**Building a Better Honeypot**

Alexander Rossetti

Kentros

Kentros

May 11, 2020

**Student Objectives**

In my project I attempted to create two medium-interaction SSH honeypots that were secure, enticing to hackers, and could successfully analyse and categorize current threats with the data produced from the deployment of the created honeypots; in order to learn and help those interested in information/cyber security and those wanting to strengthen existing SSH systems. While there are many tools to help protect systems in the IT industry, only a Honeypot has the ability to entrap an attacker while simultaneously studying techniques of hackers, their methods of attack, how the attacks happen, their targets and motivations, as well as to adapt to these attacks as cybersecurity grows. The main objective of my project was to modify and improve a honeypot system using applications with personal tools and techniques to process attack-related data, to automate analysis and to create an intelligent honeypot to compare to an existing honeypot framework, Cowrie.

A top priority of this project was to learn how to create an advanced, stable, and secure honeypot system to ensure that if something went wrong, no other devices could be compromised. Similar to penetration testing, honeypots are the opposite of security by design. It is very much an educational exercise, where one creates machines to tell them where they may be vulnerable. While honeypots produce useful cyber-security information, it is essentially painting a bullseye on oneself to hackers and malicious activity. An improperly configured honeypot security system could pose a high security risk to a network and other internal systems, or even be used as part of a botnet to attack other networks. Doing such a project taught me the basics of information-security as well as how to defend and secure a linux environment for multiple systems against different types of attacks and threats my system might face. I got first hand experience on improving security within a network. To solve the threat of my systems and honeypot being compromised, I wrote many python scripts to program these features and communicate with different ports and tools. Outside of gaining python programming skills, my project challenged my ability to incorporate my knowledge of python and it’s command line to communicate with the inner workings of vm’s, linux, and its security mechanisms.

Major components of this project were the quality of the data my honeypots produced, as well as the management of the information compiled by the honeypot. I aimed to have easy-to-understand data sets that provide insight on potential risks. Knowing what attackers may try to do if they gain unauthorized access will help us better defend systems.

Activity on the honeypot was recorded by several data analysis and network packet captured tools, of which I gained experience with, and intimate knowledge of. During this project I considered whether or not the type of information that was obtained from the honeypot, however, was useful from a technical perspective and whether it was sufficiently detailed enough to be able to obtain a trace from the actions developed. For this reason, the honeypot had to be able to present the information in an organized manner. The sheer amount of data and logs generated by most honeypots can grow massive in size when there is heavy attacked traffic in the system, thus consuming a lot of disk space. I developed data analysis and forensic skills by organizing, interpreting, and improving upon the collection and analysis of the network traffic in my honeypots. The skills I gained while managing the honeypot database included log management and examination, database use, remote log storage, formatting stored information and generating reports.

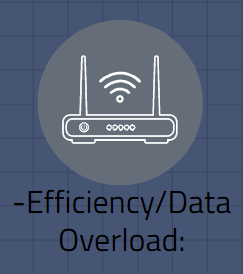
The reason for choosing Linux for this project was that it was open-source and highly customizable, making it easy to interchange features of this project if future development were to happen. A major milestone when working on my project was my ssh honeypot successfully running as a server in the Linux operating system environment while I hosted this project on multiple virtual machines. This provided hands-on experience with deploying, using, and securing virtual machines as well as using Linux. Completing all these goals not only greatly improved my programming and network/vm security skills, but also examined my ability to manage a major project in an organized and timely manner.

**Problem Specification**

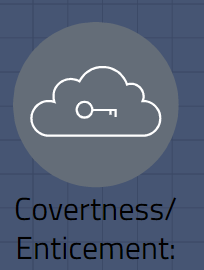
When considering what to do for my capstone project, I thought about what the utmost important issue facing the digital world today is, and that is information-security. Privacy and security are basic needs for all people, and with 3.77 billion internet users globally, nearly all of these users have had personal and confidential data online, and where there is personal data, there is always someone trying to gain access. Privacy and security are under threat more than ever before. As more and more devices and systems become internet-connected, the importance of battling back against those who use the internet as a weapon will only increase. My overall goal was to help combat the growing threat of cybercrime, and to do this I improved upon an essential tool in current cyber defensive arsenals, and to specifically create a honeypot that strives to improve upon many of the issues current honeypots have. A honeypot is a decoy computer system with voluntary vulnerabilities to threats, designed to look like a legitimate system, which tempts an intruder to break into it while unknown to the intruder, he is being covertly observed. These current issues include the work to sift through the massive amount of data, the honeypots ability to efficiently categorize and analyze traffic, the honeypots being identified by hackers, and the vulnerability of the honeypot to being compromised.



Yes, one should expect a honeypot to be probed and attacked, but one should also consider the potential for it to be exploited as well. Serious problems arise once a honeypot gets hijacked and used to attack, infiltrate, or harm other systems or organizations. One of the biggest inherent flaws with honeypots is the risk that they, and your systems, will be compromised. To assist with security and helping you monitor ports, there’s a plethora of tools I utilized to do just that. This project, on top of using tools, adds extra protection by including writing a python file to act as a honeypot. This honeypot python file has the capabilities to take on different profiles and simulations, improving security and tricking hackers.



A Honeypot aims to capture and analyze the data of intruders, observing the behavior and providing information about security threats. In conclusion, a honeypot is only as good as the data it produces. It is hard to pinpoint and identify what was important to investigate. If everything is important to investigate, then nothing is important. One of the main problems I aimed to improve upon the honeypot design was it’s efficiency at capturing, as well as the analysis of network attack traffic. In general, data collected by honeypots are stored in log files so that it can be subsequently analyzed by an analyst. However, the identification and characterization of attacks from huge log files is a tedious and challenging task because of the massive volume of collected traffic. Honeypots collect a large amount of data that contain interesting information about attackers. However, the huge amount of collected data can rapidly overwhelm the person in charge of analyzing such datasets. To avoid such a problem, I have integrated into my ssh honeypot automatic analysis techniques based on my honeypots ability to adapt to different ports as they are being scanned, spoof packets and send them to different ports. Another method was to limit the massive amount of traffic coming through to only two ports. The goal of this module was to identify and characterize the captured attacks in an automatic and highly efficient way, categorizing it into graphs and easy to understand data sets.



