National College of Ireland

An approach to enhance low -interaction honeypots by enabling them to detect spoofing attacks via network analysis

MSc Internship
Cyber Security

Rhea Bonnerji
Student ID: X18176887

School of Computing
National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Rhea Bonnerji……

Student ID: ……x18176887……..……

Programme: …MSc. Cybersecurity…… **Year:** …2019-2020.

Module: ……MSc Internship……..……

Supervisor: ……Niall Heffernan……..……

Submission Due Date: ……17/08/2020……..……

Project Title: An approach to enhance low-interaction honeypots by enabling them to detect spoofing attacks via network analysis

Word Count: ……7301…… **Page Count**……17……..……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.
I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature: ……Rhea Bonnerji……

Date: ……17/08/2020……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

An approach to enhance low -interaction honeypots by enabling them to detect spoofing attacks via network analysis

Rhea Bonnerji
Student ID: 18176887

Abstract

A spoofing attack is when a malicious party imitates another system on a network as being from a known, trusted source and initiates attacks on network administrators to steal data, bypass network access or perform DDoS attacks. Spoofing attacks are a rising problem for companies both big and small costing them billions of dollars for the same. Honeypots are often used in organisations to both bait the attackers and to detect underway attacks. Low-interaction honeypots that imitate basic network services, internet protocols and operating systems are more likely to be used because of their cost-effectiveness but since they are limited in their abilities, organizations prefer high-interaction honeypots which are both expensive and resource heavy. Therefore, it was necessary to propose a method of detection of spoofing attacks with minimal resources to meet the needs of all. This paper proposes a spoofing attack detection mechanism for IP, ARP and DNS spoofing by enhancing the functionalities of a low interaction honeypot by incorporating some additional intelligence to make them detect basic spoofing attacks and capture all the network traffic. Tshark is used to capture the network traffic and then different scripts are implemented on them to categorize the network packets as spoofed or genuine. Next, these results are compared to the logs from the honeypots to see how our enhancement scripts have worked compared to them.

1 Introduction

Spoofing is the act of pretending to be someone from a trusted source in order to gain unauthorized access to the systems. A spoofing attack is when a device or user is impersonated on a network by a malicious party to launch attacks against network hosts. Spoofing can be used to retrieve personal information, spread malware through infected links and attachments, steal money, perform DDoS attacks and bypass network access controls [1]. The most common type of spoofing methods are IP Spoofing, ARP Spoofing and DNS Spoofing. IP spoofing is the falsifying of Internet Protocol (IP) packets using a modified source address in order to either hide the identity of the sender or to impersonate another system. It is mainly done to conduct DDoS attacks, to mask botnet device locations and to stage a reflected assault. It exploits a shortcoming that only a few networks have the motivation to fix: ensuring that the outbound packets have a valid IP address [2]. ARP spoofing is a type of attack in which an attacker sends false ARP (Address Resolution Protocol) messages over a local area network. This links the attacker's MAC address with the IP address of a legitimate computer or server on the network [3]. Once the attacker's MAC address is connected to an authentic IP address, the attacker

receives any data that is intended for that IP address. DNS (Domain Name Server) spoofing is an attack in which manipulated DNS records are used to divert online traffic to a fake website that comes off as its intended destination [4]

Spoofing attacks can be detected using honeypots. A honeypot is a sacrificial computer system designed to draw cyber-attacks. It is like luring attackers to a trap. It imitates a hacker target, and uses its intrusion efforts to obtain details about malicious actors and how they function or distract them from other targets. They are used to redirect a hacker's malicious intentions to a contained and monitored environment instead of a network where they can cause serious harm. They are also used to defend against threats that firewalls don't block. Honeypots are segregated according to their interaction level - low, medium and high. Usually a low interaction honeypot imitates a limited number of internet protocols and network services, just enough to fool the attacker [5] and can be manipulated by using the trusted source IP address. Even a simple idle scan on Nmap can sometimes deceive the low interaction honeypot if the IP is spoofed to a trusted IP. Low interactions are more likely to be used at organisations because it is easy to deploy and maintain and they do not require extensive resources. However, since it fails to capture all the network traffic and captures only limited amounts of information, most organisations spend a huge amount of money and resources and resort to high interaction honeypots. This study is important so that the correct utilisation of low interaction honeypots can be made without making it an expensive resource while enabling it to detect spoofing attacks by implementing certain scripts that could be incorporated in low interaction honeypots to enhance their traffic capturing and detection abilities.

In this proposal, we will send spoofed IP, ARP and DNS packets to two low interaction honeypots and see what they log. Then we will run our scripts and send the same spoofed packets to see what they log. Since we know that low interaction honeypots don't capture all network traffic, we will be using tshark in our scripts to ensure that all of the network traffic is captured. Then, this generated pcap file will be fed into our detection scripts to categorize the traffic as genuine or spoofed. This will be discussed in detail in the Research Methodology section of the paper.

Before reviewing the proposal submitted by this paper, the Literature Review portion of the document is presented. The next section covers several approaches done on spoofing attack detection by various researchers. It is extremely important to study the current methods of detection and prevention of spoofing attacks so that models can be developed to overcome their limitations. Multiple models have been built to detect spoofing attacks, but they still do not hold up against the current sophisticated spoofing attacks performed by the attackers. Hence, there was a need to come up with a novel approach that not only detects spoofing attacks efficiently but while maintaining a balance between its cost and efficiency. This is the paper's intended flow to ensure that the reader is in a better position to evaluate and understand the methods needed to achieve a comprehensive understanding of the subject. Following the Literature Review section of the paper is the Research Methodology section followed by the Design Specification of our setup. This paper will also contain the detailed implementation procedure of the project in the Implementation section along with an Evaluation and Results section discussing the competence of our proposed model and the results it has produced. This model is only practical if doesn't come with its share of problems. Therefore, we will also be proposing the possible solutions to its shortcomings in the Future Work section of the paper.

