Faculty of Mathematics and Computer Science

Heidelberg University

Master thesis

in Computer Science

submitted by

Stefan Machmeier

born in Heidelberg

2021

Honeypot Implementation in a Cloud Environment

This Master thesis has been carried out by Stefan Machmeier

at the

Engineering Mathematics and Computing Lab

under the supervision of

Prof. Dr. Vincent Heuveline

Abstract

Nowadays, services which are accessible from the Internet face more attacks than ever. An attackers' objective is to exploit systems and use them for their own vicious purposes. Such efforts are on the rise as faulty systems can be discovered and compromised through Internet-wide scanning. Besides traditional security leverages, one known methodolgy is to learn from those who attacks it. A Honeypot helps to gather information about an attacker. It is a security resource whose value lies in being probed, attacked, or compromised. Thus, it is an interesting question how honeypots can contribute to more secure infrastructure. In this work we will present a honeypot solution to investigate cybercrime activities in heiCLOUD, and show that attacks have increased significantly. We will try to catch attackers in restricted network zones at the Heidelberg University, and we discovered leaks in the firewall allowing attackers to send malicious packets to the network. Moreover, we will consider an attacker's point of view by present a method to detect honeypots on transport level. Lastly, we mitigate these efforts with a customized OpenSSH version.

Zusammenfassung

Heutzutage sind Dienste, die über das Internet zugänglich sind, mehr Angriffen ausgesetzt als je zuvor. Das Ziel von Angreifern ist es, Systeme auszunutzen und sie für ihre eigenen bösartigen Zwecke zu verwenden. Derartige Bemühungen nehmen zu, da fehlerhafte Systeme durch internetweites Scannen entdeckt und kompromittiert werden können. Neben den traditionellen Sicherheitsmaßnahmen besteht auch die Möglichkeit, von den Angreifern zu lernen. Ein Honeypot hilft dabei, Informationen über Angreifer zu sammeln. Er ist eine Sicherheitsressource, deren Wert darin liegt, dass sie untersucht, angegriffen oder kompromittiert wird. Daher ist es eine interessante Frage, wie Honeypots zu einer sichereren Infrastruktur beitragen können. In dieser Arbeit werden wir eine Honeypot-Lösung zur Untersuchung von Cybercrime-Aktivitäten in der heiCLOUD vorstellen und zeigen, dass die Angriffe aus dem Internet erheblich zugenommen haben. Des Weiteren werden wir versuchen, Angreifer in eingeschränkten Netzwerkzonen der Universität Heidelberg zu entdecken. Wir werden zeigen, dass die Firewall lücken aufweist und Angreifer in der Lage waren gewisse Bereiche zu scannen. Zusätzlich werden wir die Sichtweise eines Angreifers einnehmen und eine Methode zur Erkennung von Honeypots auf Transportebene vorstellen. Abschließend stellen wir ein Konzept zur Entschärfung dieser Maßnahmen vor.

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den (Datum) ....

Acknowledgements

First and foremost, I am deeply grateful to my supervisor Prof. Dr. Vincent Heuveline for his valuable and constructive input. Without his guidance and mentorship I would have not been able to finish this thesis. Moreover, I was able to grow as a researcher, and I am immensely grateful for the oppurtunity to continue my research as a future PhD candidate under his supervision.

I would like to thank Stefan Steiger, and Olaf Pichler from the Computing Centre at the Heidelberg University for their invaluable advice at crucial times and their sincere interest in my topic. No matter what happened, they always had time and provided me ample support.

I am indebted to Joachim Peek for his continuous support . His helped me to understand the quirks, and pushed my

Last but not least, I would like to thank my girlfriend Carmen for

Contents

Acronyms

ACL Access Control List

ADB Android Debug Bridge

ADC Application Delivery Controller

ASA Adaptive Security Appliance

AWS Amazon Web Services

BelWÜ Baden-Württembergs extended LAN

BPP Binary Packet Protocol

CERT Computer Emergency Response Team

CHARGEN Character Generator Protocol

CVE Common Vulnerabilities and Exposures

DaaS Data-as-a-Service

DDoS Distributed Denial of Service

DICOM Digital Imaging and Communications in Medicine

DMZ demilitarized zone

DNS Domain Name System

DTK Deception Toolkit

EU European Union

FHIR Fast Healthcare Interoperability Resources

GCP Google Cloud Platform

HaaS Hardware-as-a-Service

HTTP Hypertext Transfer Protocol

IaaS Infrastructure-as-a-Service

ICS Industrial control systems

IDENT Identification Protocol

IDS intrusion detection system

IOCTA Internet Organised Crime Threat Assessment

IPD intrusion prevention system

IPP Internet Printing Protocol

MAC Message Authentication Code

NIST National Institute of Standards and Technology

NSM network security monitoring

NTP Network Time Protocol

OS operating system

PaaS Platform-as-a-Service

RDP Remote Desktop Protocol

SaaS Software-as-a-Service

SCADA Supervisory Control and Data Acquisition

SNMP Simple Network Management Protocol

SOHO small office/home office

SSDP Simple Service Discovery Protocol

VM virtual machine

VMM virtual machine monitors

VPN virtual private network

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Recently, Europol[1] rose awareness of new cyber threats related to the ongoing pandamic. As stated in their yearly Internet Organised Crime Threat Assessment (IOCTA) report, scanning corporate infrastructures have been skyrocketing within the last 12 months by ransomware groups respectively increasing malware usage. Attackers use scans to find potential vulnerabilities in remote desktop sharing software, or virtual private networks (VPNs) in order to deploy malware and blackmail the company. [27] The rapid increase dates back to the pandamic and the shift to home office, forcing companies to quickly adapt their infrastructure. Such changes come with the downside of adding new threats to an organization. The latest incident at the SRH University Heidelberg points out the obsenity that institutions face when ransomware groups have access and exploit various parts of an infrastructure with malware. An unkown group infected systems with malware, and distributed internal data in the darknet. Such incidents lay emphasis for the rise of cyber criminal activties.

Especially in cloud computing due to access to large pools of data and computational resources, controlling access to services is becoming a tougher challenge nowadays. Besides traditional security leverages such as firewalls or intrusion detection systems, one known methodology to strengthen infrastructures is to learn from those who attacks it. Honeypots are a proper instrument with the objective to gather information about attackers. It is “a security resource whose value lies in being probed, attacked, or compromised” [64]. Unlike production systems, honeypots have to be easy to exploit. Collecting attacks can reveal shell-code exploitation or bot activity. In retrospect, this would help to harden infrastructures before proper damage occurs. As a cloud provider, it is a crucial point if and how attacks on their service can be prevented from happening. Considering the Global Security Report by Trustwave, the amount of attacks doubled in 2019, and increased by 20% in 2020 [1]. Respectively puttting cloud providers to the third most targeted environments for cyberattacks, behind corporate and internal networks.

[1] An agency that fights against terrorism, cybercrime, and other threats [28]

The Heidelberg University offers its own cloud provider, called heiCLOUD. It enables users to easily maintain and control computational resources. Thus, it is interesting to elaborate the value of honeypots for this cloud solution. As a conclusion, this thesis tries to answer the overall research question if honeypots contribute to a more secure infrastructure in the context of a cloud environment. This includes deploying a honeypot solution in heiCLOUD and presenting the results. Prior to that, we will give an insight of a recent study investigating honeypots for the cloud providers AWS, GCP and Microsoft Azure. These findings assist to validate our results. In addition, we focus on the university network and try to find potential leaks in the stateless firewall. Therefore, we outline a concept using a honeypot-like detection tool of the BSI, and deploy it on desktop computers inside a university building. Lastly, we will consider an attacker's point of view by present recent work to detect honeypots on transport level. On the contrary, we draft a solution to mitigate these efforts and present our results.

This thesis is structured in six chapters. Besides the introduction, chapter 2 outlines the background knowledge that is needed to comprehend the upcoming experiments. It gives the reader a profound understanding of cloud computing, honeypots, and virtualization. In chapter 3, *Analyze Honeypot Attacks in the Cloud*, we show the status quo of cybercrime activities in heiCLOUD. In the beginning, we present the results that Kelly et al. [41] claim for AWS, GCP and Microsoft Azure. Next, we give an insight of the T-Pot solution that has been used to collect our data, and show our results after collecting them over a period of three weeks. The chapter 4, *Catching Attackers in Restricted Network Zones*, investigates the university network in which we deployed our new concept over a period of three weeks. We show that we were able to adapt the firewall, thus, improving the network security at the university. Finally, chapter 5, *Mitigate Fingerprint Activities of Honeypots*, presents two experiments. First, we will describe the prilimnary work to detect honeypots, and finish with an experiment to prove this assumption. Next, we draft the counterpart of mitigating this activity, also closing up with an experiment. Lastly, we finish this thesis with our conclusion that summarizes the results and describes future work in this regard.

Chapter 2

Background

> A honeypot is a security resource whose value lies in being probed, attacked, or compromised.
>
> *Lance Spitzner*

Using honeypots in a cloud environment merge two varying principals together. This chapter introduces the fundamental knowledge that is needed to comprehend the upcoming experiments. If the reader has a profound understanding of cloud computing, honeypots, and virtualization he can skip this chapter.

2.1 Virtualization

Virtualization, often referred to virtual machine (VM), is defined by Kreuter [44] as "an abstraction layer or environment between hardware component and the end-user". A VM runs on top of the operating system's (OS's) core components. Through an abstraction layer, the virtual machine is connected with the real machine by hypervisors or virtual machine monitors (VMM). Hypervisors can use real machine hardware components, but also support virtual machine's operating systems and configurations. Both are similar to emulators, that are defined by HA [33] as a "process whereby one computer is set up to permit the execution of programs written for another computer". This allows to manage multiple VMs with real machine resources. There are three different types of virtualization, (i) software virtual machines, (ii) hardware virtual machines, and (iii) virtual OS/containers. Software virtual machines manage interactions between the host operating system and guest operating system. Hardware virtual machines offers direct and fast access to the underlying resources. It uses hypervisors, modified code, or APIs. Lastly, virtual OS/container partitions the host operating system into containers or zones. [21]

2.2 Cloud Computing

Cloud Computing has become a very popular keyword these days. It has been used by various large companies such as Google and Amazon. However, the term "cloud computing" dates back to the late 1996, when a small group of technology executives of Compaq Computer framed new business ideas around the Internet.[55] Starting from 2007 cloud computing evolved into a serious competitor and outnumbered the keywords' "virtualization", and "grid computing" as reported by Google trends [72]. Shortly, various cloud providers become publicly available, each with their own strengths and weaknesses. For example IBM's Cloud[1], Amazon Web Services[2], and Google Cloud[3]. Why are clouds so attractive in practice?

- It offers major advantages in terms of cost and reliability. When demand is needed, consumers do not have to invest in hardware when launching new services. Pay-as-you-go allows flexibility.
- Consumers can easily scale with demand. When more computational resources are required due to more requests, scaling up instances in conjunction with a suited price model are straightforward.
- Geographically distributed capabilities supply the need for world-wide scattered services.

2.2.1 Definition of Cloud Computing

With reference to the definition by Brian Hayes, cloud computing is "a shift in the geography of computation" [34]. Thus, computational workload is moved away from local instances towards services and datacenters that provide the user's needs[3].

The National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [47]. NIST not only reflects the geographical shift of resources such as datacenters, but also mentions on-demand usage that contributes to a flexible resource management. Moreover, NIST composes the term in five essential characteristics, three service models (see subsection 2.2.2), and four deployment models (see subsection 2.2.3) [47]

On-demand-self-service refers to the unilateral provision computing capabilities. Consumers can acquire server time and network storage on demand without a human interaction.

[1] `https://www.ibm.com/cloud`

[2] `https://aws.amazon.com/`

[3] `https://cloud.google.com/`

Broad network access characterizes the access of capabilities of the network through standard protocols such as HTTP. Heterogeneous thin and thick client platforms should be supported.

Resource pooling allows the provider's computing resources to be pooled across several consumers. A multi-tanent model with different physical and virtual resources are assigned on demand. Other aspects such as location are independent and cannot be controlled on a low-level by consumers. Moreover, high-level access to specify continent, state, or datacenter can be available.

Rapid elasticity offers consumers to extend and release capabilities easily. Further automatization to quickly increase resources when demand skyrockets significantly can be supported regardless of the limit and quantity at any time.

Measured service handles resources in an automated and optimized manner. It uses additional metering capabilities to trace storage, processing, bandwidth, and active user accounts. This helps to monitor, and control resource usage. Thus, contributing to transparency between provider and consumer.

2.2.2 Service models

Service models are categorized by NIST into three basic models based on usage and abstraction level. Figure 2.1 shows the connection between each model whereas cloud resource are defined in subsection 2.2.3. Due to the vast range of functionalities, Infrastructure-as-a-Service (IaaS) builds the foundation of service models. Each model on top represents a user-friendly abstraction with derated capabilities.

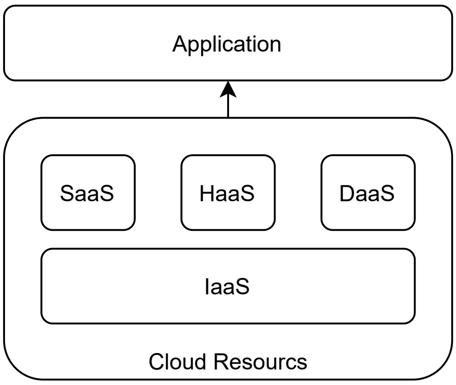


Figure 2.1: Abstract visualization of service models. The lowest level within the container "cloud resources" represents the depth of functionalities. Therefore, Infrastructure-as-a-Service (IaaS) offers the most functionalities whereas the others have a user-friendly abstraction.

Software-as-a-Service (SaaS) is a high-level abstraction to consumers. Controlling the underlying infrastructure is not supported. Providers often use a multi-tenancy system architecture to organize each consumer's application in a separate environment. It helps to employ scaling with respect to speed, security availability, disaster recovery, and maintenance. Main objective of SaaS is to host consumer's software or application that can be accessed over the Internet using either a thin or rich client. [23] "Limited user-specific application configuration settings" can be made [47].

Platform-as-a-Service (PaaS) pivots on the full "Software Lifecycle" of an application whereas SaaS distinct on hosting complete applications. PaaS offers ongoing development and includes programming environment, tools, configuration management, and other services. In addition, the underlying infrastructure is not managed by the consumer. [47]

Infrastructure-as-a-Service (IaaS) offers a low-level abstraction to consumers with the ability to run arbitrary software regardless of the operating system or application. In contrast to SaaS, IT infrastructure capabilities (such as storage, networks) can be used. It strongly depends on virtualization due to integration, or decomposition of physical resources. [47]

Data-as-a-Service (DaaS) serves as a virtualized data storage service on demand. Motivations behind such services could be upfront costs of on-premise enterprise database systems. [23] Mostly they require "dedicated server, software license, post-delivery services, and in-house IT maintenance" [23] whereas DaaS costs solely what consumer's need. When dealing with a tremendous amount of data, file systems and RDBMS often lack in performance. DaaS outruns such weak links by employing a table-style abstraction that can be scaled. [23]

Hardware-as-a-Service (HaaS) offers IT hardware, or datacenters to buy as a pay-as-you-go subscription service. The term dates back to 2006 during a time when hardware virtualization became more powerful. It is flexible, scalable and manageable. [72]

2.2.3 Deployment models

Deployment models are categorized by NIST into four basic models. Each differs in data privacy, location, and manageability [47].

Private clouds offer the highest level of control in regard of data privacy, and utilization. Mostly, such clouds are deployed within in a single organization, either managed by in-house teams or third party suppliers. In addition, it can be on or off premise. Within private clouds, consumers have full control of their data. Especially for European data privacy laws, it is not negligible when data is stored abroad, and thus, under law of foreign countries. However, its popularity has not been diminished due to the immense cost of switching to public clouds. [23, 47]

Community clouds can be seen as a conglomerate of multiple organizations that merge their infrastructure with respect to a commonly defined policy, terms, and condition beforehand. [47]

Public clouds represent the most used deployment models. In contrary, to private one, public clouds are fully owned by the service provider such as business, academics, or government organization. Consumers do not know where their data is distributed. In addition, contracts underlie custom policies. [47]

Hybrid cloud is a mixture of two or more cloud infrastructures, such as private and public cloud. However, each entity keeps its core element. Hybrid clouds defines "standardized or proprietary technology to enables data and application portability"[47].

2.3 Honeypots

The term "honeypot" exists since more than a decade. 1997 was the first time that a free honeypot solution became public. Deception Toolkit (DTK), developed by Fred Cohen, released the first honeypot solution. However, the earliest drafts of honeypots are from 1990/91, and built the foundation for Fred Cohen's DTK. Clifford Stoll's book "The Cuckoo's Egg"[65], and Bill Cheswick's whitepaper "An Evening With Berferd"[7] describe concepts that are considered nowadays as honeypots.[64] A honeypot itself is a security instrument that collects information on buzzing attacks. It disguises itself as a system, or application with weak links, so that it gets exploited and gathers knowledge about the adversary. In 2002 a Solaris honeypot helped to detect an unknown `dtspcd` exploit. Interestingly, a year before in 2001 the Coordination Center of CERT[4] shared their concerns regarding the dtspcd. Communities were aware that the service could be exploited to get access and remotely compromise any Unix system. However, during this time such an exploit was not known, and experts did not expect any in the near future. Gladly, early instances based on honeypot technologies could detect new exploits and avoid further incidents. Such events lay emphasis on the importance of honeypots.

2.3.1 Definition of a Honeypot

A dozen of definitions for honeypots circulate through the web that causes confusion, and misunderstandings. In general, the objective of a honeypot is to gather information about attacks, or attack patterns [50]. Thus, contributing as an additional source of security measure. See subsection 2.3.3 for a detailed view regarding

[4] Computer Emergency Response Team is an expert group that handles computer security incidents[31]

honeypots in the security concept. As Spitzner [64] has listed, most misleading definitions are: honeypot is a tool for deception, it is a weapon to lure adversaries, or a part of an intrusion detection system. In order to get a basic understanding, we want to exhibit some key definitions. Spitzner [64] defines honeypots as a "security resource whose value lies in being probed, attacked, or compromised". Independent of its source (e.g. server, application, or router), we expect that our instance is getting probed, attacked, and eventually exploited. If a honeypot does not match this behaviour, it will not provide any value. It is important to mention that honeypots do not have any production value, thus, any communication that is acquired is suspicious by nature [64]. In addition, Spitzner [64] points out that honeypots are not bounded to solve a single problem, hence, they function as a generic perimeter, and fit into different situations. Such functions are attack detection, capturing automated attacks, or alert/warning generator. Figure 2.2 shows an example how honeypots could be used in an IT infrastructure.

In general, we differentiate two types of honeypots (i) production honeypots (ii) research honeypots. This categorization has their origin from Mark Rosch, a developer of Snort, during his work at GTE Internetworking.

Production honeypots are the common type of honeypots that people would think of. The objective is to protect production environments, and to mitigate the risk of attacks. Normally, production honeypots are easy to deploy within an organization. Mostly, low-interaction honeypots are chosen due to a significant reduce in risk. Thus, adversaries might not be able to exploit honeypots to attack other systems. The downside of a low-interaction honeypot is a lack of information, that means, only standard information like the origin of attacks, or what exploits have been used can be collected, whereas insides about communication of attackers, or deployment of such attacks are unlikely to obtain. In contrast, research honeypots do fulfill this objective. [64]

Research honeypots are used to learn more in detail about attacks. The objective is to collect information about clandestine organizations, new tools for attacks, or the origin of attacks. Research honeypots are unlikely suitable for production environments due to a higher increase of risk. Facing an increase in deployment complexity, and maintenance does not attract a production usage either. [64]

It is worth to mention that there is no exact line between research or production honeypots. Possible cases are honeypots that could function as either a production or a research honeypot. Due to their dynamic range in which they are applicable it makes it hard to distinguish.

In addition, Provos [54] adds a differentiation for the virtual honeypot framework and splits it into the following types:

- Physical honeypots are "real machines on the network with its own IP address" [54]

- Virtual honeypots are “simulated by another machine that responds to network traffic sent to the virtual honeypot” [54]

2.3.2 Level of Interaction

When building and deploying a honeypot, the depth of information has to be defined beforehand. Should it gather unauthorized activities, such as an NMAP scan? Do you want to learn about buzzing tools and tactics? Each depth brings a different level of interaction because some information depends on more actions of adversaries. Therefore, honeypots differ in level of interaction.

Low-interaction honeypots provide the lowest level of interaction between an attacker and a system. Only a small set of services like SSH, Telnet, or FTP are supported which contributes to the deployment time. In terms of risk, a low-interaction honeypot does not give access to the underlying OS which makes it safe to use in a production environment. For example, using an SSH honeypot with emulated services enables attackers to log in by brute force or by guessing, and execute commands. However, the adversary will never gain more access because it is not a real OS. However, safety comes with the downside of less information. Collection is limited for statistical purpose such as (i) time and data of attack (ii) source IP address and source port of the attack (iii) destination IP address and destination port of the attack. Transactional information can not be collected. [64]

A medium-interaction honeypot offers more sophisticated services with higher level of interaction. It is capable to respond to certain activities. For example, a Microsoft IIS Web server honeypot could be able to respond in a way that a worm is expecting it. The worm would get emulated answers, and could be able to interact with it more in detail. Thus, gathering more severe information about the attack, including privilege assessment, toolkit capturing, and command execution. In comparison, medium-interaction honeypots allocate more time to install and configure. Also, more security checks have to be performed due to a higher interaction level than low-interaction honeypots. [64]

High-interaction honeypots represent a real OS to provide a full set of interactions to attackers. They are so powerful because other production servers do not differ much to high-interaction honeypots. They represent real systems in a controlled environment. Obviously, the amount of information is tremendous. It helps to learn about (i) new tools (ii) finding new bugs in the OS (iii) the black hat community. However, the risk of such a honeypot is extremely high. It needs severe deployment and maintenance processes, thus, it is time-consuming.

2.3.3 Security concepts

Security concepts are classified by Schneier [62] in prevention, detection, and reaction. Prevention includes any process that (i) discourages intruders and (ii) hardens systems to avoid any kind of breaches. Detection scrutinizes the identification of attacks that threatens the systems' (i) confidentiality (ii) integrity and (iii) availability. Reaction treats the active part of the security concept. When attacks are detected, it conducts reactive measures to remove the threat. Each part is designed to be sophisticated so that all of them contribute to a secure environment. [50]

Honeypots contribute to the security concept like firewalls, or intrusion detection system (IDS). Regarding prevention, honeypots add only a small value because security breaches cannot be identified. Moreover, attackers would avoid wasting time on honeypots and go straight for production systems instead.

However, detection is one of the strengths of honeypots. Attacks often vanish in the sheer quantity of production activities. If any connection is obtained to a honeypot it is suspicious by nature. In conjunction with an alerting tool, attacks can be detected.

Honeypots strongly supply reaction tools due to their clear data. In production environments, finding attacks for further data analysis are not easy to grasp. Often data submerge with other activities which complicates the process of reaction. [50] Nawrocki et al. [50] distinct honeypots from other objectives such as firewall, or log-monitoring.

Table 2.1: Distinction between security concepts based on areas of operations (derived from [50]).

