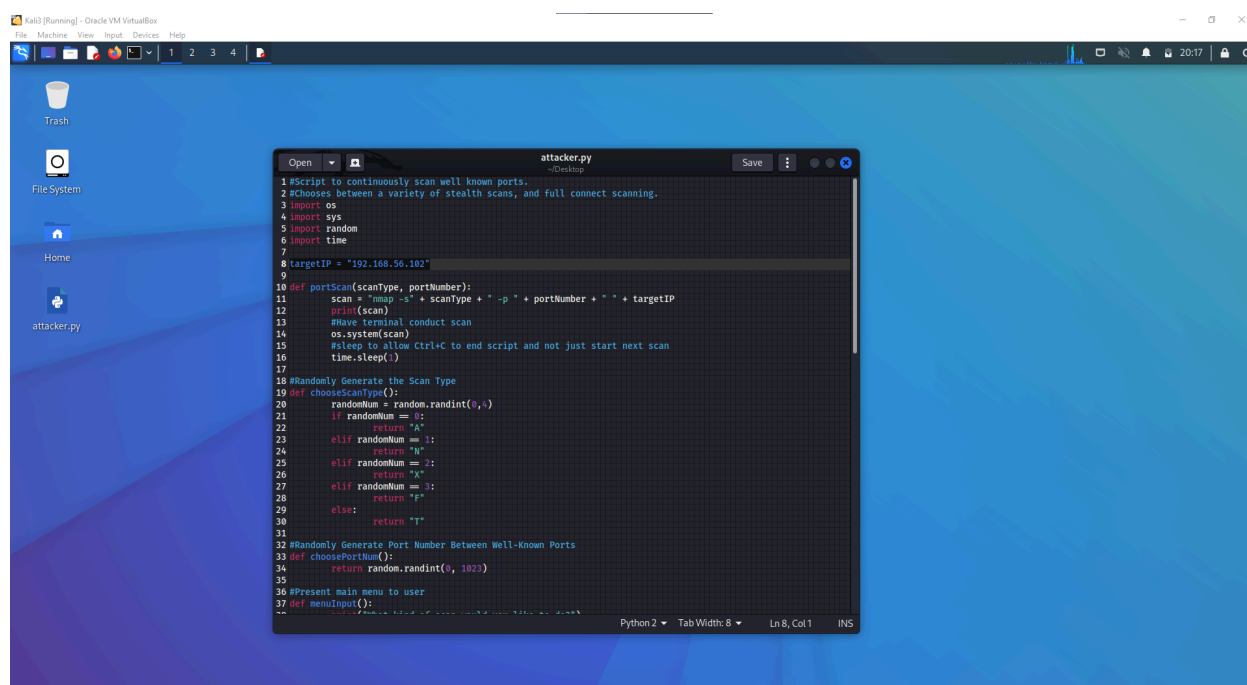
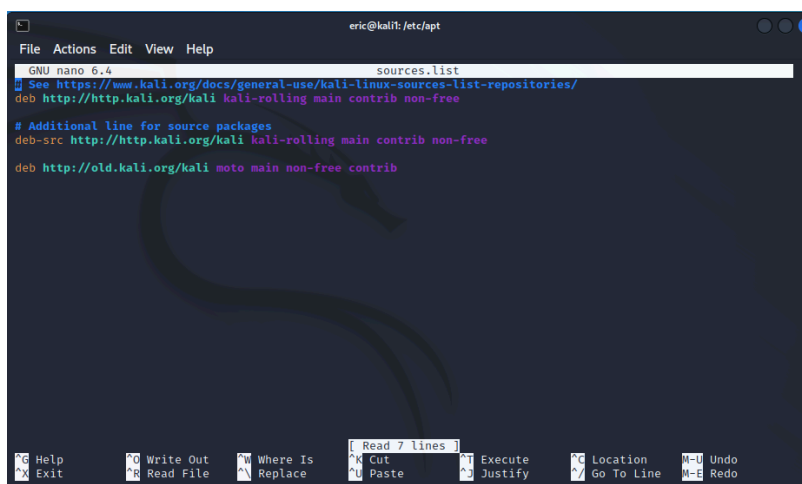


