UNIVERSITY OF TARTU
Institute of Computer Science
Cyber Security Curriculum

Ahmed Elazazy

HoneyProxy Implementation in Cloud Environment with Docker Container HoneyFarm

Master's Thesis (30 ECTS)

Supervisor: Anton Vedeshin, MSc
Supervisor: Truls Tuxen Ringkjob, MSc
Supervisor: Raimundas Matulevicius, PhD

Tartu 2018

HoneyProxy Implementation in Cloud Environment with Docker HoneyFarm

Abstract:
Cloud hosting services is a common trend nowadays for small startups, medium sized business and even for large big cooperations, that is helping the agility and scaling of resources and spare the overhead of controlling, managing and administrating the data-centers. The fast growing technology raised security questions of how to control the access to the services hosted on the cloud, and whether the performance and the latency of the solutions offered to address these questions are within the bearable limits. This research is introducing the honeypots to the cloud in a revolutionary way that exposes and applies what is called a HoneyProxy to work as a honeynet gateway for a reverse proxy that is controlling the incoming and outgoing flow to the back-end services. This HoneyProxy is connected to a HoneyFarm that is hosted on the same machine (cloud server) each honeypot is serviced in a docker container dedicated for every unique IP, so that each attack session can be isolated within one container with the ability to switch between different types of containers that can fool the attacker without suspecting the existence of a honeypot. This defending mechanism can detect and log attackers behavior which can reveal new attack techniques and even zero day exploits. The contribution of this work is introducing the framework to implement the HoneyProxy on the cloud services using Docker containers.

Keywords:
honeynet , honeyproxy, dockers, malicious threats

CERCS: P170, Computer science, numerical analysis, systems, control

HoneyProxy implementeerimine pilvekeskkonnas Docker konteineritel põhineva HoneyFarm lahendusega

Lühikokkuvõte:
Pilveteenustel põhinev infotehnoloogia süsteemide taristu on saamas tavapäraseks nii idufirmades, keskmise suurusega ettevõtetes kui ka suurtes korporatsioonides, toetades agiilsemat tarkvara arendust ning lihtsustades andmekeskuste haldamist, kontrollimist ja administreerimist. See kiirelt arenev tehnoloogiavaldkond tõstatas palju turvalisusega seotud küsimusi seoses pilves hoitavate teenuste ligipääsetavuse kontrollimisega ning sellega, kas pakutud lahenduste jõudlus ning viiteaeg (latentsus) jäävad aktsepteeritavatesse piiridesse. Käesolev teadustöö tutvustab honeypot peibutusmehhanismi pilves revolutsioonilisel viisil, mis rakendab HoneyProxy lahendust honeynet lüüsina pöördproksile, mis kontrollib sissetulevaid ja väljaminevaid päringuid back-end teenustesse. Vastav HoneyProxy on ühendatud HoneyFarm lahendusega, mida käitatakse samal masinal (pilveserveril). Iga honeypot jookseb eraldi Docker'i konteineris ning omab unikaalset IP-d, mistõttu on võimalik igat ründesessiooni isoleerida ühte konteinerisse

võimalusega vahetada erinevate konteineritüüpide vahel, ajades ründaja segadusse honeypot'i kasutust paljastamata. See kaitsemehhanism suudab tuvastada ja logida ründaja tegevusi, mis võivad omakorda paljastada uusi ründetehnikaid ning isegi "nullpäeva" (zero-day) haavatavusi. Käesoleva töö fookus on tutvustada raamistikku HoneyProxy implementeerimiseks pilveteenustel Docker'i konteinereid kasutades.

Võtmesõnad:
honeynet , honeyproxy, dockers, pahatahtlik ohud

CERCS:P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

List of Figures

List of Tables

1 Introduction

As long as time goes by, there would be no system secure enough. For this reason, the most hardened system is still vulnerable to unknown attacks and zero-day exploits. Honeypots are one of the solutions for predicting, the always evolving attacker techniques, they started as a way to fool them into the trap of capturing their intentions.

Honeypots were presented for the past two decades as a trap point that can lure an attacker into, making them exposed, traced and reveal their attack trial to a dedicated system to capture their behavior, techniques and scan any malicious payloads or exploits which can early detect and prevent a zero-day exploit or uncover a weakness point in a production system, and could be eventually patched before it can be approached by any attacker later on.

Many architectures for honeynets were introduced to control an attacker and capture their traffic, these will be discussed in section 2, their efforts were moving towards building a system with low complexity in terms of deployment and management, not to mention centralizing the logs collected for all the sessions. Starting from the HoneyFarm [3] that went for that cause and ending with the most recent HoneyProxy[7], which is mostly influencing this research with it's design.

Nevertheless the efforts toward applying a honeynet design in a cloud environment was not well covered in a satisfying way that can work with the cloud nature of on demand resources in a best practice to work efficiently in different aspects like performance, cost, isolation and resources utilization.

As there were no sufficient research in this all the background papers presented were mostly dated between 2015 till 2017 which reflects that this research is following up an on-going researches and cover the gap point of hosting such systems in a cloud environment.

That encouraged introducing the container-based technology into the honeynet systems as they are customizable ready-made, easy to deploy images that can run independent from the hosting infrastructure, scale their instances proportionally on demand with the volume of traffic and controllably with the predefined limits, and since it's a fast developing technology there were no solid efforts in integrating them in a model to ease the automation of running what the author calls a container sandbox and trap the attacker within its borders.

Thus the main contribution in this research is building a model honeynet suitable to work on a cloud environment and take advantage of its infrastructure and efficiently utilize

its resources, in addition to integrating the container technology with the honeypots deployments so that we can make each service trap scalable on demand and to make use of the containers' agility and isolation, and finally to control forwarding incoming and outgoing traffics, reverse proxy will be used to decide which traffic should be proxied to the production environment and which requests should be forwarded to the Docker HoneyFarm system.

A dedicated container with respect to type of attacking protocol will be launched for each attacking session with a unique IP address so that all the incoming traffic from this source will be forwarded to the same container. A cron job will be running every fixed interval e.g. 5 minutes to remove idle containers, or force removing them to make space for new ones.

The flow of work starts with Introducing the problem statement, research questions and the legal aspects in section 1, then a summary of background work of all related research efforts in the field and their influence on the implementation in hand through section 2. Next, organized practical work steps and implementation guide with the type of the technologies used in section 3. Section 4 shows the outcomes, insights and validation of the work presented and how the work results was realized through. Finally Section 5 summaries the research plans and achievements and answer the research questions.

1.1 Problem Statement

Honeynet is network architecture that is meant to deceive an attacker and capture their techniques for further analysis of the attacking session, they have been evolving through the last two decades, but without coping with the continuously advanced adversaries techniques. Recently a research was made by a team in Arizona State University which came up with a new revolutionary honeynet architecture they called HoneyProxy that can address several challenges in protecting the network through a centralized reverse proxy module which could multicast malicious traffic to honeypots and select the response that does not contain a fingerprint indication.

Applying extra security tools and measures to a cloud environment is highly demanded for the sensitive resources and services with high availability that are hosted on the cloud, with that in mind integrating a new security technique may cause a network and performance overhead that might affect the underlined service and thus increase the latency dramatically.

Experimenting and evaluating a recently developed honeynet architecture "HoneyProxy", that was implemented only on physical servers, in a cloud environment and adding the utilization of the docker containers to the honeynet network is the point to be addressed

in this research, in order to answer the question of the latency overhead, the isolation and agility of the containers with high resources utilization.

1.2 Research Questions

1. How can we protect cloud-hosted services from malicious attacks using Docker containers?
2. Would a normal honeypot be enough to protect the backend services?
3. How much deploying a Honeynet would give information about future attacks?
4. What are the benefits of deploying the honeynet's honeypot on docker containers?
5. How to avoid honeypots fingerprints?
6. What are the advantages of using docker containers versus using virtual machines?

1.3 Research Scope

The goal for this research is to reach a fully working and optimized design model for a honeynet defensing mechanism for a cloud hosted services, however as this thesis is bounded by time and resources, the research scope must be focusing on the initial objectives of that goal, as to build a working prototype and an implementation guide for the following researches.

The work for building this model was represented only in the basic prototype that is a starting point for many further researches, however the author explained and highlighted all the possible and potential features that can be added or merged to the model. As a master thesis the scope was limited with giving the best effort for the final goal more than implementing every small detail, nevertheless, examples and solid recommendations were provided to follow as a future work.

As for the containers, one Docker image will be used per protocol and one hosted on a virtual machine and connected to the reverse proxy that is hosted on another virtual machine, and will be using the host machine as a source of traffic to simulate different type of attacks and scenarios for testing and validation.

The Validation will be working with different honeypot scenarios for a successful connection, which would be using the naming convention suitable for each type of communication protocol or for the type of honeypot. However duplicating the traffic and response

selection process would need further time and research, which will not be included in this work more than the future work recommendations.

1.4 Legal Aspects

Tracking and recording traffic for malicious activities seem legit as a logical concept from the network's owner perspective, however the law protects everyone's privacy in general, sometimes without even considering the intentions of the trial. The research aim to detect the style and the technique of the attack rather than identifying the attacker's identity, so that potential vulnerabilities could be spotted and avoided. Legally the laws should be flexible enough to facilitate these intentions without exposing captured identities.

According to article-2 in the *Directive 95/46/EC*[1], 'personal data' is defined as "any information relating to an identified or identifiable natural person", and 'processing of personal data' definition includes."any operation or set of operations which is performed upon personal data, whether or not by automatic means, such as collection, recording, organization, storage, ...". Nowadays, when doing research in the information technology field, it is usually hard to draw an exact rigid line between what is legal and what is not. Every research introduces new service, technology, or platform, how can you specify an exact law for what you do not know yet!.

The implementation of honeypots have always brought concerns about the 'privacy rights of attackers' and the 'trapping of spammers'. Several researches have highlighted the obstacles that impact the lead in the research field. In his paper *The Honeynet Project: Trapping the Hackers* [2] in 2003, Richard Salgado drew the attention of the need to hire a lawyer before implementing a honeypot in th United States!. He presented the concerns about monitoring the traffic that is not yours, and the risk that results from the acquisition of your established honeypot by a hacker to attack others who you have been already monitoring.

Several authors have discussed the need to consider the IP as a personal data as it can be used with today's technology to identify a person [13, 14]. On the other hand *Directive 95/46/EC* in article (6a and 6e) allows a limit usage of personal data for statistical and scientific reasons, if the collection and processing is done for legitimate specific reason and for known period of time. For any further processing and storage of the data, the researcher need to securely anonymize and store it in a manner that prevents identifying the original data subjects.

[1] EU Data Protection Directive

In their 2017's research *Honeypots and honeynets: issues of privacy*, Sokol et al. summarize the challenges that have restricted the researchers who use this technology, and differentiate between the 'production honeypots' and the 'research honeypot' [13]. They also brief the EU laws concerned with digital data and the accepted scope of using data and personal data for legitimate research purpose.

Finally, it is worth to highlight that the data that has been used in this work belongs to data subjects who have given their consent to track their traffic for the purpose of this research. Furthermore, the data will be erased as soon as this research is published.

2 Background

This section presents a summarization of different research efforts, this work was built on, from how the reverse proxy is working, through Honeypots structures and finally how to utilize them as Docker containers, highlighting which ones this research was most influenced with.

2.1 Examples of Reverse Proxy Implementation

Valeur et al. have used the reverse proxy back in 2006 to mitigate the impact of attacks targeting the web-based applications [15]. The team chose to provide an alternative design for the e-commerce application as shown in Figure 1. The main idea was to (i) separate the back-end database that contains product information and (ii) have the functionalities of the application replicated on different servers, A, B, and C. According to the anomalous level of each user query, the request will be directed to server-A (if highly anomalous), server-B (if moderately anomalous), or server-C (if confirmed legitimate request). Queries that are directed to server-A are dropped, because server-A does not have connection with the back-end database. Queries directed to server-B, will have limited response. And users queries on server-C will be totally fulfilled. Figure 1 outlines the full platform.

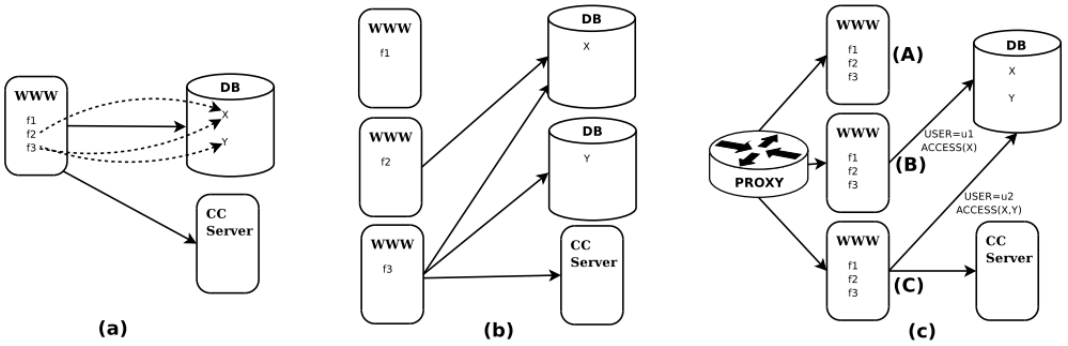


Figure 1. Website designs. (f1,f2,f3=different web functions, DB=Database, CC=Credit Card) [15].