Objective	**Prevention**	**Detection**	**Reaction**
Honeypot	+	++	+++
Firewall	+++	++	+
Intrusion Detection Sys.	+	+++	+
Intrusion Prevention Sys.	++	+++	++
Anti-Virus	++	++	++
Log-Monitoring	+	++	+
Cybersecurity Standard	+++	+	+

2.3.4 Value of Honeypots

To assess the value of honeypots we want to take a closer look to their advantages and disadvantages. [49, 40, 64]

Advantages

- Data Value: Collected data is often immaculate and does not contain noise from other activities. Thus, reducing the total size of data, and speeding up the analyzation.
- Resources: Firewalls, and IDS are often overwhelmed by the gigabits of traffic, thus, dropping network packets for analyzation. This results in a far less effective detection for malicious network activities. However, honeypots are independent of resources because they only capture their activities at itself. Due to resource limitation, expensive hardware is not needed.
- Simplicity: A honeypot do not require any complex algorithms, or databases. A user should be able to quickly deploy it somewhere. Research honeypots might come with a certain increase of complexity. However, if a honeypot is complex, it will lead to misconfigurations, breakdowns, and failures.
- Return on Investment: Capturing attacks immediately informs users that suspicious activities occur on the infrastructure. This helps to demonstrate their value, and contributes to new investments in other security measurements.

In addition, Nawrocki et al. [50] listed four more advantages of honeypots:

- Independent of Workload: Honeypots only process traffic that is direct to them.
- Zero-Day-Exploit Detection: It helps to detect unknown strategies and zero-day-exploits.
- Flexibility: Well-adjusted honeypots for a variety of specific tasks are available.
- Reduced False Positives and Negatives: Any traffic or connection to a honeypot is suspicious. Client-honeypots verify such attacks based on system state changes. This results in either false positive, or false negatives.

Disadvantages

- Narrow Field of View: Only direct attacks on honeypots can be investigated whereas attacks on production system are not detected.
- Fingerprinting: A honeypot often has a certain fingerprint that can be identified by attackers. Especially commercial ones can be detected by their responses or behaviors.
- Risk to the Environment: Using honeypots in an environment always increase the risk. However, it depends on the level of interaction.

2.3.5 Honeynets

Instead of having single honeypots that can be attacked, a honeynet offers a complete network of standard production systems such as you would find in an organization. Those systems are high-interaction honeypots, thus, allowing to fully interact with the OS, and applications. The key idea is that an adversary can probe, attack, and exploit these systems so that we can derive interaction within this network. It should be mentioned that a honeynet have to be protected by firewalls. Figure 2.3 represents such a honeynet within an organization.

In comparison to a regular honeypot, the greatest value of honeynets is the usage of true production systems. Black hats often do not know that they attack a honeynet, thus, adding value to prevention. However, the downsides are high complexity and maintenance that is needed to keep a honeynet running. [64]

2.3.6 Legal Issues

Considering questions related to legal issues of honeypots can easily exceed this thesis. In this regard, we restrict the study to the country we reside in. Thus, we are only concerned about the European Union (EU) regulations, EU directives, and international agreements. Honeypots collect (i) content data that is used for communication, and (ii) transactional data that is used to establish the connection. Sokol et al. [63] studied the legal conditions for data collection and data retention. They have come to the conclusion that administrators of honeypots have a legal ground of legitimate interest to store and process personal data, such as IP addresses. Moreover, for production honeypots the legitimate interest is to secure services. Regarding the length of data retention, the principle of data minimization has to be considered which means there is no clear answer for it. Any published data of research honeypots needs to be anonymized.

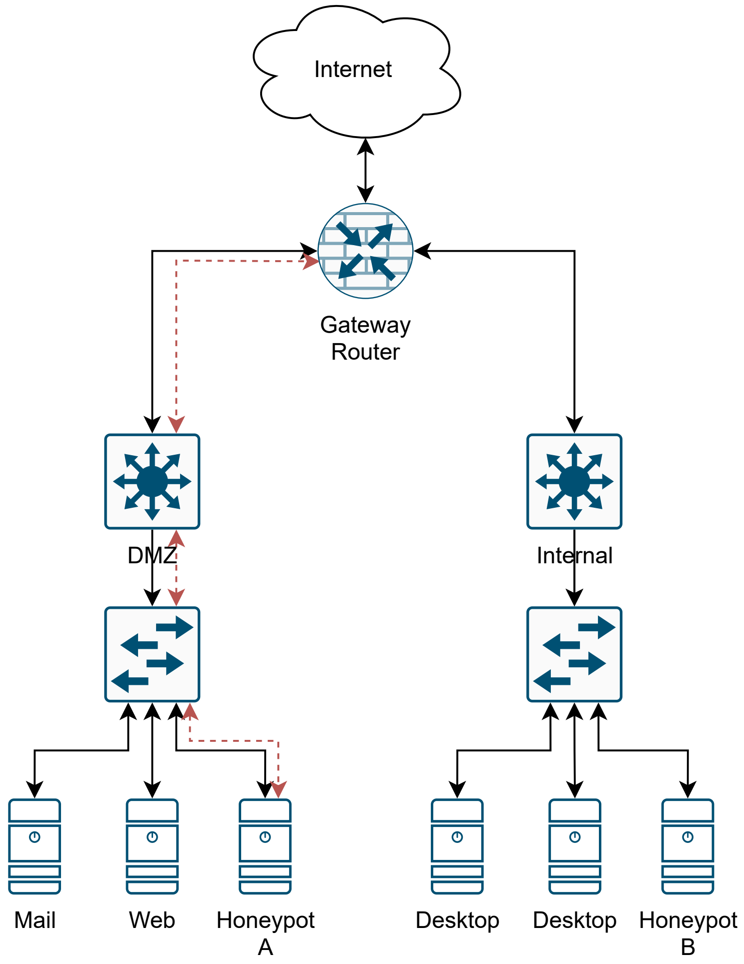


Figure 2.2: Example of honeypots in a simplified network (derived from [64]). Each of the demilitarized zone (DMZ), and internal network are separated by a router and a Layer-3 switch. As derived from above, in each network a honeypot is available (honeypot A, B). The read path symbolizes the path of an attacker coming from the gateway router.

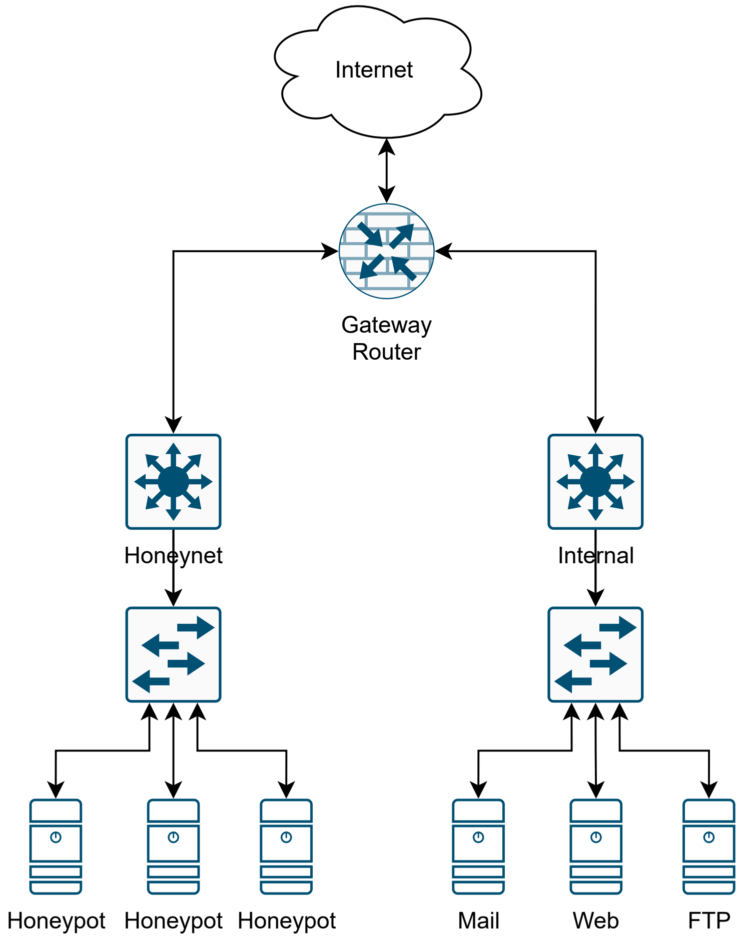


Figure 2.3: Example of honeynets in a simplified network (derived from [64]). On the left, this network presents the honeynet consisting of several other honeypots. On the right the network presents an ordinary subnet consisting of mail, web, and FTP server.

Chapter 3

Analyze Honeypot Attacks in the Cloud

Attacks from the Internet are often originated from bots. A bot, short for "robot", is an automated process that interacts with different network services. Besides good intentions, bots can be used for malicious purposes. Mostly, bots try to self-propagate malware across the Internet and try to capture hosts that merge into a botnet [29]. Recently, Universities in Germany received more cyberattacks than ever, respectively increasing their costs for damage repairs. Honeypots are a good solution to catch attackers and learn from their exploits. However, there is no conclusion if honeypots in times of bots give a proper countermeasure to avoid such damage. Following the rise of cyberattacks, we introduce a method to collect and analyze cyberattacks in a cloud environment. We further propose an answer if honeypots are helpful to detect bot activities.

3.1 Introduction

As previously mentioned in section 2.2, using cloud resources are becoming the go-to option for new services and applications. Kelly et al. [41] thoroughly investigated honeypots on Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP). Followingly, we present their results that we want to compare on with the ones heiCLOUD achieves. The results are collected by T-Pot version 20.06.0 over a duration of 3 weeks. In addition, Kelly et al. [41] considered different server geographical locations. They have collected data from East US, West Europe, and Southeast Asia. Table 3.1 shows the results presented by Kelly et al. [41]. Dionaea (a honeypot to capture malicious payload), Cowrie (SSH and Telnet honeypot), and Conpot (industrial honeypot for ICS, and SCADA) are the most attacked honeypots in comparison to the others. Regarding AWS, Dionaea account 91% of the total attacks, Glutton and Cowrie are minor with 5%, and 2%. Interestingly, Cowrie

Table 3.1: Overview of attacks on cloud providers. For a better overview, only the three most attacked honeypots are listed. The others combine several honeypots.

Provider	Honeypot				In Total
	Dionaea	**Cowrie**	**Glutton**	**others**	
Amazon Web Services	228,075	4,503	11,878	3,688	248,144
Google Cloud Platform	162,570	297,818	84,375	36,403	581,116
Microsoft Azure	308,102	9,012	17,256	6,365	340,735

reported several attacks related to the COVID-19 pandemic to enable social engineering methods. In contrast to AWS, Cowrie logged the majority of attacks with 51% on GCP. Beside several automated attacks trying to log in with default credentials, adversaries tried to gather information about CPU architecture, scheduled tasks, and privilege escalation. Microsoft Azure reflects nearly the same results as the other two cloud providers beforehand.

The overall results show an average ratio of 55,000 attacks per day, summing up to roughly 1.3 million in total. Similar results for different regions could have been reproduced. Their results clearly show the disparity of the regions Europe, US, and Asia. An important question that Kelly et al. [41] answered is if attackers target services on cloud providers based on the cloud providers' market share. The study could not confirm this assumption based on the fact that Google Cloud with the smallest market share received most of the attacks. In total, most of the attacks are originated from Vietnam, Russia, United States, and China. Due to technologies such as VPN or Tor, the geolocation only indicates the last node, so location data might be distorted. Across all providers roughly 80% of the source IP addresses had a bad reputation and could have been filtered by the organization. The operating devices used for attacking the services are mostly Windows 7 or 8, and different Linux kernels and distributions. Windows devices target vulnerabilities in remote desktop sharing software. Such vulnerabilities are (i) CVE-2006-2369[14] (RealVNC) in the US region, (ii) CVE-2001-0540[11] (RDP) in EU and Asia regions, (iii) CVE-2012-0152[15] (RDP) in the Asia region, and (iv) CVE-2005-4050[13] (VoIP) in EU region. In addition, attackers were also capable of disguising any fingerprinting activity of p0f.

We want to compare the findings Kelly et al. [41] claimed in the paper "A Comparative Analysis of Honeypots on Different Cloud Platforms" with ours using the Heidelberg University's cloud solution. First, a short introduction of heiCLOUD is presented, followed by a closer lookup of T-Pot that is used to acquire data. Lastly, we present our results and do a thorough comparison closing up with a discussion based on a technical report of the Cambridge University.

3.2 Methodology

Our foremost goal is to track as many attacks as possible. Figure 3.1 sketches our concept to achieve our goal to gather various attacks from the Internet. Honeypots should be deployed on a single instance, and their data, or log files are stored in a database. By the help of data visualization tools, we analyze the attacks. For security reasons, honeypots should run in a virtualized environment to avoid any harm to our host system. We use Debian as a base Linux distribution. Our instance runs on heiCLOUD, a cloud service provided by Heidelberg University. It is capable of 16 GB of RAM, 8 VCPUs, and a volatile memory of 30 GB. In addition, we mount a 125 GB permanent volume to securely store our data. In its very early stages, we compared different approaches to achieve this goal. As an example, we compared native implementation approaches, using additional frameworks, and ready-to-use solutions. However, the T-Pot, developed by Telekom, offers a profoundly ready-to-use solution with major advantages. It combines several honeypots in conjunction with various analytic tools to trace the newest attacks. Furthermore, it helps to compare our findings with the ones Kelly et al. [41] claim.

Running our instance and exposing it to the Internet needs some adjustments beforehand. Therefore, a virtual network with subnet `192.168.145.0/24` has been created wherein our instance with IP address `192.168.145.4` is assigned to. The instance is accessible from the outside with floating IP address `129.206.5.74`. Access rules are similar to a stateless firewall, and thus, do not block any attacks. Ports $1-64000$ are exposed and can be attacked by anyone. Ports higher than 64000 are only accessible through the university network `129.206.0.0/16` or eduroam `147.142.0.0/16` and should provide a basic authentication with username and password.

3.2.1 heiCLOUD

University Computing Center Heidelberg [68] offers a "IaaS specially tailored for higher education and research institutions" called heiCLOUD. It supplies multiple institutes at Heidelberg University with storage, virtual machines, or network components. In addition, heiCLOUD is a DFN[1] member, and offers others to use their services. As stated on their information website[67], it is (i) capable of freely manageable IT resources, (ii) beholds a stable and fast connection, (iii) ensures high availability and scalability, (iv) has freely selectable VM operating systems, and (v) has a transparent payment model [67]. Based on the open source application OpenStack, users can easily create own network areas, and manage their space individually. Unlike well-known cloud providers, heiCLOUD servers are located within

[1] German National Research and Education Network, the communications network for Science and research in Germany

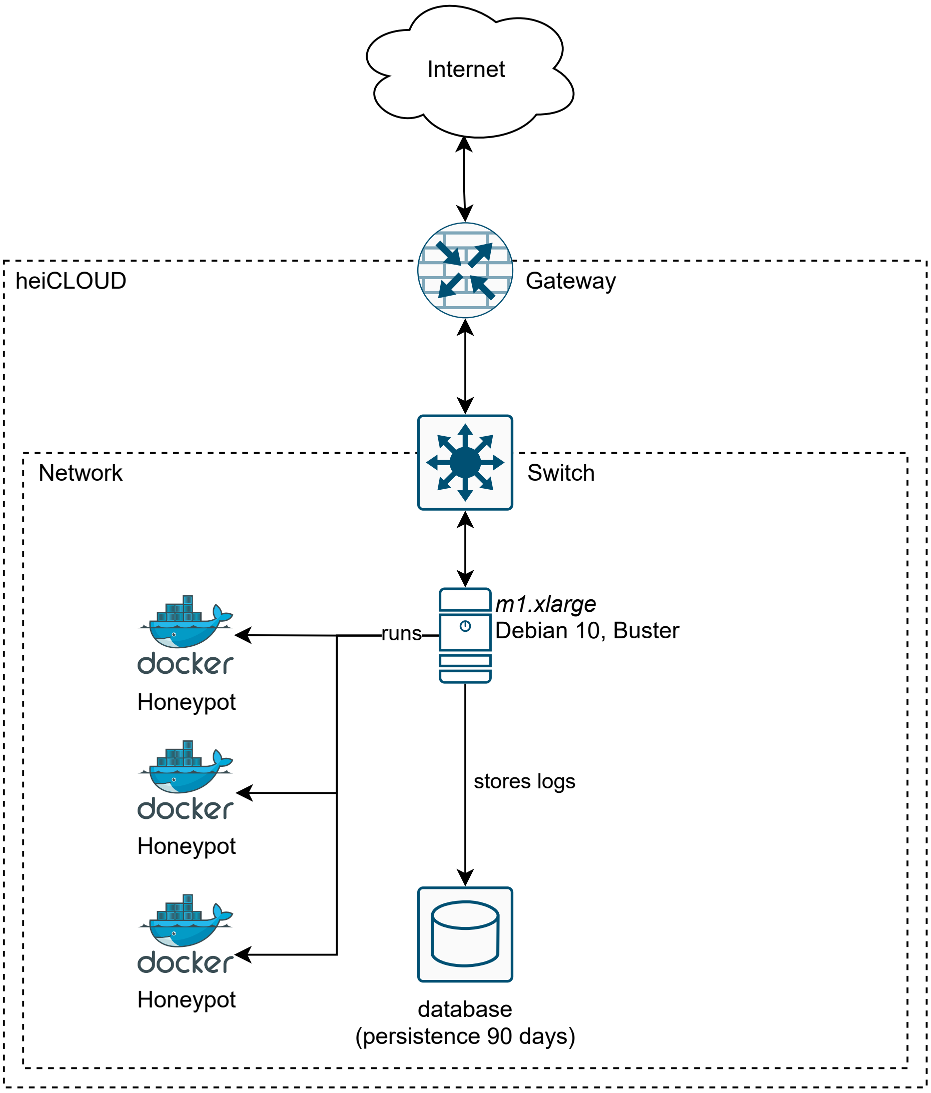


Figure 3.1: Concept to collect honeypot attacks

Germany, thus, abide by the European data privacy law. HeiCLOUD have never considered honeypots for additional cybersecurity measurements.

3.2.2 T-Pot

To be able to compare our results with Kelly et al. [41], we use the same approach to capture recent cyberattacks. The T-Pot solution, a mixture of Telekom and Honeypot, stands out with their sheer quantity of various honeypots. It requires at least 8 GB of RAM and a minimum of 128 GB hard drive storage. Based on a Debian 10 Buster distribution, it relies on Docker to run their services [25]. T-Pot has to be deployed in a reachable network where intruders are expected. Either

TCP and UDP traffic are forwarded without filtering to the network interface, or it runs behind a firewall with forwarding rules. Specified ports for attackers are 1-64000, higher ports are reserved for trusted IPs, thus, a reverse proxy asks for basic authentication. All daemons and tools run on the same network interface whereas some of them are encapsulated in their own Docker network. Docker is a lightweight virtualization technology that uses containers to run on the host system [9]. Unlike virtual machines, Docker reduces overhead with the downside of a greater attack surface. To mitigate attacks, Docker wraps containers in an isolated environment. This is achieved by restricting the kernel namespace and control groups (`cgroups`). Figure 3.2 visualizes the technical concept of T-Pot. Each service has dedicated ports or port ranges that are exposed. Attackers can communicate either with TCP or UDP. All honeypots and tools create log files that are used to get any knowledge about attackers. In order to view and trace current attacks, T-Pot uses the ELK stack. ELK is the acronym of Elasticsearch, Logstash and Kibana [26]. Elasticsearch is a search engine based on Lucene. It is multitenant-capable and offers full-text search via HTTP. Logstash is used to feed Elasticsearch. In general, it offers an open server-side data processing pipeline that helps to send data from multiple sources to an Elasticsearch node. Kibana is the main data visualization tool. It offers users to create plots and dashboards, crawl Elasticsearch, and trace the system health. All logs of the honeypots and tools are forwarded to the search engine Elasticsearch by Logstash. The ELK stack is not directly exposed to the Internet, thus, an authentication is not needed. Users can monitor all log files with Kibana by pre-defined dashboards, or custom search queries. In addition, T-Pot features different services types, namely (i) standard, (ii) sensor, (iii) industrial, (iv) collector, (v) next generation, and (vi) medical. Each service type has a different set of honeypots and tools tailored to their core idea. T-Pot feeds their data to an external Telekom service, however, this data submission can be turned off. For this chapter we restrict ourselves to the latest version 20.06.0. Newer versions might be available by the end of this study and could differ from ours.

Honeypots

T-Pot consists of 20 honeypots. Albeit the sheer quantity of it, we will give a short explanation. In addition, Table 3.2 gives a quick overview of all available honeypots in conjunction with (i) the port they are running on, (ii) their interaction level, and (iii) a short description.

ADBHoney [8] is a low interaction Android Debug Bridge (ADB) honeypot over TCP/IP. The importance of it lies in the ADB protocol that is used to debug and push content to an Android device. However, unlike USB it does not support any kind of ample mechanisms of authentication and protection. By exposing the ADB service over any port, an adversary could connect and exploit it. ADBHoney is designed to catch malware that has been pushed onto devices.

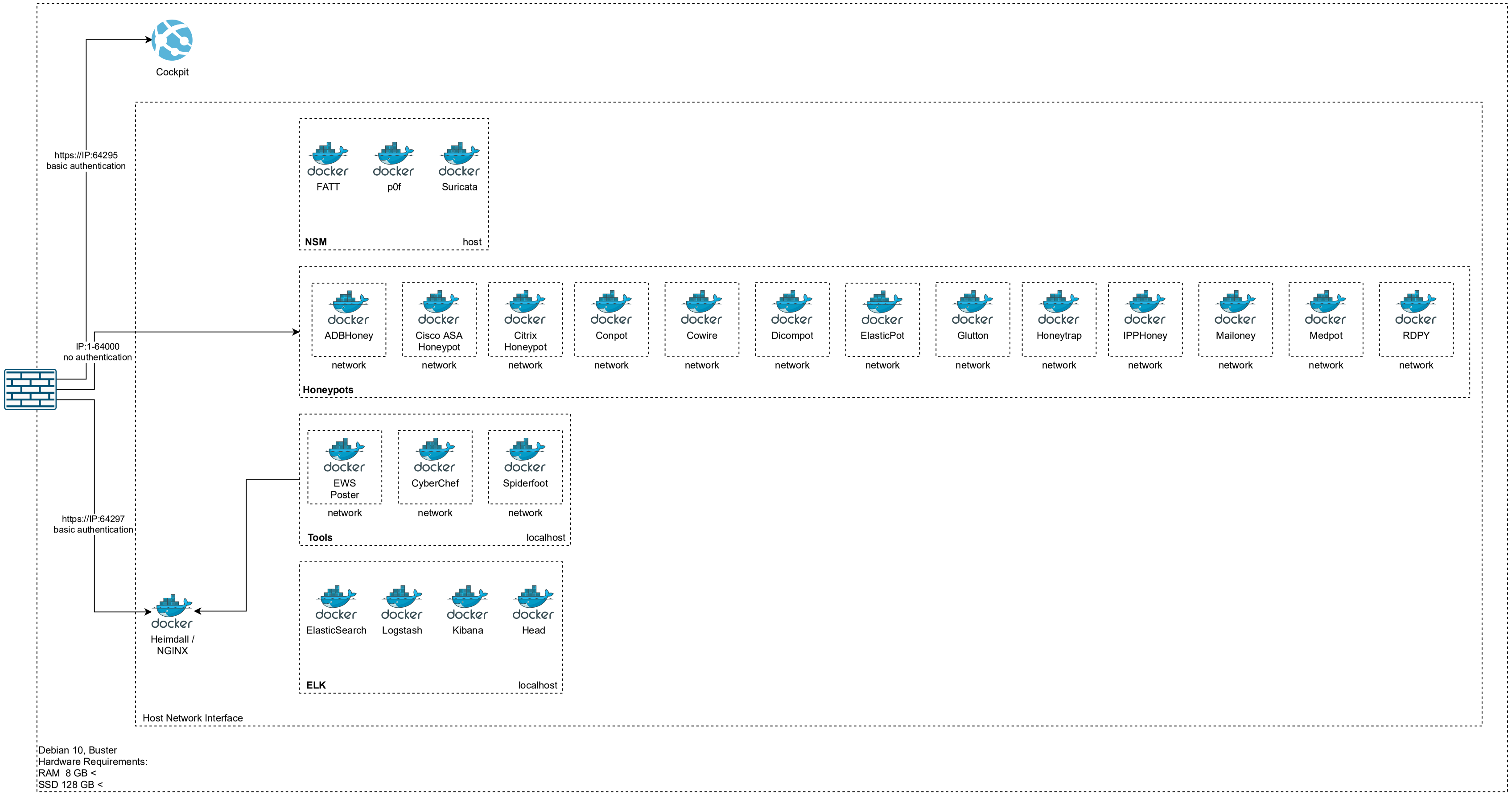


Figure 3.2: T-Pot architecture derived from [51]. Honeypots are encapsulated in their own network. NSM runs on the host network, and thus, receives every packet. ELK and tools run on localhost, and are accessible through NGINX.