A major problem with many honeypots that I attempted to solve is enticement and covertness. Simply setting up a fake server on a network may not be enough to attract attackers. This is best done by making the honeypot security system appear to contain information that is useful or valuable to the attacker, such as customers’ credit card information, human resource records, company financial data, trade secrets, confidential email and documents, and so forth. I must create a realistic environment for my honeypot that simulates real network activity. For example, giving the server an enticing name, running services and applications that are generally used for such purposes, naming files to indicate they hold this type of data, and actually providing false information in the contents of the files will get the attention of hackers. The honeypot must appear to hackers to be a real production machine that is vulnerable to attack, and I used dummy items and hosted a website that is a high-value target on my honeypot system to do just that. The goal of data capture in honeypots is to capture all the attacker's activity without detection, as once a honeypot is spotted, it is often flagged and the attack data is impaired. With this knowledge, my honeypot's ability to adapt to different ports as they are being scanned and spoof packets and send them to different ports also helped the security and covertness of the system, as it confused attackers, and prevented the system from being identified as a honeypot.

**Implementation/Journal**

The first decision I had to make was the device that was going to act as a honeypot. I have numerous old PC’s laying around, and there was no reason why this would not work on a standard PC with an operating system such as Cent O/S or Ubuntu. However, I needed a machine that was powerful enough to host three VM’s simultaneously, so I chose my fairly-powerful old gaming laptop.

**1. Installing VMware Workstation:**

I navigated to https://www.vmware.com/products/workstation-pro.html and purchased VMWare Workstation. After purchasing, I downloaded the installer provided on the website. I ran the installer and followed the install wizard to configure it for the machine it's running on.VMware is now installed.

**2. Installing a CentOS Virtual Machine:**

I Navigated to <https://www.osboxes.org/centos/#centos-7-x-info> and downloaded the latest VMware version of CentOS.iso file. When this file is downloaded, extract it using WinRaR or a similar tool. I opened VMware Workstation, clicked file, then new virtual machine. I then hit custom installation, then installer disk image file, and navigated to the folder which was extracted earlier and opened the virtual machine inside. I allocated VM ram and system memory (I recommend at least 4g of ram and 100 gb of harddrive space) CentOS is now installed.

**3. Installing a Kali Linux VM:**

I Navigated to <https://www.oensive-security.com/kali-linux-vm-vmware-virtualbox-hyperv-image-download/> and downloaded the .iso VMware version of Kali 64 Bit. When this file was downloaded, I extracted it using WinRaR or a similar tool. I opened VMware Workstation, and clicked file, then new virtual machine. I hit the custom installation installer disk image file, and navigated to the folder which was extracted earlier and opened the virtual machine inside. I allocated VM ram and system memory (I recommend at least 4g of ram and 40 gb of harddrive space) Kali is now installed.

**4. Installing ScaPy on the CentOS VM:**

I Opened the command line on the CentOS VM and entered the following command to enable the EPEL repo:

sudo yum install epel-release

Now that the EPEL repo is enabled, it can be used to install pip using the following command:

sudo yum -y install python3-pip

Finally to install ScaPy I entered the following command:

sudo pip install scapy

**5. Installing T-shark on the CentOS VM:**

To install T-Shark, the installation steps for installing Wireshark must be followed as it is packaged together. To install Wireshark, I simply entered the following command:

sudo yum install wireshark

Wireshark and T-shark were then installed.

**6. Disabled services or packets using ports 22 or 445 on the CentOS VM:**

To remove SSH from the Virtual Machine, which uses port 22, I entered the following in the command line:

sudo yum remove openssh-server

As this is a Linux based machine, microsoft-ds will not be installed on it. This means port 445 is not already in use.

**7. A basic python script has been created.**

From the command line I opened a vim text editor for creating the python script:

vim capture.py

I added the following lines of code inside this for a basic hello world script:

#!/usr/local/bin/python2

print("hello world")

To close the VIM you press escape, I typed :wq

To run the python script, in the command line I typed:

python3 capture.py

**8. T-shark can be called from a python script.Open the python script again in VIM:**

vim capture.py

I added the following line near the top of the python file, this is importing the OS library which was used to call tshark on the command line from python:

import os

I added the following line near the bottom of the script, this calls tshark on the command line from the python script:

os.system("tshark -i any -w honey.pcap -q ")

When this script was run, it started a tshark capture and logged the files in honey.pcap.

**9. Making the script capable of logging any activities on ports 22 and 445:**

To log the activity on only port 22 and port 445, I edited the same line as above to look like the following:

os.system("tshark n"tcp port 22n" or n"tcp port 445n" -i any -w honey.pcap -q ")

The script then logged activity only to ports 445 and 22.

**10. Manipulating the script to only show every port as open:**

In order to show each port as open using ScaPy, the following libraries were imported into the code near the top:

from scapy import all

from scapy.all import \*

Next, in order to craft scapy packets to make the ports appear as open, I defined a method and added the following lines to the code (remember to tab the code):

os.system("iptables -A OUTPUT -p tcp -o eth0 {sport 1:65535 {tcp-

ags RST

RST -j DROP")

def packet(pkt):

if pkt[TCP].

ags == 2:

print('SYN packet detected port : ' + str(pkt[TCP].sport) + ' from IP Src : ' +

pkt[IP].src)

send(IP(dst=pkt[IP].src, src=pkt[IP].dst)/TCP(dport=pkt[TCP].sport,

sport=pkt[TCP].dport,

ack=pkt[TCP].seq + 1,

ags='SA'))

sni

(iface="eth0", prn=packet, lter="tcp[0xd]&18=2",count=100)

os.system("iptables -D OUTPUT -p tcp -o eth0 {sport 1:65535 {tcp-

ags RST

RST -j DROP")

When this python script is run, it shows each port as open, however after each port is scanned they remain closed again, so I added a while loop around the script to keep it running continuously. To do this I added this line before the packet crafting section of the script and tab the rest of the section out :

while True:

**11. Manipulating the script to only show ports 445 and 22 as open:**

Instead of every port appearing as open, I only wanted ports 445 and port 22 to appear as open. To do this, an if else function was needed inside the python script. These can be seen here in blue:

os.system("iptables -A OUTPUT -p tcp -o eth0 {sport 1:65535 {tcp-

ags RST

RST -j DROP")

def packet(pkt):

if pkt[TCP].

ags == 2:

if(str(pkt[TCP].dport)) == "22":

print('SYN packet detected port : ' + str(pkt[TCP].sport) + ' from IP Src : ' +

pkt[IP].src)

send(IP(dst=pkt[IP].src, src=pkt[IP].dst)/TCP(dport=pkt[TCP].sport,

sport=pkt[TCP].dport,

ack=pkt[TCP].seq + 1,

ags='SA'))

sni

(iface="eth0", prn=packet, lter="tcp[0xd]&18=2",count=100)

os.system("iptables -D OUTPUT -p tcp -o eth0 {sport 1:65535 {tcp-

ags RST

RST -j DROP")

if pkt[TCP].

ags == 2:

elif(str(pkt[TCP].dport)) == "445":

print('SYN packet detected port : ' + str(pkt[TCP].sport) + ' from IP Src : ' +

pkt[IP].src)

send(IP(dst=pkt[IP].src, src=pkt[IP].dst)/TCP(dport=pkt[TCP].sport,

sport=pkt[TCP].dport,

ack=pkt[TCP].seq + 1,

ags='SA'))

sni

(iface="eth0", prn=packet, lter="tcp[0xd]&18=2",count=100)

os.system("iptables -D OUTPUT -p tcp -o eth0 {sport 1:65535 {tcp-

ags RST

RST -j DROP")

Only ports 22 and 445 appeared as open now.