In 2009, Wurzinger et al. used the reverse proxy to mitigate XSS Attacks [1]. There product SWAP (Secure Web Application Proxy), shown in Figure 2, works on detecting JavaScript component before the user's request is fulfilled. Legitimate known scripts are checked against their encoded IDs in the database. Upon detection of other script,

the request is dropped. Else, the proxy decodes legitimate scripts IDs and allows the response to pass to the client.

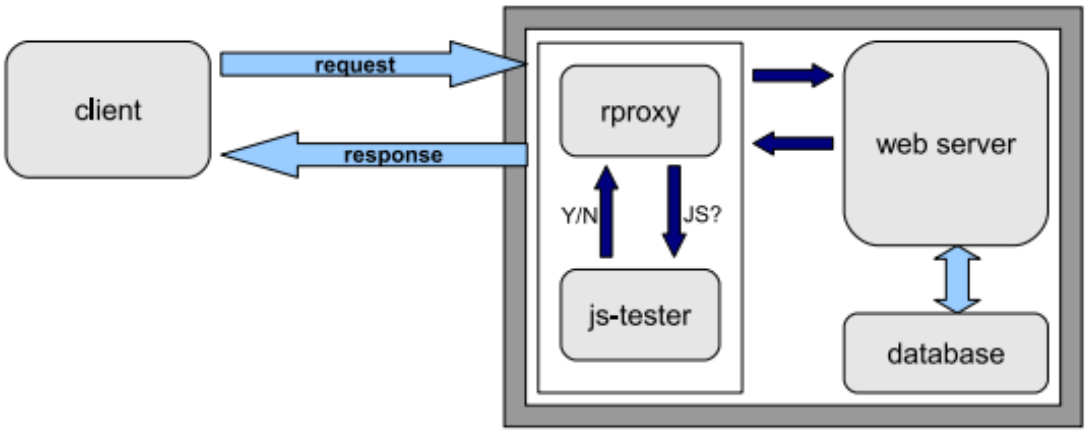


Figure 2. Scheme of SWAP setup [1].

2.2 Honeynet Architecture and Structure

According to the honeypot mailist [16], a public forum of 5000 security professionals, *"A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource."*, ever-since then the honeypot project evolved and in 1999 started forming the first generation 'Gen-I' [2] of honeynet technologies that was basic in terms of controlling the attacker and collecting encrypted activities, however it was successful enough to capture automated attacks known back then such as autorooters[2], in Figure 3 gives an overview of Gen-I honeynet, which is a contained environment with yellow highlighted target systems.

In order to have more advanced techniques to control the attackers and monitor their activities, in 2002, the second generation 'Gen-II' of honeynet was developed to making honeynets simpler to deploy and allowing more space for the attacker to go far without detecting the honeynet presence, thus decrease the possibilities of compromising other networks by controlling outbound connections, Figure 4 shows the honeynet sensor working as an isolator bridge for layer-2, however, attackers could still detect the honeynets from their fingerprints.

[2]Symantec: Introduction to Autorooters

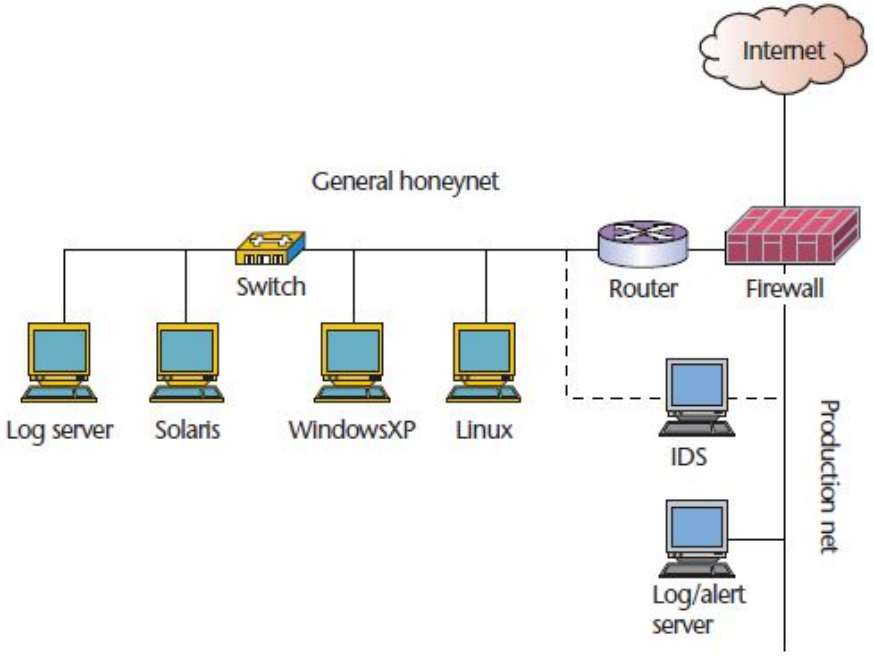


Figure 3. Gen-I honeynet architecture [2].

Further improvements were done on the same architecture from deployment and management perspectives to introduce 'Gen-III' honeynets with Sebek server[3] built in the gateway, it utilizes 3 interfaces two of them are for connecting external network to the internal honeypot, leaving the third interface for management configurations.

Later-on several techniques were introduced to simplify and cover different needs to monitor, control and capture the data.

Honeypot Farm [3], is about deploying honeypots in large networks were the need to deploy a honeypot in each network is substituted with deploying a consolidated honeypot, so whenever an attack trial is performed on any of the networks, it gets redirected to the consolidated honeypot farm (Figure 5) without the attacker knowing, however this techniques is vulnerable to internal propogation of malware.
Honeybird [4], is a hybrid honeynet network that combines between low and high interaction honeypots, it's made of four main components[17], 1) Decision engine that is used for filtering incoming attacks, 2) Redirection engine, decides if the traffic needs to be redirected for further analysis, 3) Control engine, block the compromised honeypot's

[3]Symantec: Tracking the attackers
[4]Honeybrid

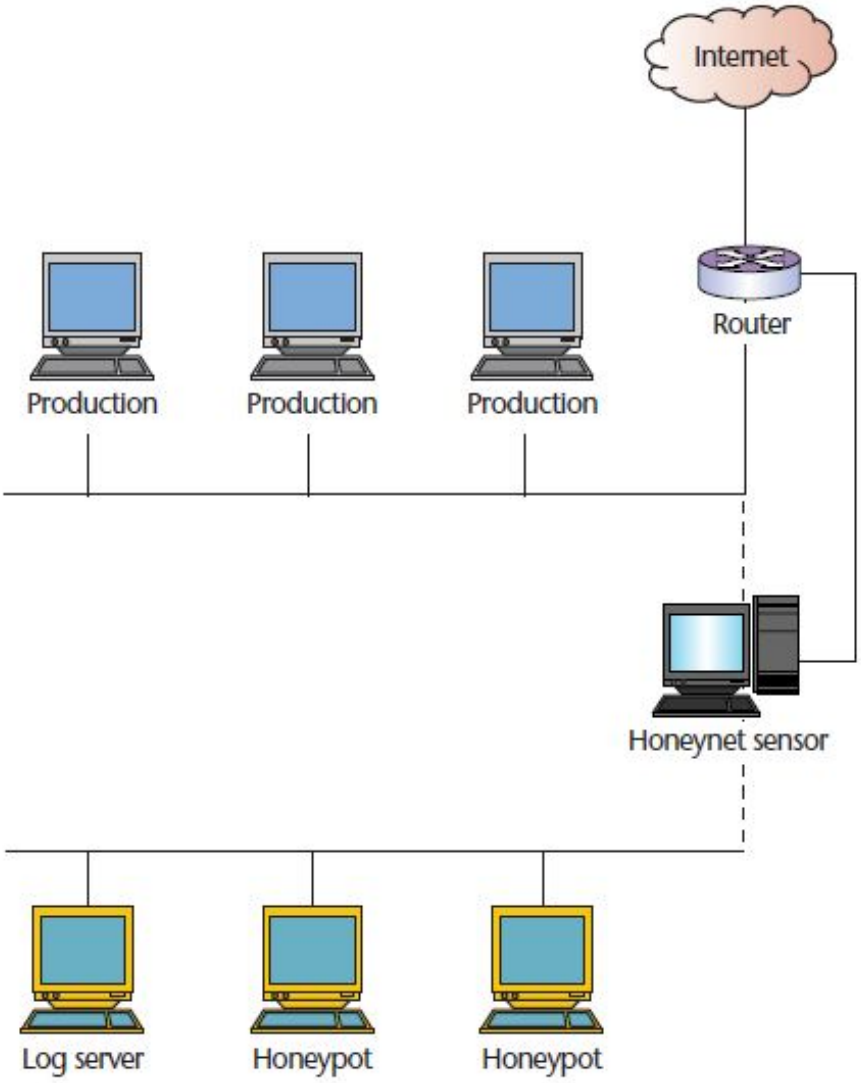


Figure 4. Gen-II honeynet architecture[2].

outgoing traffic, 4) Log engine, store all the processed traffic. The overall framework of different software represented in Figure 6.

Through this design, it was observed that the initial scanning attacks are handled by low-interaction honeypot, however, during most of the connection only the high-interaction honeypot are active.

Collapsar [5], was introduced in 2006 to realize the idea of a honeyFarm with *"decen-*

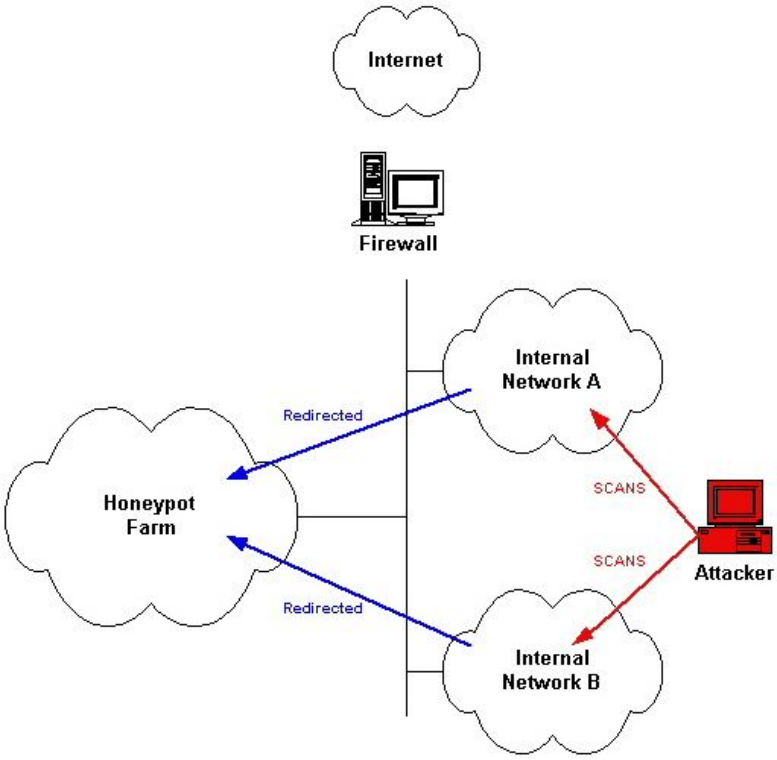


Figure 5. Honeypot Farm architecture [3].

tralized presence and centralized management." of honeypots, where they are created as virtual machines distributed logically multiple networks, but physically they exist in dedicated local network. the team has also incroduced reverse-HoneyFarm which is based on a client-side honeypots that crawl the Internet with server requests, to the server they appear to be from different domains. Figure 7a and 7b shows how the traffic connection flow for Collapsar HoneyFarm and Collapsar reverse HoneyFarm respectively.