Cisco Adaptive Security Appliance (ASA) [56] is a low interaction honeypot that detects CVE-2018-0101[16]. It is a vulnerability that could allow an unauthenticated remote attacker to cause a reload of the affected system and to remotely execute code. This can be achieved by flooding a webvpn-configured interface with crafted XML packets. Consequently, the attacker obtain full control by executing arbitrary code.

Citrix Application Delivery Controller (ADC) honeypot [35] detects and logs CVE-2019-19781[18] scans and exploitation attempts. This vulnerability allows adversaries to perform directory traversal attacks. Files are accessible by path strings to denote the file or directory. In addition, some file systems include special character to easily traverse the hierarchy. Attackers take advantage of it by combining special characters in order to get access to restricted areas. [30]

Conpot [57] is a low interaction industrial honeypot for ICS, and SCADA. It provides a variety of different common industrial control protocols. An adversary should be tricked by the complex infrastructure, and lure him into attacks. In addition, a custom human machine interface can be connected to increase the attack surface. By randomly delaying the response time, Conpot tries to emulate a real machine handling a certain amount of load.

Cowrie [52] is a medium to high interaction SSH and Telnet honeypot. It offers to log brute-force attacks and shell interactions with attackers. In medium interaction mode cowrie emulates a UNIX shell in Python, whereas in high interaction mode it proxies all commands to another system.

DDoSPot [22] is a low interaction honeypot to log and detect UDP-based Distributed Denial of Service (DDoS) attacks. It is used as a platform to support various plugins for different honeypot services, and servers. Currently, it supports DNS, NTP, SSDP, CHARGEN, and random/mock UDP server.

Dicompot [42] is a low interaction honeypot for the Digital Imaging and Communications in Medicine (DICOM) protocol. As other honeypots before, it mocks a DICOM server in Go to collect logs and detect attacks.

Dionaea [24] is a medium interaction honeypot that tries to capture malware copies by exposing services. It supports a vast variety of protocols such as FTP, SMB, and HTTP. Several modules can be integrated to work with Dionaea such as VirusTotal for further malware results.

Elasticpot [4] is a low interaction honeypot for Elasticsearch, a search engine based on the Lucene library.

Glutton [58] is a generic low interaction honeypot that works as a MitM for SSH and TCP. However, lacking documentation does not provide a deeper insight of this honeypot.

Heralding [70] is a credential catching honeypot for protocols like FTP, Telnet, SSH, HTTP, or IMAP.

HoneyPy [32] is a low to medium interaction honeypot that supports several protocols such as UDP, or TCP. New protocols can be added by writing a custom plugin for it. HoneyPy should give the freedom of easily deploying and extending honeypots.

HoneySAP [32] is a low interaction honeypot tailored for SAP services.

Honeytrap [74] is a low interaction honeypot network security tool. As stated by Werner [74], Honeytrap is vulnerable to buffer overflow attacks.

IPPHoney [5] is a low interaction Internet Printing Protocol (IPP) honeypot.

Mailoney [46] is a low interaction SMTP honeypot written in Python.

MEDpot [61] is a low interaction honeypot focused on Fast Healthcare Interoperability Resources (FHIR). It is a standard description data format to transfer and exchange medical health records.

RDPY [53] is a low interaction honeypot of the Microsoft Remote Desktop Protocol (RDP) written in Python. It features client and server side, and it is based on the event driven network engine Twisted. It supports authentication over SSL and NLA.

SNARE and TANNER [59, 60] is a honeypot project. SNARE is an abbreviation for Super Next generation Advanced Reactive honEypot. It is a successor of Glastopf, a web application sensor. In addition, it supports the feature of converting existing webpages into attack surfaces. TANNER [60] can be seen as SNARES's brain. Whenever a request has been sent to SNARE, TANNER decides how the response should be.

Tools

T-Pot integrates tools to screen network traffic, and to block DoS attacks.

FATT [38] is used to extract metadata and fingerprints such as JA3 [2] and HASSH [39] from captured packets. JA3 is a method for "creating SSL/TLS client fingerprints" whereas HASSH is a "network fingerprinting standard which can be used to identify specific client and server SSH implementations". In addition, it features live network traffic. As noted by the author, FATT is based on a python wrapper for tshark, namely pyshark, and thus having performance downturns. T-Pot applies FATT on every request made on the host network.

Spiderfoot [48] is an open source intelligence automation tool that helps to screen targets to get information about what is exposed over the Internet. It can target different entities such as IP address, domain, hostname or network subnet. In addition, it features more than 200 modules that can be integrated as an extension. T-Pot uses it to scan defensively and thus not include any other module.

Suricata [66] is "a high performance IDS, intrusion prevention system (IPD) and network security monitoring (NSM) engine". T-Pot lets Suricata analyze and assess any request made on the host network.

P0f [76] is a fingerprinting tool that uses passive traffic fingerprinting mechanisms to check TCP/IP communications. T-Pot lets p0f passively check any request made on the host network.

Endlessh [73] is an SSH server that sends an endless, random SSH banner. The key idea is to lock up SSH clients that try to connect to the SSH server. It lowers the transaction speed by intentionally inserting delays. Due to connection establishing before cryptographic exchange, this module does not require any cryptographic libraries.

HellPot [36] is an "endless honeypot". Connecting to this honeypot results in a memory overflow. Its key idea is to send an endless stream of data to the attacker until its memory, or storage runs out.

3.3 Results

Our T-Pot has been deployed for 3 weeks (from 21st of September to 16th of October) and collected in total 607,747 attacks. Overall, RDPY (46.08%), Honeytrap (33.23%), and Cowrie (12.42%) received most of the attacks with a total amount of 540,398 attacks. Figure 3.3 shows the distribution of honeypot attacks. The total numbers are based on Table 3.3.

What is striking is the large disparity between the previously mentioned attacks on AWS, GCP, and Azure. Especially with the honeypot Dionaea it is unclear why only 2,368 attacks have been performed. 96% of IP addresses connected to Dionaea are known attackers, and 70% were acquired on port 81, unofficially known for Tor routing. Neither any malware nor suspicious payload could be identified. We assume that the packets run through a static filter. Heidelberg itself has a centralized stateless firewall which indicates that certain ports or protocols are excluded. To prove our assumption, a NMAP TCP SYN scan (`nmap -sS -A 129.206.5.74`) has been performed. Our result clearly shows that port 139 for SMB is filtered, although the access security explicitly allows it. We also checked the stateless firewall running in front of heiCLOUD, and it turned out that this port is filtered with many others. Based on this, we assume that most of the attacks on Dionaea are carried out via SMB which would explain the total number of attacks. We have consulted the administrator of the university firewall to exclude our T-Pot instance to validate if the

Table 3.2: Overview of all available honeypots and tools of T-Pot with interaction level, port, and a short description. Ports are marked with either TCP or UDP, if a port misses any definition, both TCP and UDP are allowed.

Honeypots			
	Port	**Interaction-level**	**Description**
ADBHoney [8]	5555/TCP	low	ADB protocol honeypot
Cisco ASA [56]	5000/UDP, 8443/TCP	low	honeypot for CVE-2018-0101[16] detection
Citrix honeypot [35]	443/TCP	low	detects and logs CVE-2019-19781[18] scans and exploitation attempts
Conpot [57]	80, 102, 161, 502, 623, 1025, 2404, 10001, 44818, 47808, 50100	low	industrial honeypot for ICS, and SCADA
Cowrie [52]	2222, 23	high	SSH and Telnet honeypot
DDoSPot [22]	1112/TCP	low	log and detect UDP-based DDoS attacks
Dicompot [42]	1112/TCP	medium	honeypot for the DICOM protocol
Dionaea [24]	21, 42, 69/UDP, 8081, 135, 443, 445, 1433, 1723, 1883, 1900/UDP, 3306, 5060/UDP, 5061/UDP	low	capture malware copies
Elasticpot [4]	9200	low	honeypot for Elasticsearch
Glutton [58]	NFQ	medium	MitM proxy for SSH and TCP
Heralding [70]	21, 22, 23, 25, 80, 110, 143, 443, 993, 995, 1080, 5432, 5900	low	credential catching honeypot
HoneyPy [32]	7, 8, 2048, 2323, 2324, 4096, 9200	low	extendable honeypot
HoneySAP [32]	3299/TCP	low	honeypot for SAP services
Honeytrap [74]	NFQ	medium	captures attacks via unknown protocols
IPPHoney [5]	631	low	IPP honeypot
Mailoney [46]	25	low	SMTP honeypot
MEDpot [61]	2575	low	FHIR honeypot
RDPY [53]	3389	low	Microsoft RDP honeypot
SNARE/TANNER [59]	80	low	web application honeypot

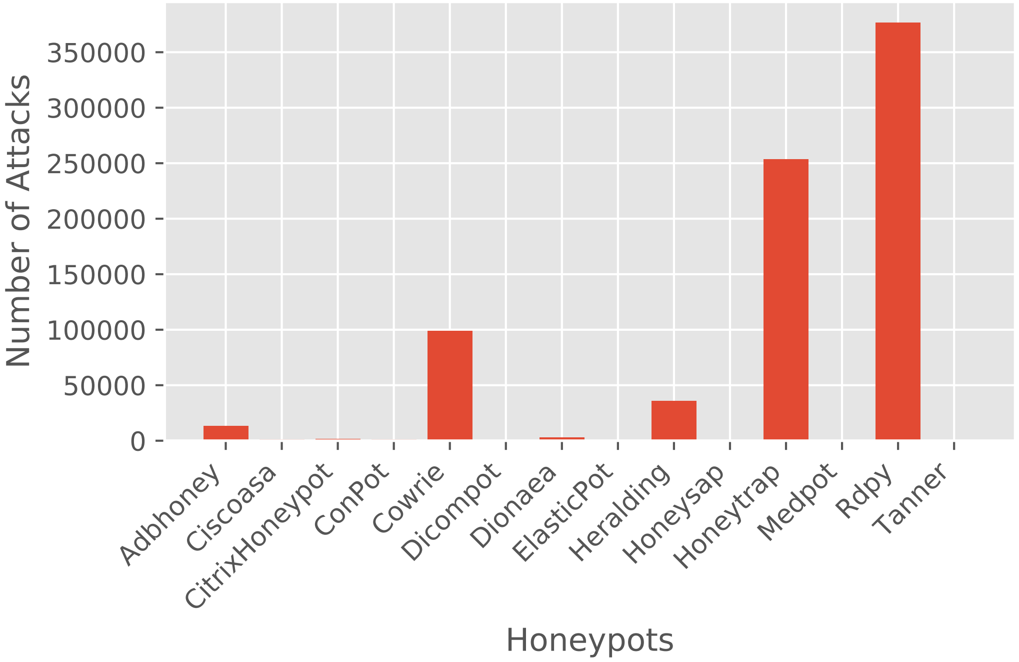


Figure 3.3: Distribution of honeypot attacks. Timestamp; 21st of September to 16th of October.

real number is even higher without any packet filtering in front of it. Respectively, for a period of three weeks (2nd of December until 23rd of December) no stateless packet filter has been applied to our T-Pot. We could identify a drastic increase in Dionaea attacks with a total number of 213,053. Overall, 93% of all attacks are on the smbd protocol followed by many database protocols such as mongod, and mssqld. This confirms our assumption that without the packet filter in front of our instance, a higher total number of attacks would be the result.

Comparing the number with Kelly et al. [41] we see that our number surpasses every other cloud provider. However, Dionaea attacks will not be included in our later results because normally a server is not allowed to be excluded from the university firewall. Only for our research purpose to assess the effect of the packet filter we were allowed to get a permit exclusion.

Logstash uses GeoLite2 to resolve the source IP address with information such as location, ASN, continent code, country name, and AS organization. Figure 3.4 indicates the geographical location of connections acquired to any honeypot. Most attacks are originated from the United States, Germany, Russia, and China. Large security scans of DFN or Baden-Württembergs extended LAN (BelWÜ) pushes Germany to second place, therefore, Germany can be considered negligible. On the contrary, the geographical location of an IP address merely indicates the true origin.

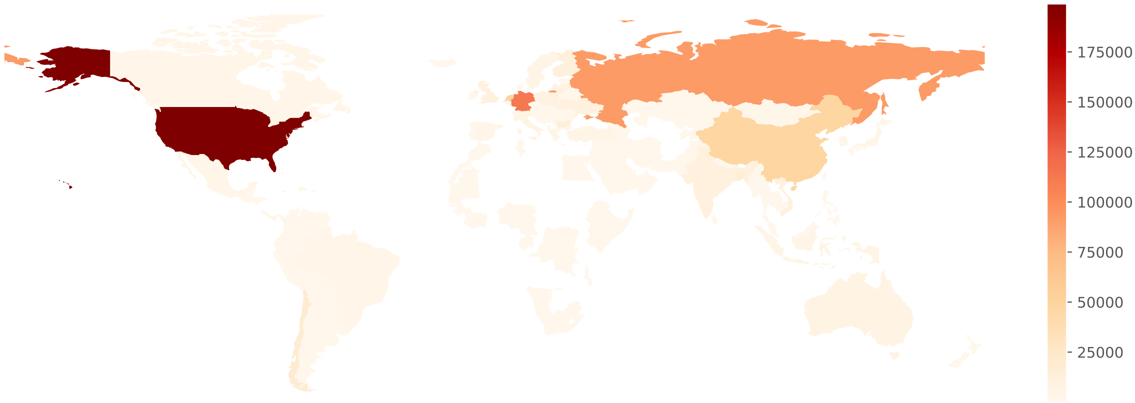


Figure 3.4: Attack distribution of T-Pot. USA, Russia, China, and Germany are the most attacking countries. Timestamp; 21st of September to 16th of October.

Due to technologies like VPN or Tor, the last known node of an IP address could be spoofed, and thus as stated by Kelly et al. [41], would remain insufficient to use. Hence, we do not strongly rely on the geographical information.

Attacks are not equally distributed among all honeypots, and thus, different protocols and applications receive more attention than others. Figure 3.5 shows the timeline of attacks that are executed on our instance separated by honeypots. RDPY, Honeytrap, and Cowrie are clearly the most attacked honeypots. The high peak of Honeytrap in the middle indicates a full NMAP scan from Germany that has been done to get an insight of the packet filtering at the Heidelberg University. We clearly identify a bias towards remote desktop protocol attacks, shell-code exploitation, and commands to retrieve information about CPU, scheduled tasks (`cat /proc/cpuinfo`, or `crontab`), or privilege escalation.

Suricata registered several alerts and CVEs. The vast majority of alerts are RDP related policies, VNC authentication failures, and NMAP scans. Most used vulnerabilities are (i) CVE-2001-0540[11] which is a memory leak in terminal servers in Windows NT and Windows 2000 causing a denial of service (memory exhaustion) by malformed RDP requests, (ii) CVE-2006-2369[14] which is a RealVNC vulnerability allowing hackers to bypass authentication, and (iii) CVE-2012-0152[15] which enables attackers for RDP in Microsoft Windows Server 2008 R2 and R2 SP1 and Windows 7 Gold and SP1 to cause a denial of service by sending a series of crafted packets. As derived from Figure 3.6, our T-Pot has not received many attacks in the first week. Starting from 28th of September the number of alerts are skyrocketing. This would indicate that bots crawl IP address ranges to find new machines and probe them. Interestingly, zero-day exploits like the Apache vulnerability [20] that came with version 2.49.0 got registered in CVE on the 6th of October, and immediately recognized by Suricata on the 15th of October. Attackers could perform a remote code execution using path traversal attacks when the CGI scripts of Apache are en-

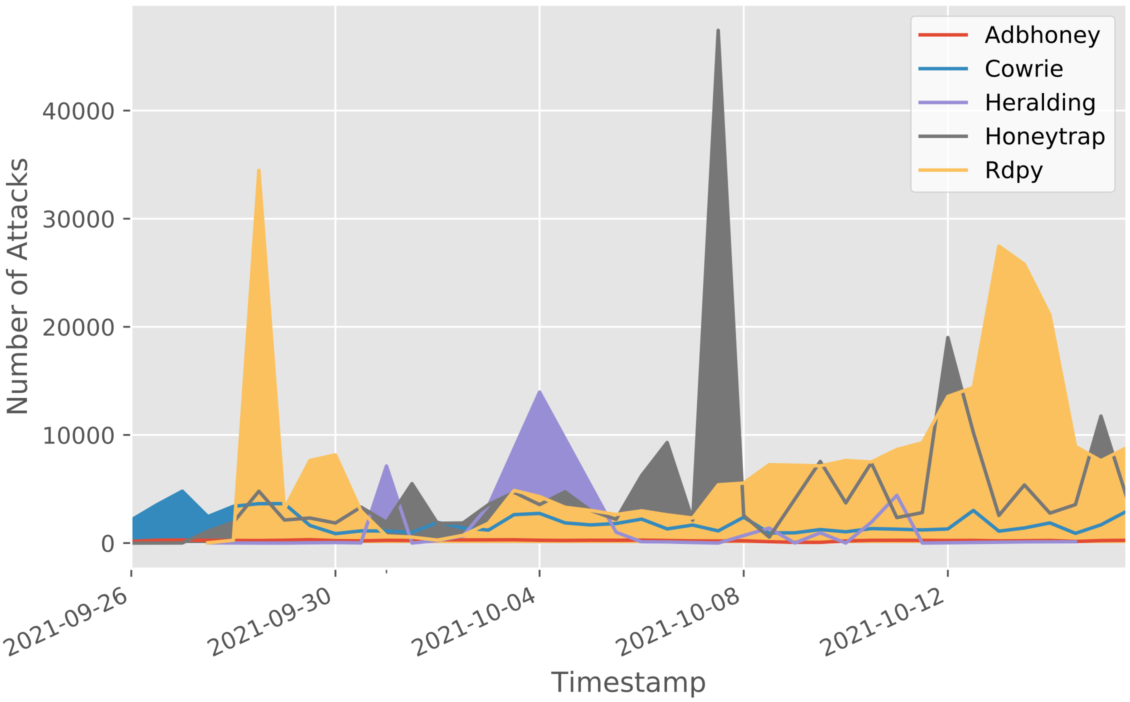


Figure 3.5: Attack histogram of T-Pot. Only the five most attacked honeypots are considered. Timestamp; 21st of September to 16th of October.

abled. We could trace back similar attacks like `/cgi-bin/.\%2e/\%2e\%2e/bin/sh` until the 7th of October, leaving an even smaller time frame to adapt to new exposures. This shows how fast bots adapt to new vulnerabilities in order to compromise more systems.

The results from RDPY in Figure 3.7 backups our assumption that attacks originate from bots. It shows that only a small margin represents unique source IP addresses. The rest of the attacks result in either bad reputation, bot, crawler, or known attacker. Figure 3.6 shows the distribution of alert categories that Suricata identified. Respectively, misc activities sum up to roughly 1.5 million entries, RDP related alerts account two-third of it. Several RDP attacks from 2021 back to 2001 had been executed on our T-Pot. Respectively, CVE-2012-0152 and CVE-2001-0540 coincide with the ones Kelly et al. [41] claim.

For NFQ related attacks, Honeytrap could identify three major services that are not provided by default. Honeytrap functions as a honeypot to provide a service on ports which are not specified by default. NFQ intercepts incoming TCP connections during the TCP handshake and Honeytrap provides a service for it. Most of these interceptions are made on (i) port 5038 which is merely used by a machine learning database called MLDB, (ii) port 5905 which is merely used by an Intel Online Connect Access on Windows machines, and (iii) port 7070 which is merely used by Apple's QuickTime streaming server (RTSP). On nearly all ports attacks focused

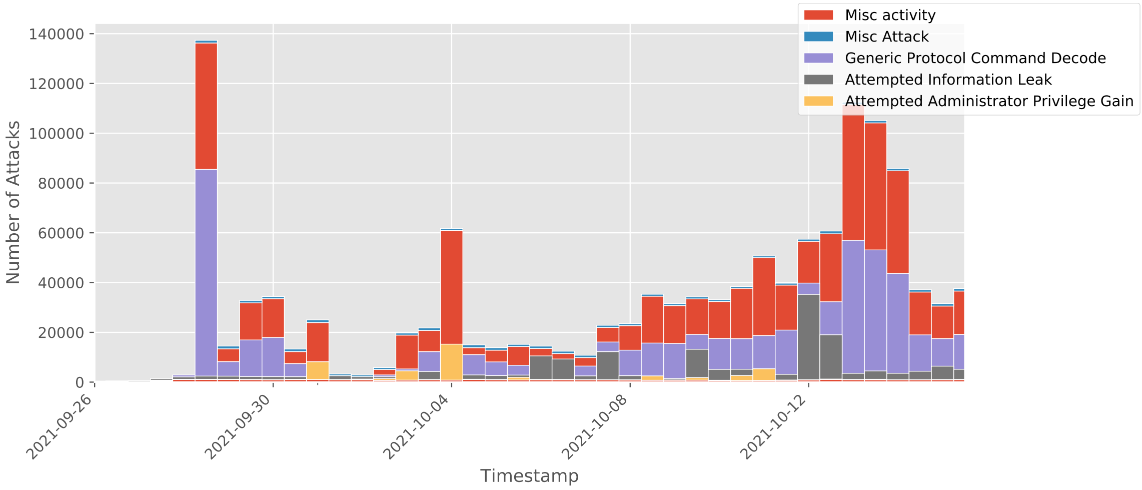


Figure 3.6: Suricata results of T-Pot. Displays the five most listed alert categories. Timestamp; 21st of September to 16th of October.

on RDP connection attempts (`Cookie: mstshash=Administr`). However, 94% of all connected IP addresses on Honeytrap are resolved as known attackers.