**12. Running the script when the OS boots:**

In order to do this Crontab is used. I opened a Crontab editor in the command line using:

crontab -e

I Added the following line to the crontab to make the script run on reboot:

@reboot python /home/osboxes/script/capture.py

The script now ran on startup for the root user.

**13. Setting up a Slack Channel for message alerting:**

I navigated to slack.com in a web browser. I clicked to get started, and then clicked create a new workspace. I registered for a Slack account, entering a full name and display name for the user which appeared on the Slack channel. I set a password. I told Slack about the Team which will be using the Channel. I entered a name for the Slack Channel. Finally, I hit create workspace and the channel was created.

**14. Configure a Chatbot on Slack for honeypot alerts:**

I logged in to the Slack Channel previously set up. On the left hand pane, I hit the "plus" button next to Apps. In the Search bar that appears, I searched for Webhook. I found the App named Incoming WebHook from the list and hit install next to the App. I gave the WebHook a name which appeared in the Channel, and uploaded an icon for it. I recorded the WebHook URL given, as it was used later. The Chatbot was then congured.

**15. A script has been created to send the Honeypot alerts to slack:**

I created a new python script called notifyslack.py using:

vim notifyslack.py

At the top of the python script, the following libraries were imported:

from collections import Counter

import os

import slackweb

Next the Slack WebHook link from earlier was entered in a variable in the script:

slack =slackweb.Slack(url="ENTER WEBHOOK URL HERE")

Next towards the end of the script, but above the main method, a method for getting a list of IP's from the pcap file and sorting them by most common was made:

def getIPs():

The FIRST step of this method was to open the pcap file and read each ip source address found in it, into a txt file:

os.system("tshark -r .cong/honey.pcap -T elds -e ip.src ! :config=honey:txt")

After the txt file is created, the IP address of the local machine was read into a variable which was used later:

localIP = socket.gethostbyname(socket.gethostname())

Then an IP array was created for reading in each IP address from the txt file, while ignoring the IP address from the local machine:

ip array=[]

with open('.cong/honey.txt') as ip\_file:

for ip in ip\_file:

ip = ip.strip()

if ip != str(localIP):

ip array.append(ip)

This array was then sorted into another array, which sorted it by most common and recorded the count of each repeated IP address:

count ips=(Counter(ip array).most common())

Next a different method for sending the Slack alerts was created:

def notification(count ips):

Arrays for both the top ip addresses and the number of attacks were then denied:

def notification(count ips):

TopIP=[]

NumberAttack=[]

for ip, attacks in count ips:

TopIP.append(ip)

NumberAttack.append(attacks)

A slack alert was then generated inside the same method for the top 3 or less attacks:

if not TopIP:

attachment = f"color": "#008000",

"title": "No IP's attacked today"g

else:

if len(TopIP) == 1:

attachment = f"color": "#FFFF00",

"title": "Top Honeypot Attacker IP:",

"text" : "1." + str(TopIP[0]) + " - " + str(NumberAttack[0]) + " attacks"g

I repeated the second part of the if statement, adding on extra parts of the TopIP and NumberAttack arrays for the alert on 2 IP's and alert on 3 IP's. I sent the Slack alerts using the following lines of code:

attachments.append(attachment)

slack.notify(attachments=attachments)

The alert script then sent alerts showing the top 3 or less attackers.

**16. Hosted a website on the Honeypot.**

From the terminal, I entered the following command to install Apache:

sudo yum install httpd

I changed directory to the httpd files where the website will be stored:

cd /var/www/html/

In a web browser, I navigated to <https://github.com/dogecoin/dogecoin.com> and cloned the files from this repository into the httpd files and extracted them. I restarted Apache from the command line using:

service httpd restart

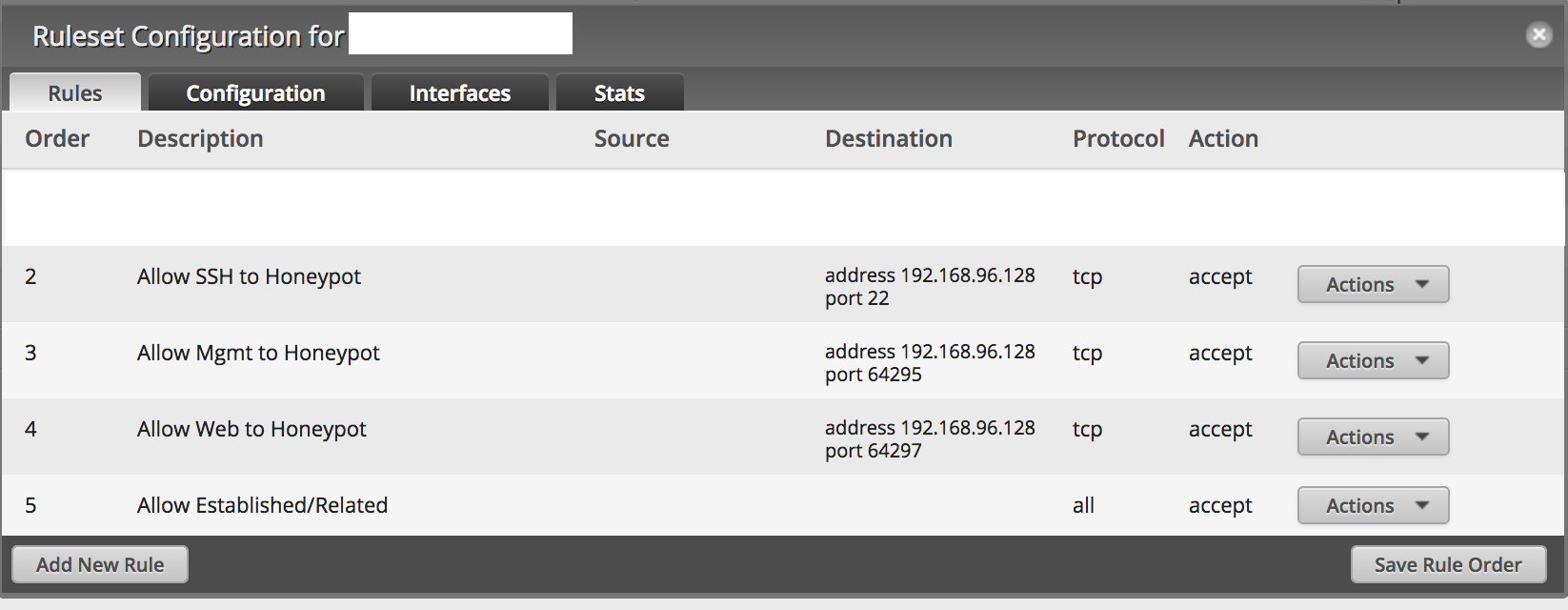
A website was then hosted on the VM, which was used to attract attackers to this Honeypot.