HoneyMix [6], introduced, in 2016, SDN(Software Defined Networking)-based intelligent honeynet that leverages the programmable nature of SDN to enable the four controller modules that works as mechanism for attacker detection, these modules are the main components of the HoneyMix: *1) Response Scrubber,* avoid exposing the honeypot to be detected by scrubbing the fingerprinted responses. *2) Forwarding Desicion Engine,* "Service map", decides the nature of the request and to which network should the traffic be redirected to. *3) Connection Selection Engine,* maintains a connection with the attacker to the SDN while establish multiple connections the honeypots and pipe only the selected and filtered response fit for the type of attack. *4) Behavior Learner,* put weights for connections between SDN switch and the honeypots dependant on different parameters like modifications and active connections. In Figure 8 summarizes HoneyMix

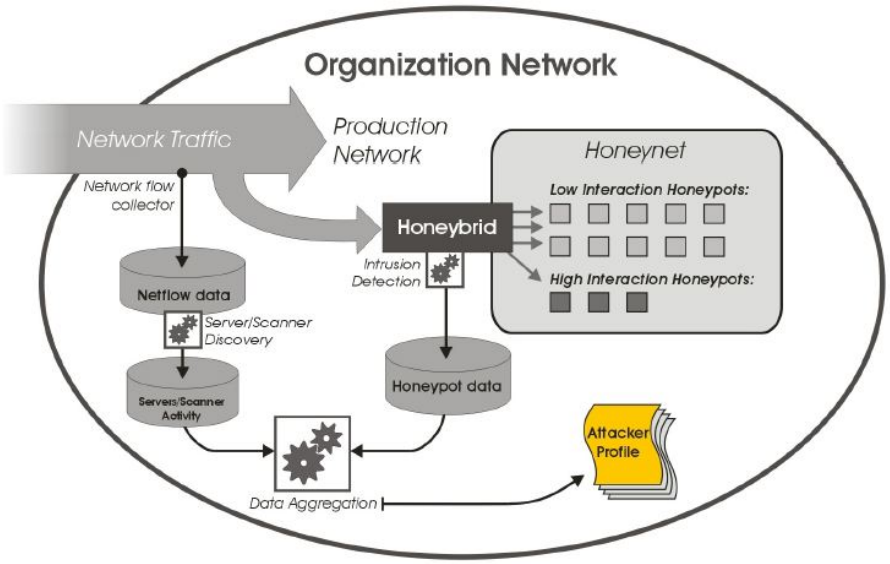
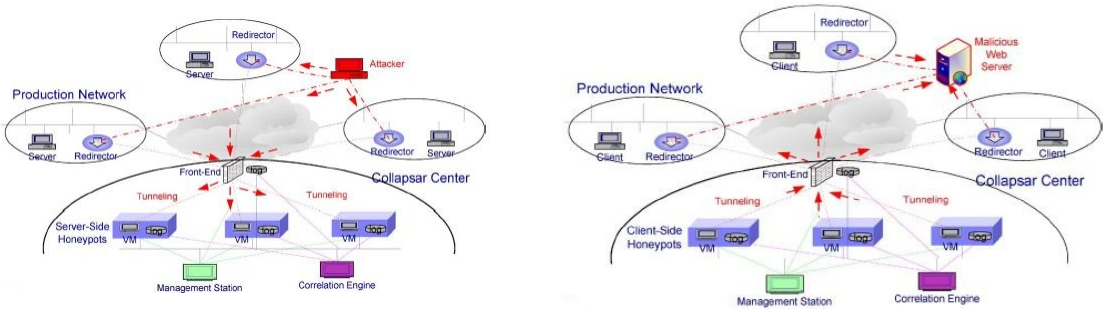


Figure 6. Overview of traffic analysis framework[4].



(a) HoneyFarm server-side honeypots.

(b) Reverse HoneyFarm client-side honeypots.

Figure 7. Collapsar architecture [5].

architecture with traffic steps in the network.

HoneyProxy [7], was a very recent design introduced in 2017 that was greatly influenced by HoneyMix, it is SDN-based as well that supports a very dynamic method of transitions between honeypots, and control captured data against fingerprinting attacks by multicasting the traffic to relevant honeypots and choose the response that had least fingerprinting indicators, in Table 1, an example of popular honeypot SSH Kippo[5] and its known fingerprints that are easy to detect.
The HoneyProxy SDN controller depend mainly on a tagging technique in the traffic

[5]Kippo honeypot

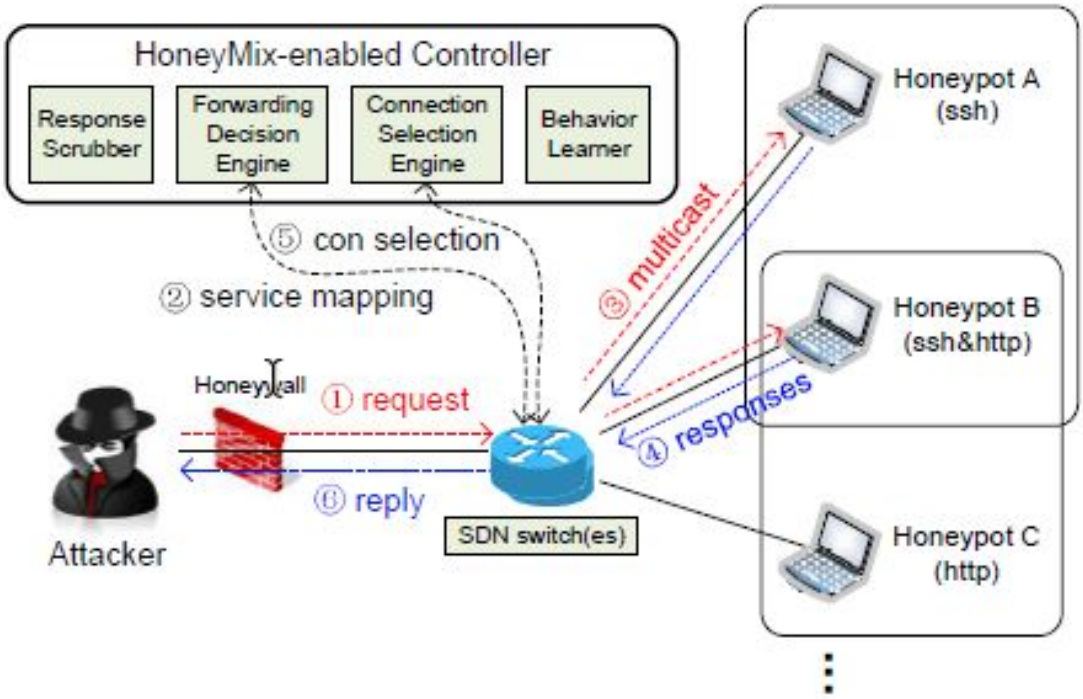


Figure 8. HoneyMix architecture [6].

Table 1. Example of fingerprints known in Kippo SSH honeypot [7].

Request		Response	
type	payload	type	payload
exact match	uname -a	exact match	Wed Nov 4 20:45:37 UTC 2009
pattern	.{7,}\n	exact match	bad packet length
exact match	vi	exact match	E558: Terminal entry not found in terminfo
exact match	ifconfig	exact match	HWaddr 00:4c:a8:ab:32:f4

header using the reverse proxy, then the SDN rules check the tags to redirect the traffic through the SDN switches to the relevant honeypots as seen in Figure9.

The HoneyProxy made a big leap in the honeynet architecture, instead of running multiple honeypots and being limited to maintain one connection with the attacker without switching dynamically, and not to mention configuring the honeypots manually, it made the honeynet work as a big entity that has different vulnerabilities integrated together as one connection from the attacker perspective, Figure 10a and 10b shows how the HoneyProxy converted the honeynet to work as big entity.

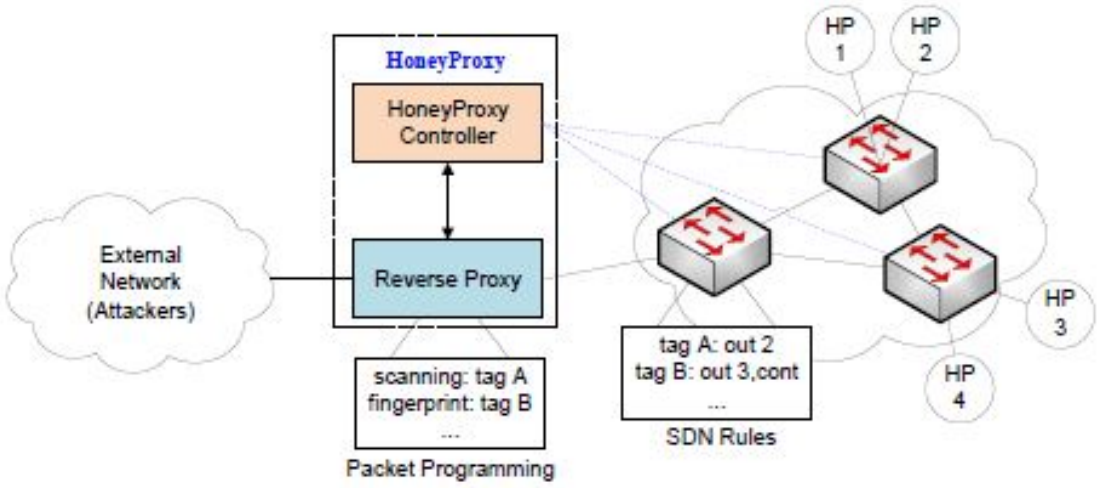
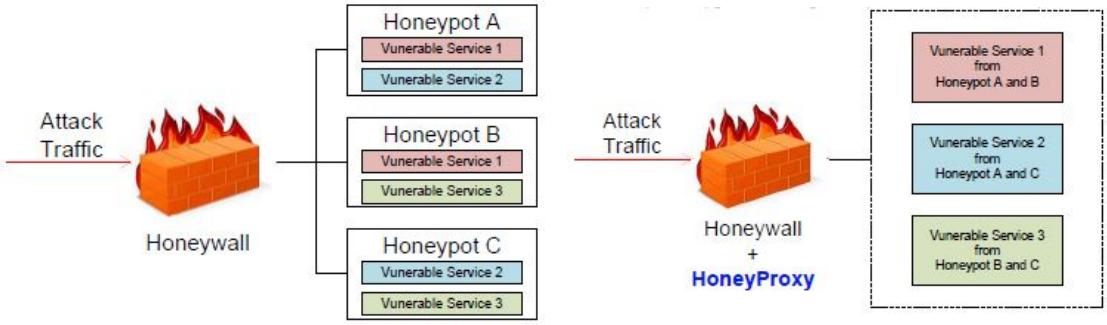


Figure 9. Overview of HoneyProxy and how tagging works [7].



(a) Normal honeypots run separately behind HoneyWall

(b) Vulnerabilities are combined from different honeypots as one entity.

Figure 10. HoneyProxy big leap[7].

The HoneyProxy team tested their design on a two physical servers and tested out their honeynet architecture using 4 types of honeypots as shown in Figure 11, it achieved a line rate throughput 8.23 Gbps on a 10 Gbps link and a negligble latency overhead of (0.5 - 1.2) milliseconds.

2.3 Honeypot as a Docker Container

Several researchers discussed the containers as new revolutionary technology that can be utilized in an efficient manner to help scientific research and production environments

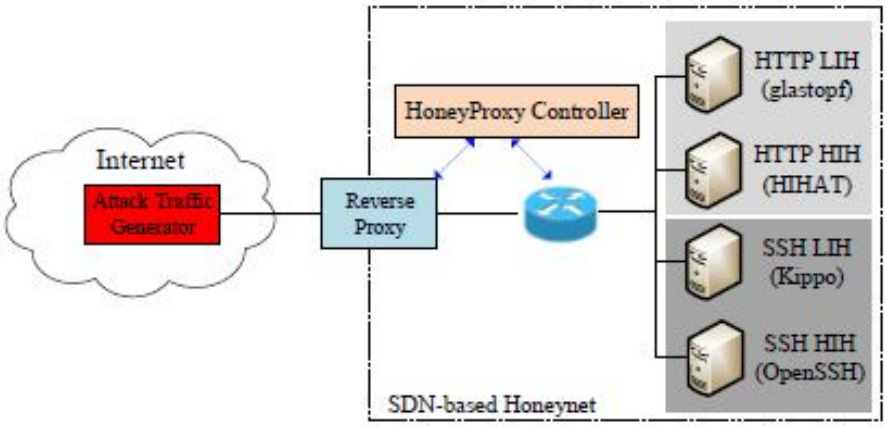


Figure 11. HoneyProxy setup [7].

with aspects of agility, performance advantages and the maintenance overhead compared to different technologies like normal virtual machines.

In 2017, Majithia [8] used the model of running three type honeypots on a Docker server, shown in Figure 12 with a logging management mechanism that is built on top of ELK framework[6] and discussed issues and security concerns associated with each honeypot. The honeypots used were HoneySMB[7], HoneyWEB-SQLi, an http protocol honeypot that includes SQL injection vulnerability and HoneyDB, a honeypot built for mysql databases vulnerabilities, the work displayed analysis of the attacks using unique IPs and the distribution among the honeypots.

In 2015 Adufu et al. [9] investigated and compared running molecular modeling simulation software ,autodock3[8], on a container-based virtualization technology systems and hypervisor-based virtualization technology systems, and concluded that the Container-based systems are managing the memory resources in an efficient manner even when memory allocated to instances are higher than physical resources, not to mention that the amount of execution times were reduced for multiple containers running in parallel as shown in the graphs in Figure 13.
Meanwhile in the same year EFTIMIE et al. [10] highlighted the advantages of using containers in their usage as a base for honeypot systems and how they can help overcoming issues of complexity in management and deployment, and eventually host them on a cloud environment.