Third most compromised honeypot is Cowrie with a strong bias towards SSH, and FTP. Figure 3.9 shows all attacks executed on Cowrie separated by their port. Respectively, SSH port 22 is the most considered port, resulting in a high use for privilege escalation. Besides using default credentials to log in (`username: root`, `password: root`, see Figure 3.10 for top 10 credentials), adversaries used various commands to retrieve any information about the host system (`nproc;uname -a`, `cat /proc/cpuinfo`). We could identify a unique information gathering attack that has been widely used on our T-Pot. Listing 3.1 shows all shell commands that are executed. Attackers try to gain knowledge about running processes on the system (`/bin/busybox`). Interestingly, crypto mining attacks are getting more attractive to criminals. For example, XMRig has been the most downloaded malware for cryptocurrency mining. Some adversaries even executed complex tailored shell commands to exploit the host machine as a crypto miner (Listing 3.2). With respect to the current time it is not surprising that such attacks gain attraction. Attackers could exploit machines for crypto mining in order to earn more money. This looks more appealing than acquiring mining machines and hijacking electricity from surrounding apartments.

P0f identified different Windows versions and Linux distributions in conjunction with various SSH clients that are used to compromise our T-Pot. Like Kelly et al. [41], Windows 7 or 8 and Windows NT Kernel are the most used OS with 81%. Unfortunately, disguising OS fingerprinting activities account 84% of all fingerprints. Lastly, we cleaned up our results and excluded all IPs from DFN and BelWÜ from our results. Both scan frequently and check if any vulnerability exists. This distorts our findings, and thus, we filtered them based on their subnet addresses. However,

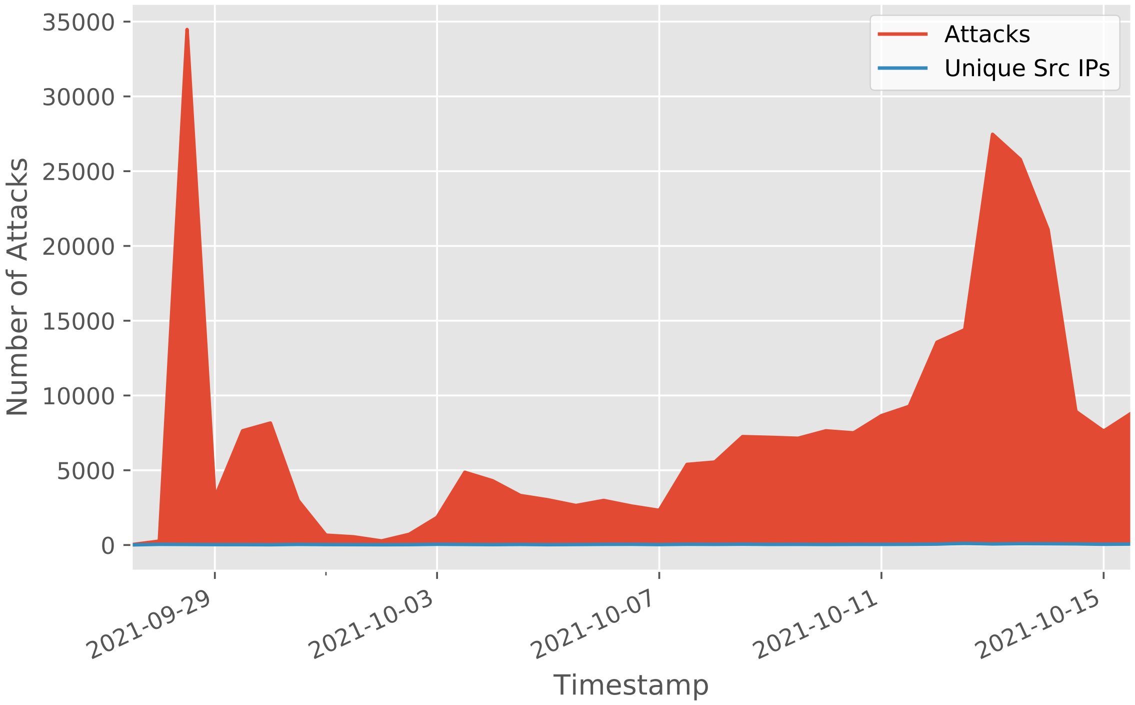


Figure 3.7: RDPY attacks separated in attacks and unique source IP addresses. Timestamp; 21st of September to 16th of October.

the results show no big changes. The total number of attacks were merely influenced by it. This indicates that our own scans do not greatly interfere with our findings. Due to completeness, we leave our results as they are and do not show the filtered data.

On average, heiCLOUD has received 48.23% more than Azure, GCP, and AWS. Attacks on Cowrie, RDPY, and Honeytrap are the most compromised honeypots. In contrast to Kelly et al. [41], Dionaea and Glutton used to be the most considered honeypots for adversaries. We assume that attacks by bots have increased significantly since last year when Kelly et al. [41] did their research. Respectively, one question we are not able to answer is if other cloud providers filter their network traffic. It would explain the major difference between Heidelberg University and the big tech companies. The cause for such an increase remains doubtful. One explanation could root back to the Corona pandemic and the skyrocketing increase in home office activities. Related to that is a higher usage in screen sharing software. Considering the BSI[2] report for cybersecurity 2021 [6], they revealed an increase of attack surfaces during the pandemic. Respectively, the IT infrastructure could not keep up with this fast change and widen the attack surface of the company. Their conclusion overlays our assumption that attackers took advantage and increased their

[2] The Federal Office for Information Security is responsible for managing communication security for the German government. Each year they publish a report for recent cybersecurity threats.

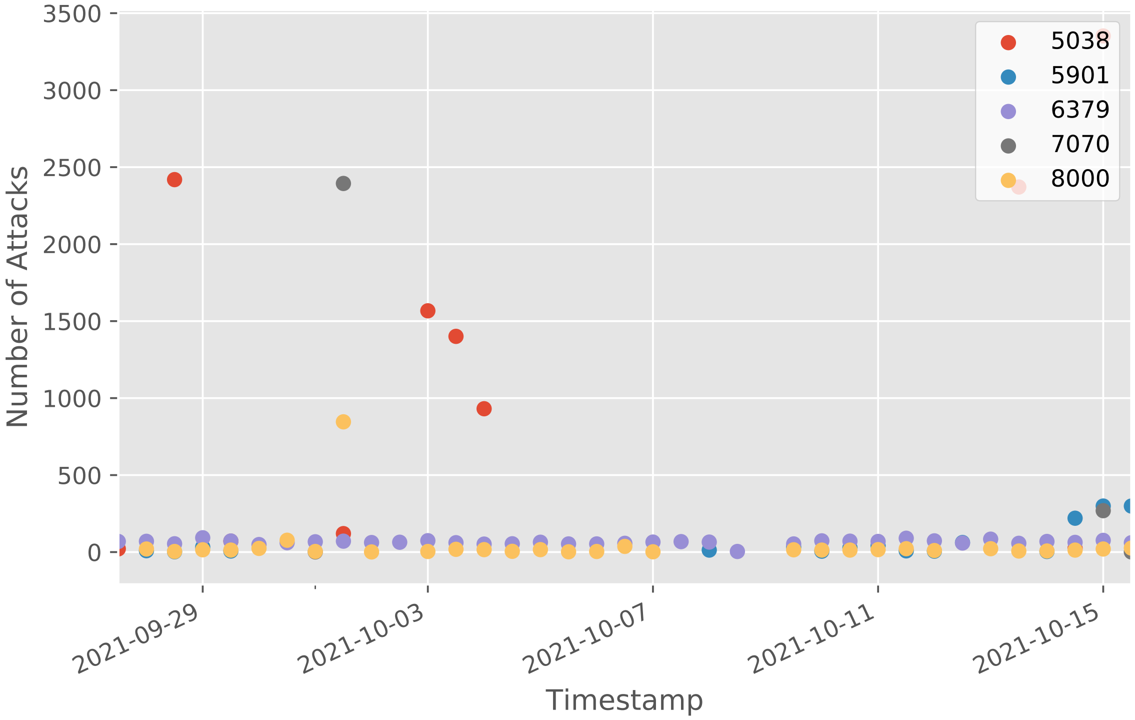


Figure 3.8: Honeytrap results of T-Pot. Timestamp; 21st of September to 16th of October.

activities. This phenomenon shows that nearly all attacks originate from bots which scan through IP address ranges. In total, 73% of all IP address are unresolved. The known attacker reputation represents the largest part of resolved IP addresses with 23%. Fortunately, such reputations could technically be filtered by an organization's firewall and would lower the chance of an exploit. Interestingly, after 3 weeks the number of attacks originated from China decreased to almost zero percent. This might be an indication that our honeypot has been exposed, and further attacks represent a risk of revealing their compromises. However, this assumption cannot be confirmed due to the lax geographical reliability of IP addresses.

Our results lay emphasis on the importance of honeypots. It gives a proper security measure of an IT infrastructure, and helps to identify potential leaks or vulnerabilities. Moreover, it shows that T-Pot helps to detect recent bot activities, and gives an outlook to the newest trends of attacks.

3.4 Discussion

One downside of T-Pot is the static hostname representation of Cowrie. It always returns `#1 SMP Debian 3.2.68-1+deb7u1` (`uname -a`) as hostname information, leaving a tiny footprint when bots crawl through the web. A random choice

Table 3.3: Overview of attacks on heiCLOUD, AWS, GC and Azure. Only the top 10 most attacked honeypots are considered. "-" entails that a honeypot is not part of the top 10.

HONEYPOTS	BASIS			COMPARISON					
	HEICLOUD			AWS		GC		AZURE	
	Number	**Pct.**		**Number**	**Pct.**	**Number**	**Pct.**	**Number**	**Pct.**
ADBHoney	9,302	1.65%	↑	413	0.13%	2,497	0.43%	442	0.13%
Cisco ASA	674	0.11%	↑	260	0.08%	750	0.13%	134	0.04%
Citrix honeypot	1,121	0.18%		-	-	-	-	-	-
Conpot	615	0.10%		-	-	-	-	-	-
Cowrie	75,511	11.97%	↓	4,503	1.46%	297,818	51.25%	9,012	2.64%
DDoSPot	0	0%		-	-	-	-	-	-
Dicompot	22	0%		-	-	-	-	-	-
Dionaea	2,368	0.40%	↓	288,075	93.49%	162,570	27.98%	308,102	90.42%
Elasticpot	385	0.06%		-	-	-	-	-	-
Glutton	0	0%	↓	11,878	3.85%	84,375	15.52%	17,256	5.06%
Heralding	35,680	4.34%	↑	1,885	0.61%	12,255	2.11%	3,370	0.99%
HoneyPy	0	0%	↓	172	0.06%	2,149	0.37%	497	0.15%
HoneySAP	15	0%		-	-	-	-	-	-
Honeytrap	201,949	32.01%		-	-	-	-	-	-
IPPHoney	0	0%		-	-	-	-	-	-
Mailoney	0	0%	↓	720	0.23%	9,419	1.62%	146	0.04%
MEDpot	2	0%		-	-	-	-	-	-
RDPY	280,040	49.15%	↑	100	0.03%	7,916	1.36%	1,463	0.43%
SNARE/TANNER	63	0.02%	↓	138	0.04%	1,367	0.24%	313	0.09%
IN TOTAL	607,747	100%		308,144	100%	581,116	100%	340,735	100%

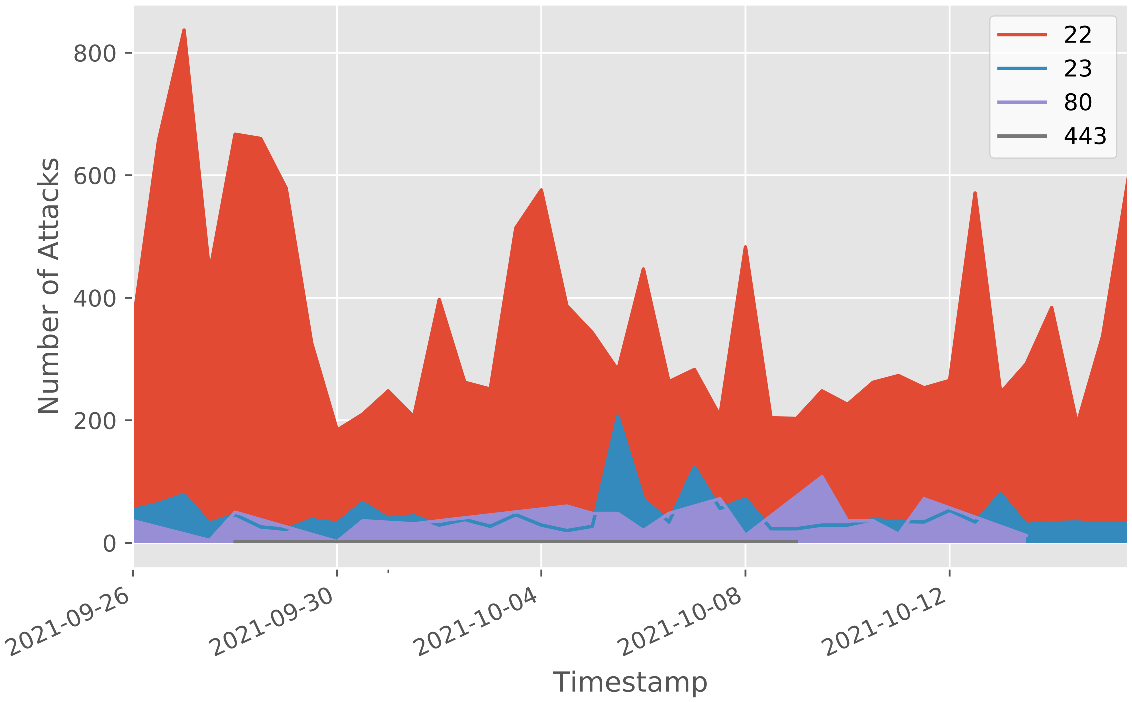


Figure 3.9: Cowrie results of T-Pot. Timestamp; 21st of September to 16th of October.

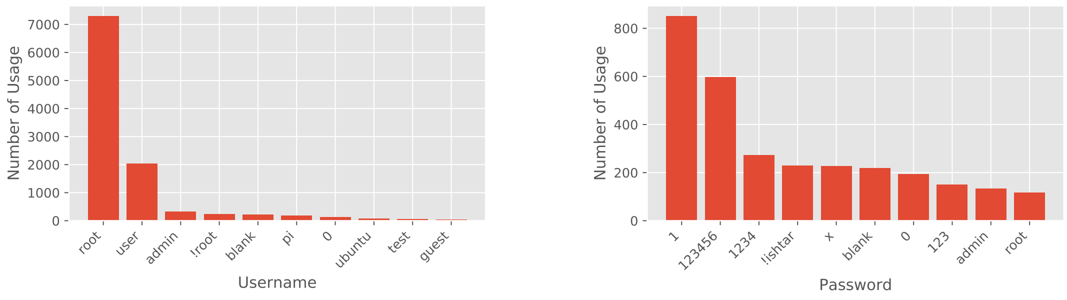
of hostname information could harden Cowrie from being exposed. Next, if attackers would scan open ports on T-Pot, it might be suspicious when many ports with services are open. From a technical perspective, bots could check this state if it is uncommon, and thus, exclude T-Pot from being probed. However, T-Pot includes good preventions like a random hostname and scheduled tasks. Another major drawback is the latest endeavor to detect honeypots on transport level. As recently investigated by Vetterl [71], detecting honeypots is becoming easier due to a fatal flaw in the underlying protocol implementation. Vetterl [71] states that attackers always try to prevent their methods, exploits and tools from being divulged. Therefore, detecting honeypots before attack them is a strong motivation for black hats. In chapter 5 we will present a way to avoid such fingerprint activities with the honeypot Cowrie.

Listing 3.1: Cowrie attack to gather various information about the system

```
enable
system
shell
sh
cat /proc/mounts; /bin/busybox $PROCESS_NAME
cd /dev/shm; cat .s || cp /bin/echo .s; /bin/busybox $PROCESS_NAME
tftp; wget; /bin/busybox $PROCESS_NAME
dd bs=52 count=1 if=.s || cat .s || while read i; do echo $i; done < .s
while read i
/bin/busybox $PROCESS_NAME
rm .s; exit
```

Listing 3.2: Cowrie attack to exploit the host machine as a crypto miner

```
mkdir -p /home/osmc/.ssh/
echo ssh-rsa  $RSA_KEY >> /home/osmc/.ssh//authorized_keys
echo '<cmd7uname>'; uname -a
echo '</cmd7uname><cmd7uptime>'; uptime
echo '</cmd7uptime><cmd7w>'; w
echo '</cmd7w><cmd7who>'; who
echo '</cmd7who><cmd7last>'; last
echo '</cmd7last><cmd7lastlog>'; lastlog
echo '</cmd7lastlog><cmd7authkey>'; cat /home/osmc/.ssh//authorized_keys
echo '</cmd7authkey><cmd7lshome>'; ls -la /home
echo '</cmd7lshome><cmd7passwd>'; cat /etc/passwd
echo '</cmd7passwd><cmd7shadow>'; sudo -n cat /etc/shadow
echo '</cmd7shadow><cmd7psfaux>'; ps -faux
echo '</cmd7psfaux><cmd7netstat>'; netstat -npta
echo '</cmd7netstat><cmd7arpan>'; /usr/sbin/arp -an
echo '</cmd7arpan><cmd7ifconfig>'
/usr/sbin/ifconfig
echo '</cmd7ifconfig><cmd7localconf>'; cat /home/ethos/local.conf
echo '</cmd7localconf><cmd7remoteconf>'
cat /home/ethos/remote.conf
echo '</cmd7remoteconf><cmd7rclocal>'
cat /etc/rc.local
echo '</cmd7rclocal><cmd7claymorestub>'; cat /home/ethos/claymore.stub.conf
cat /hive-config/rig.conf; cat /hive-config/wallet.conf
cat /hive-config/vnc-password.txt
echo '</cmd7claymorestub><cmd7claymorezstub>'
cat /home/ethos/claymore-zcash.stub.conf
echo '</cmd7claymorezstub><cmd7sgminerconf>'
cat /var/run/ethos/sgminer.conf
echo '</cmd7sgminerconf><cmd7iptables>'
sudo -n iptables -S  && sudo -n iptables -t nat -S
echo '</cmd7iptables><cmdcrontab>'; crontab -l; echo '</cmdcrontab>'
exit
```



(a) Cowrie username credentials

(b) Cowrie password credentials

Figure 3.10: Cowrie top 10 credentials used on T-Pot. Timestamp 22nd of September to 18th of October

Chapter 4

Catching Attackers in Restricted Network Zones

Our T-Pot identified a flood of threads when it was available on the Internet. However, capacious networks do have separated compartments and usually services are not directly available without any protection. Zoning is a well-known method to segment a network. Heidelberg University applies zoning, and thus, it is an interesting question if any attacker probes services outside or within the network. We present a concept that uses a honeypot-like detection tool to detect any dubious packets in the network. We show that attacks in a restricted network zone of the Heidelberg University's internal network occur, and contributed to an adaption of the stateless firewall. Thus, improving the security of the network.

4.1 University Network

Honeypots that are accessible via the Internet do receive a broad range of attacks. As Spitzner [64] noted, a honeypot is not strictly bound to run in a demilitarized zone (DMZ) or in a network with direct Internet access. The correct location has to be chosen based on the goals of the honeypot. For example, one goal could be to catch attackers behind a perimeter firewall to reveal leaks or vulnerabilities. Beforehand, our honeypot was broadly available on the Internet, and attackers could probe it easily. It collected on average 29,840 attacks per day, resulting in a total amount of 607,747 attacks. Zoning a network into logical groups mitigates the risk of an open network. Thus, our T-Pot would receive significantly fewer attacks in a controlled network zone. A network infrastructure is segmented into the same communication security policies and security requirements. For example, the Canadian government created their own baseline for infrastructures, called Baseline Security Architecture Requirements for Network Security Zones in the Government of Canada (ITSG-22). The four most common zones are: (i) Public Zone (PZ) which is entirely open, (ii) Public Access Zone (PAZ) which interacts as an interface between the PZ and internal services, (iii) Operations Zone (OZ) which processes sensitive information, and (iv) Restricted Zone (RZ) which includes business critical services. A network zone restricts access and controls data communication flows. [10]

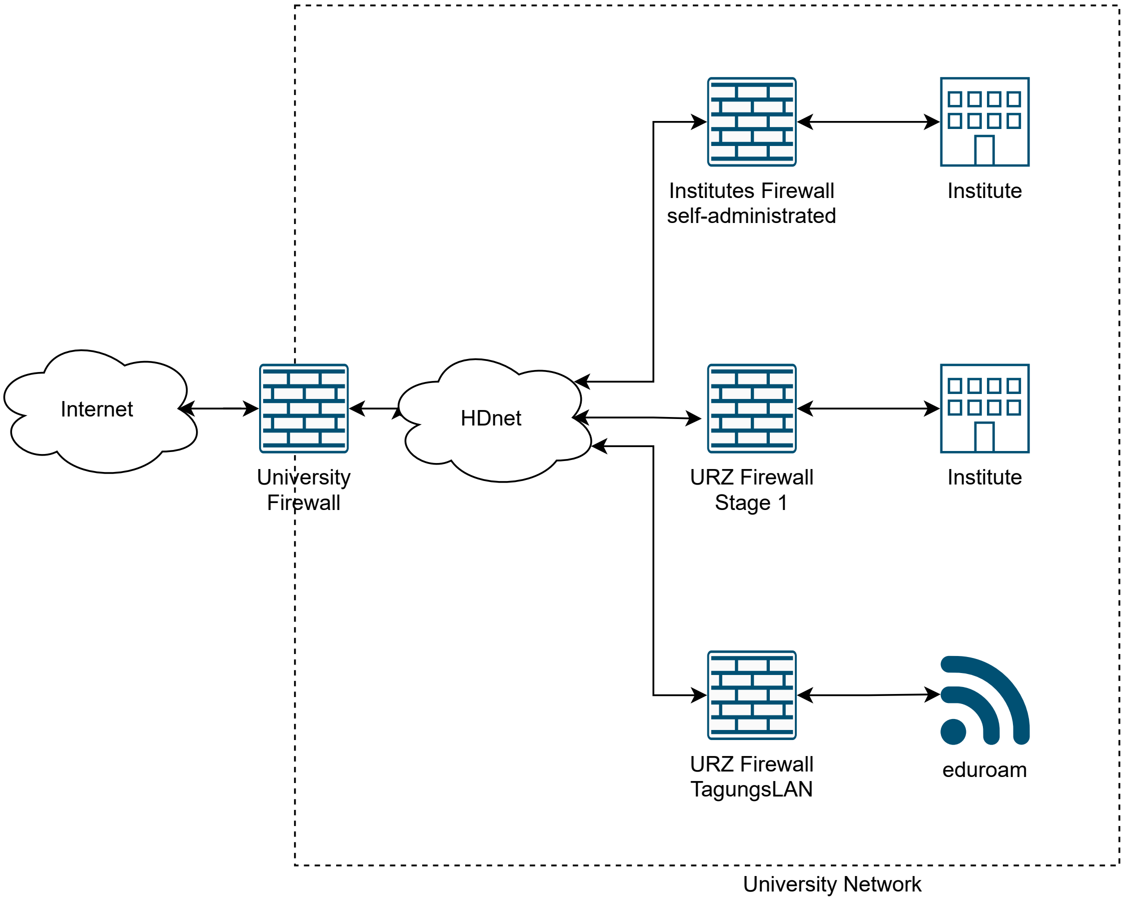


Figure 4.1: Draft of the University network.