2 Related Work

This section includes a list of academic sources related to our study of detecting spoofing attacks to contextualize our research with respect to existing information. There are a few broad categories of performing this task, namely, by using network traffic analysis, machine learning algorithms and a fuzzy approach. We will be taking a brief look at all of them.

Detection of spoofing attacks using the different properties of network traffic:

A research proposal called 'A Novel Method for Detecting and Preventing IP Spoofing Attack in Data Network' by Dr. N. Arumugam proposes a novel detection method where spoofed packets will not even reach the network server and will get discarded at an Authentication Server (AS). The configuration consists of a Subnet Host Identification Database that consists of IP addresses with their corresponding MAC addresses. The AS is in-charge of setting up the TCP connection. It computes the value of the hop count to the source of the received packet. Based on the difference it can identify spoofing of packets. If the hop count is 0, the data stored in the database will be checked as an added check. It sends a request to the IP source to obtain the MAC address. Then it examines whether the details on the database match. If it does, then it forwards the packet through. Only if the hop count isn't 0 (meaning it could be from an unknown external source) then another validation step will be conducted. The AS now sends the IPSO a traceroute request so it can obtain the t-hop value to calculate the difference. When the difference is 0 then a genuine source can be confirmed and the request is sent across. For this, a threshold is set, crossing which indicates that the packet should not go beyond the AS [6].

This paper is relevant for our study because it gives us an idea of a novel way of detecting IP spoofing attacks. However, the disadvantage of this approach is that it is a costly process because of the Authentication Server acting as the connection service provider between the sender and the receiver. Also, the time required to establish a TCP connection is higher because there is a response only after verification which when combined with the database search time makes it infeasible to use.

Yu et al. suggested a spoofing detection strategy based on physical network features in their paper, 'A system for detecting MAC and IP spoofing attacks with network characteristics. It employs techniques which artificial intelligence cannot imitate. It can monitor users' physical network characteristics in real time and check whether there have been any significant changes in the calculations to accurately detect spoofing attacks. For example, the fingerprint of a system can be observed if we take the signal strength. Unless it isn't an attempt at a spoofing attack, two packets should have identical characteristics in two consecutive windows. This can be easily identified as a spoofing attack if two packets with the same MAC and IP address have somewhat different properties in two consecutive windows. The measurement parameters selected include RSSI, RTT, and LQI. These represent the transmission power of the radio signal received, the delay time and the performance of the connection, respectively. For each

parameter, a threshold has been determined, exceeding which means that the packet is possibly a spoofed packet [7].

This is an interesting approach for the detection of spoofing attacks using physical network characteristics which gives us an idea of using network analysis to come up with our own approach of detecting spoofing attacks. However, attempting to use this strategy could pose a few challenges. Firstly, if there are multiple devices in a small range then we need multiple features that are appropriately influenced by range and robust enough as well. Next, if the parameter 's physical characteristics change then the model won't yield the correct result. However, these physical features can be quickly manipulated because of the complexity of attacks nowadays. The model's intent is to monitor the network continuously which costs a lot of extra power making it less viable. Also, it is not a simple task to produce the fingerprint of each device, since every new device has to be configured manually. It is also easily exploitable if someone has the set-up information.

Sadasivam et al. proposed a distributed honeynet architecture consisting of low-interaction honeypots for identifying malicious network activities. Each honeypot is installed in this architecture to detect and catch the attack packets unique to its port of installation. This assists in improving the surface of the attack. The paper discusses how effective this approach is due to the honeypots' low-interaction capability, which does not allow complete access to the operating system and thus prevents the honeypot from targeting other devices. The data from the honeypot revealed that there were multiple attack attempts made but very few malwares were caught using the distributed architecture [8]. This was particularly interesting for our research because it inspired our proposal to have a distributed low-interaction honeypot architecture running on different ports to capture traffic related to the specific port only.

Another innovative method to detect IP spoofing was proposed by Ayman et al. using a modified hop count algorithm inspired by the Hop Count Filtering algorithm. But the proposed algorithm modifies the HCF's learning process to incorporate all possible hop count values by making a profile for every IP with which it interacts so the destination can know all the correct hop counts viewed from the source in case there are many routes present with different hop counts. This strengthens the HCF results by appropriately classifying the valid traffic, and therefore the false alarm rate is decreased. This increases the overall efficiency of the detection system by 9% [9].

The upside to approaches like this one is that one can learn the multiple factors that can determine changes in network traffic and then come up with their own logic to detect spoofing attacks. In this case, when the algorithm is implemented between the router and the firewall, if there is an attack, the model can filter the traffic protecting the end servers because everything will pass through it. However, it is not feasible as a defence for DoS attacks that target the bandwidth of the Internet links.

Detection of spoofing attacks using the Machine Learning algorithms:

This subsection deals with network traffic analysis using machine learning algorithms. There is a whole bunch of papers where machine learning is used to predict different attacks and network traffics. We will be looking at some of them.