[6]ELK framework
[7]HoneySMB
[8]Autodock

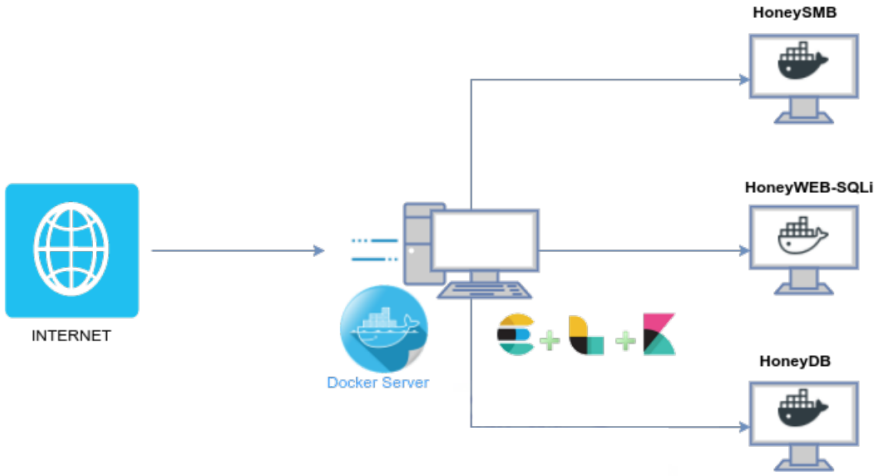
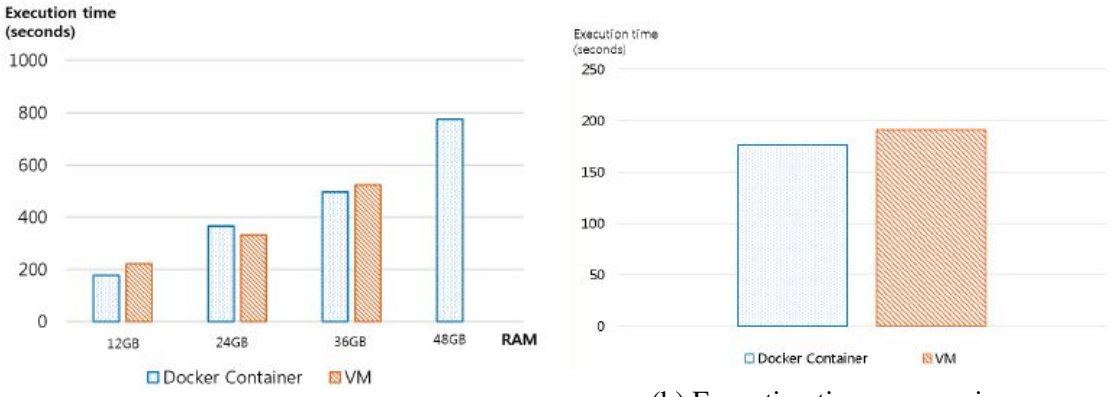


Figure 12. Docker Server working with 3 different honeypots [8]



(a) Memory comparison.

(b) Execution times comparison.

Figure 13. Container advantage against VMs [9].

The authors have also summarized the benefits of using the light containers to solve different overhead with much flexibility in these points, 1) They require no dependencies leaving no complications in interfacing with platform services, 2) They are interdependent from the cloud provider as the application built on containers can be running on different environments, 3) They can work portably and in an automated form to ease the deployment at any given time, 4) They are isolated and can be governed through the platform outside the container's level which reduce the complexity tremendously.

They built their Docker container host with a group of low interaction honeypots presented in Figure 14 to distribute the web services and give the them the right firewall rules to

forward the incoming and outgoing traffic, to have the container accept the malicious requests and assemble the connection with automating the iptables rule for the established connection.

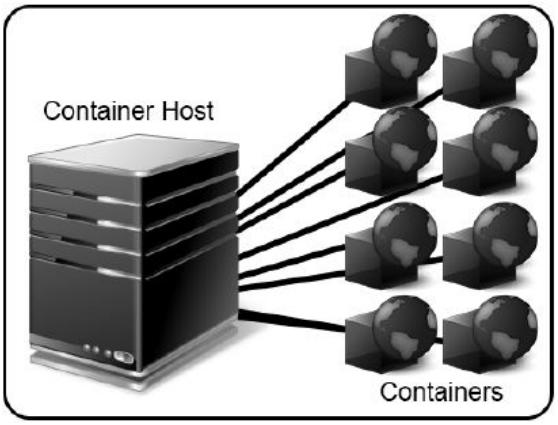


Figure 14. Honeypot hosted in Container-based environment as an on demand service [10].

As a survey of the existing honeypots introduced, in addition to highlighting their strength and weakness points, Nawrocki et al. presented in 2016 an overview of honeypot software and analysis techniques for the data collected by them. The team defined the evolution of attacking techniques, patterns and propagation, giving a comparison between different honeypot projects and the efforts that were contributed through them to this field, moreover they presented a general list of different honeypots, their services, type of interaction and design details shown selectively in Table 2.

In the same year, Fan et al. [11] presented a novel program based on security and decoy model and illustrated how their design and how the constraint in the features work in Figure 15 according to the three levels of cyber attack. The research gave some beneficial insights of honeypots application for future systems.

2.4 Summary

The work presented showed different architectures and models for honeynet and how new technologies affected these designs. However there was not too many researches contributing to this approach as it was not covered thoroughly.

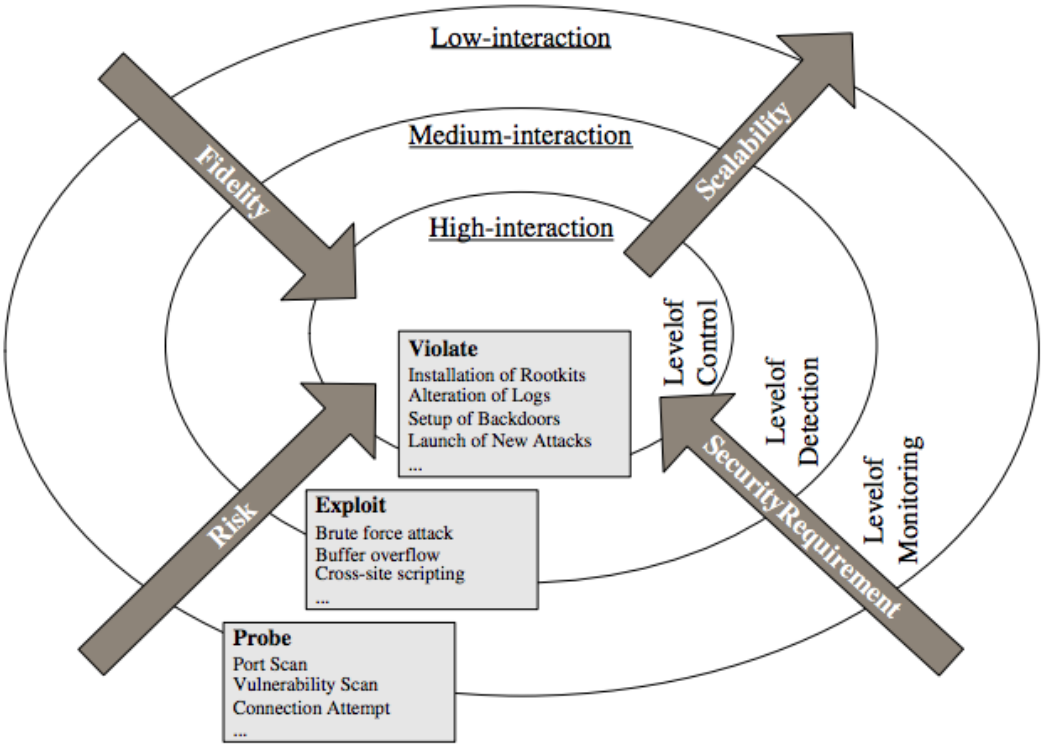


Figure 15. Attack levels mapped to honeypot's levels of interaction [11]

Table 2. Selective chronologically ordered Honeypots with scopes and remarks [12]

Software	First	Last	Free	Services / Applications	Design / Details
DTK	1997	1999	✓	SMB, SSH, DNS, FTP, Netstat(++)	implement many known vulnerabilities
BOF	1998	1999	✓	Back Orifice, Telnet, SMTP(+)	waste intruders time, easy deployment
NetFacade	1998	2002*	✗	not specified	class C network emulation
CyberCop String	1999	1999	✗	Telnet, FTP, SendMail, SNMP	emulating different network devices
Specter	1999	2005	✓	SMTP, FTP, HTTP and Telnet(+)	commercial deployment, decoy files
Sandtrap	2002*	2002*	✗	dialup modem	war dialing trapping
single-honeypot	2002	2002	✓	all ports, but no emulation	mere logging, KISS architecture
HoneyWeb	2002	2003	✓	HTTP	various web server header emulation
SMTPot	2002	2003	✓	SMTP	spam accumulation, KISS
Jackpot	2002	2004	✓	SMTP	delay spam, utilizing spam databases
FakeAP	2002	2005	✓	802.11b AP beacons	p.o.c wireless honeypots
HoneyBot	2002*	2007*	✓	SSH, SMTP, FTP, HTML(++)	windows vulnerabilities and GUI
Spampot	2003	2003	✓	SMTP	platform independence
HoneyPerl	2003	2003	✓	HTTP, FTP, SMTP, Telnet(+)	extensibility by modules
HoneyD	2003	2008	✓	HTTP, POP3, SMTP, FTP(+)	emulating heterogeneous networks
SpamD	2003	2015*	✓	SMTP	tarpit against spam
Kojoney	2005	2006	✓	SSH (shell activity)	first dedicated SSH honeypot
Mwcollect	2005	2009	✓	compare Nepenthes, Honeytrap	merging Nepenthes and Honeytrap
Nepenthes	2005	2009	✓	FTP, HTTP, TFTP, MSSQL(++)	capture worm payload
GHH	2005	2013	✓	HHTP-Apache, PHP, MSSQL	crawler and search engines
Honeytrap	2005	2015	✓	HTML, FTP(+), dyn. emulation	attacks via unknown protocols
HoneyPoint	2006	2014	✗	not specified	ICS/Scada, back tracking intruders
Dionaea	2009	2013	✓	SMB, FTP, SIP, MYSQL(++)	nepenthes successor, capture payload
Kippo	2009	2014	✓	SSH (shell activity)	emulate entire shell interaction
Artemisa	2010	2011	✓	VoIP, SIP	Bluetooth Malware
bluepot	2010	2015	✓	Bluetooth	Bluetooth Malware
HoneySink	2011	2011	✓	DNS, HTTP, FTP, IRC	bot sink holing
HoneyDroid	2011	2014*	✓	compare Kippo, HoneyTrap	p.o.c Android OS honeypot
Glastopf	2011	2015	✓	HTML, PHP, SQL	web applications, vulnerability types
Kojoney2	2012	2015	✓	SSH (shell activity)	applying Kojoneys lessons learned
Conpots	2013	2015	✓	kamstrup, BACnet, mosbus	ICS and SCADA architectures
IoTPOT	2014*	2015	✓	telnet	IoT (ARM, MIPS, and PPC)
honeypot-camera	2014	2015	✓	HTTP	Tornado Web, Webcam Server
Shockpot	2014	2015	✓	Apache, Bash	Shellshock vulnerability
Cowrie	2014	2015	✓	SSH (shell activity)	Kippos successor
Canarytokens	2015	2016	✓	URLs, bitcoin, PDF	honeypot tokens
elastichoney	2015	2015	✓	elasticsearch	elasticsearch RCEs
Sebek	2003	2011	✓	Win32 and Linux systems	attackers OS activities, state-based
Honeywall	2005	2009	✓	compare Sebek, CentOS	live bootable CD
HoneyBow	2006	2007	✓	Win32 Systems	extraction of malware, state-based
Argos	2006	2014	✓	Linux, Windows XP-7	0-day exploits identification, tainting
HIHAT	2007	2007	✓	php-BB,-Nuke,-Shell,-Myadmin	PHP framework extension, state-based
HoneyC	2004	2007	✓	HTTP	identify malicious servers with snort
Monkey-Spider	2007	2009	✓	HTTP, JavaScript	threat database creation, several AV
PhoneyC	2007	2011	✓	HTML, JavaScript, PDF, ActiveX(+)	browser identities, dyn. analysis
Thug	] 2011	2015	✓	HTML, JavaScript, PDF, Flash(+)	complete emulation, stat/dyn. analysis
YALIH	2014	2015	✓	HTML, JavaScript, (IMAP)	precise by combining analysis methods
HoneyClient	2004	2010	✓	Windows (Firefox, IE)	proof of concept, state-based
Capture-HPC	2004	2009	✓	Linux, Windows (Firefox, Office (+))	efficiency, scalability, state-based
UW-Spycrawler	2005	2006*	✓	Windows (IE)	spyware detection, state-based
HoneyMonkey	2005	2007*	✗	Windows (IE)	IE vulnerabilities, state-based
WEF	2006	2007	✓	Windows (IE)	drive-by download attacks, state-based
HoneyIM	2007	2007*	✓	compare Capture-HPC	instant messaging
SHELIA	2008	2009	✓	Windows (IMAP, POP)	email malware, call-tracing
Trigona	2010	2010	✓	Windows (Browsers)	high throughput, —
HoneySpider	2011	2015	✓	Capture-HPC, THUG	hybrid client honeypot framework

3 Practical Work

3.1 Introduction

The initial quest for the implementation was following a network topology that would realize the main goal of the research which is to build a flexible HoneyProxy on a Cloud environment that is able to handle all unauthorized access or untrusted sources of connection and open for that a dedicated docker container per attacker remote IP session, Figure 16 describes how the main plan for the implementation would work. The author have tested two types of containers in this design (SSH and HTTP), the SNORT honeypot container was added as an example of the different types of honeypots that can be utilized in the model.