The network at the Heidelberg University includes a central stateless firewall (Access Control List (ACL)) that enfolds all institutes. It entails a default blacklist that blocks certain services (such as SMTP, NCP, or Simple Network Management Protocol (SNMP)) and a stateless filter provided by BelWÜ. Inside the network, each institute has the possibility to either use a pre-defined stateless firewall provided by the University Computing Center Heidelberg, or use a self-administrated firewall. Figure 4.1 outlines the association between these components. The internal "HDnet" enables the communication between institutes without leaving the internal network. Institute firewalls can be set up by each institute and is self-administrated. They do have the possibility to use SOHO routers[1] to disconnect certain network zones from the network. It is recommended to configure the global ACL as a fall back solution

[1] A small office/home office router is a broadband router used in small offices and home offices environments.

Table 4.1: Overview of firewall stages at the Heidelberg University. As an example we applied the rules to subnet `129.206.218.0/24`. Rules apply to any subnet.

Name	Description	range
Stage 0	Filters broadcast communication	`129.206.218.0-15/24`
	No filtering	`129.206.239.16-255/24`
Stage 1	Allows common network protocol	`129.206.239.0-255/24`
	Allows services	`129.206.239.240-255/24`
Stage 3	Internet access only via internal proxies	`129.206.239.0-255/24`
Stage 4	Only internal network communication	`129.206.239.0-255/24`

in case of any downtime. The University Computing Center Heidelberg offers stateless firewalls for router interfaces or VLANs. This stateful firewall whitelists certain services and splits up into four stages. Each stage can be individually activated per router interface. Its key value is to maintain a baseline security to avoid any misconfigurations and port scans. Table 4.1 outlines these stages including the IP address range. Before applying one of these zones, the respective network has to oblige to client IP addresses below `129.206.218.240/24`. In addition, `129.206.218.1` is allocated for the gateway. A network has to adhere to these obligations if it applies any of these pre-defined stages.

An interesting question is if attackers have access to restricted zones at the Heidelberg University. It arises during the research of T-Pot if an adversary would try to probe any hosts in the internal university network. In order to detect such events we present a honeypot-like packet detection application that helps to identify any threats in a network. In addition, it offers to deploy multiple instance and collect their data at a centralized instance.

4.2 Honeypot-like Connection Detection Tool

Recording and investigating connection attempts assimilates new honeypots. Respectively, we will present a new honeypot-like detection tool called MADCAT. It has been developed by the BSI, and helps us to log any connection attempt being made on our host machine. The acronym MADCAT stands for Mass Attack Detection Connection Acceptance Tools. It works as a honeypot-like detection application with low interaction level. Its key idea is to log every connection attempt and further process it to retrieve credentials, or shell exploitation. Figure 4.3 gives an insight how MADCAT works. It runs on an Ubuntu distribution, either 18.04 or 20.04. We have tested it on Ubuntu 18.04. It processes packets from any interface that has been configured. As an example, we could process Ethernet and wireless packets. MADCAT itself consists of six independent modules for TCP, UDP, ICMP, and RAW packets that communicates with each other through a pipeline. A module is responsible for analyzing packets and logging the results in a queue. In addition, UDP and TCP offers a proxy to tunnel packets to another service. Every 5 seconds TCP postprocessor reads the newly arrived TCP packets and processes them accordingly. It resolves packets to log data including source IP address, protocol, and event type. The enrichment processor is the final process step. Its purpose is to log all written packets of the queue in a specified format for further analysis. The key idea of MADCAT is to get an insight if attackers have access to a certain network. In contrast to T-Pot, we do not want to know what specific attacks are operated on our honeypot. Instead, we do want to ensure that no one else than authorized users have access. Especially in high confidential areas, no attacker should be capable of sending even a single packet to a host in the network. Tracking packets on a detailed level is not provided by the vast range of honeypots.

In addition, we will deploy a T-Pot instance to have comparison data to our new concept. We focus on the `129.206.218.0/24` and `147.142.0.0/16` subnet. The `129.206.218.0/24` subnet is used within University Computing Center Heidelberg building. Every client in the building has a compelling connection in this subnet. Otherwise, an Internet connection would not be feasible. The subnet `147.142.0.0/16` connects clients to "eduroam"[2]. Like the four stages of the institute firewall, the "eduroam" network, also called "Tagungslan", builds various permits into the subnet. One essential difference is that services like SMTP and HTTP are not allowed, so that, attackers are not able to deploy traps for users. Moreover, each client is encapsulated in its own subnet that disables any communication to other clients. Our instances are located in the building with IP addresses `129.206.219.65` and `129.206.219.88`. Figure 4.2 outlines our concept using MADCAT and a separate instance to visualize our data. The first instance with IP address `129.206.5.157` provides Kibana, and Elasticsearch to visualize and crawl logs. The honeypot with IP address `129.206.5.88` consists of MADCAT in conjunction with P0f, Suricata, and FATT. Like T-Pot, we use Logstash to forward our data to Elasticsearch. One benefit is the centralized approach to store data. This allows us to randomly deploy more instances to collect data from other zones.

4.3 Results

Our MADCAT (28th of October till 17th of November) and T-Pot (16th of November till 6th of December) have been deployed for three weeks respectively. All instances had a connection to both subnets. First, the results obtained in the subnet

[2] The eduroam is an international Wi-Fi internet access point for researchers.

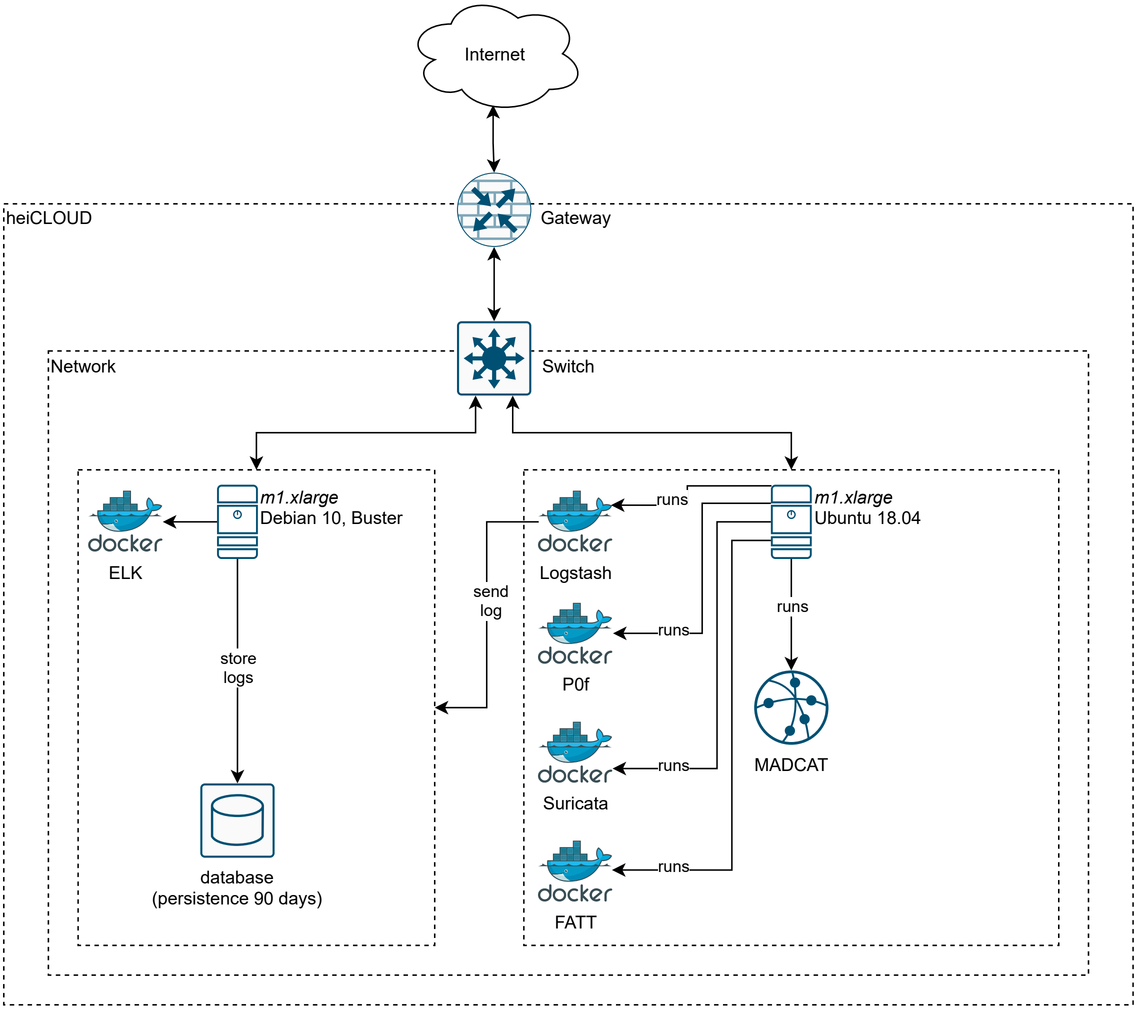


Figure 4.2: Concept to detect connection attempts. Kibana and Elasticsearch are deployed in heiCLOUD.

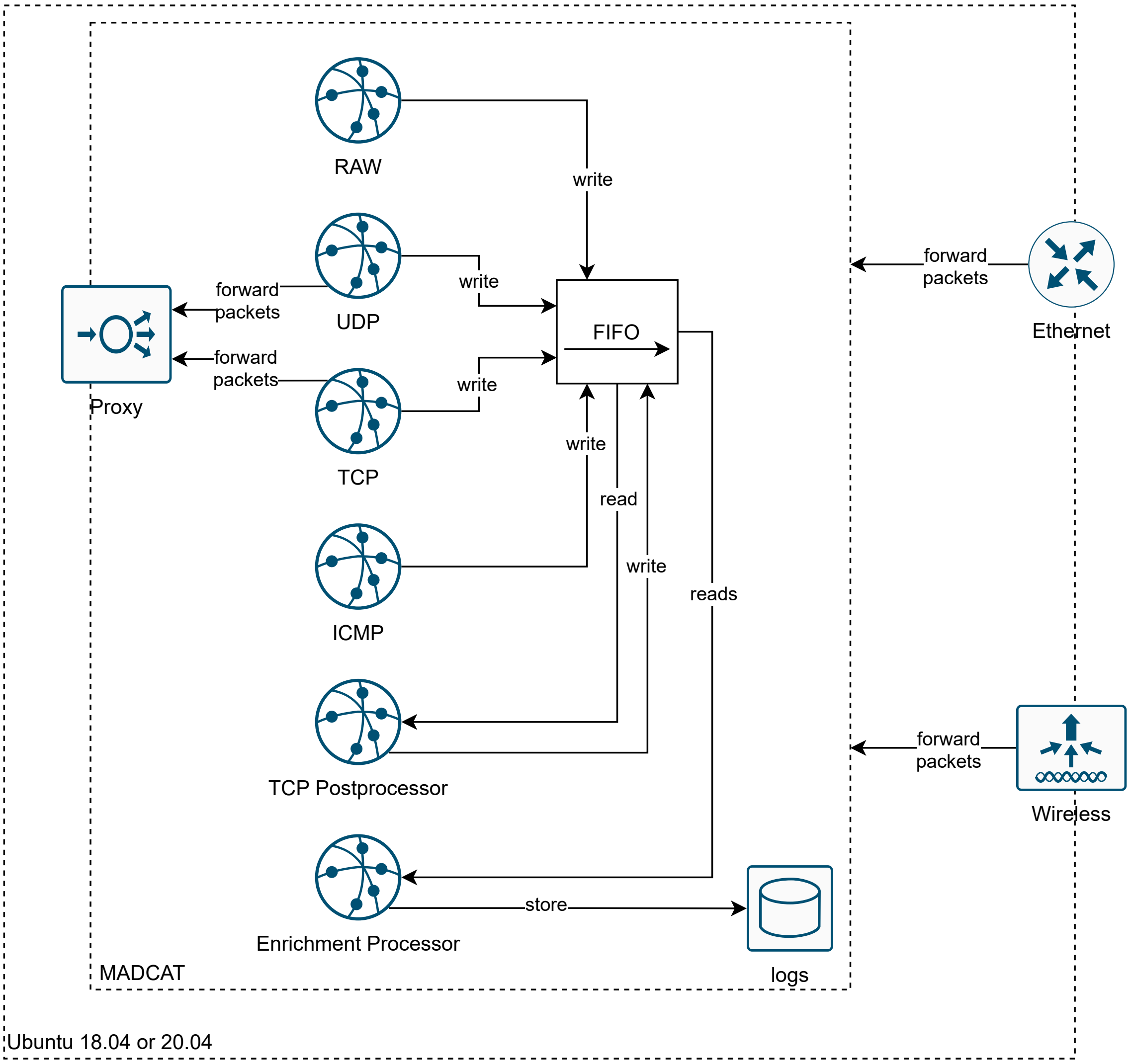


Figure 4.3: Visualization of the MADCAT packet flow starting at the network interface. The Ethernet and wireless interface forwards packets to the desired module.

`129.206.218.0/24` will be presented, closing up with the ones we claim in the eduroam network.

In total, MADCAT received 35,372 packets. Overall, the modules TCP (66.62%) and RAW (33.26%) received the majority of all connection attempts. The minority with less than one percent are suspicious packets which have individual TCP flags like reset or syn set. On the contrary, we could not identify any harmful activity based on these packets. Overall, ConPot (56.98%), Honeytrap (31.35%), and Dionaea (7.09%) received most of the attacks with a total number of 437. Interestingly, we could identify SNMP connections which are used by print servers to discover printers.

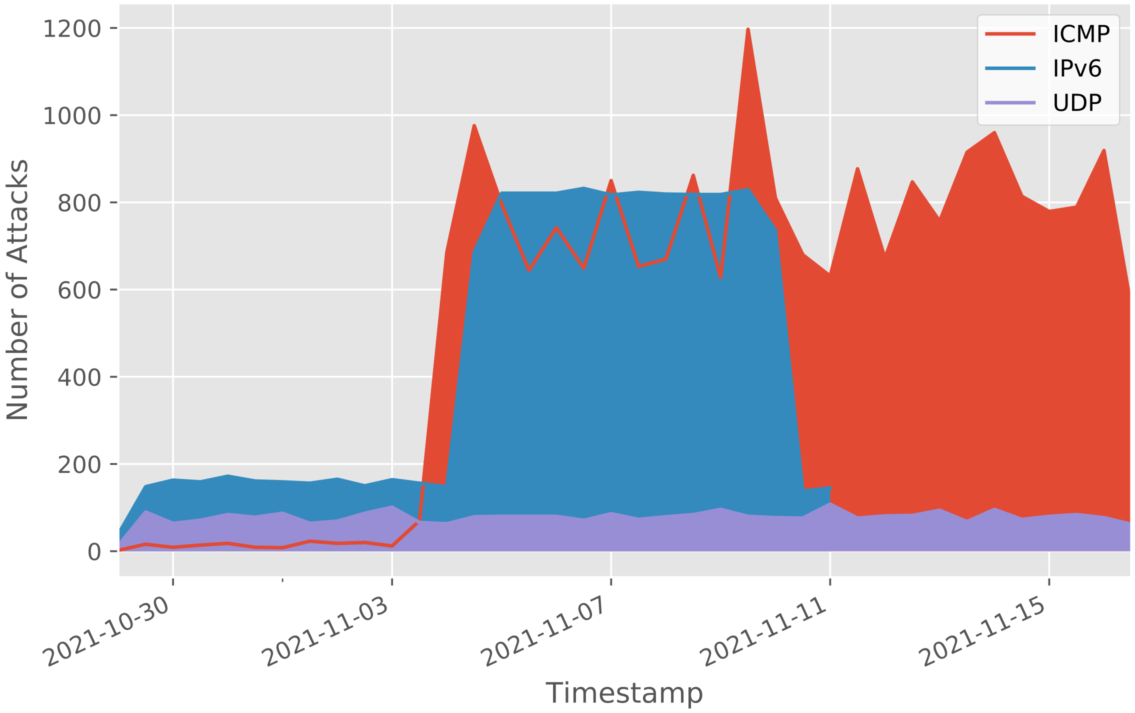


Figure 4.4: Protocol distribution of MADCAT. ICMP, IPv6 and UDP are the most used protocols. Timestamp; 28th of October to 17th of November.

Figure 4.4 shows the protocol distribution indicating a high amount of ICMP and IPv6 packets. Only 11.59% of all IP address reputations could be resolved, splitting up into known attacker (11.26%), mass scanner (0.14%), bad reputation (0.12%), and tor exit node (0.08%). Focusing on TCP packets, 88.3% are known attackers with source port 113 as the main target. The port 113 is officially known as the Identification Protocol (IDENT)[37] used for identification/authorization on a remote server such as POP, IMAP, and SMTP. Comparing the results with the stateless firewall settings, we spot a potential leak that allows adversaries to send IDENT requests to our network. Decoding the payload of these TCP packets shows that attackers rather used this port to get an SMB connection than deploying IDENT protocol attacks. We identified attempts to acquire an SSH session, using SMB and SIP connection attempts, as well as various HTTP requests. For example, we take a closer look at two payloads that have been sent to our instance. Listing 4.1 outlines an SIP probe that checks if any VoIP service is active by answering the request packet. Next, Listing 4.2 shows an SMB probe trying to achieve the same. The IP address reputation could help to answer if these packets are sent by a real user or an attacker. Considering our two examples, both IP addresses were resolved as known attacker, thus, we identified them as a probe packet before executing their attack. A vital security interest in port 113 is negligible, however, our concept helps to detect such leaks especially when stateless firewalls are the main doorkeeper for packets.

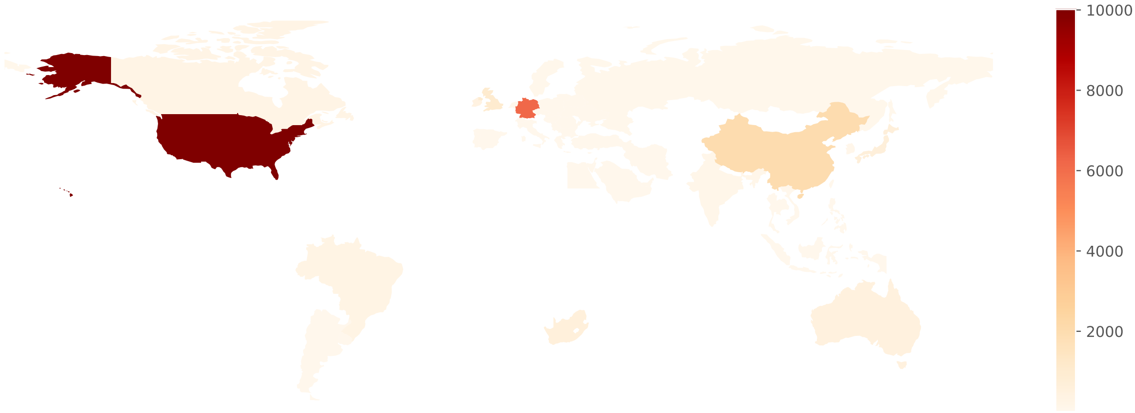


Figure 4.5: Attack distribution of MADCAT. USA, Russia, China, and Germany are the most attacking countries. Timestamp; 28th of October to 17th of November.

Figure 4.5 shows the attack distribution indicating the origin of an IP address. Most of the connections originate from the United States, Germany, and China. As we have shown beforehand in chapter 3, geographical information only outlines the last known location of a node. Like our results in heiCLOUD, we assume that this information is not reliable as an indicator from where attacks take place. Nevertheless, it is an interesting insight to see where the last node originated from.

Suricata identified odd behaviors in the network (Figure 4.6). In total, it detected 292,953 alerts and CVEs. Besides minor alerts like NMAP scans, Suricata registered alerts in SNMP requests, TCP stack, and Wind River VxWorks. CVE-2020-11899 [19] accounts nearly 99% with a total number of 102,838. This CVE is one of 19 others forming the “Ripple20” vulnerability in the low-level TCP/IP library developed by Treck, Inc.. One of the task of the Track TCP/IP stack is to reassemble fragmented packets. Whenever a fragmented packet arrives, the stack tries to validate the total length in the IP header. If the total length is not correct, it trims the data. However, this leads to an inconsistency, and thus, resulting in a buffer overflow when someone sends fragmented packets through tunnel. A detailed de-

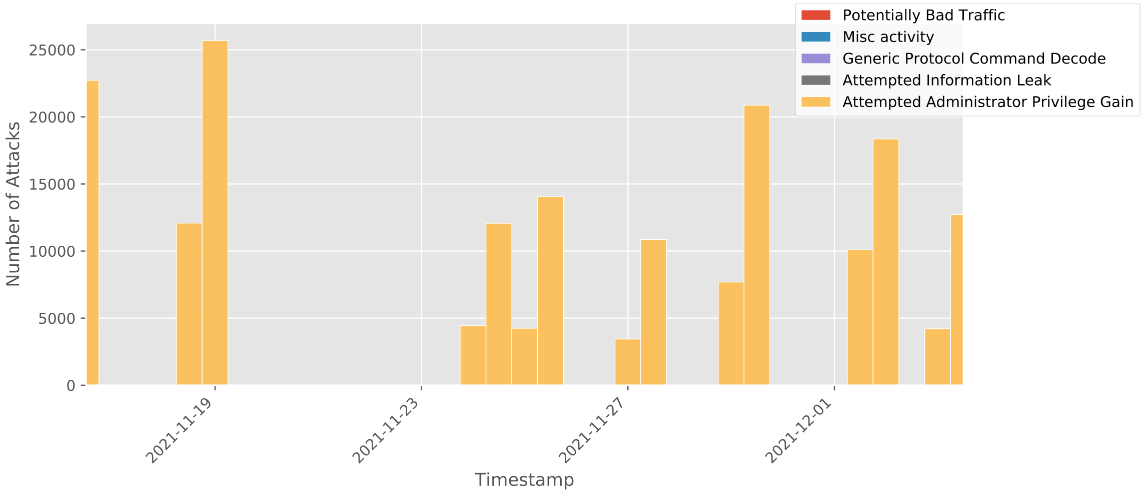


Figure 4.6: Suricata results of T-Pot. Timestamp; from 16th of November to 6th of December

scription of the vulnerability can be found in [43]. Adversary could send malformed IPv6 packets that cause an Out-of-bounds Read which results in a potential remote code execution. Only TCP/IP stack versions until 6.0.1.66 are affected by this vulnerability. Nevertheless, the tremendous amount of alerts shows the importance of adapting the IPv6 permits. Second most recorded vulnerability with the highest score is CVE-2002-0013 [12] that allows remote attackers to cause a denial of service or gain privileges in the SNMPv1 protocol. The root cause for the CVE alert is the usage of the default public community for broadcast requests instead of configuring a private community with mandatory authentication. To compromise SNMP, attackers have to have access to the network. However, the university firewall blocks SNMP port 161 and 162 for TCP and UDP, thus, restricting any access from outside. If adversaries plan to deploy an attack on the SNMP protocol, they need to have a connection to the internal network. Acquiring such a connection is rather hard to accomplish without any credentials. On the contrary, all connection attempts registered by our concept have been made within the network, and they do reflect a normal SNMP communication. Lastly, Wind River VxWorks 6.9.4 and vx7 in CVE-2019-12263 [17] cause a buffer overflow due to the underlying TCP component that results in a race condition. Each connection attempt with CVE-2019-12263 is originated from Russia. Hence, we assume that the source IP address had a vicious intention to send an urgent flag. However, for the other CVEs the IP reputation could not be resolved.