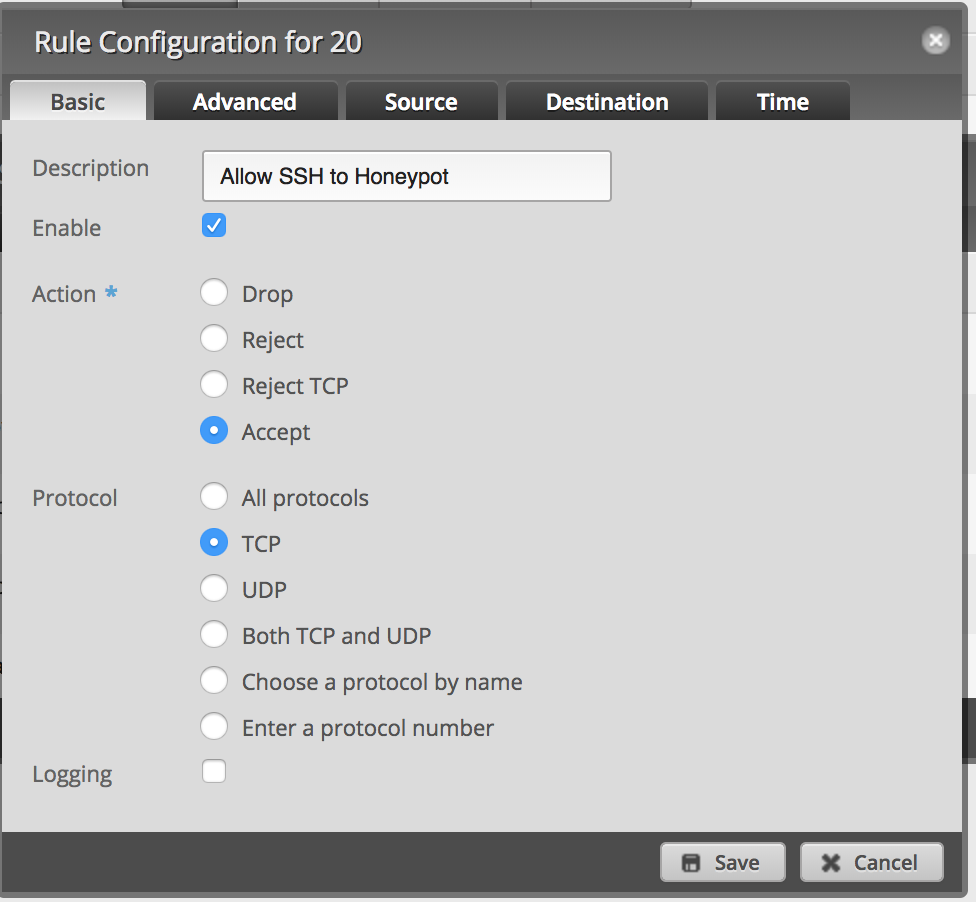
**17. Deploying Cowrie using T-Pot**

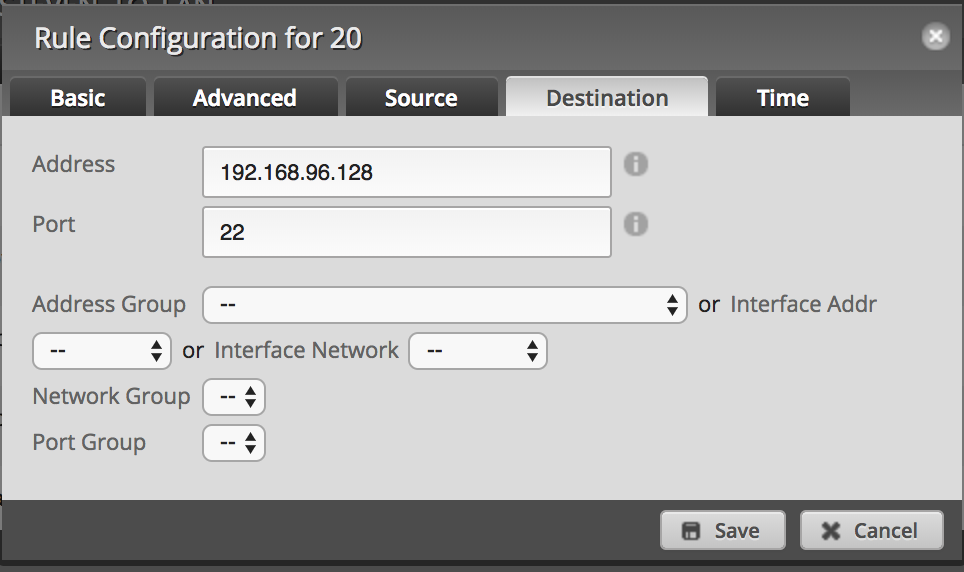
I decided to deploy Cowrie using T-pot, a python library and download the iso image for it from <https://github.com/dtag-dev-sec/tpotce/releases>. I allocated 128GB HDD, 2 CPUs and 4 GB RAM, which made it run smoothly (The recommended is 8GB RAM, so do as you feel is appropriate for you). I encrypted the drive and the home directory, just in case. I then cloned the auto-install scripts and ran through the process. As with all scripts that you download, please go through it before you run it to make sure nothing terrible is happening. The script does require you to run it as the root user, so assume this machine is hostile from the start and segment appropriately. The installer itself is straightforward, the biggest complexity is the choice of installation. I went with Standard which contains Cowrie, Suricata (the threat detector), and ELK (data organizer).