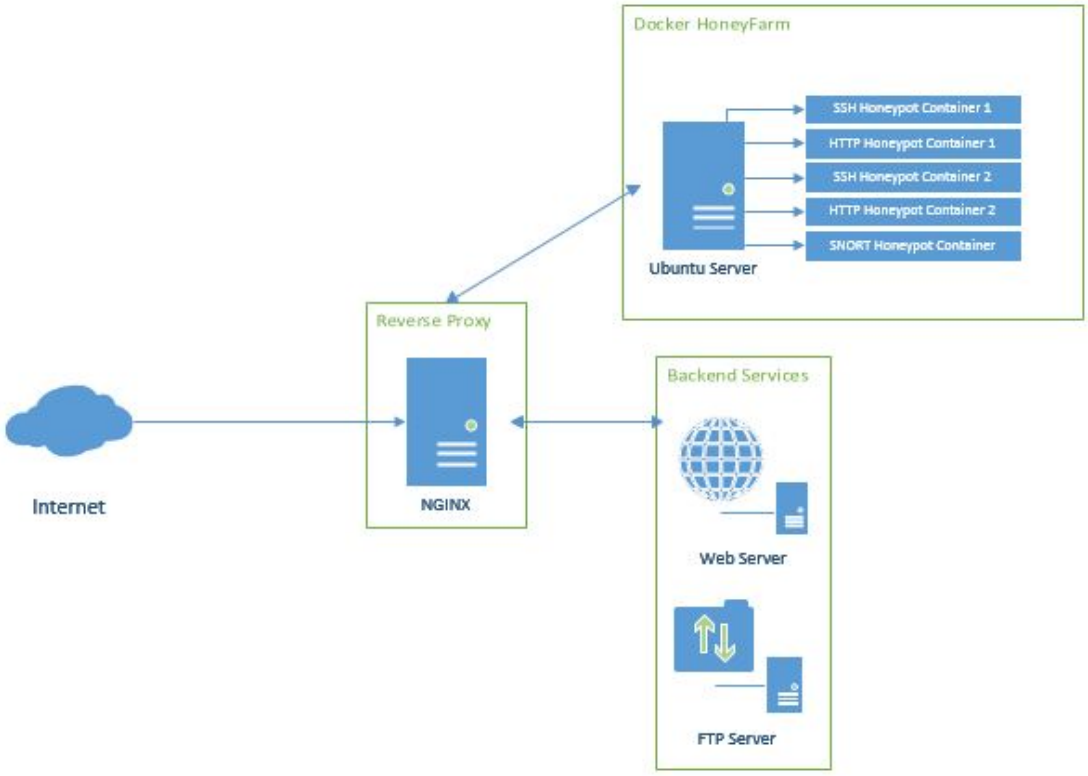


Figure 16. Implementation outline frame.

The traffic flow, summarized in the Flowchart below (Figure 17), would be starting from the Internet targeting NGINX the reverse proxy with an HTTP or SSH request to the web server, the NGINX would have a white-list of allowed IP ranges to access the production web server, other than that the traffic will be forwarded to the Honeypot Farm that will

launch two containers for each request matching the corresponding protocol.

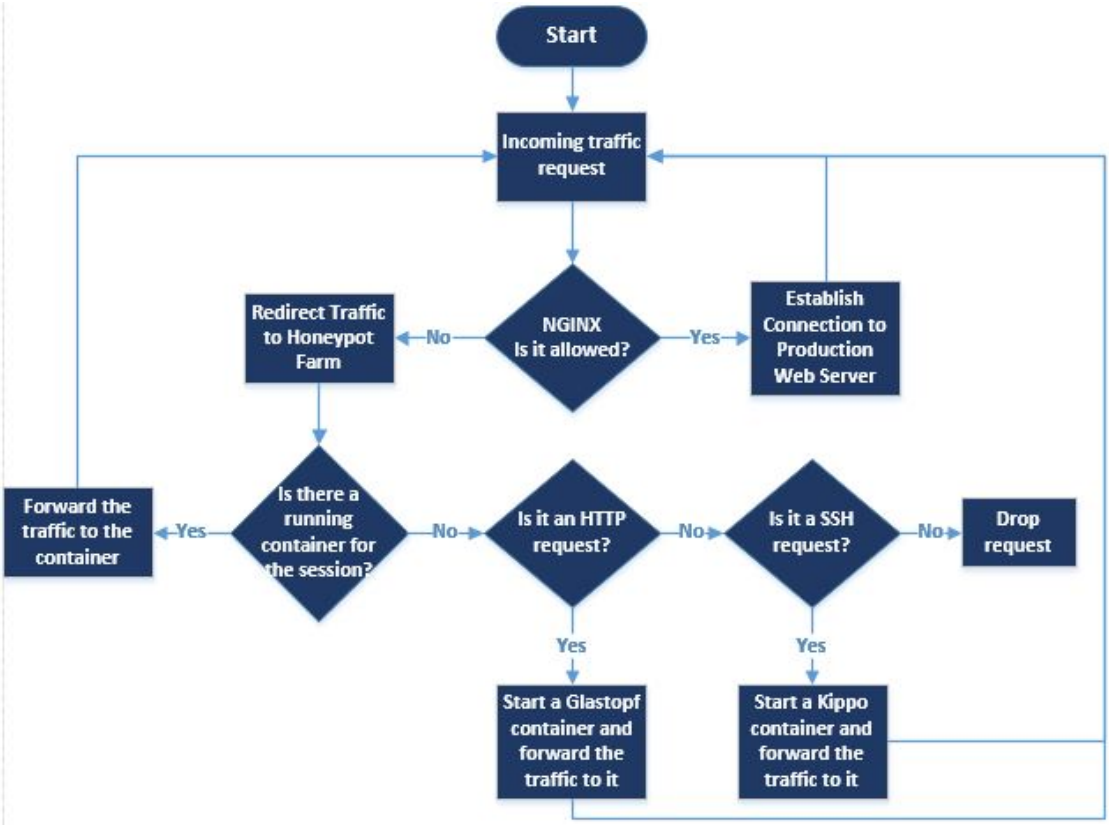


Figure 17. Traffic Flowchart

For study reasons the traffic would be duplicated and sent to both containers, but only one of them will be primary and would respond to the remote attacker, while the other honeypot response will be dropped, yet logged for later stages tuning and comparison for the least fingerprinted response.
The containers will stay running as long as the session is active, as there will be a clean-up cron job running every 5 minutes to clear and remove the old containers, thus better utilization for the server resources.

In order to start building the correct or optimum configuration for the above network architecture, two phases are made for this work to establish the connection and make a model for the final phase which the evaluation of this thesis research will be based on.

The first phase would be running locally on my personal laptop using VirtualBox that

would run two virtual machines, first for the reverse proxy and back-end services, and second for the Docker honeypots farm. As a future work, the solution would be running on a public cloud environment (preferably AWS) and apply the final configuration produced from the first phase with the final evaluation and observations.

Two virtual machines will be used for running Ubuntu Server 16.04 for both the honeypot farm and the reverse proxy with the back-end services. Virtual host-only network card was used for the VirtualBox with a DHCP service to connect the machines to my Host machine and to make them communicate with each other as well, also in this setup, the host machine was used as a source of traffic to simulate normal connection requests and attack trials.

3.2 Building the Honeypot Farm

Honeypots in general are designed either to emulate a production service or file storage which can fool the adversaries with capturing details about their tools and attacks, or at least send alerts of an attack trial or mis-usage of resources.
There are different level of interactions for honeypots that can categorize them into two types, 1) Low interaction; that allow only limited interaction for an attacker or a malware, which make them not vulnerable themselves and cannot be infected by exploit attempt, however, the rigid behavior makes it easy to be detected and avoided. 2) High interaction; that involves real operating systems and applications, since nothing is emulated and everything is real, it can provide more details of an attack or an intrusion, thus help in identifying unknown vulnerabilities, however, this nature exposes the honeypot and increase the risk of an attacker using it to compromise production systems

In this research we are not after how strong and robust the honeypots can be more than building a model for best utilization of such monitoring and defensive techniques, thus we are using a standard honeypot made into containers available through Docker Hub[9].

3.3 Using Docker Containers

Docker[10] defines the container as "an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly." (Fig.18) versus the virtual machines that has a

[9]Docker hub
[10]Docker

full operating systems, taking more space and takes much time to boot, which makes containers more portable, agile and efficient.

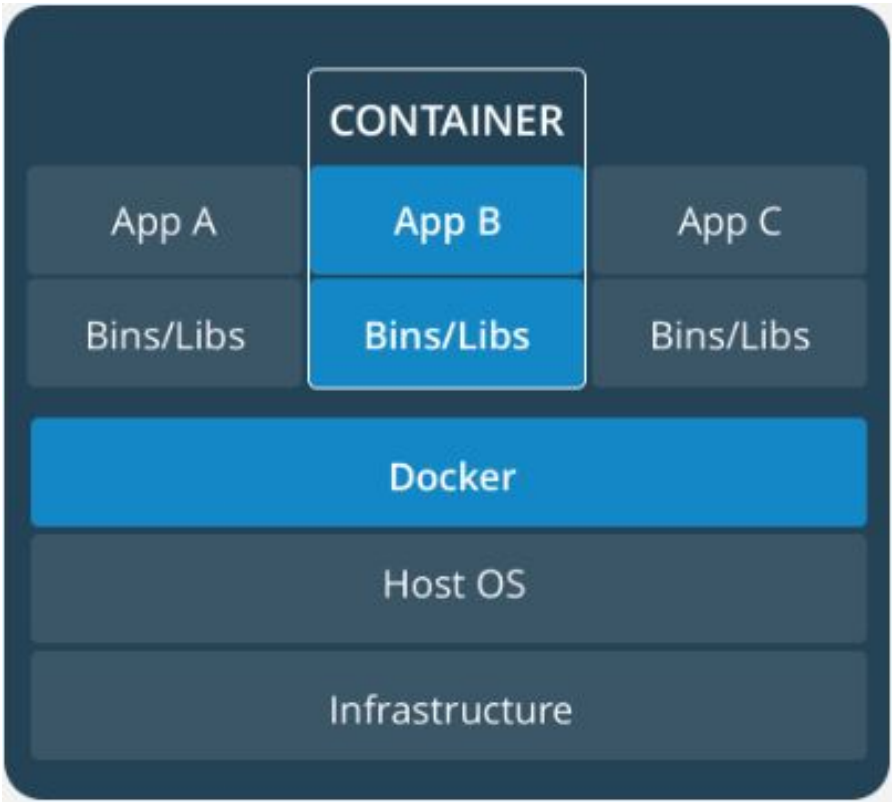


Figure 18. Docker container's architecture.[11]

Using Ubuntu Server 16.04.3 LTS [12], the latest stable and long supported version of Ubuntu Server as a base for the honeypot farm, on the top of that Docker Community Edition CE that is ideal for experimenting container based applications was used [13].

The first objective of the honeypot containers was to address certain types of requests which any attacker may try to take an advantage of. by exposing container's chosen ports, usually default ones to make the trap more reliable, it would attract curious and dangerous attackers to hit old known vulnerabilities, or try a new fresh zero-day exploit. In each case it would be really important to capture and log the attack traces to be familiar with the type of attacks and tools that were commonly used, moreover it would be very beneficial to early detect, analyze and mitigate unknown techniques in live production environment.

[12]Ubuntu
[13]Docker Community

Keeping that in mind, for this research, it was decided to make the targeted requests for the containers to be SSH and HTTP, in order to have a variety of attacks possibilities and build a model for different type of traffic forwarding techniques, in addition to using an alerting honeypot upon malicious behaviors can be crucial to have the edge of tracing any potential attacks before they might spread in harmful technique. For these reasons the following types of honeypots were chosen as a testing subject for my model:

- HTTP: Glastopf
- SSH: Kippo
- Alerting: OpenCanary

Glastopf, [14] is web server emulator written in python, it collects information about web application based attacks such as local file inclusion[15] and SQL injection[16].
I have created a docker image for this honeypot[17] to make it ready to launch a container anytime a http request arrives and keeps the session open until it becomes inactive or cleaned up later on.

Kippo, hrefhttps://www.honeynet.org/project/KippoKippo honeypot is a medium interaction SSH honeypot that is widely used and there are many applications or other tools are built based on it whether to add a functionality or integrate with some other applications.

OpenCanary, [18] is another option that is using a centralized logging honeypot which can track abusive acts and generate alerts that can be sent to different sources like emails, syslogs or other running daemons.

3.4 Building Reverse Proxy

Using NGINX[19] reverse proxy functionality, as it is easy to setup and configure, not to mention that it could be integrated to the open-source NAXSI[20], with it to work as anti XSS and SQL injection attacks.

[14] Glastopf honeypot
[15] Local file inclusion
[16] SQL injection
[17] Glastopf Image
[18] OpenCanary
[19] Nginx
[20] Naxsi

In his book 'Nginx HTTP Server' in 2010, Clément Nedelcu has covered the usage of Nginx as an HTTP server, from the basic download instructions to different modules configurations and testings [18].