Results from our T-Pot instance are exiguous, and in short, no real attacks such as shell exploitation have been performed. Overall, all connection attempts originated from Germany within the same network, and are made on port 161 and 4567. Conpot registered minor SNMPv2 Get, SNMPv1 Get, and GetNext requests. A possible attack vector could be an SNMP reflection/amplification attack. As previ-

Listing 4.1: MADCAT connection attempt to exploit SIP connection. Received on the 16th of November. IP reputation: known attacker. Location Germany.

```
OPTIONS sip:nm SIP/2.0 Via: SIP /2.0/TCP nm;
branch=foo From: <sip:nm@nm>;
tag=root To: <sip:nm2@nm2> Call-ID: 50000 CSeq: 42
    ↪ OPTIONS Max-Forwards: 70 Content-Length: 0 Contact:
    ↪ <sip:nm@nm> Accept: application/sdp
```

ously discussed, our assumption is that devices within the network have a misconfigured printer and send broadcast requests frequently to find the machines. These SNMP requests affiliate with day-to-day traffic in an internal network, and thus, are not suspicious. Second most attacked honeypot is Honeytrap which received numerous packets on different ports whereas 39% evince an empty payload. All of these received packets have a resolved IP address in the subnet `129.206.0.0/16`. It remains unclear if these connections are malicious or are acquired by accident. Investigating the payload of outliers does not confirm the assumption of a vicious intention. Thus, declaring these results as negligible. Overall, most of the connection attempt received by our instance are from these IP addresses: `129.206.217.118`, `129.206.218.23`, and `129.206.218.194`.

Lastly, we consider the results from the eduroam network. Neither T-Pot nor MADCAT could identify any significant behavior over a period of three weeks. Unlike the subnet `129.206.218.0/24`, our honeypot did not register any suspicious packets, TCP flags, or other CVEs. In retrospect, the eduroam configuration has shown to work as designed. Thus, our client seemed to be encapsulated from others, and receives no other packets.

Besides the subtle output we have received, our results have given an insight of the value of honeypots in a restricted network zone. For the Heidelberg University, using honeypots to evaluate their stateless firewall have never been considered. We have shown that our initial concept delivered minor findings in the subnet `129.206.218.0/24` with stage 1 firewall. As a result the port 113 used for the IDENT protocol will be removed in the future to reduce the attack surface, thus, contributing to the firewall definition. Overall, our two instances received numerous packets containing interesting payloads. In comparison to our T-Pot which has been used in heiCLOUD, our results are as expected delicate and data analysis turns out to be more detailed. The statement from Spitzner [64] that honeypots only receive little input and nearly every input is suspicious matches our results only half way. As shown beforehand, our results are dramatically little, however, only a few requests seemed suspicious. Nonetheless, we could answer our initial question if attackers have any access to the restricted network zone at the Heidelberg University.

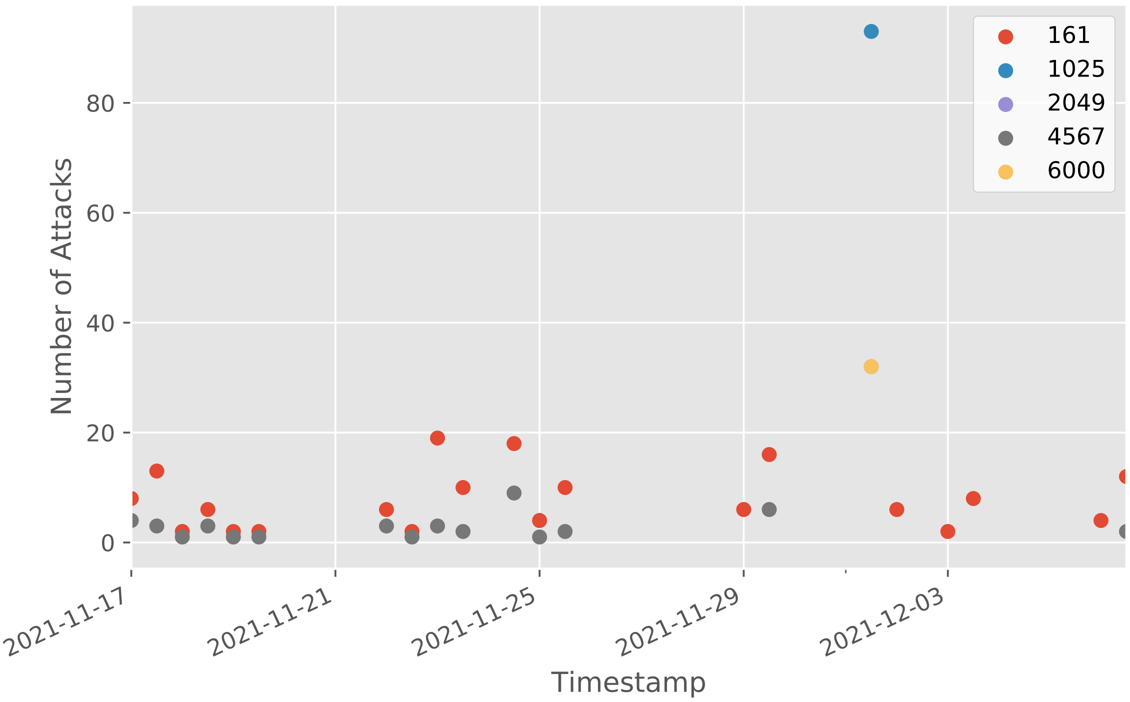


Figure 4.7: Attack port histogram of T-Pot. Timestamp; from 16th of November to 6th of December

Listing 4.2: MADCAT connection attempt to exploit SMB connection. Received on the 16th of November. IP reputation: known attacker. Location Germany.

```
PC NETWORK PROGRAM 1.0 MICROSOFT NETWORKS 1.03 MICROSOFT NETWORKS 3.0 LANMAN1.0 LM12X002 Samba NT LANMAN 1.0 NT LM 0.12.
```

4.4 Discussion

We have seen that honeypots do help to find potential leaks in restricted network zones. Though, it remains questionable if our concept is capable of delivering accurate results. Our instance has been running for three weeks in the two different subnets. To deliver any meaningful data, our honeypot has to be detected as a vulnerable target. However, we could not detect any large scans on our instance, thus, it is very likely that either an attacker could not find our instance or no one had any kind of access. In the eduroam network large scans are negligible due to the firewall permits. We assume that the results are accurate, and do not show any discrepency. Considering the subnet with stage 1 institute firewall, we identified attacks on port 113 resulting in an adaption of the stage 1 permits. On the contrary, we could not register any other odd packets on other ports. We assume that a detailed investigation could resolve upcoming questions regarding the accuracy. Especially, an exclusion of the honeypot to permit any kind of communication would help us to compare our results. Such an approach has been done in chapter 3 to prove that certain protocols are blocked in heiCLOUD.

Chapter 5

Mitigate Fingerprint Activities of Honeypots

> There is a generic weakness in the current generation of low- and medium-interaction honeypots because of their reliance on off-the-shelf libraries to implement large parts of the transport layer.
>
> *Alexander Vetterl*

Detecting honeypots before launching attacks helps to avoid disclosure of information. In chapter 3, we have seen that bot activities are on the rise, and more attacks than ever have been launched. However, the vast majority of attacks have been identified to be repetitive. In this chapter, we will conduct two experiments related to the question if it is possible to fingerprint honeypots. First, we want to reproduce the findings that Vetterl [71] claims to prove the initial question if any fingerprint activity is feasible. Lastly, we present a concept to disguise Cowrie, and verfiy our assumption with an experiment.

5.1 OpenSSH

OpenSSH is one of the most used applications that enables SSH. Before proceeding with generic weaknesses of honeypots, we want to give a short intermezzo about OpenSSH itself.

OpenSSH consists of three major layers, namely `ssh-connection`, `ssh-userauth`, and `ssh-transport` (Figure 5.1). Last layer is the most important one because it provides the basic functionalities for crypto operations such as the key exchange and encryption.

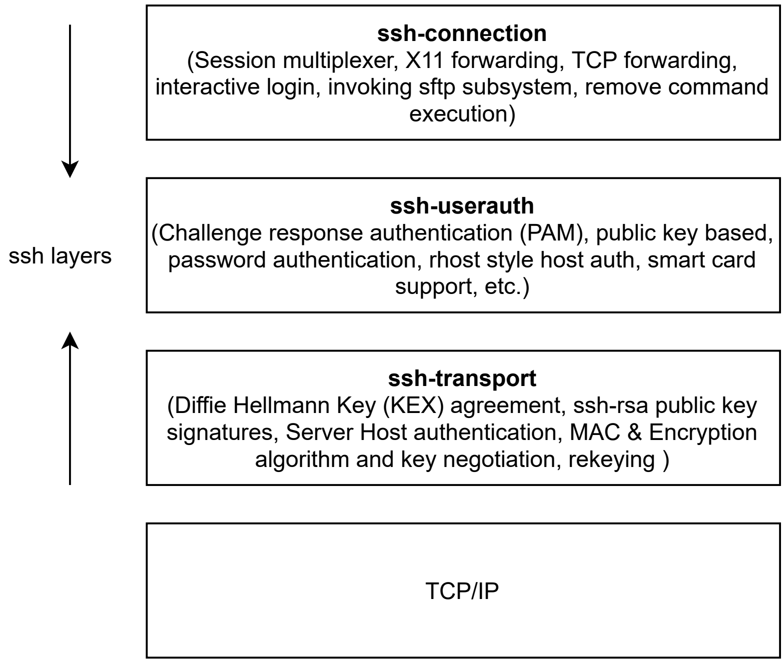

ssh-connection
(Session multiplexer, X11 forwarding, TCP forwarding, interactive login, invoking sftp subsystem, remove command execution)
ssh layers
ssh-userauth
(Challenge response authentication (PAM), public key based, password authentication, rhost style host auth, smart card support, etc.)
ssh-transport
(Diffie Hellmann Key (KEX) agreement, ssh-rsa public key signatures, Server Host authentication, MAC & Encryption algorithm and key negotiation, rekeying)
TCP/IP

Figure 5.1: OpenSSH architecture (derived from [69])

The first layer is responsible for authenticating the user to the sshd daemon. Based on two-way authentication, the client authenticates the sshd daemon by the help of the `ssh-transport`. Finally, a secure connection is established, and the key exchange is done. Next step is to authenticate the user of the client. It offers various authentication methods such as username/password, public key, or smart-card authentication. If the `ssh-userauth` layer is successful, it will establish a secure channel through the `ssh-connection` layer. Each session is handled in a so-called channel.

The `ssh-connection` layer handles multiple sessions simultaneously over a single `ssh-userauth` layer with the TCP/IP layer below. It is responsible for executing arbitrary commands, forwarding X11 connections, establishing VPN tunnels and more.

In addition, OpenSSH comes with built-in features such as keep alive messages, redirecting stdin to /dev/null for specialized X11 windows.

Figure 5.5 outlines a sample session between a client and a server. The key exchange initialization is the first message between them to negotiate all ciphers and keys for communication. For this chapter, no other than this message will be considered.

5.2 Preliminary Work

Attackers have a strong motivation to reveal honeypots before launching an attack. Without any protection attackers would disclose their methods, and thus, newly developed attacks would become useless. As shown in chapter 3, attackers do try to get information about the host system. Vetterl [71] discussed various methods of fingerprinting, however, executing commands in a shell and examining the response leaves precarious information to the honeypot itself. In his work he evaluated methods to detect honeypots at the transport level. As stated, the value of a honeypot would be merely zero if a detection on transport level would work. He presents fingerprinting methods for SSH, Telnet, and HTTP/Web. Due to the complexity of each method, we focus on SSH fingerprinting using the honeypot Cowrie. The idea to detect SSH honeypots is to look for deviations in the response. Therefore, Vetterl [71] sends a set of probes $P = \{P_1, P_2, \ldots, P_n\}$ to a given set of implementations of a network protocol $I = \{I_1, I_2, \ldots, I_n\}$ and stores the set of responses $R = \{R_1, R_2, \ldots, R_n\}$. For the given set of responses he calculated the cosine similarity coefficient C. Goal is to find the best P_i where the sum of C is the lowest. Figure 5.2 presents these steps.

Cosine similarity outputs the similarity between vectors of numerical attributes. It is widely used in text semantics to measure the similarity of sets of information such as two sentences. Vetterl [71] outlines that it can be used in "traffic analysis to find abnormalities and to measure domain similarity". Mathematically, it computes the angle between two vectors. For each set of information A, we create a vector D_A. Referring to our use case with SSH, we use the response from the server as information A. If θ is the angle between D_A and D_B, then:

$$\cos\theta = \frac{D_A \cdot D_B}{\|D_A\| \|D_B\|} \tag{5.1}$$

where "·" is the dot product obtained by:

$$D_A \cdot D_B = \sum_{i=1}^{n} (D_{A_i} \times D_{B_i}) \tag{5.2}$$

and $\|D_A\|$ (resp. $\|D_B\|$) is the Euclidean norm, obtained by $\sqrt{\sum_{i=1}^{n} D_{A_i}^2}$ (resp. $\sqrt{\sum_{i=1}^{n} D_{B_i}^2}$). The values of vectors are non-negative. The similarity between items is the value $\cos\theta$, $\cos\theta = 1$ indicates equality.

In order to find the best P_i for SSH, Vetterl [71] first created different SSH version strings based on the format: `SSH-protoversion-swversion SP comment crlf`. He

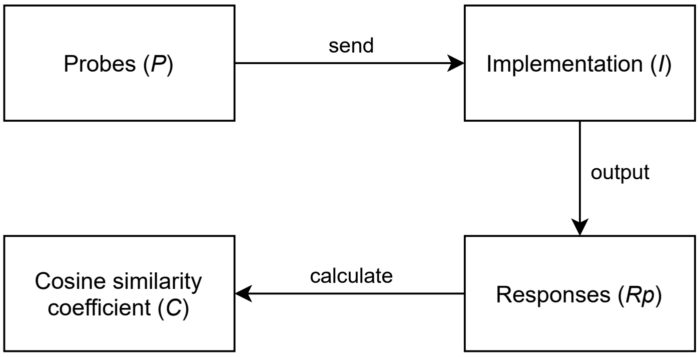


Figure 5.2: Outline to obtain the cosine similarity coefficient (derived from [71]).

Listing 5.1: OpenSSH connection attempt with probed SSH packet. All non-essential debug information have been removed to lay emphasis on the modified key exchange initialization.

```
Local version string SSH-2.2-OpenSSH
SSH2_MSG_KEXINIT sent
SSH2_MSG_KEXINIT received
kex: algorithm: ecdh-sha2-nistp521
kex: host key algorithm: ssh-dss
kex: server->client cipher: blowfish-cbc@openssh.com MAC: <implicit> compression: zlib@openssh.com
kex: client->server cipher: blowfish-cbc@openssh.com MAC: <implicit> compression: zlib@openssh.com
```

used different lower and upper case variations, 12 different protoversions ranging from 0.0 to 3.2, swversion set to “OpenSSH” or empty string, comment set to “FreeBSD” or empty string, and crlf to either `\r\n` or empty string. In total, summing up to 192 client version strings. Second, he created different `SSH2_MSG_KEXINIT` packets with 16 key-exchange algorithms, 2 host key algorithms, 15 encryption algorithms, 5 MAC algorithms and 3 compression algorithms. In total, he sent 58,752 `SSH2_MSG_KEXINIT` packets. Combining them with the 192 client versions, he ended up sending 157,925,376 packets. The version string `SSH-2.2-OpenSSH \r\n` and the `SSH2_MSG_KEXINIT` packet including ecdh-sha2-nistp521 as key-exchange algorithm, ssh-dss as host key algorithm, blowfish-cbc as encryption algorithm, hmac-sha1 as mac algorithm and zlib@openssh.com as compression algorithm, with the wrong padding result in the lowest cosine similarity coefficient C. Listing 5.1 shows the SSH debug information with the modified version string, and key exchange message.

Table 5.1 has been derived from Vetterl [71] to present his results of the cosine similarity of OpenSSH, Twisted, and Cowrie. Twisted has been added to have an example with an older SSH honeypot. We can see that it differs fundamentally from

OpenSSH. At most, it scores 0.52 whereas various OpenSSH versions start at 0.98. The number of hosts significantly decreases with a cosine similarity score of 0.90 and higher. Cowrie responses are not too far away to OpenSSH with an average of 0.80. However, scanning through the web with a minimum score of 0.90 and higher would exclude all honeypots. Thus, distinguishing Cowrie from OpenSSH with SSH packets is a feasible method. Moreover, Vetterl [71] performed an Internet-wide scan, and detected 758 Kippo and 2,021 Cowrie honeypots. These results show that the values of honeypots would decrease to zero when large fingerprinting activities are used.

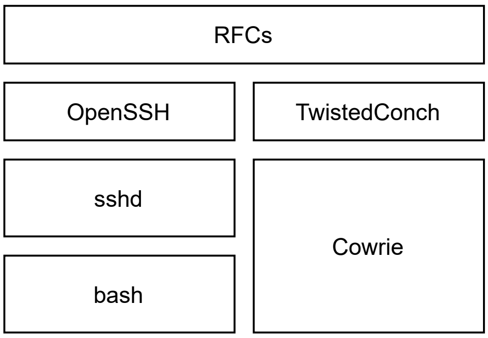


Figure 5.3: Architecture of OpenSSH and Cowrie. OpenSSH and TwistedConch have subtle differences (derived from [71])

Vetterl [71] states that current low- and medium-interaction honeypots have a generic weakness due to the underlying off-the-shelf libraries. Cowrie is based on TwistedConch[1], a Python 2/3 library that implements the SSH protocol. Any bash command and its response are tweaked by Cowrie, and thus, resulting in a discrepancy to OpenSSH. For example, Cowrie version 1.1.0 missed `tftp`[2] that later came with version 1.2.0. Therefore, it is a continuous struggle of adding new commands to avoid early disclosures of Cowrie.

Figure 5.3 shows the difference between OpenSSH and Cowrie. Both have to fulfill the RFC4250 [45] which defines the protocol. OpenSSH and TwistedConch implement the RFC requirement. As an example, Vetterl [71] found that Cowrie used to have random bytes for the `SSH2_MSG_KEXINIT` packet[3]. With respect to RFC4253 [75] that defines the Binary Packet Protocol (BPP) of SSH, the random padding is used to solidify the total length of the packet to be a multiple of the cipher block size. The RFC in section 6 defines that the padding have to consist of 4 random bytes. Based on the statement of the OpenSSH authors, random bytes have been changed to `NULL` characters due to no security implications. Thus, an adversary could have detected a Cowrie honeypot with a single `SSH2_MSG_KEXINIT` packet.

[1] TwistedConch 12.0.0 on GitHub

[2] Trivial File Transfer Protocol (TFTP) is a lockstep File Transfer Protocol

[3] Each packet consists of the packet and padding length, the Message Authentication Code (MAC), a payload, and a random padding.

Table 5.1: Overview of the cosine similarity of OpenSSH, Cowrie, and Twisted

		A	B	C	D	E	F	G	H	I	J
OpenSSH 6.6	A	-	0.98	0.98	0.94	0.94	0.42	0.78	0.79	0.79	0.79
OpenSSH 6.7	B		-	0.98	0.98	0.98	0.41	0.80	0.81	0.81	0.80
OpenSSH 6.8	C			-	0.96	0.96	0.42	0.78	0.79	0.79	0.79
OpenSSH 7.2	D				-	0.98	0.42	0.80	0.80	0.80	0.80
OpenSSH 7.5	E					-	0.42	0.78	0.79	0.79	0.79
Twisted 15.2.1	F						-	0.50	0.51	0.51	0.52
Cowrie 96ca2ba	G							-	0.98	0.98	0.98
Cowrie dc45961	H								-	0.99	0.99
Cowrie dbe88ed	I									-	0.99
Cowrie fd801d1	J										-

Nowadays, Cowrie adapted itself to have `NULL` characters as padding to mitigate such an exploit. However, these subtle differences give adversaries precautionary information and influence the cosine similarity coefficient.

5.3 Experiment 1: Reproduce Vetterl et al.'s findings

First, the reproduction of the outdated OpenSSH library that Vetterl [71] used will be investigated. In his work he used OpenSSH 7.5$P1$ which deviates from the latest version 8.8$P1$. Older versions rely on OpenSSL 1.0.2 which includes outdated algorithms and functions. For the `SSH2_MSG_KEXINIT` packet, the encryption algorithm blowfish-cbc has been removed with version 7.6$P1$, and thus, are outdated. Building the version 7.5$P1$ requires the libraries libssl (1.0.2), libssl-dev (1.0), libssh-dev ($0.7.3-2$), and libssh-4 ($0.9.6-1$). All of these libraries are outdated, and have been removed from any Debian installation. Using the latest versions of these libraries result in errors such as missing encryption algorithms and host key algorithms. Thus, replacing the libraries is a necessary task. It required to download the libraries, remove the current versions, and install the outdated ones. The version 7.5$P1$ allows to modify the proposal of the key exchange initialization message in a single file. On the contrary, this has been removed starting from version 7.6$P1$. After compiling the application, we test its behavior with a Debian 11 Buster and a Debian Jessie 9 Docker image. Both are new machines that have no other packages installed than the OpenSSH daemon (`sshd`). Debian 11 uses the latest OpenSSH version whereas

Listing 5.2: OpenSSH connection attempt for version 7.5P1 and 8.8P1 with probed key exchange initialization message. All non-essential debug information have been removed to lay emphasis on the modified key exchange initialization.

```
OpenSSH_7.5p1, OpenSSL 1.0.2u  20 Dec 2019
Local version string SSH-2.2-OpenSSH
SSH2_MSG_KEXINIT sent
SSH2_MSG_KEXINIT received
kex: algorithm: ecdh-sha2-nistp256
kex: host key algorithm: ssh-dss
Unable to negotiate with ::1 port 22: no matching cipher found. Their offer: aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,chacha20-poly1305@openssh.com
```

Jessie is at 6.7P1. These environments help us to uniquely identify variations in the protocol version.