One such paper by Banerjee et al. deals with the detection of botnets using honeynet data. By analysing network traffic and defining features that have a major effect on botnet traffic filtering, it uses classification techniques to detect network behaviour. Three honeypots serving three different purposes are deployed on the network to capture malicious network activities. These packets are then converted to network flows and then python is used to assign integer values to them so that machine learning classifiers can be applied on them to distinguish normal traffic from botnet traffic [10]. Another such work by Chinta et al. to detect botnets in a host using network behaviour analysis takes a similar approach. They are making use of a dataset that contains details of different botnet attacks. They use Wireshark to capture the data unlike the previously discussed paper and then implement machine learning classifiers on them. They used parameters that are significant in detecting network traffic changes which is an important takeaway for anyone attempting to analyse network traffic. Then they made a comparison between all the classifiers to come up with the best classifier for detecting bots [11].

These kinds of approaches come in handy for matching patterns. Like, in this case, the algorithm focuses on identifying patterns and behaviours that do not conform to regular network traffic, rendering it to be botnet traffic. Results of machine learning algorithms can be called accurate to a certain point but never can be a cent percent relied on as it depends hugely on the training dataset selected, data pre-processing, parameter selection and the choice of the algorithm (i.e., if the dataset is appropriate for that particular algorithm).

Fuzzy approach for detecting spoofing attacks:

In the research work by Nitin et al., they propose a fuzzy approach for detecting IP and ARP spoofing attacks using data from low interaction honeypots. They consider the parameters that are most significant in detecting the spoofing attack like Unusual ARP packets, Rate of ARP packets and Rate of Trusted IP Address and set up a threshold for each of the parameters. If the thresholds are exceeded, it means that it is a possibly spoofed packet. Next, these parameters are fed into MATLAB's Fuzzy Logic Designer to predict the probability of the packet being a spoofed one or not. This system is customised to not only predict but also alert the user about the potential spoofing attack. [12]

The method proposed in this paper is based on exactly this concept where additional features can be added to low interaction honeypots to make them be able to detect spoofing attacks. Here, they used the fuzzy technique to enhance the abilities of the honeypot but a lot of other techniques can also be implemented to improve the capabilities of the low interaction honeypot. The disadvantage here is that, to predict a spoofing attack, a lot of captured traffic is needed and often it generates false positives because they have patterns that are similar to those provided in the detection mechanism. Hence, in our proposal we have been able to successfully negate the generation of false positives.

This work was further modified by Shang et al. by adding a feature called a Dynamic Fuzzy Rule Interpolation (D-FRI) system to the pre-existing fuzzy system. D-FRI was commonly used for network security applications but was never tried on a honeypot before this. The study also establishes a methodology for learning and maintaining a complex base of

rules to help identify potential spoofing attacks with respect to the evolving traffic conditions of a computer network. As an addition to the previous work, this research has a few more parameters to detect spoofing attacks. Because honeypots cannot collect all the traffic, the D-FRI method performs rule interpolation to have an approximate result if any of the pre-existing rules are not met by a specific observation. Such results are processed for dynamic learning that supports the simultaneous rules generalised from the interpolated results stored up to the current sparse rule base. This way, D-FRI constantly updates the rule base to actual traffic flow and decreases false positive and false negative forecasts to improve the efficiency of the method of detection of spoofing attacks [13].

The D-FRI system led us to come up with a less expensive resource to ensure all the network traffic was captured which honeypots fail to do. The addition of new parameters in the detection mechanism inspired this paper's chosen parameters to detect spoofing attacks. However, a major drawback of this work is that if there is an issue with the traffic capture, it would be difficult to apply this model, as it relies heavily on the traffic captured which our proposed framework has been able to overcome.

Another interesting paper by Jenkins et al. uses a fuzzy system for predict a fingerprinting attack on a honeypot. Fuzzy logic has the ability to link all attack behaviour within a given range and label a fingerprinting attack as a result. Low interaction honeypots are not capable of logging sophisticated fingerprinting attacks hence making the correlation impossible. This paper uses the same concept as [14] to predict OS fingerprinting attack probability using TCP, UDP and ICMP packets.
This paper comes in handy to protect the low interaction honeypots from fingerprinting attacks because most tools available today to prevent such attacks are resource extensive and expensive to implement unlike this one [15].

Another work by Maria et al. focuses on a hybrid fuzzy system that is constructed on the source's MAC address, passive fingerprinting, GeoIP, hop count, and browser user agent. Here, unlike some papers discussed previously do not require continuous hops but instead makes use of the GeoIP information and the subnet IP address. The fuzzy system can enhance the detection procedure by cross-checking the number of hops and reducing network and resource requirements for multiple traceroute queries. This system works in various phases where first the number of compromised hosts are checked. Next, the OS information from the user agent is checked using OS fingerprinting which continues through the TTL. Then, it searches for IP hops until the TTL is specified, and it calculates the difference between the set TTL and the hop count when it detects hops for an IP. This TTL is compared with the TTL value reported in the network data to detect irregularities and count the number of times that they change. These results are gathered from the fuzzy system.

This work inspired in formulating our proposal's main logic. Their approach works perfectly in local networks due to its MAC address and its corresponding IP address pairing and for other networks it uses other factors like hop count, OS fingerprinting and TTL values which do a pretty good job in determining spoofing attacks. The only drawback to this approach is the fact that it needs a database with constantly updated TTL values of as many Ips as possible along with its IP hops, OS signatures, etc. [16].