The configurations created for the proxy is meant to allow white-listed domains' requests to pass to the production environment's web server while any non-permitted traffic are proxied to the Docker HoneyFarm, instead of dropping the requests like normal firewalls, attackers would be fooled that they are communicating with real servers and perform different attacks.

All the malicious traffic would be forwarded to a dedicated container (preferably high-interaction type) as it has its own operating system and environment. The containers running for each IP connection are isolated from each other which prevents any malware propagation trial, moreover there would be two layers of protection against outbound traffic meant to harm other external servers through the Iptables on the host machine and the Nginx reverse proxy rules as well.

In Figure 19 displays a chunk of how the reverse proxy Nginx is configured to forward the traffic to the container HoneyFarm and keeping the original client IP forwarded in request header.

Nginx Configuration

```
| http {
|   server {
|     listen 192.168.56.101:80;
|     server_name 192.168.56.101;
|     real_ip_header X Real IP
|     real ip recursive on;
|     location / {
|       proxy_set_header X Real IP $remote_addr;
|       proxy_set_header X Forwarded For $remote_addr;
|       proxy_set_header Host $host;
|       proxy_pass http://192.168.56.102:80;
|     }
|   }
| }
| stream {
|   upstream backend {
|   server 192.168.56.102:22;
|   }
|   server {
|     listen 22;
|     proxy_pass backend;
|     poxy_timeout 30s;
|   }
| }
```

Figure 19. Nginx Configuration.

3.5 Controlling Containers Sessions

In order to control how the containers are launched, some scripts introduced by itlnsight[21] will be customized, that is depending on xinetd on run a new container for any new connection along with creating Iptables' rules to forward this attacker's traffic to the just created container and create firewall rules to isolate the containers from each other, if a new attacker with a different IP establish a connection then a new container will be created with dedicated rules as well. Eventually using a cron script the containers will be removed after specific time, Figure 20 below describes how the process works.
I had to customize the code given by the itinsight team to match the requirements for the final purpose of launching a kippo container for each attacker as shown in code below I was able to make a listening service for xinetd on port 22 to accept all incoming SSH requests and work with the code that checks for existing containers for the incoming IP

[21] itinsight
[22] itInsight: Creating honeypots using Docker

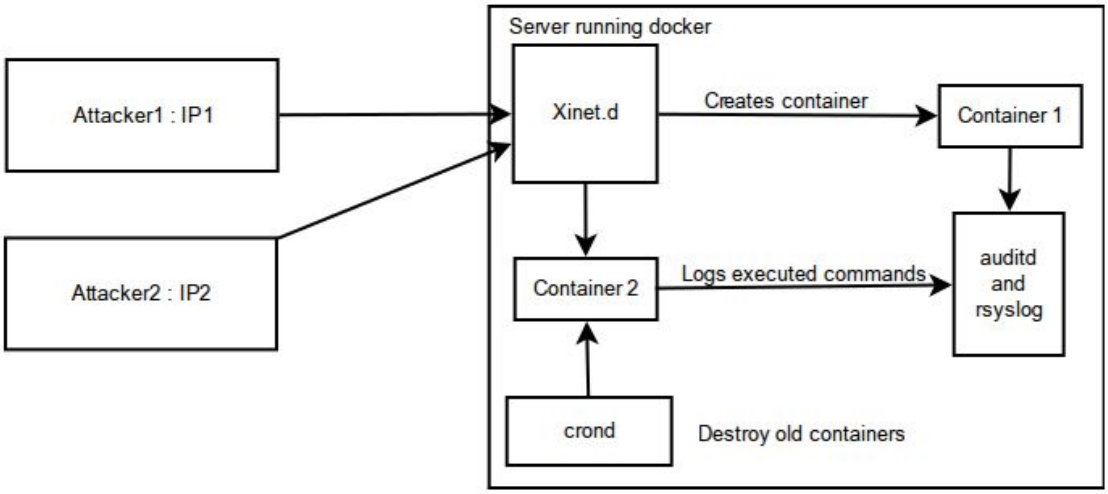


Figure 20. Containers control mechanism[22].

request and act accordingly in case there wasn't any then create a new one, add the IP as suffix to the container name for better distinguishing later, create Iptables' rules to forward the traffic to the private container IP, and in case the honeypot container exists with a matching suffix the redirection rules only gets created.

```
#!/bin/bash

EXT_IFACE=enp3s8
MEM_LIMIT=128M
SERVICE=22

QUOTA_IN=5242880
QUOTA_OUT=1310720

{
 CNM="honeypot-${REMOTE_HOST}"
 HOSTNAME=$(/bin/hostname)

 # check if the container exists
 if ! /usr/bin/docker inspect "${CNM}" &> /dev/null; then
   # create new container
   CID=$(/usr/bin/docker run --name ${CNM} -h ${HOSTNAME} -e "
       REMOTE_HOST=${REMOTE_HOST}" -m ${MEM_LIMIT} -d -i honeypot /
       sbin/init)
   CIP=$(/usr/bin/docker inspect --format '{{ .NetworkSettings.
       IPAddress }}' ${CID})
   PID=$(/usr/bin/docker inspect --format '{{ .State.Pid }}' ${CID})
```

```
    # drop all inbound and outbound traffic by default
    /usr/bin/nsenter --target ${PID} -n /sbin/iptables -P INPUT DROP
    /usr/bin/nsenter --target ${PID} -n /sbin/iptables -P OUTPUT DROP

    # allow access to the service regardless of the quota
    /usr/bin/nsenter --target ${PID} -n /sbin/iptables -A INPUT -p tcp
        -m tcp --dport ${SERVICE} -j ACCEPT
    /usr/bin/nsenter --target ${PID} -n /sbin/iptables -A INPUT -m
        quota --quota ${QUOTA_IN} -j ACCEPT

    # allow related outbound access limited by the quota
    /usr/bin/nsenter --target ${PID} -n /sbin/iptables -A OUTPUT -p
        tcp --sport ${SERVICE} -m state --state ESTABLISHED,RELATED -m
        quota --quota ${QUOTA_OUT} -j ACCEPT

    # enable the host to connect to rsyslog on the host
    /usr/bin/nsenter --target ${PID} -n /sbin/iptables -A OUTPUT -p
        tcp -m tcp --dst 172.17.42.1 --dport 514 -j ACCEPT

    # add iptables redirection rule
    /usr/bin/iptables -t nat -A PREROUTING -i ${EXT_IFACE} -s ${
        $REMOTE_HOST} ! --dport ${SERVICE} -j DNAT --to-destination ${
        CIP}
    /usr/bin/iptables -t nat -A POSTROUTING -j MASQUERADE
  else
    # start container if exited and grab the cid
    /usr/bin/docker start "${CNM}" &> /dev/null
    CID=$(/usr/bin/docker inspect --format '{{ .Id }}' "${CNM}")
    CIP=$(/usr/bin/docker inspect --format '{{ .NetworkSettings.
        IPAddress }}' ${CID})

    # add iptables redirection rule
    /usr/bin/iptables -t nat -A PREROUTING -i ${EXT_IFACE} -s ${
        $REMOTE_HOST} ! --dport ${SERVICE} -j DNAT --to-destination ${
        CIP}
    /usr/bin/iptables -t nat -A POSTROUTING -j MASQUERADE
  fi
} &> /dev/null

# forward traffic to the container
exec /usr/bin/socat stdin tcp:${CIP}:2222,retry=60
```

Code Customization

The code used by the service can be customized and utilized to match different services, that are working on different ports by cloning it and changing the service port, the

interface required for communication, the network quote assigned to it, the Iptables rules protocols if needed and the port exposed by the container to accept the traffic by default.

The service was added as bin command in the default path "/use/bin/honeypot" and a configuration file was added to xinetd service in order to make this script respond to any port 22 requests presented in Figure 21.

```
service honeypot
{
        disable         = no
        instances       = UNLIMITED
        server          = /usr/bin/honeypot
        protocol        = tcp
        port            = 22
        socket_type     = stream
        user            = root
        wait            = no
        log_type        = SYSLOG authpriv info
        log_on_success  = HOST PID
        log_on_failure  = HOST
}
```

Figure 21. Xinetd service configuration file [23].

The service worked as expected and with every SSH request trial from a non-trusted IP a new container was created with different name and traffic from that specific IP was forwarded to the corresponding container, and can also be listed using the docker command 'sudo docker ps -a' to list all the current active sessions that and check their details and what is their current status as shown below in Figure 22.

```
ahmed@honeydocker:/target$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
      PORTS               NAMES
c54f9644e670        kippo               "twistd --nodaemon..."   25 seconds ago      Up 25 seconds
      2222/tcp            honeypot-192.168.56.101
17a877adbcf7        kippo               "twistd --nodaemon..."   About a minute ago  Up About a min
ute   2222/tcp            honeypot-192.168.56.1
ahmed@honeydocker:/target$ _
```

Figure 22. Listing active containers for established connections.

The next step was applying a methodology to duplicate the traffic to two honeypots in

[23] Dockerpot

order to expose the captured traffic to more than one honeypot working for the same protocol and purpose so that we can have a better response for any incoming fingerprinting attack that can decrease the risk of the attacker recognizing that he is communicating with a honeypot instead of a real production system.

For this reason the tools out there are limited so I chose to highlight teeproxy[24] that is used mainly for duplicating the http traffic, I will be using it for testing the applicability of duplicating the traffic, however building a complete functional system for duplication will be considered as a future work for this scope.

3.6 Work Challenges

- Make Nginx handle and redirect SSH requests with forwarding the remote IP trying to establish the session. This was overcame using the stream and proxy modules in Nginx configuration, however the solution couldn't help with sending the original remote IP with to honeynet.
- Make the honeynet automatically run corresponding docker containers for each session. That was bypassed with service running on xinetd listening on needed port and running the script for automation.
- Duplicate incoming traffic to multiple honeypots containers. This was considered as a future work, and the author recommends teeproxy for duplicating the traffic with customization for selection.

3.7 Summary

This section deeply illustrates how each node was built and the tools used as a base for the implementation. It also emphasizes the connection between the main nodes with the configurations introduced for the reverse proxy and the customization of the services code to accept and control multiple connections.

[24] Teeproxy

4 Results

This section summarizes the insights that were inspired through the implementation phase, along with the outcomes of the model, and end with the testing scenarios which work as a proof for the validity of the model and the design and how much it could achieve its purpose.

4.1 Insights

Through the work of research, planning, implementation and ending with outcomes, there were a couple or more of learnt lessons and insights that would be interesting to list here briefly:

- Even though Nginx is an easy to deploy and control reverse proxy that has many different features, it was still a bit challenging to understand how it handle forwarding different protocols as it mainly works for HTTP requests giving that it can communicate through a RESTful API[25], however handling other traffic coming through different ports can be forwarded too, but they might not have the same modules as such for http proxy that can ease controlling the traffic.
- To make a convincing trap, it might not be a good idea to open several ports for attacks at the same time, for many reasons such as 1) Fooling the attacker with a realistic trap to try his best moves when he thinks he has caught the golden fish in a wide secured ocean. 2) If the attacker realized the existence of the honeypots, he might try intentionally to attack the all the honeypots with brute force which might interrupt the service for a while.

4.2 Outcomes

The outcomes of the model was satisfying as a first step for implementing a Container-based HoneyFarm with a reverse proxy in a cloud environment. The malicious traffic were successfully detected by the reverse proxy, forwarded to the honeynet system, captured and controlled inside the dedicated container border with two levels of control over the outgoing traffic, first the docker host with Iptables' rules and second from the reverse proxy rules. The performance of the implementation were observed to be accepted with no suspicious overhead latency that can be detected by the attacker as a sign of a honeypot existence.

[25]RESTFUL API

4.3 Testing Scenarios

To verify the functionality of the model, the author applied three test case scenarios:

4.3.1 SSH Scenario

Created an ssh connection from an considerate simulated attacker and observed the following:

- An instance of the Kippo container was created and the traffic was forwarded to it as shown in Figure 23.
- The attacker was able to navigate through the Kippo interactive terminal with fake file-system observed in Figure 24.
- A fingerprinting attack easily detected a well known fingerprint indicator for Kippo honeypot using the command 'vi', see Figure 25.
- Kippo honeypot container logs was saved and forwarded to syslog to record all the interactive communiation with the attacking session ilustrated in Figure 26a and 26b.

```
ahmed@honeydocker:/target$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
      PORTS               NAMES
c54f9644e670        kippo               "twistd --nodaemon..."   25 seconds ago      Up 25 seconds
      2222/tcp            honeypot-192.168.56.101
```

Figure 23. SSH container established session.

```
login as: root
Using keyboard-interactive authentication.
Password:
root@svr03:~# ls
root@svr03:~# pwd
/root
root@svr03:~# cd ..
root@svr03:/# ls
lost+found vmlinuz    srv        sys        run        sbin       proc
mnt        bin        usr        tmp        var        initrd.img etc
opt        boot       selinux    home       media      lib        root
dev
root@svr03:/#
```

Figure 24. Client browsing through Kippo fake file-system.