Listing 5.2 shows the connection attempt with our adjusted version string and `SSH2_MSG_KEXINIT` packet. Both Debian machines return the same response. Using the outdated OpenSSH version 7.5P1 results in an incompatibility. OpenSSH outlines that blowfish-cbc is not supported anymore. OpenSSH kept the encryption algorithm usable for compatibility reasons for clients until 7.6P1. Later patches removed the blowfish-cbc from SSH, and thus, a reproduction of Vetterl [71] remains not feasible with state-of-the-art OpenSSH versions. However, testing it with version 7.3P1 that has been compiled on the machine results in a successful connection attempt. Vetterl [71] does not outline any expected response of OpenSSH, thus, we have to assume that a connection attempt would have been successful due to the existing ciphers during that time. Adapting OpenSSH version 8.8P1 with chacha20-poly1305 instead of blowfish-cbc for the encryption algorithm results in a successful connection attempt. Thus, we have adapted the key exchange initialization to use chacha20-poly1305 as encryption algorithm instead. Next, the DSA host key algorithms are marked as too weak, and are not included automatically during the key exchange initialization. Using ssh-dss requires the extra flag `-oHostKeyAlgorithms=+ssh-dss`. In addition, we have tested it with the ssh-ed25519 host key algorithm, and the response has been promising to probe instances. So far, the `SSH2_MSG_KEXINIT` packet with ecdh-sha2-nistp521 as key-exchange algorithm, ssh-ed25519 as host key algorithm, chacha20-poly1305 as encryption algorithm, hmac-sha1 as mac algorithm and zlib@openssh.com as compression algorithm has been successfully tested on our two Debian instances.

The most interesting question remains about Cowrie's response deviation. Vetterl [71] claims that it results in a `bad version *` exception. After testing, we

Listing 5.3: Cowrie connection attempt with probed key exchange initialization message. All non-essential debug information have been removed to lay emphasis on the modified key exchange initialization.

```
OpenSSH_8.8p1, OpenSSL 1.1.1l  24 Aug 2021
Local version string SSH-2.2-OpenSSH
SSH2_MSG_KEXINIT sent
Bad packet length 1349676916.
ssh_dispatch_run_fatal: Connection to 129.206.5.74 port
    ↪ 22: message authentication code incorrect
```

conclude that this issue has been fixed in the meantime by Cowrie, and thus, do not leak vulnerable information anymore. For instance, we use the default Cowrie implementation version $v.2.3.0$[4] of our T-Pot instance. Listing 5.3 outlines the connection attempt. Unambiguously, Cowrie results in a `bad packet length *` exception, and thus, deviates fundamentally from an OpenSSH response. The underlying off-the-shelf library TwistedConch checks if a packet is within 1,048,576 bytes (1 MB) (Listing 5.4). Any packet that exceeds that threshold causes this exception, which results in a loss of connection for the client. When Cowrie tries to get the packet of a request, this static check is performed. It remains dubious why TwistedConch has added it whenever an SSH packet has to be returned. In the RFC4253, the minimum packet size is 5 bytes whereas maximum packet size is set to 32,768 bytes (256 KB). Debugging Cowrie shows that the exception occurs during the version string validation (Listing 5.5). The server validates if the version string matches the allowed versions 1.99 and 2.0. Any higher or lower version will result in a `Protocol major versions differ.\n` exception by calling the function `_unsupportedVersionReceived`. This response would match the behavior of OpenSSH.

We have tested Cowrie and OpenSSH with the version strings 1.0, 2.0 and 2.2. As a result, Cowrie's `bad packet length *` exception occurs only when the version does not match the expected one. This result diverges from OpenSSH, as only versions under 1.99 lead to the same exception as Cowrie. For any higher version, the connection can be established successfully. We assume that Cowrie has an error in validating the version string. Debugging Cowrie shows that the method to return the `Protocol major versions differ.\n` exception is called, but our client does not receive this message. Hence, we assume that the underlying library TwistedConch is responsible for the incorrect message.

As a conclusion, these are the protocol deviations that Vetterl [71] has presented in his work. Thus, we could successfully recreate his findings by detecting Cowrie on transport level. Adversaries who modify their SSH client to send our specific version

[4] Cowrie v2.3.0 on GitHub

Listing 5.4: TwistedConch packet length validation.

```
def getPacket(self):
    ...
    if packetLen > 1048576: # 1024 ** 2
        self.sendDisconnect(DISCONNECT_PROTOCOL_ERROR,
                            'bad packet length %s' %
                               ↪ packetLen)
    return
    ...
```

string and key exchange initialization message could detect Cowrie honeypots and stop any further activity.

5.4 Attempt to Disguise Cowrie

Cowrie has to be tweaked to hide its generic weakness. Fixing the major flaws in Cowrie to avoid early detection remains an ephemeral patch. The continued use of libraries that reimplement the behavior of OpenSSH leads attackers to try to find subtle protocol differences and exclude any host machine that deviates. Such approaches could be achieved by arbitrary Internet-wide scanning and calculation of the cosine similarity coefficient. Thus, the value of honeypots would decrease to almost zero. Therefore, a new solution is required to disguise SSH honeypots. Vetterl [71] presented a solution to use OpenSSH as an intermediary instance between the attacker and Cowrie. Unfortunately, this solution is outdated, and newer versions contain significant changes in structure and functions. Our concept is based on Vetterl [71] solution, but due to newer versions available we have updated it to the latest OpenSSH version. By default, OpenSSH itself is not able to act as an intermediary, therefore we need to customize the latest OpenSSH version to enable this feature. Figure 5.4 visualizes the flow of SSH packets between an attacker and Cowrie. Cowrie is hidden in the background, and it is only accessible via the loop back address `127.0.0.1:65522`. Our updated sshd daemon is exposed to the Internet, and it is accessible via `129.206.5.157:22`. Each connection to OpenSSH is forwarded to our honeypot through a NAT rule (iptables -t nat -A PREROUTING -p tcp –dport 22 -j REDIRECT –to-port 65222). Accordingly, an attacker should not be able to detect Cowrie through response deviations.

For instance, we use the latest OpenSSH version 8.8$P1$[5]. Our implementation is based on Vetterl [71] version 6.3$P1$[6]. As mentioned beforehand, due to major dif-

[5] OpenSSH 8.8P1 on GitHub

[6] sshd-honeypot on GitHub

Listing 5.5: Cowrie version string validation. It tweaks the same results as OpenSSH.

```python
def _unsupportedVersionReceived(self, remoteVersion: bytes) -> None:
    """
    Change message to be like OpenSSH
    """
    self.transport.write(b"Protocol major versions differ.\n")
    self.transport.loseConnection()

def dataReceived(self, data: bytes) -> None
    ...
    if not self.gotVersion:
        ...
        self.otherVersionString = self.buf.split(b"\n")[0].strip()
        ...
        # Checks if the version string has a correct format
        m = re.match(br"SSH-(\d+.\d+)-(.*)", self.otherVersionString)
        if m is None:
            ...
            self.transport.write(b"Invalid SSH identification string.\n")
            self.transport.loseConnection()
            return
        else:
            ...
            # Checks if version string is either 1.99 or 2.0
            if remote_version not in self.supportedVersions:
                self._unsupportedVersionReceived(self.otherVersionString)
                return
            ...
        ...
    ...
```

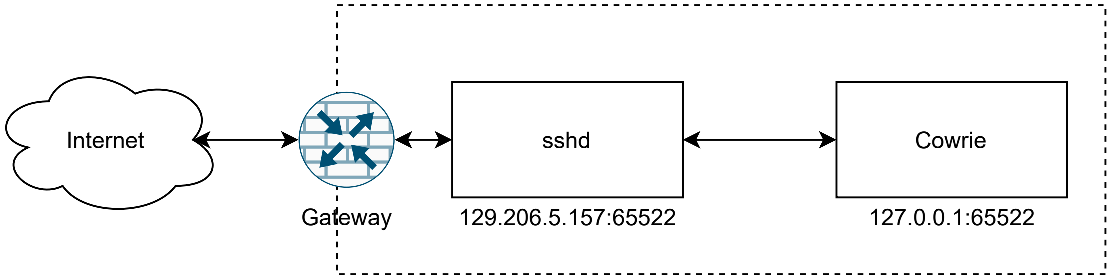


Figure 5.4: Architecture of OpenSSH and Cowrie. OpenSSH and TwistedConch have subtle differences (derived from [71])

ferences between both versions a smooth transition is unattainable without modifications. However, the basic idea to morph OpenSSH into an intermediary instance stays the same. To change the sshd daemon to work as an intermediary, a total of the following steps are required:

- A separate channel to communicate with the attacker and forward the packets to Cowrie.
- An authentication that permits any connection to Cowrie.
- Tweaking the session to write packets in the new channel.

Respectively, a detailed description of the adaption will be presented. The easiest step is to tweak the authentication in the `auth-passwd.c` to permit any session so that an incoming connection can be forwarded to Cowrie. Originally, OpenSSH checks each session to see if the chosen authentication method returns true. Our server has to return true for any client in the `allowed_user` function to skip the authentication process. Cowrie continues the authentication process, and communicates with the attacker. In OpenSSH, communications are handled in channels as seen beforehand in section 5.1. Technically, the sshd daemon opens a SOCKS connection for each session to communicate with the client. SOCKS is a network protocol to exchange packets between a server and a client. To communicate with Cowrie, the sshd daemon needs a separate channel to store the attacker's session and forward packets. The channel is implemented in version $6.3P1$ and can be used in $8.8P1$ with minor adaptions. In the main method, when sshd is started, the channel is created and a connection to the running Cowrie instance is opened so that a new session can be forwarded. If Cowrie is unavailable, the startup would fail. Thus, Cowrie has to run prior to SSH. Next, the server loop that is responsible to connect the client to the correct port has to be modified in order to put direct TCP/IP connections in our honeypot channel. Without this adaption, Cowrie would not receive any packet. The function `server_request_direct_tcpip` handles these connections. For instance, it checks if the TCP forwarding port for Cowrie is defined, and connects the current session to the respective port only if a communication to Cowrie is feasible. Lastly, we have to adapt the sshd daemon to start and set

Listing 5.6: Cowrie log information.

```
New connection: 127.0.0.1:65522 [session: 2ca9a619ceb8]
Remote SSH version: SSH-2.0-libssh_0.9.6
SSH client hassh fingerprint: ....
kex alg=b'curve25519-sha256' key alg=b'ssh-ed25519'
outgoing: b'aes256-ctr' b'hmac-sha2-512' b'none'
incoming: b'aes256-ctr' b'hmac-sha2-512' b'none'
connection lost
```

up the channel. In addition, for easy configuration of the daemon, an extension of the configuration to set the Cowrie IP address and port has been included. After compiling our version, a brief test proofed a valid connection to the sshd daemon.

5.5 Experiment 2: Avoid fingerprinting of Cowrie

The last experiment to conclude this chapter is to test if our concept helps to disguise Cowrie and avoid fingerprinting based on a custom local string version and key exchange initialization message. For instance, we have used Vetterl [71] original 6.3P1 sshd honeypot, as well as the newly implemented version 8.8P1 with Cowrie version 2.3.0. Both versions have been tested in heiCLOUD and locally in an encapsulated environment. All requests do not result in a bad packet length, and thus, are behaving like an original sshd daemon. Only the older version 6.3P1 had problems to coup with new encryption and host key algorithms. Thus, the concept is capable of forwarding any related packet to Cowrie, and hiding the generic weakness of TwistedConch. In Listing 5.6, Cowrie receives the connection and log information respectively. Currently, one downside is the connection loss that happens due to timeout restrictions. This issue is a minor bug, and could be fixed in the future. As a conclusion, this experiment has shown that the initial idea of hiding Cowrie in the background and directing the message through OpenSSH prevents fingerprint activities of any adversary.

5.6 Discussion

Depending on the interaction-level, honeypots will always deviate from production instances. As we have seen in the two experiments beforehand, detecting a generic weakness is doable in a respective time, as well as mitigating it. Thus, finding and fixing weaknesses of honeypots becomes a continuous cycle. However, this chapter also outlined the importance of the libraries that are used. TwistedConch is the bottleneck of Cowrie and has not been updated for 10 years. Libraries that reimplement protocols have to be always up-to-date. As a conclusion, such libraries should be chosen carefully to avoid any bugs which leaves harmful information to attackers.

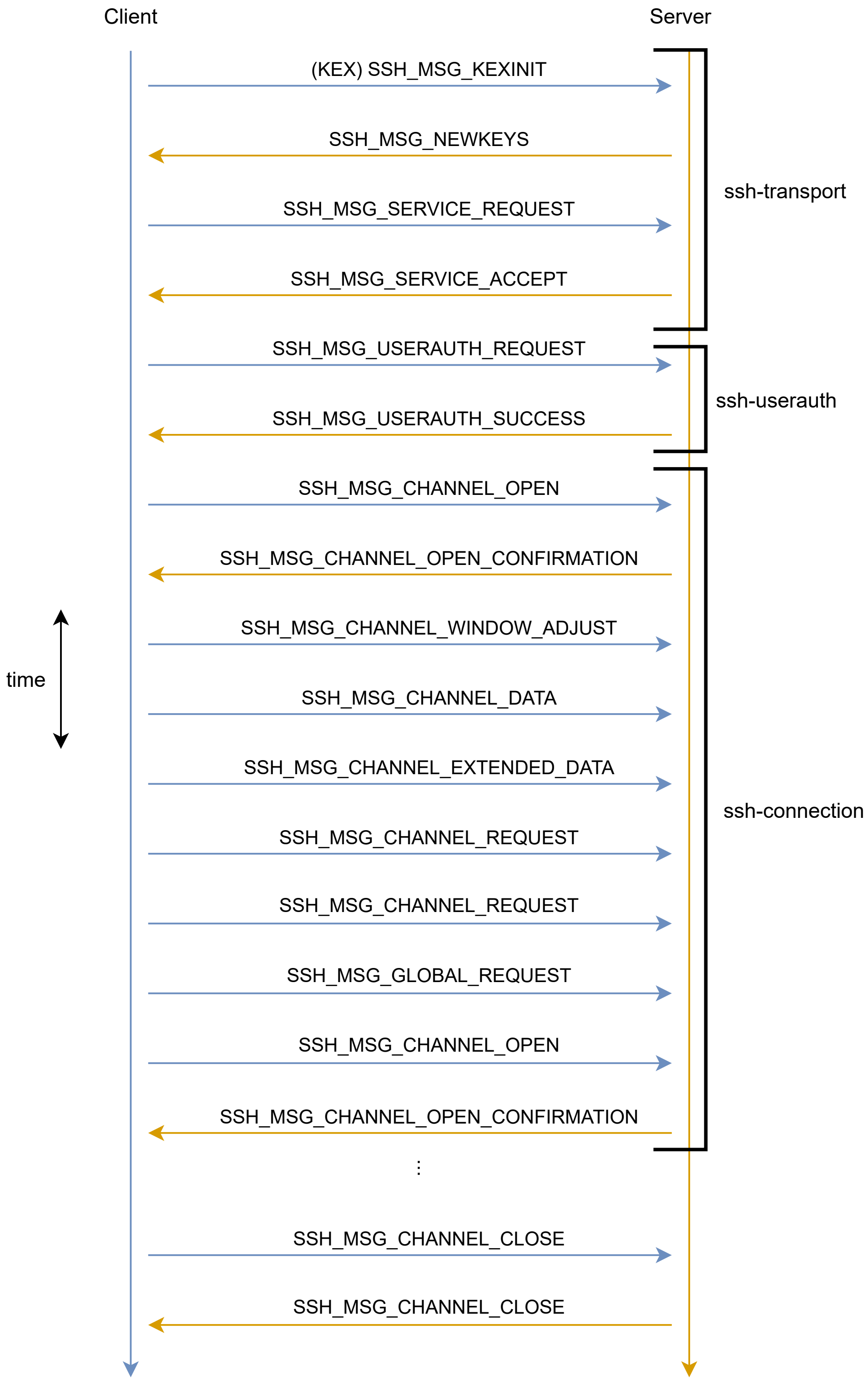


Figure 5.5: OpenSSH sample session flow diagram (derived from [69]). In addition, on the right side indicates the layers that are responsible for handling the messages.

Chapter 6

Conclusion

We have shown that cybercrime activities can be spotted by using honeypot solutions. Thus, the initial question if honeypots contribute to a more secure infrastructure has been answered successfully. We can confirm this assumption based on our results we have achieved in the cloud and in the university network. Our first approach was to collect data by the help of the T-Pot solution and compare them to a previous study of similiar cloud providers. It has shown that these activities increased significantly. Considering the services that are excluded by default of the firewall, heiCLOUD has received more attacks than ever, putting it on the first place compared to other cloud providers. We have seen various attacks such as in RDP, VoIP, and SSH. Outstanding are the amount of cryptocurrency related attacks which reflects the current situation of highly traded GPUs on the market. In addition, the latest attacks like the Apache vulnerability in version 2.49.0 could be traced back to very early stages, showing how fast attackers adapt to new vulnerabilities. Our assumption is that the vast majority of executed attacks on our instance originated from bots.

Next, we have focused at the internal network of the university and implemented a new concept to detect every single packet that have been sent to a host machine. The MADCAT solution in conjunction with IDS tools helped us to identify the open port 113 that has been used to deploy attacks. We have shown that known attackers with an IP address originating from Russia have probed our instance, and we assume further attacks would have been carried out. In retrospect, this helped to remove the port from the permits in the firewall, thus, we have improved the security at the Heidelberg University. Any other suspicious behavior in the eduroam network could not be registered, proving that the firewall works by design.

Moreover, we have shown that honeypots like Cowrie have a fundamental flaw because they rely on off-the-shelf libraries. These libraries often re-implement protocol behaviors like OpenSSH, and adding a subtle difference to the response. On the contrary, this deviation of responses can be used to detect honeypots on transport level. Based on a cosine similiartiy coefficient, adversaries could spot honeypots before deploying any attack, thus, avoid exposures of newly developed attacks. We have re-created the findings Vetterl [71] claims in his work by adapting OpenSSH $8.8P1$, and testing it on different Debian instances. Due to outdated algorithms, we have updated the key exchange initialization message to work with the latest version. We have shown that the lastest Cowrie version 2.3.0 results in a bad packet length because the local version string does not match the expected ones of the underlying library TwistedConch. This result deviates fundamentally from OpenSSH. Lastly, we tried to protect Cowrie from early exposure by hiding it in the background and tunneling requests through a customized OpenSSH daemon. We have successfully fixed the generic weakness of Cowrie, so that, connecting to Cowrie works without running into a bad packet length error. Our last chapter shows that honeypots are not flawless, and developers should be careful during the decision of additional libraries.

In conclusion, we have presented concepts to catch attackers for different scenarios, and shown that cybercrime activities have increased tremendously. In addition, we have taken a deep dive into an edge-breaking study to detect honeypots on transport level, and have disguised Cowrie to block such activities. Thus, using honeypots as a security parameter have been proven promising for further scientific research.

Bibliography

[1] Fahim Abbasi. 2020 trustwave global security report. *Trustwave*, 2020.

[2] John B. Althouse, Jeff Atkinson, and Josh Atkins. JA3 - a method for profiling ssl/tls clients. `https://github.com/salesforce/ja3`, 2021. Accessed: 2021-09-26.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, April 2010. doi: 10.1145/1721654.1721672. URL `https://doi.org/10.1145/1721654.1721672`.

[4] Vesselin Bontchev. Elasticpot: an elasticsearch honeypot. `https://gitlab.com/bontchev/elasticpot`, 2021. Accessed: 2021-09-26.

[5] Vesselin Bontchev. Ipphoney: an internet printing protocol honeypot. `https://gitlab.com/bontchev/ipphoney`, 2021. Accessed: 2021-09-26.

[6] BSI. Die lage der it-sicherheit in deutschland 2021. Technical Report BSI-LB21/510, Bundesamt für Sicherheit in der Informationstechnik, Sep 2021. URL `https://www.bsi.bund.de/DE/Service-Navi/Publikationen/Lagebericht/lagebericht_node.html`.

[7] Bill Cheswick. An evening with berferd in which a cracker is lured, endured, and studied. In *In Proc. Winter USENIX Conference*, pages 163–174, 1992.

[8] Gabriel Cirlig. ADBHoney. `https://github.com/huuck/ADBHoney`, 2021. Accessed: 2021-09-26.

[9] Theo Combe, Antony Martin, and Roberto Di Pietro. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5):54–62, 2016. doi: 10.1109/MCC.2016.100.

[10] CSEC. Information technology security guideline. Technical Report ITSG-38, Communications Security Establishment Canada, May 2009. URL `https://cyber.gc.ca/sites/default/files/publications/itsg-38-eng.pdf`.

[11] CVE-2001-0540. CVE-2001-0540. Available from MITRE, CVE-ID CVE-2001-0540., March 09 2002. URL `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0540`.

[12] CVE-2002-0013. CVE-2002-0013. Available from MITRE, CVE-ID CVE-2002-0013., February 13 2002. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0013.

[13] CVE-2005-4050. CVE-2005-4050. Available from MITRE, CVE-ID CVE-2005-4050., December 07 2005. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4050.

[14] CVE-2006-2369. CVE-2006-2369. Available from MITRE, CVE-ID CVE-2006-2369., May 15 2006. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2369.

[15] CVE-2012-0152. CVE-2012-0152. Available from MITRE, CVE-ID CVE-2012-0152., December 13 2011. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0152.

[16] CVE-2018-0101. CVE-2018-0101. Available from MITRE, CVE-ID CVE-2018-0101., November 27 2017. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-0101.

[17] CVE-2019-12263. CVE-2019-12263. Available from MITRE, CVE-ID CVE-2019-12263., September 07 2020. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12263.

[18] CVE-2019-19781. CVE-2019-19781. Available from MITRE, CVE-ID CVE-2019-19781., December 13 2019. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-19781.

[19] CVE-2020-11899. CVE-2020-11899. Available from MITRE, CVE-ID CVE-2020-11899., July 17 2020. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-11899.

[20] CVE-2021-42013. CVE-2021-42013. Available from MITRE, CVE-ID CVE-2021-42013., October 06 2021. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-42013.

[21] Jeff Daniels. Server virtualization architecture and implementation. *XRDS*, 16(1):8–12, September 2009. ISSN 1528-4972. doi: 10.1145/1618588.1618592. URL https://doi.org/10.1145/1618588.1618592.

[22] ddosspot. DDoSPot. https://github.com/aelth/ddospot, 2021. Accessed: 2021-09-26.

[23] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: Issues and challenges. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. IEEE, 2010. doi: 10.1109/aina.2010.187. URL https://doi.org/10.1109/aina.2010.187.

[24] dionaea. dionaea - catches bugs. https://github.com/DinoTools/dionaea, 2021. Accessed: 2021-09-26.

[25] Docker. Docker overview. https://docs.docker.com/get-started/overview/, 2021. Accessed: 2021-09-21.

[26] elasticsearch. The Elastic Stack. https://www.elastic.co/elastic-stack/, 2021. Accessed: 2021-09-26.

[27] Europol. Internet organised crime threat assessment (iocta). *European Union Agency for Law Enforcement Cooperation*, 9(1), 2020.

[28] Europol. About europol. https://www.europol.europa.eu/about-europol, 2021. Accessed: 2021-09-04.

[29] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. A survey of botnet and botnet detection. In *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, pages 268–273, 2009. doi: 10.1109/SECURWARE.2009.48.

[30] Michael Flanders. A simple and intuitive algorithm for preventing directory traversal attacks, 2019.

[31] Federal Office for Information Security. Cert-bund. https://www.bsi.bund.de/EN/Topics/IT-Crisis-Management/CERT-Bund/cert-bund_node.html, 2021. Accessed: 2021-09-12.