3 Research Methodology

This portion of the paper is intended for explaining the details of the proposed research technique to enhance the functionalities of a low interaction honeypot to enable them to detect spoofing attacks like IP, ARP and DNS.

Low-interaction honeypots come in use in organisations because they are easy to deploy and maintain and they do enough to distract the hackers from the main target systems. They, however, fail to capture all the network traffic which already makes them less efficient. Also, they do not detect basic spoofing attacks making them fall prey to the same.

Low-interaction honeypots are not developed for substantial load and functionality. However, they could potentially capture almost every internal security incident, but it will have a huge effect on the efficiency of the honeypot communicating with external attackers. This could be accomplished by using more advanced tools, levying additional costs and computational resources which is already a problem in the case of low-interaction honeypots as they work on low powered devices. So, we decided to write individual scripts to enhance their functionality by enabling them to detect IP, ARP and DNS spoofing attacks.

From the previous Related Work section of the paper, we saw multiple broad categories of detecting spoofing attacks like using different properties of network analysis, fuzzy approach, and the machine learning approach to detect spoofing attacks. Out of the three types, both fuzzy logic implementations and machine learning algorithm approaches make the detection procedure resource extensive and expensive. They are also not able to perform the detection mechanism in real time. Thus, making the two approaches infeasible. Also, these approaches do not guarantee hundred percent accurate results. Hence, we are left with the usage of different properties of network analysis to perform the detection which always guarantees accurate results. Although, some approaches in this subheading did make the approach, time consuming and/or expensive, the proposed methodology aims to solve just this. In our proposed methodology, we are writing five python scripts to enhance the abilities of the low-interaction honeypots that could be modified to run in real time while not being resource extensive or expensive. These scripts first capture all the network traffic using tshark which already is the first enhancement feature for the honeypot. Next, we use the basic concepts of network analysis to formulate logics for detection of spoofing attacks.

In our proposed approach, we tested the honeypots with spoofed packets and saw that they failed to even capture the attack attempts. Next, we tested our scripts against the same spoofed packets, and we got a cent percent accurate logging of the data. Not only was the attempts logged, but was also categorised as genuine or spoofed with the reason in case there was uncertainty as to why a packet was called a spoofed or genuine packet. The scripts were also tested with genuine network traffic and they were also logged with the reason for genuinity.

4 Design Specification

This section contains the architecture and design flow of this proposal. First let us understand the network configuration of the machine.

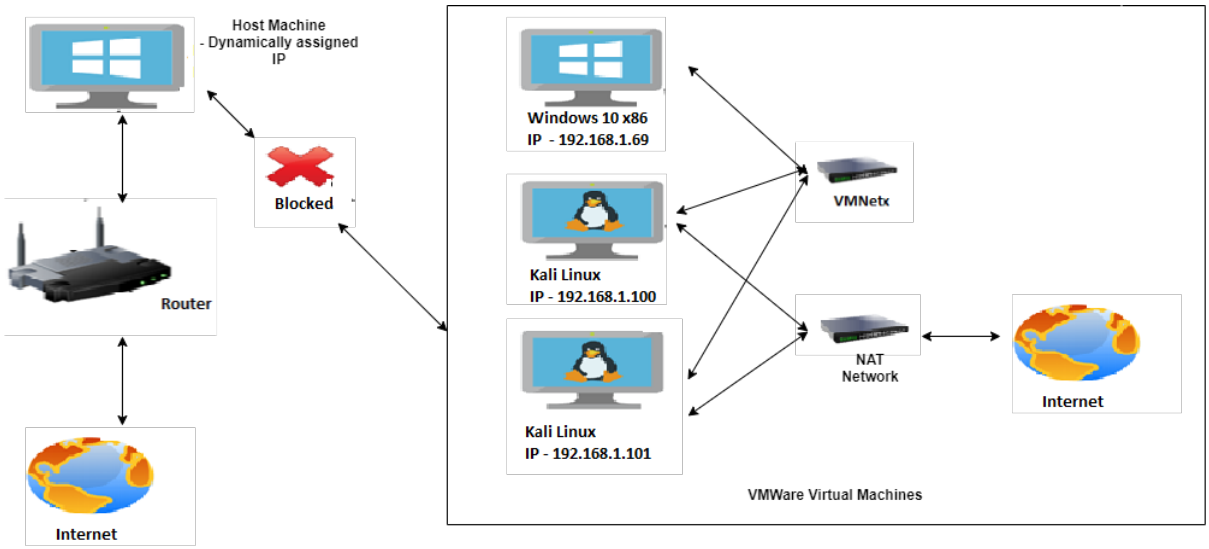

Figure 2: Network Diagram

This network consists of three virtual machines, namely, two Kali Linux machines – one is called Kali Linux x64 Snare and the other is called Kali Linux FakeDNS Server and one Windows 10x86 machine. The two Kali machines are additionally connected to a NAT network which is connected to the internet while being on the internal VMWare network. The Windows VM on the other hand is connected to the internal VMWare network only. Kali Linux x64 Snare contains our honeynet and the enhancement scripts for the detection of IP, ARP and DNS spoofing attacks. Kali Linux FakeDNS Server hosts the fake DNS server that we set up. Windows 10x86 machine is only responsible for sending the crafted spoofed packets on the local network.

4.1 Experimental Setup

This research comprises of three parts: a packet generator called HyenaeFE to craft and send packets, a honeynet consisting of two low interaction honeypots – HoneyPy and SNARE (a successor of Glastopf) to detects hackers, worms, malwares, viruses, DDos attacks, botnets, etc. and the enhancement scripts written in python.