**18. Setting up the VLAN, Firewall, and NAT Destination Rules**

I wanted to make sure that the honeypot was in a secured VLAN so it could not access any internal resources. Running an Edgerouter Lite, I created the VLAN on the router dashboard (Add Interface -> Add VLAN). I trunked that VLAN to my ESXi host, made a new port group and placed the honeypot in that segment. Next, I needed to set up the firewall rules for that VLAN. In the Edgerouter's Firewall Policies, I created a new Ruleset "LAN\_TO\_HONEYPOT". It needs a few rules setup - allow me to access the management and web ports from my internal VLANs (so I can still manage the system and view the data) and also allow port 22 to that VLAN. I do not allow any incoming rules from the honeypot VLAN. Port 22 was already added to my "WAN\_IN" ruleset, but you will need to add that rule as well to allow SSH access from the internet.



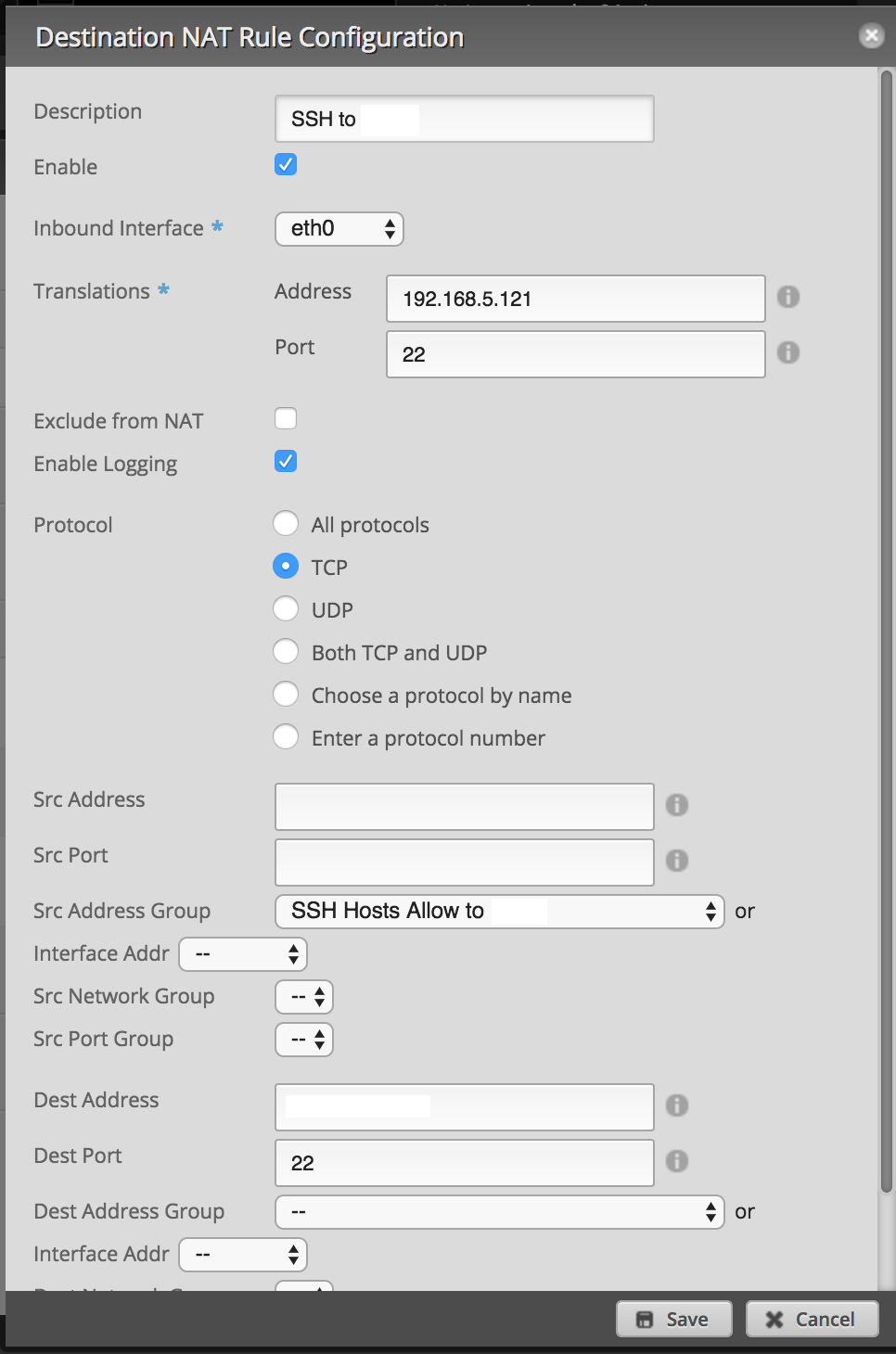


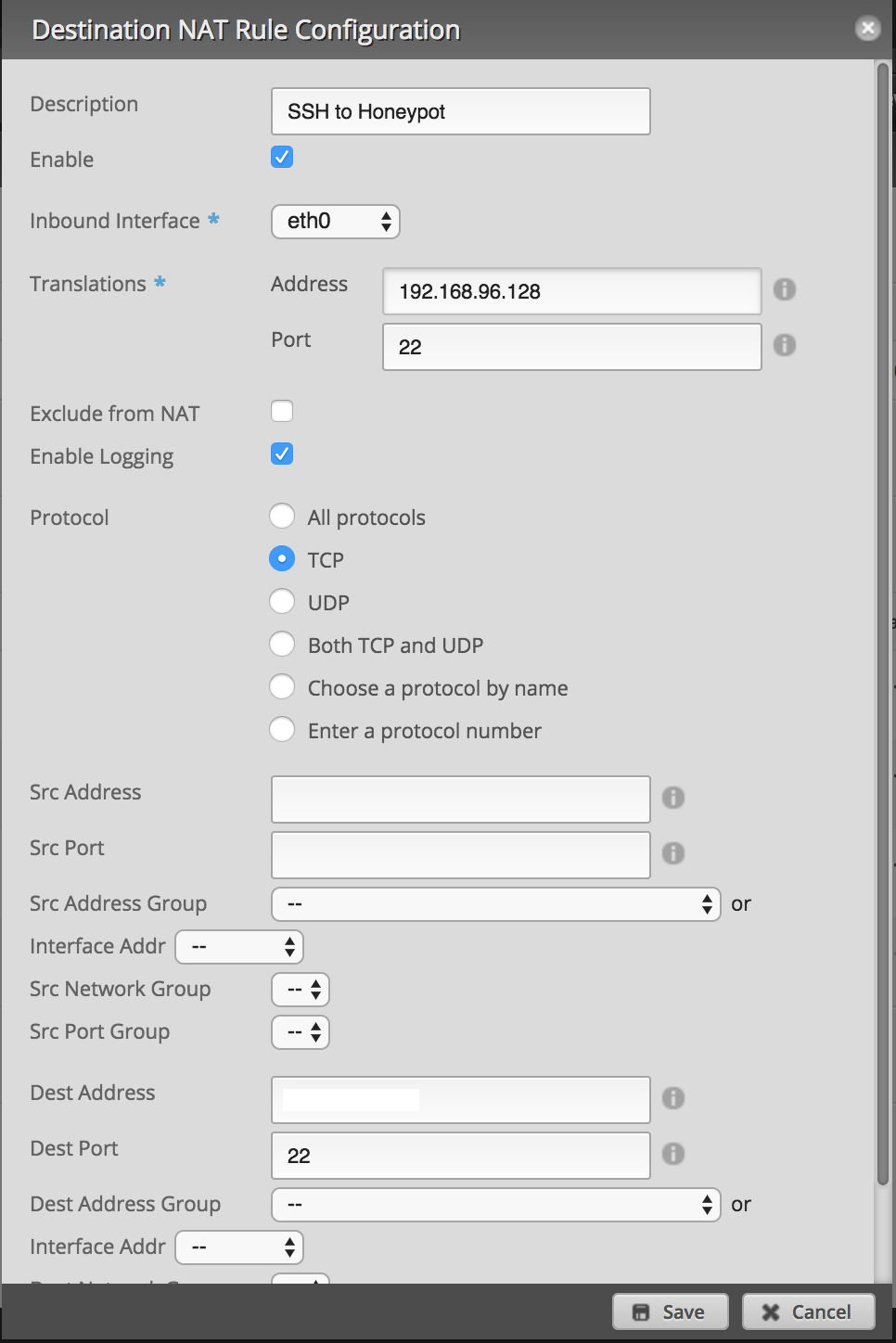


**19. SSH to Jumphost**

Since I wanted to still have my jump host running port 22, we cannot use traditional port forwarding to solve this. I wanted to set things up in such a way that if I came from certain addresses, I would be sent to the jump host and everything outside of that address set would be forwarded to the honeypot. This is done pretty simply by using Destination NAT rules. Our first step is to set up the address-group. In the Edgerouter, under Firewall/NAT is the Firewall/NAT Groups tab. I made a new group, "SSH\_Allowed" and added in the ranges I desired (my work address range, Comcast, a few others). Using this address group makes it easier to add/remove addresses versus trying to track down all the firewall/NAT rules that I added specific addresses to.

Once the group was created, I then went to the NAT tab and clicked "Add Destination NAT Rule." This can seem complex at first, but there is an idea of what goes where, it makes more sense. I made two rules, one for SSH to my jump host and a second (order matters with these rules) to catch everything else. Here are the two rules I setup:





**20. Running an nmap scan of the Honeypot:**

I right clicked on the desktop in the CentOS VM. I clicked open terminal from the list of options given. When the terminal appeared, "ifcong" gave the IP address of the CentOS Machine. I switched over to the Kali Machine and right clicked on the desktop, then clicked open terminal. Once the terminal is open, I typed "nmap -sS IPADDRESSHERE" into the terminal, replacing IPADDRESSHERE with the CentOS VM's IP. The nmap scan then ran.

**Benchmark Specifications:**

The following is a list of benchmark specifications set for this Honeypot and how they were