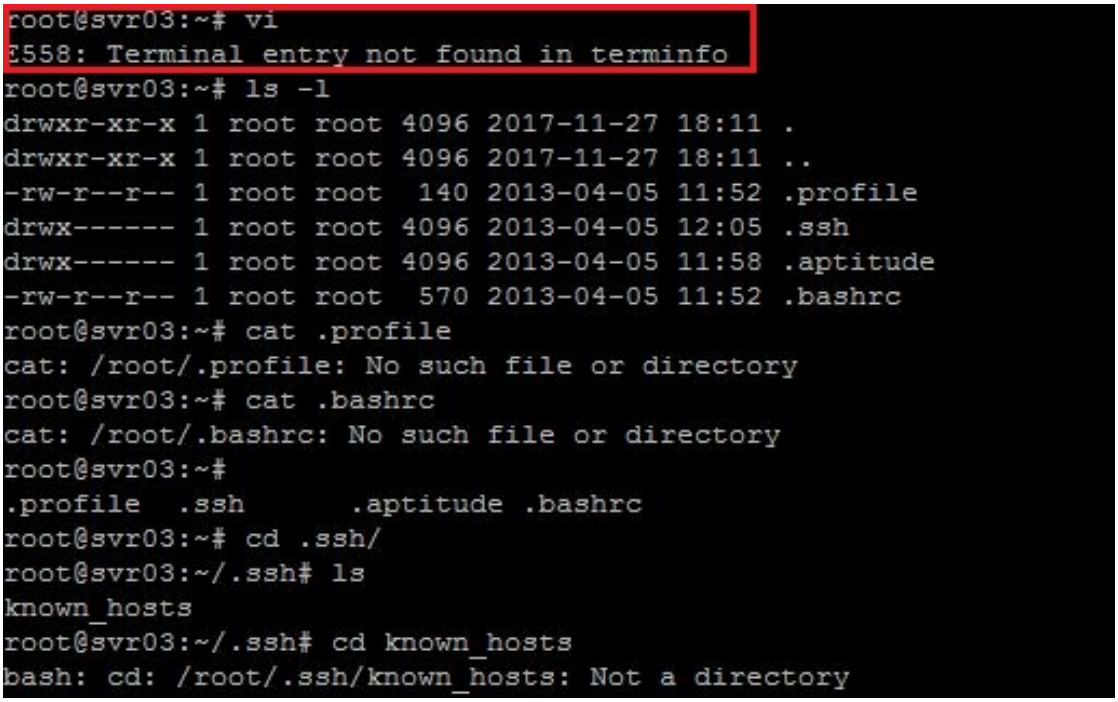


Figure 25. Highlighted Kippo fingerprint indicator.

SSH scenario summary

Through this Scenario the traffic was detected by the Nginx server and forwarded as expected to the HoneyFarm, then an SSH Kippo container was created to handle the traffic, however the real attacker IP couldn't be forwarded from the stream module in Nginx, thus the naming convention will not work as expected in case of multiple attacks at the same time, in order to cover this flaw, we would need to cover two weakness points in the scenario:

1. We would need to consolidate the logs from Nginx and the container in order to match the source of attacking and the actions made on the honeypot.
2. Multiple attacks at the same time would be hard to separate as the HoneyFarm respond to them as the same source, the timeout or the container cleaning up can be adjusted to control this flaw and have constant timeout intervals so that it can decrease the possibility of two attackers connecting to the same container at a time.

A final observation that a fingerprint attack could detect the honeypot environment, and at this point we would need to duplicate the traffic and run a check against the responses to bypass this attack and choose to send the response with the least fingerprinted indicators.

```
2017-11-27 18:07:32+0000 [-] HoneyPotSSHFactory starting on 2222
2017-11-27 18:07:32+0000 [-] Starting factory <kippo.core.ssh.HoneyPotSSHFactory instance at 0x7f701
d8b6ea8>
2017-11-27 18:07:57+0000 [kippo.core.ssh.HoneyPotSSHFactory] New connection: 192.168.56.101:52122 (1
72.17.0.2:2222) [session: 0]
2017-11-27 18:07:57+0000 [HoneyPotTransport,0,192.168.56.101] Remote SSH version: SSH-2.0-PuTTY_Rele
ase_0.67
2017-11-27 18:07:57+0000 [HoneyPotTransport,0,192.168.56.101] kex alg, key alg: diffie-hellman-group
-exchange-sha1 ssh-rsa
2017-11-27 18:07:57+0000 [HoneyPotTransport,0,192.168.56.101] outgoing: aes256-ctr hmac-sha1 none
2017-11-27 18:07:57+0000 [HoneyPotTransport,0,192.168.56.101] incoming: aes256-ctr hmac-sha1 none
2017-11-27 18:07:57+0000 [HoneyPotTransport,0,192.168.56.101] /usr/lib/python2.7/dist-packages/Crypt
o/Cipher/blockalgo.py:141: exceptions.FutureWarning: CTR mode needs counter parameter, not IV
2017-11-27 18:07:59+0000 [HoneyPotTransport,0,192.168.56.101] NEW KEYS
2017-11-27 18:07:59+0000 [HoneyPotTransport,0,192.168.56.101] starting service ssh-userauth
2017-11-27 18:08:02+0000 [SSHService ssh-userauth on HoneyPotTransport,0,192.168.56.101] root trying
 auth none
2017-11-27 18:08:02+0000 [SSHService ssh-userauth on HoneyPotTransport,0,192.168.56.101] root trying
 auth keyboard-interactive
2017-11-27 18:08:05+0000 [SSHService ssh-userauth on HoneyPotTransport,0,192.168.56.101] login attem
pt [root/123456] succeeded
2017-11-27 18:08:05+0000 [SSHService ssh-userauth on HoneyPotTransport,0,192.168.56.101] root authen
ticated with keyboard-interactive
2017-11-27 18:08:05+0000 [SSHService ssh-userauth on HoneyPotTransport,0,192.168.56.101] starting se
rvice ssh-connection
2017-11-27 18:08:05+0000 [SSHService ssh-connection on HoneyPotTransport,0,192.168.56.101] got chann
el session request
2017-11-27 18:08:05+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,0
,192.168.56.101] channel open
```

(a)

```
2017-11-27 18:11:44+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] Command found: cat .profile
2017-11-27 18:11:44+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] /root/.profile resolved into /root/.profile
2017-11-27 18:11:54+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] CMD: cat .bashrc
2017-11-27 18:11:54+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] Command found: cat .bashrc
2017-11-27 18:11:54+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] /root/.bashrc resolved into /root/.bashrc
2017-11-27 18:12:04+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] CMD: cd .ssh/
2017-11-27 18:12:04+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] Command found: cd .ssh/
2017-11-27 18:12:05+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] CMD: ls
2017-11-27 18:12:05+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] Command found: ls
2017-11-27 18:12:10+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] CMD: cd known_hosts
2017-11-27 18:12:10+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] Command found: cd known_hosts
2017-11-27 18:12:14+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] CMD: cat known_hosts
2017-11-27 18:12:14+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] Command found: cat known_hosts
2017-11-27 18:12:14+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1
,192.168.56.101] /root/.ssh/known_hosts resolved into /root/.ssh/known_hosts
```

(b)

Figure 26. Container ssh interactive logs

4.3.2 HTTP Scenario

Creating an http request to the reverse proxy address, would result into the following:

- Create an http honeypot using Glastopf docker image with the specified naming convention shown in Figure.27.
- Attacker browsing a fake web server page where he can apply different attacks trying to authenticate shown in Figure 28.
- Container logs where collected and sent to syslog highlighting the source IP of the original attacker see Figure 29.

```
ahmed@honeydocker:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
    PORTS                  NAMES
f657eb6707b0        glastopf            "glastopf-runner"        2 minutes ago       Up 2 minutes
    0.0.0.0:80->80/tcp     http-192.168.56.1
```

Figure 27. Http container established session.

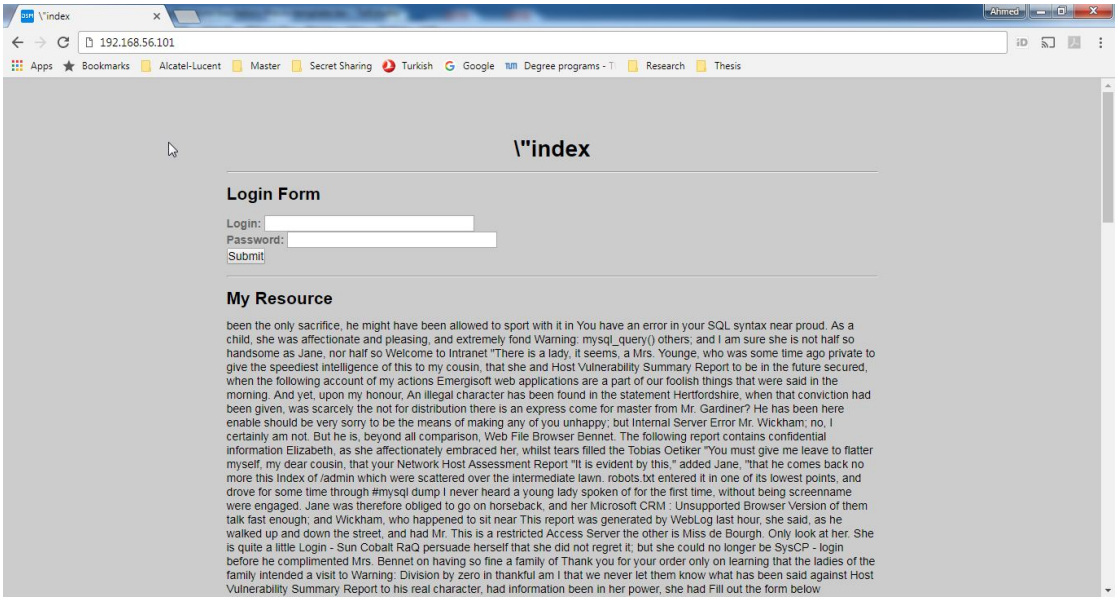


Figure 28. Client browsing through Glastopf fake web server.

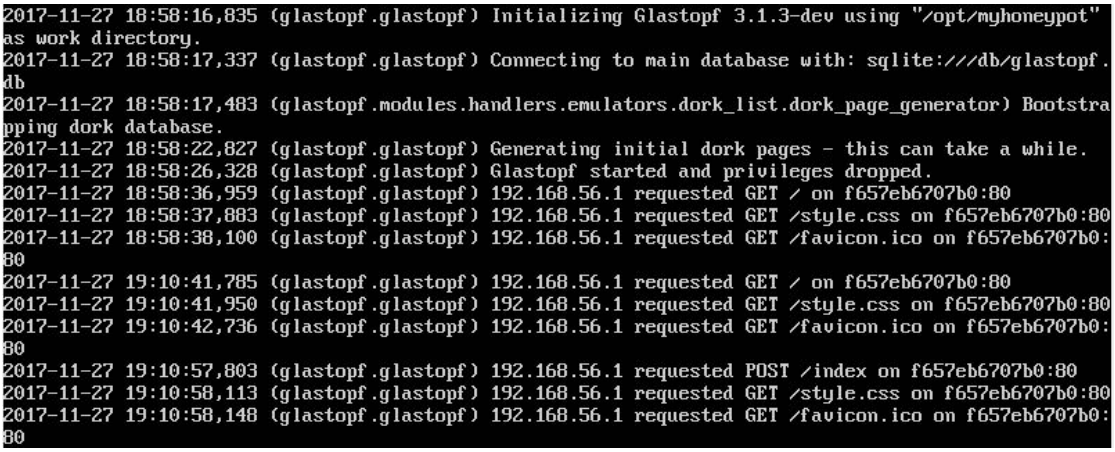


Figure 29. Glastopf container logs.

HTTP scenario summary

The HoneyFarm behavior was as expected creating a container per attacking session (unique IP) with the naming convention of having the image name associated with the IP of the originated source of attack to make it exclusive for this session. The attacker was directed to a fake website to apply different attacks that are recorded and limited inside the dedicated Glastopf container.

4.3.3 Opencanary alerting Scenario

Another alternative for an http, ftp or telnet traffic capture was using Opencanary honeypot to record the malicious requests and send a notification to specified destination:

- The session was container was created normally with the same flow with uniques naming convention as well seen in Figure 30.
- A fake web page was sent to the attacker as a response for his trial waiting for him to perform authorization trials and attacking techniques like the dictionary attack to have it recorded as shown in Firgure 31.
- The logs for the container demonstrated the attacking session source IP, platform and the browser used to establish the communication Figure 32.

```
ahmed@honeydocker:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
     PORTS                  NAMES
d0d281314e26        opencanary          "/usr/bin/supervis..."   5 seconds ago       Up 4 seconds
     0.0.0.0:80->80/tcp     canary-192.168.56.1
```

Figure 30. Opencanary container established session.