The proposal is performed in three phases: first being sending of the spoofed packets to the network by the packet generator, then letting our honeynet log the attacks and finally in the third phase running our scripts to capture the traffic and detect the spoofed packets and comparing the logs of the two.

The packet generator:

HyenaeFE: It is an open source GUI based advanced network packet generator that is platform independent and is highly flexible. It enables you to launch low-level ethernet security attacks such as DoS, MITM and DDoS to expose potential network security vulnerabilities. Hyenae comes with a clusterable remote daemon to set up distributed attack networks, in addition to clever wildcard-based address randomization and a highly customizable packet generation control with the help of an interactive attack assistant [17].

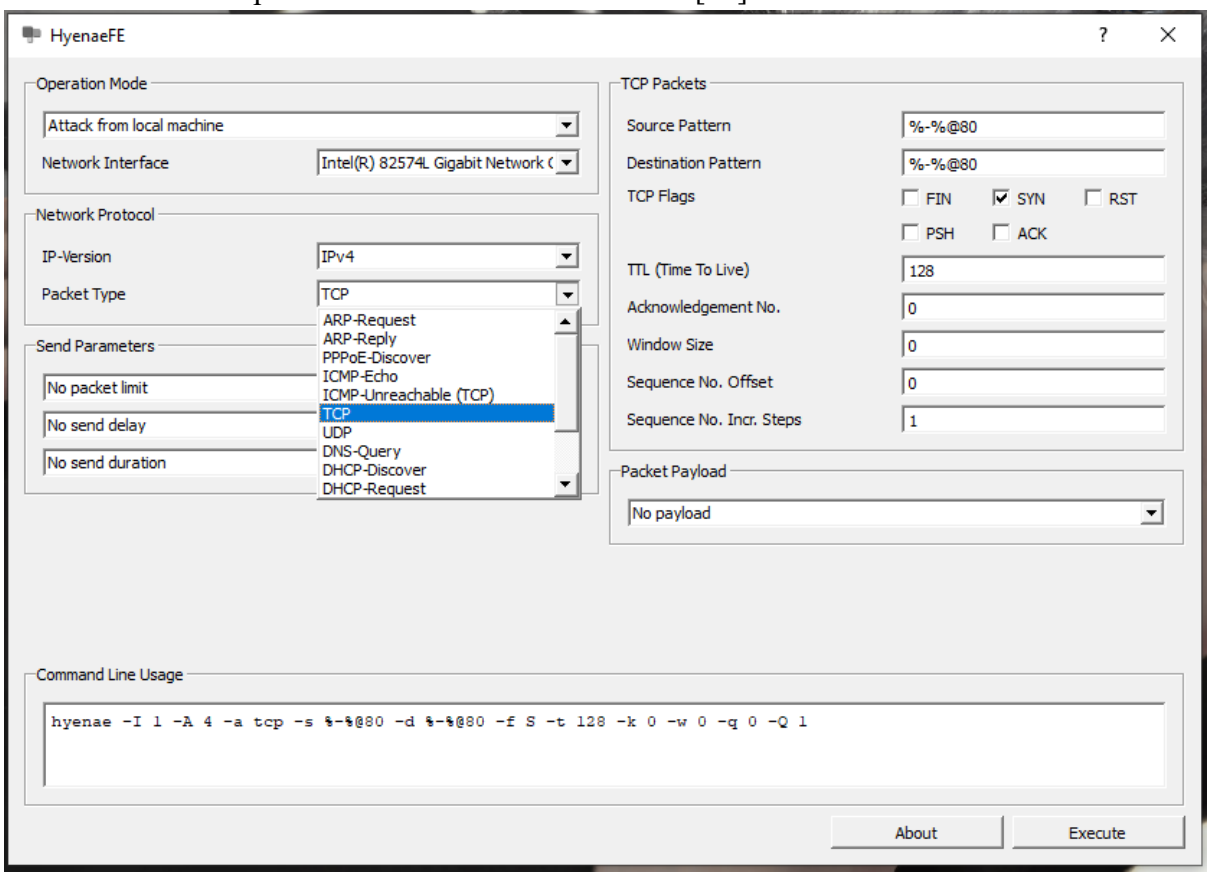

Figure 1: HyenaeFE packet generator

The honeynet:

HoneyPy: A low-interaction honeypot with the potential to have been more of a medium honeypot interaction written in Python. It is easy to configure, deploy and maintain. It was built to support plugins for TCP and UDP based protocols. Additionally, configuring which plugins we want to allow and use is as easy as modifying the services.cfg file. HoneyPy listens on ports below 1024 and since there are many system services that we would like to run, we enable port forwarding on lower ports to higher ports. Since this honeypot works on multiple ports, we have the freedom to configure it to enable whichever services we want it to [18].

SNARE: SNARE, which stands for Super Next Generation Advanced Reactive HonEypot, a successor of Glatopf, is a low interaction web application honeypot written in Python that is used to lure all sorts of malicious internet activities. It collects all the HTTP and HTTPS traffic by listening on ports 80 and 8080 [19].