validated:

**1. Ability to adapt to different ports as they are being scanned:**

This has been validated by running NMAP scans from the Kali Machine, onto the Honeypot.

If the open ports appeared different in each scan it worked.

**2. Ability to spoof packets and send them to different ports:**

This has been validated by running a NMAP scan, first when the Honeypot script is not

running, this will show there are no services on the machine. Then the NMAP scan will be ran

again at the same time that the Honeypot script is, this time it will show that port 22

or port 445 are open, showing that SSH and SMB are being spoofed on the script.

**3. Log activity on these specific ports.**

With the Honeypot script running, I again ran an nmap scan of the CentOS machine using

Kali, then opened the honey.pcap file and it was populated with packets captured

from the Kali machine.

**4. Use Linux on a virtual machine to host this system.**

I ran a NMAP -O on the CentOS Machine without the script running to verify that it is

CentOS Linux.

**5. A script which can interact with the Honeypot.**

I ran both the capture.py script and notifyslack.py scripts to validate that they both

work.

**6. The script on this system would have to be highly available with no**

**downtime.**

I tested the cron job by rebooting the CentOS VM, and the script ran on startup again.

**7. The system will also have to be highly reliable.**

Research was done before choosing CentOS as the system to host the virtual machine

and it was deemed highly reliable with no internal systems so the network would not be compromised.

**8. The Honeypot will have to have minimum capacity.**

The honeypot only records activity on ports 22 and 445 instead of recording a lot of

wasted data from all of the ports, and this can be validated by opening up the capture.py

script and looking at the line which begins the packet capture.

**9. The response time of this system to an attacker would have to be near**

**Instant.**

This can be validated with yet another nmap scan on the CentOS machine, while the

capture.py script is running. As the nmap scan is running, one can see the script printing

out messages reacting to the scan

**Tools List:**

Git (Using GitHub repositories): Version Control for software.

VMware Workstation: Used in order to host the CentOS and Kali virtual machines.

CentOS Virtual Machine: Was chosen to act as the OS to host the Honeypot on.

Kali Linux VM: Used for testing the “ new and improved” honeypot

Cowrie: a well known SSH honeypot based on the older Kippo. It records username/password attempts.