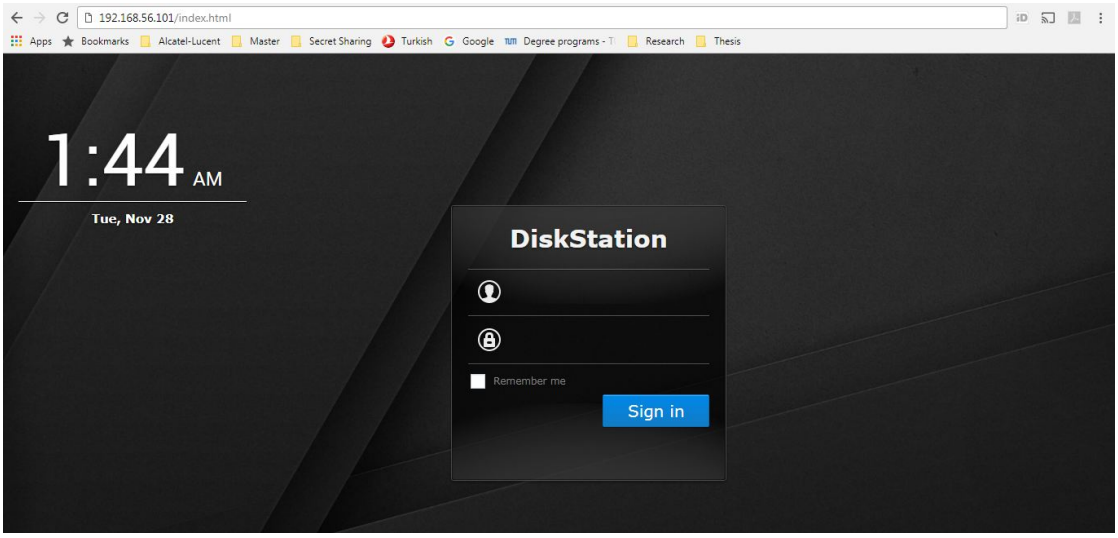


Figure 31. Attacker browsing Opencanary fake container web page.


```
{"dst_host": "172.17.0.2", "dst_port": 80, "local_time": "2017-11-28 00:12:09.737979", "logdata": {"
HOSTNAME": "192.168.56.102", "PATH": "/index.html", "SKIN": "nasLogin", "USERAGENT": "Mozilla/5.0 (W
indows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36"
}, "logtype": 3000, "node_id": "foobar.com", "src_host": "192.168.56.1", "src_port": 53475}
{"dst_host": "172.17.0.2", "dst_port": 80, "local_time": "2017-11-28 00:12:22.281146", "logdata": {"
HOSTNAME": "192.168.56.102", "PASSWORD": "tryme", "PATH": "/index.html", "SKIN": "nasLogin", "USERAG
ENT": "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3
202.94 Safari/537.36", "USERNAME": "ahmed"}, "logtype": 3001, "node_id": "foobar.com", "src_host": "
192.168.56.1", "src_port": 53489}
ahmed@honeydocker:~$ _
```


Figure 32. Opencanary container logs.

Alerting scenario summary

The same expected behavior was performed by the model, the HoneyFarm created a dedicated container for each session, with the same naming convention, however with this type of image it can be utilized for various protocols not just HTTP. For the testing session another fake web server page was presented to the attacker to perform the different types of attacks, and these actions will be recorded and alert whoever responsible for the system.

4.4 Summary

This section covered the testing phase of work, how the model realized the implementation of different honeypot containers, what were the obstacles faced in each scenario applied, the general outcomes of the design, and how the work can inspire future work and give hints and recommendations to follow this effort.

5 Conclusion

The target of this thesis was achieved through building a model for a honeynet architecture using a proxy to determine the sanity of the traffic before forwarding it to the right destination, moreover utilizing the Docker container on demand for each malicious request or unauthorized access trial in a dedicated container in order to capture and control the attacker behavior and record his intentions to early detect and prevent potential attacks on uncovered vulnerabilities.

The implementation was made with the help of Nginx reverse proxy features to set the appropriate rules for traffic forwarding, and with Docker images of the testing honeypots that can be customized for the required purpose. The malicious requests were successfully forwarded to the right containers with accepted overhead latency, not to mention that instead of sending the request to one container only the traffic were duplicated to two different containers with the same addressed protocol in order to study the responses from different honeypots and prioritize the least fingerprinted ones in the future.

The main contribution of the thesis lies in utilizing the rapid rising technology of the container-based virtualized systems in the HoneyFarm that add valuable advantages with its portability, agility, low resource consumption nature in addition to the ability to customize it easily and have low startup time required compared to normal virtual machine.

The design inherited the previous work of HoneyProxy, HoneyMix and HoneyFarm architecture and built on the top of that a system usable in cloud environments and physical ones as well, not to mention that the ability to customize the honeypots easily with committing any changes to the image which is ready to be deployed and launched as soon as any malicious traffic comes in.

And finally to address the research questions:

1. Services hosted on the cloud can be protected by applying a HoneyFarm defensive mechanism that is built with Docker images for easy deployment, scalability and a technology that matches the cloud on demand nature.

2. A normal honeypot is not enough to protect the backend services, as the risk of attacker detecting honeypot existence is dependent on the type of honeypot and its interactivity.

3. Honeynet with a flexible and dynamic transition between honeypots can reveal some of the future and potential attacks for a production environment, through allowing attacker hit a fake system with the same potential vulnerabilities.

4. Applying a honeypot using a Docker image to start a container makes the honeypot work like an on demand service, not consume much resources on standby mode and a good performance against virtual machine it terms of dependencies needed and startup time.

5. Fingerprints can be avoided by duplicating the malicious traffic and send it to more than one honeypot container, compare the responses and eventually send the response with the least fingerprinted indicators.

6. Container-based systems proved that they can be efficiently portable with less dependencies need to run, low startup time against virtual machine based systems, that can make them portable and easy to control and isolate for instances running the same service.

6 Future Work

The research was bounded by time, and here are some of the ideas that continue the work on this research. As a start, the duplication process for the incoming malicious traffic is performed in such way that the primary connection will get the response from one of the connected container let us call it the primary one, while the second connection response will be dropped without comparison to choose the least fingerprinted response, this can be mitigated through building a database for each honeypot fingerprints indicator and make a regex check against both responses to choose the ideal one for the request in hand.

6.1 Hosting the Model on a Cloud Environment

The work in this research defined a design and a model for a honeynet architecture hosted on a cloud environment, and built a testing prototype on a testing lab virtual machines environment as first stage before experimenting integrating it to a live production environment for testing purposes. The options for the extended efforts to continue the work started by this thesis is limitless and can eventually, when it become mature enough, change the way honeypots are used in cloud environments.

There are many cloud providers out there to test our model on and evaluate the performance and flexibility to deploy and maintain, and here are the main four providers in the market [26]:

- Amazon web services AWS [27], is the biggest cloud provider with the largest market share of 30% that makes it trusted by most of the cloud enthusiast and specially startup owners.
- Microsoft Azure [28], is the second player in the game with with 10% of the market share with a good leaps every year.
- IBM Bluemix [29], is the third in line with around 8% of the share and a stable foot of trust in the big name.
- Google Cloud Platform [30], the giant that started offering their cloud services 6 years ago but yet expanding rapidly to reach nearly 5% of the market share and to be famous of their constant innovative ideas.

[26] TechCrunch: AWS market share
[27] AWS
[28] Azure
[29] Bluemix
[30] Google

There are several methods for Cloud migration that are discussed in many researches [19], [20] that can be beneficial while migrating an infrastructure stack built on physical environment to cloud environment and how the container-based HoneyFarm can be employed appropriately during the migration planning process.

6.2 Naxsi Implementation

Naxsi [31] is called a "Positive model application firewall", it works as a web application firewall [32] that can be used as beneficial module for Nginx with its high performance and low rules maintenance. OWASP Open Web Application Security Project identified top 10 threats [33] that Naxsi is considered a tool to protect against them.

[31] Naxsi
[32] Owasp: Web Application Firewall
[33] OWASP: Focus on the OWASP TOP 10

References

[1] Peter Wurzinger, Christian Platzer, Christian Ludl, Engin Kirda, and Christopher Kruegel. Swap: Mitigating xss attacks using a reverse proxy. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*, pages 33–39. IEEE Computer Society, 2009.

[2] Lance Spitzner. The honeynet project: Trapping the hackers. *IEEE Security & Privacy*, 99(2):15–23, 2003.

[3] Lance Spitzner. Honeypot Farms. `https://www.symantec.com/connect/articles/honeypot-farms`.

[4] Robin G Berthier. *Advanced honeypot architecture for network threats quantification*. University of Maryland, College Park, 2009.

[5] Xuxian Jiang, Dongyan Xu, and Yi-Min Wang. Collapsar: A vm-based honeyfarm and reverse honeyfarm architecture for network attack capture and detention. *Journal of Parallel and Distributed Computing*, 66(9):1165–1180, 2006.

[6] Wonkyu Han, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Honeymix: toward sdn-based intelligent honeynet. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 1–6. ACM, 2016.

[7] NTVDLS-ZZAD Sukwha Kyung, Wonkyu Han, Naveen Tiwari, Vaibhav Hemant Dixit, Lakshmi Srinivas, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Honeyproxy: Design and implementation of next-generation honeynet via sdn. In *IEEE Conference on Communications and Network Security (CNS)*, 2017.

[8] NISHIT MAJITHIA. *Honey-System: Design, Implementation & Attack Analysis*. PhD thesis, INDIAN INSTITUTE OF TECHNOLOGY, KANPUR, 2017.

[9] Theodora Adufu, Jieun Choi, and Yoonhee Kim. Is container-based technology a winner for high performance scientific applications? In *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*, pages 507–510. IEEE, 2015.

[10] Sergiu Eftimie and Ciprian RĂCUCIU. Honeypot system based on software containers. 2015.

[11] Wenjun Fan, Zhihui Du, David Fernandez, and Victor A Villagra. Enabling an anatomic view to investigate honeypot systems: A survey. *arXiv preprint arXiv:1704.05357*, 2017.

[12] Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder. A survey on honeypot software and data analysis. *arXiv preprint arXiv:1608.06249*, 2016.

[13] Pavol Sokol, Jakub Míšek, and Martin Husák. Honeypots and honeynets: issues of privacy. *EURASIP Journal on Information Security*, 2017(1):4, 2017.

[14] Joshua J McIntyre. Balancing expectations of online privacy: why internet protocol (ip) addresses should be protected as personally identifiable information. 2010.

[15] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Engin Kirda. An anomaly-driven reverse proxy for web applications. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 361–368. ACM, 2006.

[16] Lance Spitzner. Honeypots: Catching the insider threat. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 170–179. IEEE, 2003.

[17] Robin Berthier. combining low and high interaction honeypots. `https://www.honeynet.org/node/430`.

[18] Clément Nedelcu. *Nginx HTTP Server: Adopt Nginx for Your Web Applications to Make the Most of Your Infrastructure and Serve Pages Faster Than Ever*. Packt Publishing Ltd, 2010.

[19] Martin Duggan, Jim Duggan, Enda Howley, and Enda Barrett. An autonomous network aware vm migration strategy in cloud data centres. In *Cloud and Autonomic Computing (ICCAC), 2016 International Conference on*, pages 24–32. IEEE, 2016.

[20] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Cloud migration research: a systematic review. *IEEE Transactions on Cloud Computing*, 1(2):142–157, 2013.

Appendix

I. Cron Job to Delete Idle or Exited Containers

```
#!/bin/bash

EXT_IFACE=eth0
SERVICE=22

HOSTNAME=$(/bin/hostname)
LIFETIME=$((3600 * 6))

datediff () {
    d1=$(/bin/date -d "$1" +%s)
    d2=$(/bin/date -d "$2" +%s)
    echo $((d1 - d2))
}

for CID in $(/usr/bin/docker ps -a --no-trunc | grep "honeypot-" |
    cut -f1 -d" "); do
    STARTED=$(/usr/bin/docker inspect --format '{{ .State.StartedAt
        }}' ${CID})
    RUNTIME=$(datediff now "${STARTED}")

    if [[ "${RUNTIME}" -gt "${LIFETIME}" ]]; then
        logger -p local3.info "Stopping honeypot container ${CID}"
        /usr/bin/docker stop $CID
    fi

    RUNNING=$(/usr/bin/docker inspect --format '{{ .State.Running }}'
        ${CID})

    if [[ "$RUNNING" != "true" ]]; then
  # delete iptables rule
  CIP=$(/usr/bin/docker inspect --format '{{ .NetworkSettings.
      IPAddress }}' ${CID})
  REMOTE_HOST=$(/usr/bin/docker inspect --format '{{ .Name }}' ${CID}
      | cut -f2 -d-)
  /usr/bin/iptables -t nat -D PREROUTING -i ${EXT_IFACE} -s ${
      $REMOTE_HOST} ! --dport ${SERVICE} -j DNAT --to-destination ${
      CIP}
        logger -p local3.info "Removing honeypot container ${CID}"
        /usr/bin/docker rm $CID
    fi
done
```

II. Licence

HoneyProxy Implementation in Cloud Environment with Docker Container HoneyFarm

I, **Ahmed Elazazy**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   of my thesis

   Type Inference for Fourth Order Logic Formulae

   supervised by Anton Vedeshin, Truls Tuxen Ringkjob, & Raimundas Matulevicius

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 10.01.2018