Fig. 5: Attacker machine with the attacker.py script

On the machine that you have deemed as the honeypot/IDS, Snort needs to be installed. Provided that the machine does not already have snort installed, ensure that the sources.list file found in /etc/apt has the same entries as Figure 6 below. Then run the commands “sudo apt-get update” followed by “sudo apt-get install snort.” This should install the necessary IDS onto this machine.



```

eric@kali: /etc/apt
File Actions Edit View Help
sources.list
# See https://www.kali.org/docs/general-use/kali-linux-sources-list-repositories/
deb http://http.kali.org/kali kali-rolling main contrib non-free

# Additional line for source packages
deb-src http://http.kali.org/kali kali-rolling main contrib non-free

deb http://old.kali.org/kali moto main non-free contrib
  
```

Fig. 6: Sources.list entries necessary for Snort

On the honeypot/IDS machine, the “sender.py” script is also necessary. This script should reside within the directory that any Snort alerts are being generated. For the purposes of this project, this script resides within a directory labeled as “log” within the downloads directory for easy access. The receiver IP will also need to be updated within the sender.py script. The rules file will also need to be placed within this same directory.

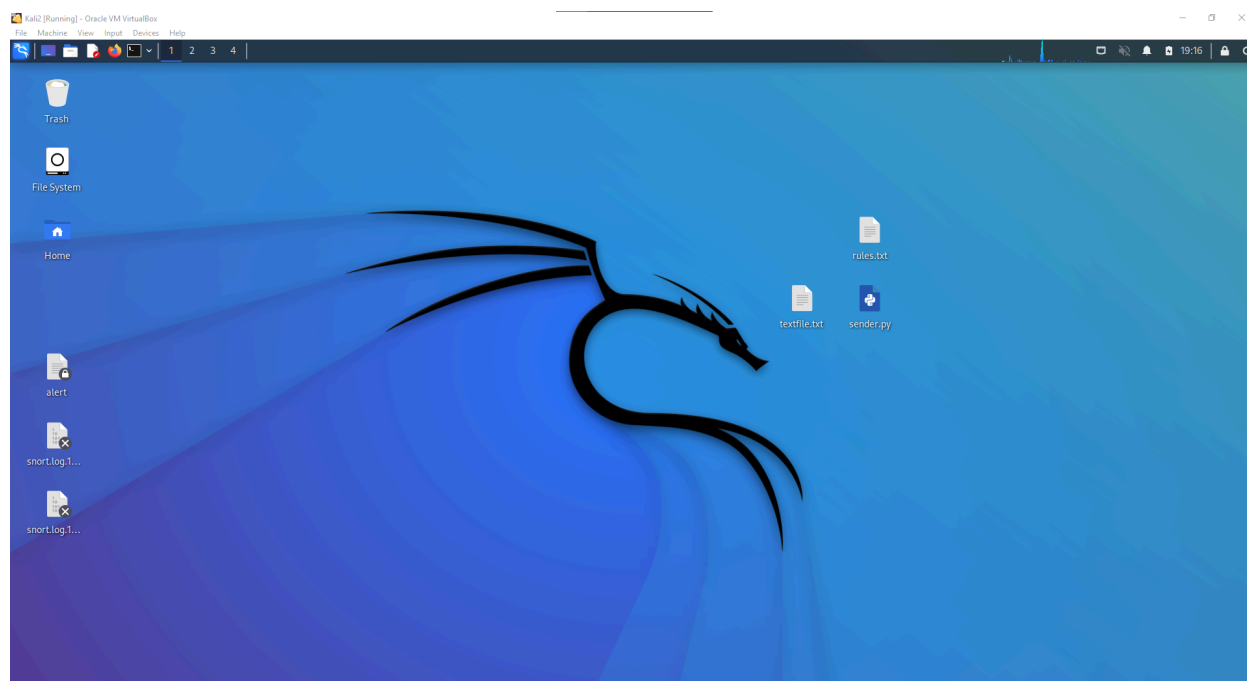


Fig. 7: Honeypot/IDS with sender script, rules and textfile.txt

Finally, the last step in setting up is downloading the parsing script and the receiver scripts. In addition, a new directory named “log” will need to be created. Within the log folder, the receiver script should be placed and the parser should be kept in the parent directory. These scripts will pull in and write the alert files from the honeypot machine to the defender machine. In addition, this program uses two textfiles named “textfile.txt” which will temporarily hold any alerts sent from the IDS. The second file is “newFile.txt” which will hold all alerts until they are parsed. These two files should be created through the scripts, however if errors occur simply create these two empty text files with their respective names.