T-shark: a packet capture tool which runs on the command line, chosen to record activity on ports

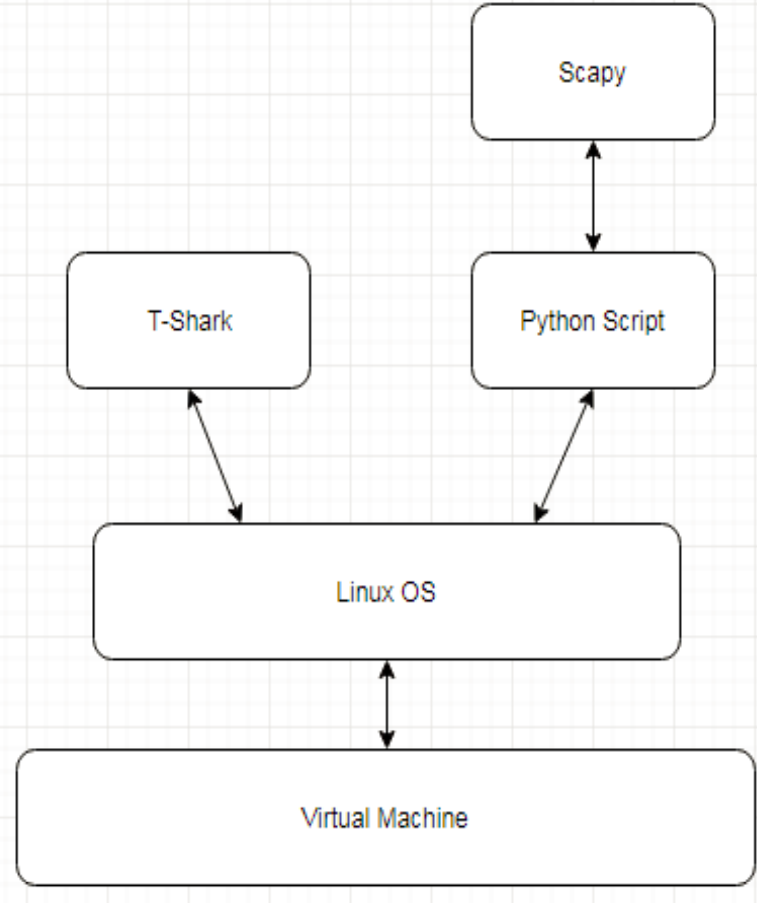
Slack: A messaging app used for creating a chatbot to alert of honeypots attackers.

Edgerouter Lite: to make sure that the cowrie honeypot was in a secured VLAN so it couldn't access any internal resources

Apache: Used to host a website on the Honeypot to lure in attackers

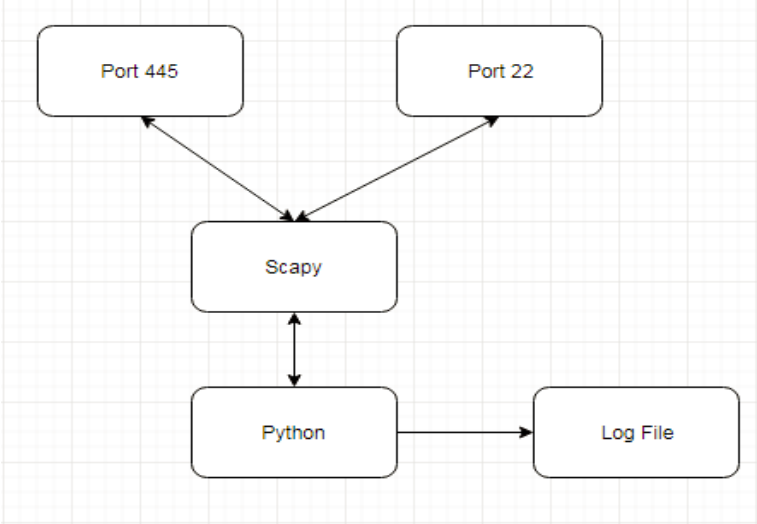
T-Pot: A unique python library used to deploy the Cowrie Honeypot with other useful applications

**Architecture of “new and improved” Honeypot**

****

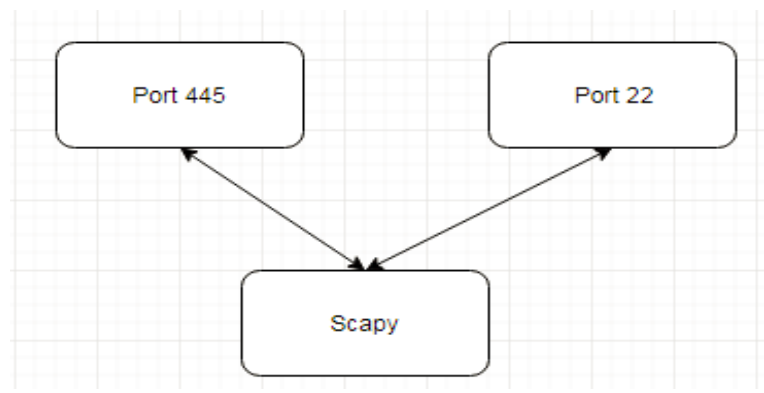
The python ﬁle itself was monitoring individual ports for malicious attacks. When these ports were probed, it used the Scapy package to craft packets to take on proﬁles while at the same time preventing the attacker from accessing the system while Python and Scapy monitored speciﬁc ports and adapted to different attacks, T-shark monitored every port in one go. This would help organizations just in case a threat tried to attack a port which the python script is unaware off, and in doing so made the Honeypot a more secure tool.

**Logging activity with Scapy:**

****

Python used Scapy to scan ports for any activity, and recorded any suspicions in a LogFile. The user of this Honeypot could then enter the system and check the log ﬁle to discover who has been trying to access this system. A python script will use Scapy to interact with the Ports. A user would be able to start or stop this Python Script.

**Scapy Packet Send**

****

(The ability to spoof packets and send them to different ports/ Ability to adapt to different ports as they are being scanned)

Scapy was used to scan each port, and adapt which packets were sent to them as it

picked up on traffic coming in. Again using Scapy, packets were crafted in Python and sent to specific ports. This would confuse the attacker into thinking that SSH or SMB were active so they could begin their attack.

**Time Schedule:**

|  |  |
| --- | --- |
| **Time Frame** | **Objective** |
| January 12 - 18 | -Download and install VMware Workstation  -Download and install CentOS to VMwareWorkstation |
| January 19 - 25 | -Ensure Python is installed on CentOS  -Install T-Shark for Linux on CentOS.  -Install Scapy for Python. |
| January 26-29 | -Download Cowrie and Setup the VLAN, Firewall, and NAT Destination Rules  -Disable Utilities running on open ports |
| January 30 – February 9 | -Begin Python script development |
| February 10 - 16 | -Manipulate the script to only show every port as open |
| February 17- 29 | -Test T-Shark capabilities  -Run T-Shark |
| March 1 - 9 | -Setting up a Slack Channel for message alerting |
| March 10 - 16 | -Host a website on the Honeypot |
| March 17 - 22 | -Test scripts |
| March 23 - 29 | -Finalize scripts |
| April 1 - 5 | -Review T-Shark logs/ELK Stack |
| April 6 - 12 | -Test all features |
| April 13 - 17 | -Fix any errors. |
| April 18 - 28 | -Compare Honeypot Datasets |
| April 29 – Capstone Presentation | -Review Project |