5 Implementation

This section contains the implementation procedure of the proposed enhancement functionality. The following scripts are all lightweight python scripts that use the packages os, scapy and getmac to add additional intelligence to low interaction honeypots thereby enabling them to detect spoofing attacks.

startCapture.py:

This script uses tshark to capture network traffic on the first Ethernet interface eth0 and creates a pcap file called dump.pcap on the desktop with all the captured internet traffic for analysis at later stages. This can be modified to capture on any interface according to the need of the user.

whitelist.py:

This script creates a whitelist (yml file) based on the live hosts found during an automated nmap scan. It then records the IP addresses and their corresponding MAC addresses of the hosts. Along with these, we also add 0.0.0.0 and 00:00:00:00:00:00 for broadcast purposes. Since ARP protocol is used to find out the destination's MAC address when the IP address of the destination on an Ethernet network is known. So, the destination MAC address field of ARP that we would put when we don't know the destination MAC address is 00:00:00:00:00:00. The ARP packet is embodied in an Ethernet frame with destination MAC address set to ff: ff: ff: ff: ff (broadcast) to reach all network machines. The machine having IP address mentioned in the ARP request packet replies with the MAC address by sending an ARP response packet. Now the source machine gets back the ARP response with the target MAC address and places it in the memory using an ARP table so it doesn't have to use ARP every time before the table entry expires. The source machine can now create an Ethernet frame with the target MAC address as its destination, using the target MAC address, and send the frame over the network. 0.0.0.0 is a wildcard address. TCP / IP software applications use 0.0.0.0 as a programming technique to track network traffic across all the currently allocated IP addresses to the interfaces on that multi-homed system. Although connected computers do not use this address, when the source of the message is unknown, messages carried over IP often contain 0.0.0.0 within the protocol header.

ARP.py:

This script is the script responsible for detecting spoofed ARP packets. First, we start our startCapture.py script so that our spoofed packets are captured using tshark. Next, we run the whitelist.py file so that we can generate a whitelist based on the number of live hosts captured by the nmap scan. Then, our packet generator is used to send spoofed ARP Request and Reply packets to the Kali VM containing our scripts. These get captured to generate the dump.pcap file for our analysis using the ARP.py script. Now, we run the ARP.py code. ARP.py parses the dump.pcap file to check if both the source and destination IP address and its corresponding MAC address is present in the whitelist or not. If both match the whitelist, then the traffic is classified as genuine else it is marked spoofed traffic. If one or both (source and destination) don't match for either the IP or the MAC address, the respective reason along

with the IP/MAC address gets logged in the generated_logs.log file that ARP.py writes to/generates.

One thing that we have to keep in mind while doing the ARP spoofing detection using this script is that if the host is heavily firewalled and it doesn't respond to requests of various types for different spoofing detection, then the host is considered to be inactive and its IP and MAC address is not considered.

IP.py:

This script is the script responsible for detecting IP spoofing attacks. First, we start our startCapture.py script so that our spoofed packets are captured using tshark. Then, our packet generator is used to send spoofed TCP and UDP packets to the Kali VM containing our scripts. These get captured to generate the dump.pcap file for our analysis using the IP.py script. Now, we run the IP.py code. This script sends a TCP SYN request to all the source and destination addresses. Then it waits for a SYN-ACK reply. If it receives a SYN-ACK request from both the source and destination address of a packet, then it sends an ACK reply to that packet and this packet is categorised to be genuine. Anything other than this (either or both source and destination is not validated), the packet is categorised to be spoofed and the appropriate message saying one or both addresses are not validated is logged in the generated_logs.log file that IP.py generates.

One thing that we have to keep in mind while doing the IP spoofing detection using this script is that if the host is heavily firewalled and it doesn't respond to requests of various types for different spoofing detection, then the host is considered to be inactive and its IP and MAC address is not considered.

DNS.py:

This script is responsible for detecting DNS spoofing attacks. First, we need to set up a DNS spoofing server on the other Kali Linux VM that we have in our network. Next, we add the Windows10 x86 VM to the target list. Now if we run the startCapture.py file, and try opening various websites on a browser in the Windows VM, spoofed traffic will get captured in the dump.pcap and upon running the DNS.py file it will get logged into the generated_logs.log file.

Now, we go back to our other Kali VM which contained the honeypots and visit various websites, while capturing these packets with startCapture.py to generate the dump.pcap file. Upon running the DNS.py script on this pcap file, we will see that the traffic that it captured happens to be genuine traffic and that gets logged into the generated_logs.log file.

This is because this Kali has a second network adapter running on NAT network that works on the eth1 channel capturing internet traffic, while our Windows VM has just one network adapter, the internal VMWare network, which captures on eth0 channel i.e., in the local network only.

Now coming to the working logic of the DNS.py file – upon processing the DNS reply packets in the dump.pcap file, it checks for the number of answers to the initial query. Next, the function gethostbyname is used. This is a python function that corresponds to nslookup which finds the IP address related to the DNS query. This is then compared to the replies that

are given for that query. If the reply contains the IP from gethostbyname() function then categorize it will get categorized as genuine traffic else it is spoofed traffic.

One thing that we have to keep in mind while performing the DNS spoofing detection using this script is that the honeypot should be connected to a genuine DNS server and it has to sit in between a local machine sending the DNS queries and a malicious DNS server sending spoofed replies.

6 Evaluation

This section entails the assessment of the performance of our proposed methodology in different sections of the approach.

6.1 Capturing of network traffic

HoneyPy was tested regarding its ability to capture network traffic. Spoofed TCP and UDP packets were sent using HyenaeFE by setting the destination IP on the packet generator to be the Kali VM containing HoneyPy. It was seen that HoneyPy failed to log any of the data that was sent.