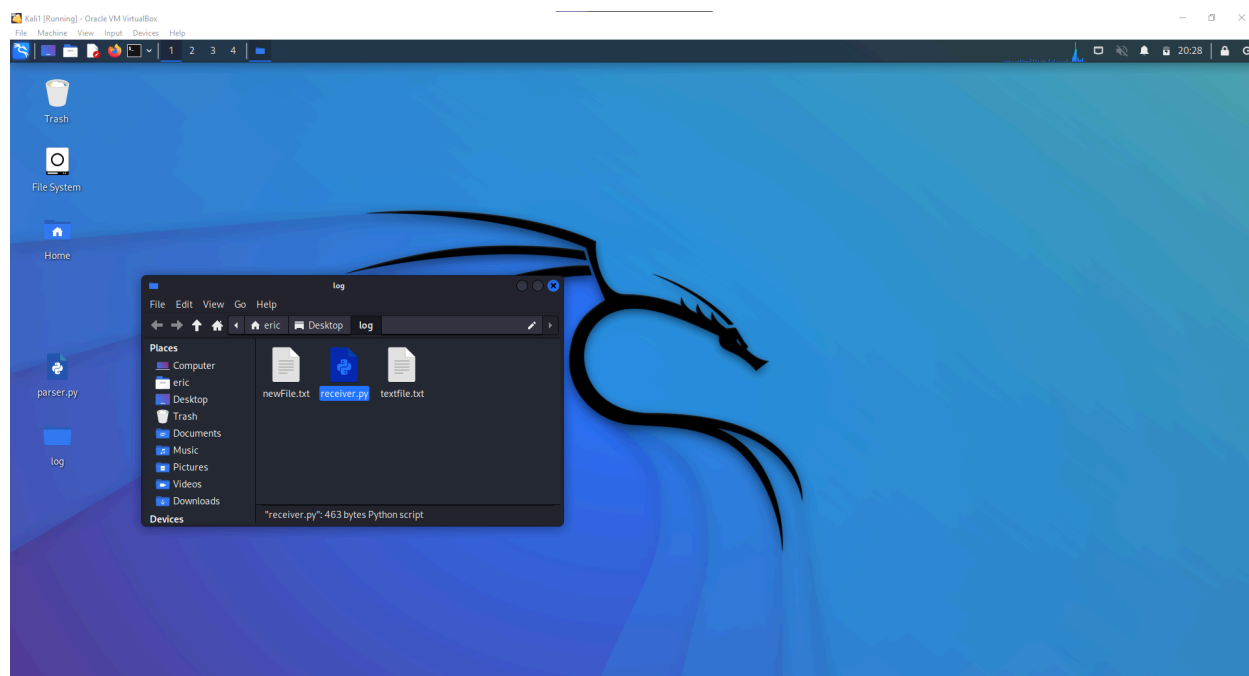


Fig. 8: Parser and receiver files placed within the defender machine

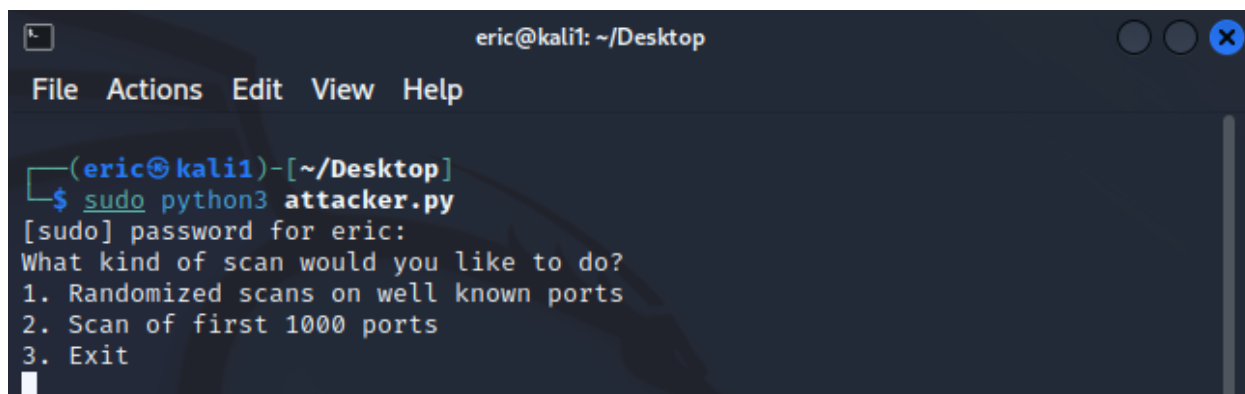
As each script provided makes use of functionality only available to the superuser group, make sure to run each python script with “sudo” to ensure the functionality works as expected.