[32] Martin Gallo. Honeysap: Sap low-interaction honeypot. https://github.com/SecureAuthCorp/HoneySAP, 2021. Accessed: 2021-09-26.

[33] Lichstein. HA. When should you emulate. *Datamation*, 15(11):205, 1969.

[34] Brian Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, July 2008. ISSN 0001-0782. doi: 10.1145/1364782.1364786. URL https://doi.org/10.1145/1364782.1364786.

[35] Marcus Hutchins. Honepot for cve-2019-19781 (citrix adc). https://github.com/MalwareTech/CitrixHoneypot, 2020. Accessed: 2021-09-26.

[36] Yung Innanet. Hellpot. https://github.com/yunginnanet/HellPot, 2021. Accessed: 2021-09-26.

[37] Michael C. St. Johns. Identification Protocol. RFC 1413, RFC Editor, February 1993. URL https://www.rfc-editor.org/rfc/rfc1413.txt.

[38] Adel Karimi. FATT /fingerprintAllTheThings - a pyshark based script for extracting network metadata and fingerprints from pcap files and live network traffic. https://github.com/0x4D31/fatt, 2021. Accessed: 2021-09-26.

[39] Adel Karimi, Ben Reardson, John Althouse, Jeff Atkinson, and Josh Atkins. HASSH - a profiling method for ssh clients and servers. https://github.com/salesforce/hassh, 2021. Accessed: 2021-09-26.

[40] Tejvir Kaur, Vimmi Malhotra, and Dheerendra Singh. Comparison of network security tools- firewall, intrusion detection system and honeypot. In *International Journal of Enhanced Research in Science Technology & Engineering*, volume 3, pages 200–204, 2014.

[41] Christopher Kelly, Nikolaos Pitropakis, Alexios Mylonas, Sean McKeown, and William J. Buchanan. A comparative analysis of honeypots on different cloud platforms. *Sensors*, 21(7):2433, April 2021. doi: 10.3390/s21072433. URL https://doi.org/10.3390/s21072433.

[42] Mikael Keri. Dicompot - A Digital Imaging and Communications in Medicine (DICOM) Honeypot. https://github.com/nsmfoo/dicompot, 2021. Accessed: 2021-09-26.

[43] Moshe Kol and Shlomi Oberman. CVE-2020-11896 RCE CVE-2020-11898 Info Leak. Technical report, JSOF Ltd., June 2020.

[44] D Kreuter. Where server virtualization was born. *Virtual Strategy Magazine*, 2004.

[45] Sami Lehtinen and Chris Lonvick. The Secure Shell (SSH) Protocol Assigned Numbers. RFC 4250, RFC Editor, January 2006. URL https://www.rfc-editor.org/rfc/rfc4250.txt.

[46] mailoney. Mailoney - an SMTP honeypot. https://github.com/phin3has/mailoney, 2021. Accessed: 2021-09-26.

[47] P M Mell and T Grance. The NIST definition of cloud computing. Technical report, National Institute of Standards and Technology, 2011. URL https://doi.org/10.6028/nist.sp.800-145.

[48] Steve Micallef. Spiderfoot automates osint for threat intelligence and mapping your attack surface. https://github.com/smicallef/spiderfoot, 2021. Accessed: 2021-09-26.

[49] Iyatiti Mokube and Michele Adams. Honeypots: Concepts, approaches, and challenges. In *Proceedings of the 45th Annual Southeast Regional Conference*, ACM-SE 45, page 321–326, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936295. doi: 10.1145/1233341.1233399. URL https://doi.org/10.1145/1233341.1233399.

[50] Marcin Nawrocki, Matthias Wählisch, Thomas C. Schmidt, Christian Keil, and Jochen Schönfelder. A survey on honeypot software and data analysis. *CoRR*, abs/1608.06249, 2016. URL http://arxiv.org/abs/1608.06249.

[51] Marco Ochse. T-Pot. https://github.com/telekom-security/tpotce, 2021. Accessed: 2021-09-26.

[52] Michel Oosterhof. Cowrie SSH/Telnet Honeypot. https://github.com/cowrie/cowrie, 2021. Accessed: 2021-09-26.

[53] Sylvain Peyrefitte. Rdpy: Remote desktop protocol in twisted python. `https://github.com/citronneur/rdpy`, 2021. Accessed: 2021-09-26.

[54] Niels Provos. Honeyd - a virtual honeypot daemon. In *10th DFN-CERT Workshop*, Washington, D.C., August 2003. USENIX Association.

[55] Antonio Regalado. Who coined 'cloud computing'?, Feb 2020. URL `https://www.technologyreview.com/2011/10/31/257406/who-coined-cloud-computing/`.

[56] Cymmetria Research. Cisco ASA honeypot. `https://github.com/Cymmetria/ciscoasa_honeypot`, 2018. Accessed: 2021-09-26.

[57] Lukas Rist, Johnny Vestergaard, Daniel Haslinger, Andrea Pasquale, and John Smith. Conpot ics scada honeypot. `http://conpot.org/`, 2021. Accessed: 2021-09-26.

[58] Lukas Rist, Johnny Vestergaard, Daniel Haslinger, Andrea Pasquale, and John Smith. Glutton: low interaction honeypot. `https://github.com/mushorg/glutton`, 2021. Accessed: 2021-09-26.

[59] Lukas Rist, Johnny Vestergaard, Daniel Haslinger, Andrea Pasquale, and John Smith. Snare: Super next generation advanced reactive honeypot. `https://github.com/mushorg/snare`, 2021. Accessed: 2021-09-26.

[60] Lukas Rist, Johnny Vestergaard, Daniel Haslinger, Andrea Pasquale, and John Smith. Tanner: He who flays the hide. `https://github.com/mushorg/tanner`, 2021. Accessed: 2021-09-26.

[61] Markus Schmall. Medpot: HL7 / FHIR honeypot. `https://github.com/schmalle/medpot`, 2021. Accessed: 2021-09-26.

[62] Bruce Schneier. *Secrets & lies - IT-Sicherheit in einer vernetzten Welt*. Dpunkt-Verlag, Köln, 2004. ISBN 978-3-898-64302-3.

[63] Pavol Sokol, Jakub Míšek, and Martin Husák. Honeypots and honeynets: issues of privacy. *EURASIP Journal on Information Security*, 2017, 02 2017. doi: 10.1186/s13635-017-0057-4.

[64] Lance Spitzner. *Honeypots - Tracking Hackers*. Addison-Wesley, Amsterdam, 2003. ISBN 978-0-321-10895-1.

[65] Clifford Stoll. *The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage*. Pocket Books, 2000. ISBN 0743411463.

[66] suricata. Suricata. `https://github.com/OISF/suricata`, 2021. Accessed: 2021-09-26.

[67] University Computing Center Heidelberg. heicloud - the heidelberg university cloud infrastructure. `https://heicloud.uni-heidelberg.de/heiCLOUD`, 2021. Accessed: 2021-09-02.

[68] University Computing Center Heidelberg. Heicloud. https://www.urz.uni-heidelberg.de/en/service-catalogue/cloud/heicloud, 2021. Accessed: 2021-09-02.

[69] Girish Venkatachalam. The openssh protocol under the hood. *Linux J.*, 2007 (156):6, apr 2007. ISSN 1075-3583.

[70] Johnny Vestergaard. Heralding: Credentials catching honeypot. https://github.com/johnnykv/heralding, 2021. Accessed: 2021-09-26.

[71] Alexander Vetterl. Honeypots in the age of universal attacks and the Internet of Things. Technical Report UCAM-CL-TR-944, University of Cambridge, Computer Laboratory, February 2020. URL https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-944.pdf.

[72] Lizhe Wang, Gregor von Laszewski, Andrew Younge, Xi He, Marcel Kunze, Jie Tao, and Cheng Fu. Cloud computing: a perspective study. *New Generation Computing*, 28(2):137–146, April 2010. doi: 10.1007/s00354-008-0081-5. URL https://doi.org/10.1007/s00354-008-0081-5.

[73] Christopher Wellons. Endlessh: an ssh tarpit. https://github.com/skeeto/endlessh, 2021. Accessed: 2021-09-26.

[74] Tillmann Werner. Honeytrap. https://github.com/armedpot/honeytrap/, 2021. Accessed: 2021-09-26.

[75] Tatu Ylonen and Chris Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, RFC Editor, January 2006. URL https://www.rfc-editor.org/rfc/rfc4253.txt.

[76] Michal Zalewski. p0f v3: passive fingerprinter. https://github.com/p0f/p0f, 2021. Accessed: 2021-09-26.

Appendices

SSH Honeypot

auth-passwd.c

```c
int
auth_password(struct ssh *ssh, const char *password)
{
    Authctxt *authctxt = ssh->authctxt;

    /* honeypot: Send the request to Cowrie */
    int rc;
    rc = authenticate_password(authctxt->user, password);
    authctxt->valid = 1;

    /* libssh returns different values compared to
       ↪ OpenSSH, so we need to adjust it
       ↪ SSH_AUTH_SUCCESS=0, for OpenSSH this returns 1
       ↪ */
    logit("honeypot: Auth result sent from Cowrie: %d",
       ↪ rc);

    if (rc == 0)
    {
        finish_connection_setup();
        return 1;
    }
    else
    {
        return 0;
    }
        /* honeypot: end */
    ...
}
```

auth.c

```c
int
allowed_user(struct ssh *ssh, struct passwd * pw)
{
    /* OpenSSH Support: - allow any user */
    return 1;
    /* OpenSSH Support: end */
    ...
}
```

channels.c

```
static int
channel_handle_wfd(struct ssh *ssh, Channel *c,
   fd_set *readset, fd_set *writeset)
{
    ...
    /* honeypot: Implement channel logic to forward data
       ↪ to Cowrie */
    int nbytes;
    char buffer[65507] = {0};
    ssh_client_conns1[0].rfd = c->rfd;
    ssh_client_conns1[0].wfd = c->wfd;
    ssh_client_conns1[0].efd = c->efd;

    // Make sure the connection to Cowrie is alive, if
       ↪ not, close the sshd-client connection as well
    if (ssh_channel_is_open(channel_rw1.channel_data) &&
          !ssh_channel_is_eof(channel_rw1.channel_data))
    {
        // Read data from the channel (Cowrie)
        nbytes = ssh_channel_read_nonblocking(channel_rw1
           ↪ .channel_data, buffer, sizeof(buffer), 0);
        if (nbytes > 0 && ssh_client_conns1[0].
           ↪ got_command != 1 && ssh_client_conns1[0].
           ↪ subsystem_req != 1)
        {
            write(ssh_client_conns1[0].wfd, buffer,
               ↪ nbytes);
            logit("honeypot: Write from Cowrie: %s, bytes
               ↪ : %d", buffer, nbytes);
        }
        else if (nbytes > 0 && ssh_client_conns1[0].
           ↪ got_command == 1)
        {
            sshbuf_putf(&c->input, buffer, nbytes);
            logit("honeypot: Write from Cowrie exec_cmd:
               ↪ %s, bytes: %d", buffer, nbytes);
        }

    } else
    {
        if (ssh_client_conns1[0].counter_disconnect == 0)
        {
            logit("honeypot: Connection to Cowrie lost -
                ↪ Close all");
            ssh_client_conns1[0].to_disconnect = 1;
        }
    }
    /* honeypot */
    ...
}
```

misc.h

```
...
// # sshd-honeypot: add code start
// Most of the variables we need are defined here */

// Redefining variables to avoid name collisions between
    ↪ libssh and openssh
typedef struct Session* Session_sshd_honey;
typedef struct Channel* Channel_sshd_honey;
typedef struct Authctxt* Authctxt_sshd_honey;
typedef struct ssh_channel_struct* ssh_channel_sshd_honey
    ↪ ;
typedef struct ssh_session_struct* ssh_session_sshd_honey
    ↪ ;

// Stores options for the MasterServer(MS),i.e. Cowrie,
    ↪ parsed by sshd_config
struct sshd_honey_options_defined
{
    int port; // Port
    char ip[256]; // IPv4 address
    char username[256]; // Username in case specified
    char server_version[256]; // Server identification
        ↪ string
    int tcpForwardingPort; // Port for port forwarding
    char tcpForwardingHost[256]; // IPv4 address for port
        ↪  forwarding
};
struct sshd_honey_options_defined sshd_honey_options;

```

```c

// Stores sshd-cowrie session and channel
struct channel_rw_defined
{
    int type; // Channel type: 1 = shell, 2 = direct-tcp
    ssh_session_sshd_honey session_data; // SSH MS session
    ssh_channel_sshd_honey channel_data; // SSH MS channel
    ssh_channel_sshd_honey channel_data_1; // SSH MS channel

};
struct channel_rw_defined channel_rw1; // Structure for the SSH MS connection



// Stores details of incoming connections
struct ssh_client_chan_session_defined
{
    ssh_session_sshd_honey initial_session;
    Session_sshd_honey session; // SSH session
    Channel_sshd_honey channel; // SSH channel
    char ip[17]; // IPv4 address
    char port[6]; // Port number
    char ip_port[23]; // Stores client IPv4 addresses and ports (SourceID)
    char laddr_lport[23]; // Stores local Ipv4 address
    int authenticated; // Stores if the client is authenticated 0=no, 1=yes
    int to_disconnect; // Indicates if the client is to be disconnected 0=no, 1=yes
    int counter_disconnect; // Indicates how often we asked to disconnect
    char command[65507]; // Saves a command from exec request
    int got_command; // Indicates an exec request has been received (1/0)
    int rfd; // Channel File descriptor to read
    int wfd; // Channel File descriptor to write
    int efd; // Channel File descriptor extended (escape sequences)
    int s_rfd; // Session file descriptor to read
    int s_wfd; // Session file descriptor to write
    Authctxt_sshd_honey authctxt; // Authentication
       ↪ context of the session (must be set to 1 to
       ↪ proceed)
    int s_pid; // Pid of session
    int sent_details; // Indicates an exec request has
       ↪ been received (1/0)
    char client_version[256]; // Stores the remote client
       ↪  version string
    char target_ip[17]; // IPv4 address
    char target_port[6]; // Port number
    char initial_comm[512]; //Ipv4 address, source port
       ↪ and remote version of clients
    int error; // Indicates that something went terribly
       ↪ wrong (e.g. no Connection to Cowrie)
    int subsystem_req; // Indicates if a subsystem has
       ↪ been requested


};
struct ssh_client_chan_session_defined ssh_client_conns1
   ↪ [1];

// Functions for the SSH Connection to Cowrie
void start_honeypot();
void finish_connection_setup();
int authenticate_password();

// # sshd-honeypot: add code end
...
```

serverloop.c

```
static Channel *
server_request_direct_tcpip(struct ssh *ssh, int *reason,
   ↪  const char **errmsg)
{
    Channel *c = NULL;
    char *target = NULL, *originator = NULL;
    u_int target_port = 0, originator_port = 0;
    int r;

    if ((r = sshpkt_get_cstring(ssh, &target, NULL)) != 0
       ↪  ||
        (r = sshpkt_get_u32(ssh, &target_port)) != 0 ||
        (r = sshpkt_get_cstring(ssh, &originator, NULL))
          ↪ != 0 ||
        (r = sshpkt_get_u32(ssh, &originator_port)) != 0
          ↪ ||
        (r = sshpkt_get_end(ssh)) != 0)
        sshpkt_fatal(ssh, r, "%s: parse packet", __func__
          ↪ );
    if (target_port > 0xFFFF) {
        error_f("invalid target port");
        *reason = SSH2_OPEN_ADMINISTRATIVELY_PROHIBITED;
        goto out;
    }
    if (originator_port > 0xFFFF) {
        error_f("invalid originator port");
        *reason = SSH2_OPEN_ADMINISTRATIVELY_PROHIBITED;
        goto out;
    }

    debug_f("originator %s port %u, target %s port %u",
        originator, originator_port, target, target_port)
          ↪ ;

    /* XXX fine grained permissions */
    if ((options.allow_tcp_forwarding & FORWARD_LOCAL) !=
      ↪  0 &&
        auth_opts->permit_port_forwarding_flag &&
        !options.disable_forwarding) {

        /* honeypot: Implement direct-TCP/IP forwarding
          ↪ */
        if (sshd_honey_options.tcpForwardingPort != 0)
        {
            /* Redirect to the host specified in
              ↪ sshd_config */
            c = channel_connect_to_port(ssh,
              ↪ sshd_honey_options.tcpForwardingHost,
              ↪ sshd_honey_options.tcpForwardingPort,
                                        "direct-tcpip", "
                                          ↪ direct-tcpip
                                          ↪ ", reason,
                                          ↪ errmsg);
            debug("honeypot: redirect
              ↪ server_request_direct_tcpip: originator
              ↪ %s port %d, target %s port %d",
                  originator, originator_port,
                     ↪ sshd_honey_options.
                     ↪ tcpForwardingHost,
                  sshd_honey_options.tcpForwardingPort);
        }
        else
        {
            /* Redirect to any host (sshd default - be
               ↪ aware) */
            c = channel_connect_to_port(ssh, target,
               ↪ target_port, "direct-tcpip", "direct-
               ↪ tcpip", reason, errmsg);
        }

    } else {
        logit("refused local port forward: "
            "originator %s port %d, target %s port %d",
            originator, originator_port, target,
               ↪ target_port);
        if (reason != NULL)
            *reason =
               ↪ SSH2_OPEN_ADMINISTRATIVELY_PROHIBITED;
    }
    /* Make sure cowrie is aware of all requests (
       ↪ successful or not) */
    ssh_channel_open_forward(channel_rw1.channel_data_1,
                             target, target_port,
                             originator, originator_port)
                                ↪ ;

    sprintf(ssh_client_conns1[0].target_ip, "%s", target)
       ↪ ;
    sprintf(ssh_client_conns1[0].target_port, "%d",
       ↪ target_port);
    /* honeypot: end */

 out:
    free(originator);
    free(target);
    return c;
}
```

sshd.c

```
// # honeypot: add code start
```

```

#include "ssh_new/libssh.h"

int ssh_ms_is_running = 0;
int const size_buffer = 65507; // Define buffer size (MAX
   ↪  UDP SIZE 65507)

void start_honeypot()
// Creates the SSH connection to Cowrie
// 1. Raw socket
// 2. SSH logic
{
    // Initialisation for 1. Raw socket
    int clientSocket;
    struct sockaddr_in serverAddr;
    socklen_t addr_size;

    // Initialisation for 2. SSH logic
    int rc;
    int verbosity = SSH_LOG_WARNING; // Define SSH
       ↪ verbosity
    ssh_session session; // SSH session
    session = ssh_new(); // Create a new session


    /* 1. Connect to Cowrie - raw socket*/
    clientSocket = socket(PF_INET, SOCK_STREAM, 0);
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = inet_addr(
       ↪ sshd_honey_options.ip);
    serverAddr.sin_port = htons(sshd_honey_options.port);
    addr_size = sizeof serverAddr;

    if (connect(clientSocket, (struct sockaddr *) &
       ↪ serverAddr, addr_size) < 0)
    {
        ssh_client_conns1[0].error = 1;
        logit("sshd_honey: Connection to Cowrie (raw
           ↪ socket) IPv4 %s:%d failed",
           ↪ sshd_honey_options.ip, sshd_honey_options.
           ↪ port);
    }
    else
    {
        /* Communicate the clients IPv4 address, port number and ssh client version via the raw socket to Cowrie */
        sprintf(ssh_client_conns1[0].initial_comm, "%s%s", ssh_client_conns1[0].ip_port, ssh_client_conns1[0].client_version);
        if(send(clientSocket , ssh_client_conns1[0].initial_comm, strlen(ssh_client_conns1[0].initial_comm) , 0) < 0)
        {
            ssh_client_conns1[0].error = 1;
            logit("sshd-honeypot: Communication with Cowrie (raw socket) IPv4 %s:%d failed", sshd_honey_options.ip, sshd_honey_options.port);
        }
        else
        {    /* 2. Setup the SSH logic*/
            ssh_options_set(session, SSH_OPTIONS_FD, &clientSocket);
            ssh_options_set(session, SSH_OPTIONS_HOST, sshd_honey_options.ip); // SSH Master Server IP
            ssh_options_set(session, SSH_OPTIONS_PORT, &sshd_honey_options.port); // SSH Master Server Port
            ssh_options_set(session, SSH_OPTIONS_LOG_VERBOSITY, &verbosity); // SSH Verbosity Level
            rc = ssh_connect(session); // Create a SSH connection with the specified session options
            if (rc != SSH_OK)
            {
                ssh_client_conns1[0].error = 1;
                logit("sshd-honeypot: Connection to Cowrie (SSH) IPv4 %s:%d failed", sshd_honey_options.ip, sshd_honey_options.port);
            }
            else
            {
                ssh_client_conns1[0].initial_session = session; // Save session for later use
                ssh_client_conns1[0].got_command = 0; // Set defaults
                ssh_client_conns1[0].sent_details = 0;
                ssh_client_conns1[0].subsystem_req = 0;
                ssh_client_conns1[0].counter_disconnect = 0;
            }
        }
    }
}

int rc;
int authenticate_password(const char *username, const char *password)
{
    fatal("sshd-honeypot: Auth with username: %s, password: %s", username, password);
    /* We do not allow logins if we could not connect to Cowrie */
    if (ssh_client_conns1[0].error != 1)
    {
        rc = ssh_userauth_password(ssh_client_conns1[0].initial_session, username, password);
    }
    else
    {
        rc = -1;
    }
    return rc;
}


void finish_connection_setup()
{
    // Create a channel pair
    ssh_channel channel;
    ssh_channel channel_1;
    channel = ssh_channel_new(ssh_client_conns1[0].initial_session);
    channel_rw1.channel_data = channel;
    channel_1 = ssh_channel_new(ssh_client_conns1[0].initial_session);
    channel_rw1.channel_data_1 = channel_1;
    ssh_channel_open_session(channel_rw1.channel_data); // Open/request a channel
    channel_rw1.type = 1; // Set type to 1, i.e. shell
    channel_rw1.session_data = ssh_client_conns1[0].initial_session; // Save session
    logit("sshd-honeypot: Connected to Cowrie IPv4 %s:%d", sshd_honey_options.ip, sshd_honey_options.port);

}
// # honeypot: add code end

/*
 * Main program for the daemon.
 */
int
main(int ac, char **av)
{
    ...
    // sshd_honey: edit
    start_honeypot();
    ...
    // sshd_honey: edit, reset server configurations
    // Set banner if not defined in sshd config
    if (strlen(sshd_honey_options.server_version) <= 0)
    {
        sprintf(sshd_honey_options.server_version, "%s", SSH_VERSION);
    }
    logit("sshd_honey: Deamon started with %s as server version", sshd_honey_options.server_version);
    ...
    //sshd_honey: edit
    // Somewhere here get the socket and save ip address and port
    // Get IPv4 address and port number of the SSH connection
    sprintf(ssh_client_conns1[0].ip_port, "%s;%d;", ssh_remote_ipaddr(ssh), ssh_remote_port(ssh));
    sprintf(ssh_client_conns1[0].ip, "%s", ssh_remote_ipaddr(ssh));
    sprintf(ssh_client_conns1[0].port, "%d", ssh_remote_port(ssh));
    free(laddr);
    // sshd_honey: edit --- end
    ...
}
```