Next, we try to SSH and FTP into the machine and we also send an HTTP request to see if it logs any of this. Upon doing this, we find that log files have been generated logging the SSH, FTP and HTTP request attempts. Therefore, we can conclude that HoneyPy fails to capture any of the spoofing attack attempts but can capture service related attempts.

So, we ran startCapture.py which is our enhancement script to ensure capture of all traffic on a given interface and it logged every single packet in a pcap file called dump.pcap. Therefore, we see that our script worked better when it came to capturing traffic than the honeypot.

6.2 ARP spoofing detection

HoneyPy was tested regarding its ability to detect ARP packets. We tried to ping the Kali VM that contained HoneyPy but the honeypot failed to capture any of those genuine ARP packets.

So, we ran startCapture.py which is our script to capture of all traffic on a given interface and it logged every single packet in the pcap file called dump.pcap. Next, we ran our ARP.py to analyse the pcap file for genuine ARP packets. This file generated the generated_logs.log file that contained a detailed list of all the genuine packets with their reason for being genuine.

Next, we sent spoofed ARP Request and Reply packets from HyenaeFE to the Kali VM containing HoneyPy. It was seen that HoneyPy failed to log any of the data that was sent.

So, we ran startCapture.py which is our script to capture of all traffic on a given interface and it logged every single packet in the pcap file called dump.pcap. Next, we ran our ARP.py to analyse the pcap file for spoofed ARP packets. This file generated the generated_logs.log file that contained a detailed list of all the spoofed packets with their reason for being spoofed. Therefore, we see that our script worked better when it came to detecting ARP spoofing.

6.3 IP spoofing detection

HoneyPy was tested regarding its ability to detect TCP/UDP packets. We sent packets with genuine source and destination IPs to the Kali VM that contained HoneyPy but the honeypot failed to capture any of those genuine TCP/UDP packets.

So, we ran startCapture.py which is our script to capture of all traffic on a given interface and it logged every single packet in the pcap file called dump.pcap. Next, we ran our IP.py to analyse the pcap file for genuine TCP/UDP packets. This file generated the generated_logs.log file that contained a detailed list of all the genuine packets with their reason for being genuine.

Next, we sent TCP/UDP packets from HyenaeFE to the Kali VM containing HoneyPy. It was seen that HoneyPy failed to log any of the data that was sent.

So, we ran startCapture.py which is our script to capture of all traffic on a given interface and it logged every single packet in the pcap file called dump.pcap. Next, we ran our IP.py to analyse the pcap file for spoofed TCP/UDP packets. This file generated the generated_logs.log file that contained a detailed list of all the spoofed packets with their reason for being spoofed. Therefore, we see that our script worked better when it came to detecting IP spoofing.

6.4 DNS spoofing detection

First, we set up a DNS spoofing server on the other Kali Linux VM that we have in our network. Next, we add the Windows10 x86 VM to the target list. Now if we run the startCapture.py file, and try opening various websites on a browser in the Windows VM, spoofed traffic will get captured in the dump.pcap and upon running the DNS.py file it will get logged into the generated_logs.log file as spoofed traffic with the reason for being called spoofed. Our HoneyPy failed to log anything at all during this attempt.

Now, we go back to our other Kali VM which contained the honeypots and visit various websites, while capturing these packets with startCapture.py to generate the dump.pcap file. Upon running the DNS.py script on this pcap file, we will see that the traffic that it captured happens to be genuine traffic and that gets logged into the generated_logs.log file. HoneyPy failed to log anything at all during this attempt. Therefore, we see that our script worked better when it came to detecting DNS spoofing.

6.5 Discussion

This section contains a thorough discussion on the findings from the performed experiment. While performing our experiment, we realised a few problems to our approach of detecting IP, ARP and DNS spoofing attacks.

<u>For IP spoofing attacks:</u>

Our approach of first capturing the traffic and then analysing the pcap file doesn't work if the hosts are not alive during the TCP spoofing analysis on the pcap file because then we will not receive the SYN-ACK reply to any of our SYN requests.

<u>For ARP spoofing attacks:</u>

If during the nmap scan, spoofed IP and MAC address combination is sent by attacker, then the whitelist is poisoned and our approach doesn't work because it works on the fundamental policy of whitelisting.

<u>For DNS spoofing:</u>

In our approach, even if n answers to a DNS query are spoofed and just one is genuine and corresponds to the gethostbyname value, the DNS reply is considered genuine.

7 Conclusion and Future Work

Our paper proposes an approach towards enhancing the functionalities of a low interaction honeypot to enable them to detect basic spoofing attacks like IP, ARP and DNS spoofing. First, we saw the capabilities of the low interaction honeypot HoneyPy when it came to detecting spoofing attacks. Then we presented the capability of our enhancement scripts in detecting the same which turned out to be a hundred percent efficient.

The problem of our approach in detecting IP spoofing talked about in the Discussion section of the paper can be overcome if both the gathering of pcap files and the analysis could happen simultaneously and in real time. Then we could know for sure if the hosts are alive. Regarding the problem with our approach in detecting ARP spoofing, this can be overcome with a simple solution. The internal network should be verified by physical or other virtual means so that the nmap scan doesn't give faulty results that lead to poisoning of the whitelist. Only when the network has been verified, should whitelisting be performed. Another thing that we should do is that multiple nmap scans should be performed at time intervals to check for new hosts or change in the IPs in case a device was reset or reconfigured. Regarding the problem with our approach in the detection mechanism of DNS spoofing - there should be a local DNS server with valid entries. The honeypot should query the local machine instead of querying any online DNS server. The DNS query replies should be matched completely with the answers acquired from the local DNS server and should be considered genuine only if all the answers in the DNS replies are the same. Lastly, if these scripts could be incorporated into the functioning of a low interaction honeypot then they can be used to detect and defend against spoofing attacks while maintaining a low cost and high efficiency and it would also extend the honeypot's lifeline.