**Grading Scheme:**

20% Functionality of python scripts/tools

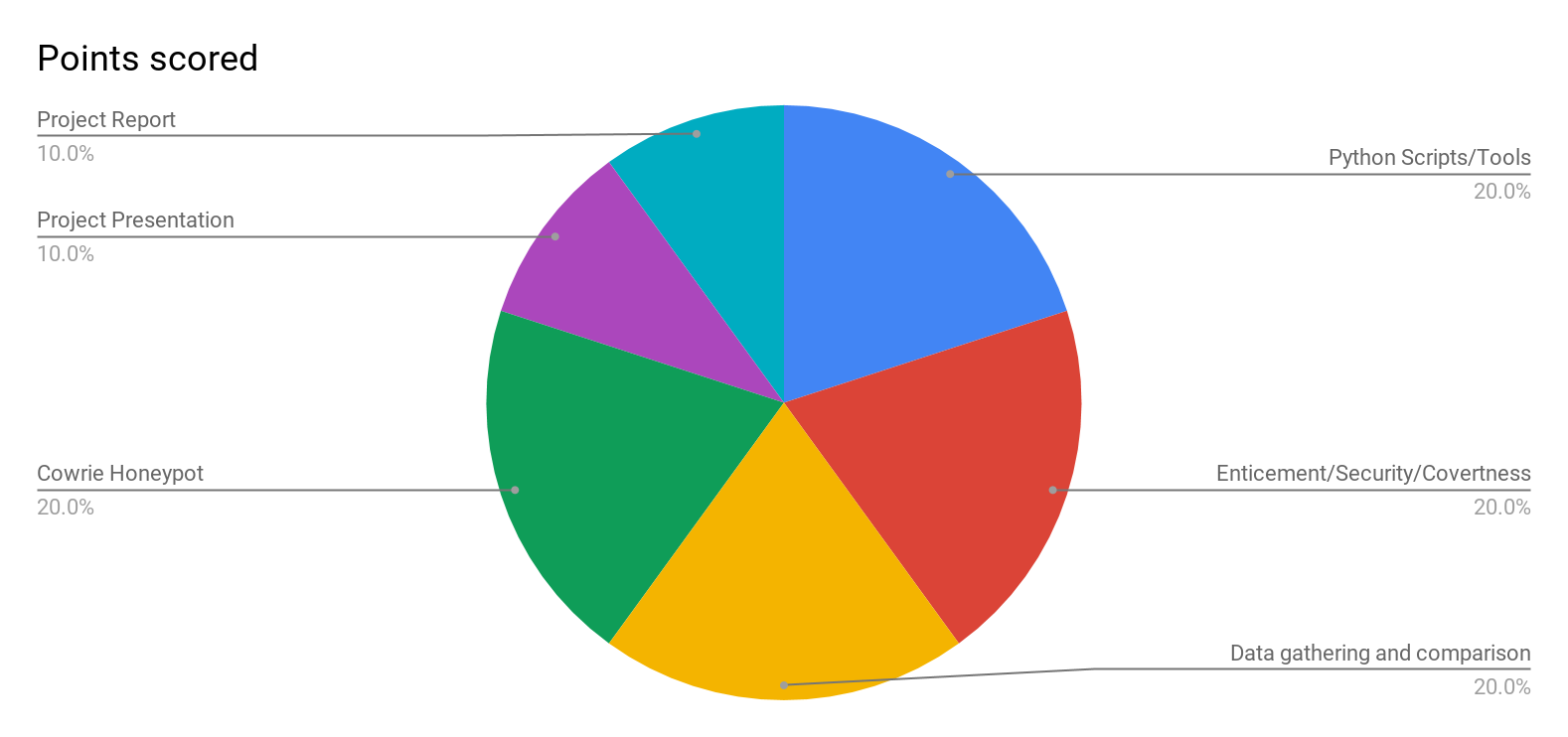
20% Enticement/Security/Covertness features of Honeypot

20% Data gathering and comparison

20% Deploying Cowrie Honeypot framework

10% Project Presentation

10% Project report



**List of Deliverables:**

* original proposal and presentation file(s) (from CSC 520)
* amendments to the proposal (approved by the project supervisor)
* system architecture diagram(s), enhanced with details determined during implementation
* appropriately commented source code
* documentation of project functionality (test results, screenshots, video capture of project execution, etc.)
* sample output (screenshots and/or reports)
* executables and/or projects
* presentation documents (used to support the presentation of the completed CSC 521 project), including any presentation file(s)
* project journal: a narrative of the progress of the project, in clear, concise English, including any problems encountered and how said problems were addressed
* project post mortem: a summary of what was learned from the project and (based on that experience) discussion of how various aspects of the project might have been approached differently
* a list of what areas of the proposal (if any) were not completed, including reasons why
* presentation of the completed project (PowerPoint format), including screenshots of the functioning
* **Images of the honeypot system**
* **Python scripts**
* **Data sets gathered (honey.pcap and elk stack screenshots)**

**Project Post-Mortem**

A major takeaway from this project is that most of the login attempts on my honeypots appeared to be automated through the use of some tools. Attackers scanned the internet for vulnerable devices and attempted to log in using a table of commonly used or factory default usernames and passwords. In this way, they would be able to assemble a botnet army and launch a massive distributed denial of service attack, leaving the targets inaccessible.

In this project I gained a deeper understanding of python, writing all my code while using 'vim' in the commandline to create and edit python files. I struggled with python syntax nearly every step of the way but it was a worthwhile learning experience. I used and gained a familiarity with many yum commands on CentOS. I learned how to bypass privilege and permission issues as tshark kept giving me "permission denied" prompts writing to (any) file in the home directory in the beginning. I learned how to create a fake bitcoin electrum .wallet file and simulate a real linux system. I learned how to deploy, use, defend, and secure a linux environment for multiple systems against different types of attacks and threats my system might face. I gained a familiarity editing router settings, setting up the VLAN, Firewall, and NAT Destination Rules for my project. I gained experience with the deployments of VM’s and a knowledge of the strengths and weaknesses of certain Linux operating systems. I learned how to use wireshark at the linux command line with T-shark, using it to monitor and analyze a network. I gained knowledge and understanding on how to capture and record specific protocols and ports, opening and closing specific ports, and spoofing ports make them appear different when scanned. I learned all about tshark capture filters and their syntax. Overall I believe my project to be an incredible learning experience that I genuinely enjoyed doing and overall a success. I did not, however, really address the differences between either of the honeypots as there was no really easy way to determine which was “more secure” or efficient at data collection (although Cowrie with elk had a much easier and aesthetic way to display data other than a .pcap file) but my honeypot exceeded all of my expectations nevertheless.