Threat Model

The threat model for the project details the main components: a scripted component and a set of manual attacks. The scripted attacks can be found within the attacker.py script that should be installed onto the attacker machine as described previously. The manual attacks use a combination of nmap scans and Scapy in order to perform various stealth scans that originate seemingly from different hosts.

Scripted Attacks

The `attacker.py` script will offer the user a simple method with which to conduct a variety of scans. As many of the scan types in the script are stealth scans, the user will need to run the script prefaced with “`sudo`.” Once the program has been run, the user will be presented with a small menu where they can choose to randomly scan well known ports or to run a scan of the first 1000 ports as seen below in Figure 9.

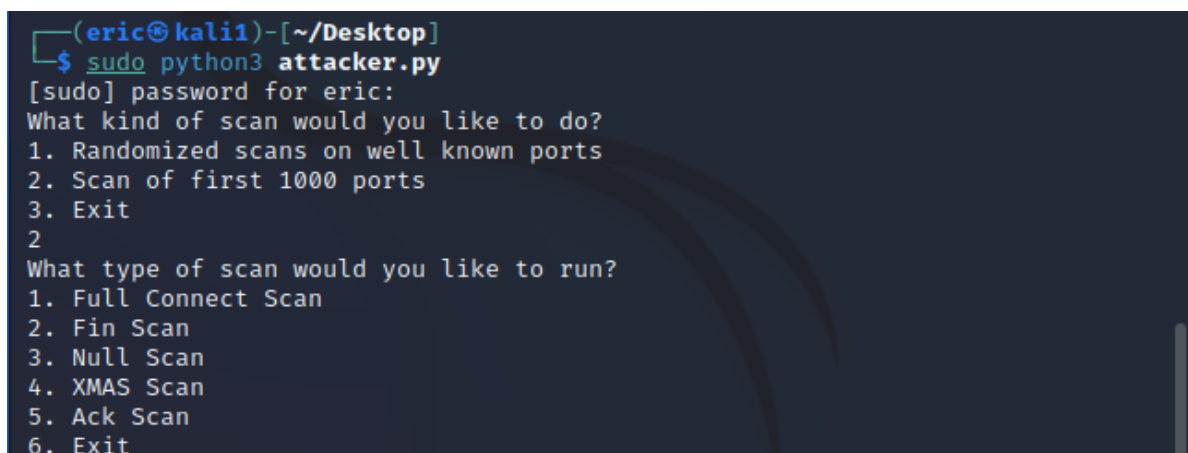


```
eric@kali1: ~/Desktop
File Actions Edit View Help

(eric@kali1)-[~/Desktop]
$ sudo python3 attacker.py
[sudo] password for eric:
What kind of scan would you like to do?
1. Randomized scans on well known ports
2. Scan of first 1000 ports
3. Exit
```

Fig. 9: Attacker scripted options

If the user chooses option 1, the script will perpetually scan the target IP on a random port between 0-1023. In conjunction with the ports being randomized, so too is the scan type. Each scan that runs will randomize between a full connect scan, an ACK scan, a FIN scan, an XMAS scan, and a NULL scan. These scans will continuously be conducted until the user ends the program with the CTRL + C hotkey. If the user chooses option 2, they are presented with a second series of options to choose from. These options can be seen below in Figure 10 wherein the user is choosing what type of scan they would like to complete on the target host.



```
(eric@kali1)-[~/Desktop]
$ sudo python3 attacker.py
[sudo] password for eric:
What kind of scan would you like to do?
1. Randomized scans on well known ports
2. Scan of first 1000 ports
3. Exit
2
What type of scan would you like to run?
1. Full Connect Scan
2. Fin Scan
3. Null Scan
4. XMAS Scan
5. Ack Scan
6. Exit
```

Fig. 10: Menu options for scan of first 1000 ports

Once the user makes a selection from this menu, nmap will run a scan on the target host for the desired scan type. This method has a simple implementation that is primarily meant to make it easier for a user to complete a scan without the need to understand or worry about the syntax regarding nmap. Once the scan is complete, the user is once again presented with the original menu again. Finally, if the user chooses to exit, the program will close.