References

[1] 'Spoofing Attack: IP, DNS & ARP', Veracode. [Online]. Available: https://www.veracode.com/security/spoofing-attack. [Accessed: 13-Feb-2020].

[2] 'New Internet Research Shows 30,000 Spoofing Attacks Per Day', Dell Technologies, 12-Dec2018. [Online]. Available: https://www.delltechnologies.com/en-us/perspectives/newinternet-research-shows-30000-spoofing-attacks-per-day/. [Accessed: 13-Feb-2020].

[3] 'Spoofing Attacks: Everything You Need to Know', Springboard Blog, 07-Jun-2018. [Accessed: 13-Feb-2020].R. Kune, P. Konugurthi, A. Agarwal, C. R. Rao, and R. Buyya, "The anatomy of big data computing," *Software: Practice and Experience*, vol. 46, no. 1, pp. 79–105, 2016.

[4] 'What Is DNS Spoofing | Cache Poisoning Attack Example | Imperva'. Learning Center, https://www.imperva.com/learn/application-security/dns-spoofing/. Accessed 17 Aug. 2020.

[5] I. Livshitz, 'Low, Medium and High Interaction Honeypot Security | Guardicore', Guardicore - Data Center and Cloud Security, 03-Jan-2019. [Online]. Available: https://www.guardicore.com/2019/1/high-interaction-honeypot-versus-low-interactionhoneypot/. [Accessed: 19-Feb-2020].

[6] A. Dr. N. Arumugam, 'A Novel Method for Detecting and Preventing IP Spoofing Attack in Data Network'', Nachimuthu Polytechnic College, Pollachi, Tamin Nadu, 2018.

[7] J. Yu, E. Kim, H. Kim, and J. Huh, ‘A Framework for Detecting MAC and IP Spoofing Attacks with Network Characteristics’, in 2016 International Conference on Software Security and Assurance (ICSSA), Aug. 2016, pp. 49–53, doi: 10.1109/ICSSA.2016.16.

[8] G. K. Sadasivam and C. Hota, ‘Scalable Honeypot Architecture for Identifying Malicious Network Activities’, 2015 International Conference on Emerging Information Technology and Engineering Solutions, 2015, doi: 10.1109/EITES.2015.15.

[9] Mukaddam, Ayman, et al. ‘IP Spoofing Detection Using Modified Hop Count’. 2014 IEEE 28th International Conference on Advanced Information Networking and Applications, 2014, pp. 512–16. IEEE Xplore, doi:10.1109/AINA.2014.62.

[10] J. of C. S. Ijcsis, ‘Network Traffic Analysis Based IoT Botnet Detection Using Honeynet Data Applying Classification Techniques’, IJCSIS Vol 17 No. 8 August Issue.

[11] S. R. Chinta, V. B. Polinati, and P. N. Srinivas, ‘Detecting Bots inside a Host using Network Behavior Analysis’, International Journal of Computer Applications, vol. 180, no. 47, pp. 1–4, Jun. 2018.

[12] N. Naik and P. Jenkins, ‘A Fuzzy Approach for Detecting and Defending Against Spoofing Attacks on Low Interaction Honeypots’, in 2018 21st International Conference on Information Fusion (FUSION), Cambridge, United Kingdom, Jul. 2018, pp. 904–910, doi: 10.23919/ICIF.2018.8455555.

[13] N. Naik, C. Shang, Q. Shen, and P. Jenkins, ‘Vigilant Dynamic Honeypot Assisted by Dynamic Fuzzy Rule Interpolation’, in 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Nov. 2018, pp. 1731–1738, doi: 10.1109/SSCI.2018.8628775.

[14] Naik, P. Jenkins, R. Cooke, and L. Yang, ‘Honeypots That Bite Back: A Fuzzy Technique for Identifying and Inhibiting Fingerprinting Attacks on Low Interaction Honeypots’, in 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Jul. 2018, pp. 1–8, doi: 10.1109/FUZZ-IEEE.2018.8491456.

[15] Naik, P. Jenkins, R. Cooke, and L. Yang, ‘Honeypots That Bite Back: A Fuzzy Technique for Identifying and Inhibiting Fingerprinting Attacks on Low Interaction Honeypots’, in 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Jul. 2018, pp. 1–8, doi: 10.1109/FUZZ-IEEE.2018.8491456.

[16] S. N. Shiaeles and M. Papadaki, ‘FHSD: An Improved IP Spoof Detection Method for Web DDoS Attacks’, Comput J, vol. 58, no. 4, pp. 892–903, Apr. 2015, doi: 10.1093/comjnl/bxu007.

[17] ‘Download HyenaeFE 0.1-1 Beta’. Softpedia, https://www.softpedia.com/get/Network-Tools/Misc-Networking-Tools/HyenaeFE.shtml. Accessed 17 Aug. 2020.

[18] ‘Introduction to HoneyPy & HoneyDB’. Signal Sciences, 1 Sept. 2016, https://www.signalsciences.com/blog/introduction-to-honeypy-honeydb/.

[19] Mushorg/Snare. 2015. MushMush, 2020. GitHub, https://github.com/mushorg/snare.