Manual Attacks

Prior to the attacker script being created, the original threat model was primarily based on manual inputs from the user. While these can still be completed, especially if the user plans to look into a specific port themselves, the script will allow for a much easier view of the project as a whole. One mechanism that does still serve a purpose is Scapy. Scapy can be used to create a wide variety of packets which can be used to create simulated scans with regards to illustrating different hosts attempting these scans. By creating a TCP packet with the proper flags set, the user can launch a simulated scan packet with a source IP that differs from their own host. As seen in Figure 11 below, after crafting a packet with the source IP of 172.168.54.101, the IDS is still able to create the proper alert. Figure 12 demonstrates how the parser is able to create the proper rule and place it into the iptables. This will be able to block further attempts from that IP address.

```

7 [**] [1:4:0] alert scan nmap FIN [**]
8 [Priority: 0]
9 05/03-20:09:55.736447 08:00:27:4E:CA:A9 → 08:00:27:7A:3B:53 type:0x800 len:0x3C
10 172.168.54.101:20 → 192.168.56.102:80 TCP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:40
11 *****F Seq: 0x0 Ack: 0x0 Win: 0x2000 TcpLen: 20
12
13

```

Fig. 11: Alert detects Scapy packet with FIN flag set.

```

eric@kali: ~/Desktop
File Actions Edit View Help
(eric@kali)~[~/Desktop]
$ sudo python3 parser.py
[sudo] password for eric:
(eric@kali)~[~/Desktop]
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source                destination
REJECT    tcp  --  192.168.56.103         anywhere             tcp flags:FIN,SYN,RST,PSH,ACK,URG/FIN,PSH,URG reject-with tcp-reset
REJECT    tcp  --  172.168.54.101         anywhere             tcp flags:FIN,SYN,RST,PSH,ACK,URG/FIN reject-with tcp-reset

```

Fig. 12: Proper rule placed into iptables.

Defense Model

Communication

Communication between the honeypot and the defender machines are handled between two simple scripts: receiver.py and sender.py. The receiver.py script is run on the defender machines and opens up port 1234 in order to listen for any traffic being sent towards it. Sender.py will hash the alert file that has been generated by Snort and if that hash was different than the last file it sent, the honeypot will forward this information to the defender. The hashing algorithm used is a simple SHA-1 hash as this was not meant to provide any cryptographic significance, merely to observe any changes to the file. This method of communication does pose a certain vulnerability to the system however, as this mode of communication is not strictly encrypted. As the scope of this project was not focused on how to effectively transmit the data in a secret manner, this was a known risk to the system that the team worked with. If this project was ever to be deployed in another system, this vulnerability would need to be addressed.

Parsing System/IPTables Creation

The parsing system plays a crucial role in identifying the type of scan indicated in the generated alert file and determining the appropriate iptables command to mitigate that specific attack. This is similar to Snort in which Snort creates a signature when alerted. It first parses the alert file for the comment designated by the Snort signature. This comment identifies the type of alert it is such as XMAS, FIN, or NULL. Then the program parses the comment and the IP address of the attacker and saves both of the information. Utilizing this saved data, the system generates an iptables rule tailored to counteract the specific attack originating from the attacker's IP address. As we strive to enhance and expand the existing system, we hope to add more iptables commands that could be specific to a wide variety of attacks or scans. Further steps would be to enable the system to act less like a low interaction honeypot and more similar to a medium interaction honeypot. A notable advancement in this direction would be simulating a service, enabling a more realistic and comprehensive engagement with potential attackers.

```
(eric@kali1)-[~/Desktop]
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target      prot opt source                destination          tcp flags:FIN,SYN,RST,PSH,ACK,
REJECT      tcp  --  192.168.56.103         anywhere              URG/FIN,PSH,URG reject-with tcp-reset

Chain FORWARD (policy ACCEPT)
target      prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
```

Fig. 13: Automatic iptables rule creation for detected xmas scan

Conclusion

In conclusion, this project approached the idea of utilizing an intrusion detection system and several python scripts in order to dynamically improve the defenses of a user's system. This project has been created in a manner that should benefit both those who are knowledgeable in the subject material as well as those who are new. The alerts being generated have messages that are meant to be easily readable and with the automatic portions of both the parser and the attacker scripts it should be considerably easier to test these methods for anyone new to nmap and iptables in particular. While the design of a system such as this is restrained by the necessity of creating all the rules that will block any malicious traffic, this project is made in a simple manner in order to be built upon further. Overall, the methodology behind this project provides any network administrators some basic tools to proactively protect their network against malicious attacks.