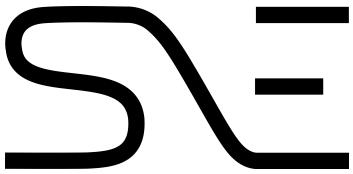

MSc System and Network Engineering
Research Project 1

Pentest Accountability By Analyzing Network Traffic & Network Traffic Metadata

Henk H.P.M. Doorn van, BSc
Henk.vanDoorn@os3.nl

Marko M.B. Spithoff, BSc
Marko.Spithoff@os3.nl

February 11, 2018

Supervisors:
A. Stavroulakis
L. Petrov

Assessor:
Prof. Dr. C.T.A.M. de Laat

Universiteit van Amsterdam

Abstract

The purpose of this study is to determine if accountability during a penetration test can be achieved by capturing network traffic. With current methods it is difficult to prove or disprove what actions have been executed during a penetration test. An isolated test environment was created to determine the feasibility of achieving accountability by capturing traffic. In this environment we researched the feasibility of capturing all network traffic and capturing network metadata during a penetration test. The results showed that capturing all network traffic is feasible apart from the libpcap, which poses severe limitations on the scalability of this method and requires capable hardware. In comparison, capturing only network metadata requires a fraction of the resources. In this we research we will show that both methods are feasible to provide accountability, but must be tailored specifically to the infrastructure that has to be tested. capturing network metadata could provide a feasible method to create accountability and only requiring little hardware recourses, but it is needed to have knowledge beforehand about what data has to be captured and stored. A capture of all network traffic requires powerful hardware and scales badly due to the limitation imposed by libpcap, but requires no knowledge beforehand about the data to be stored and captured to create accountability.

Key Words — Penetration Testing, Accountability, Network Data, Network Metadata, Standardized

Contents

List of Figures

List of Tables

1 Introduction

Penetration testing or security auditing, hereafter to be referred to as pentest or pentesting is a common procedure for firms assessing their IT-infrastructure. These firms often contact other specialized firms or individuals to execute pentests to assess their IT-infrastructure. Activities during the pentest are usually well logged and documented. Nonetheless, this is often insufficient for accountability of actions during a pentest. Pentesting firms could be questioned months after the execution of a pentest regarding specific and detailed events, which occurred during a pentest.

The logs and documents are often insufficient to prove or disprove actions done while executing a pentest. Although they give an indication of what happened during the execution of a pentest, they cannot definitively prove or disprove what action(s) have taken place. This could result in situations where firms performing pentests cannot prove for what they are accountable for.

This paper aims to propose and evaluate possible methods of creating accountability of actions by capturing pentest traffic on the network.

2 Related Work

There is little to no related work into the research of pentest accountability. We will therefore, look into the related work done in Cyber Attacks. The related work in this section is based on the execution and prevention of Cyber Attacks. After having looked into the related work of Cyber Attacks we will create usable methods to provide pentest accountability based on the methods proposed in the related work.

Bishop [8] wrote a short article about pentesting. The author defines part of the taxonomy of pentesting. The relation to this paper is that we use the definitions to determine possible actions during a pentest.

Hutchins et al. [18] wrote a paper about Intelligence-Driven Computer Network Defense. In this paper they wrote about indicators as the fundamental element of Intelligence-Driven Computer Network Defense. In this paper an Intrusion Kill Chain for the defense of computer infrastructures was proposed and the Intrusion Kill Chain was tested in a case study. Their conclusion was that the Kill Chain provided a structured way, which can be used to provide a structured way to try and protect computer infrastructures from malicious attacks.

McLaughlin et al. [23] wrote a paper about Multi-vendor Penetration Testing in the Advanced Metering Infrastructure. In this paper they proposed a new approach for penetration testing of multivendor devices. The approach proposed by [23] is based on archetypal and concrete attack trees also proposed in the same paper. As a result they showed that they were able to successfully perform penetration tests on a broad range of multivendor devices in the advanced metering infrastructure based on the proposed methods.

Worrall [30] wrote a paper about the real time sonification and visualization of network metadata. In this paper he describes how network traffic can be converted to metadata by making use of sonification and visualization to provide realtime audio and videostreaming of metadata.

Ahlers et al. [2] wrote a paper about Replicable Security Monitoring: Visualizing Time-Variant Graphs of Network Metadata. In this paper Ahlers et al. proposed a system which introduces the possibility to visualize network metadata based on Data dynamics, semantics and history based on the IF-MAP specifications. As a result they were able to visualize the current and past states of the metadata in graphs

given the limitations that data is offered via IF-MAP and all data has to be retrieved from the MAP server.

Lee and Lee [20] look into Scalable Internet Traffic Measurement by capturing traffic using libpcap and storing these to HDFS. MapReduce is used to analyse the given libpcap files. The paper tries to analyze if it is feasible to characterize Internet traffic given the scalable requirements. The research implements the well known Hadoop distributed file-system.

Feamster [16] researches a new concept of outsourcing home and small enterprise network security by making use of network metadata. They harness two trends: (1) programmable switches and (2) the capability to monitor distributed networks. The main goal of this paper is combating spam and botnets. The paper debates the ethical considerations of collection data on a large scale and discusses possible privacy concerns.

Jung et al. [19] wrote a paper about Fast Portscan Detection Using Sequential Hypothesis Testing, which delves into quick and accurate detection of port-scans. Based on their findings they developed the Threshold Random Walk, which is a detection algorithm able to identify malicious remote hosts.

Yasinsac and Leckie [31] research the possibilities of detecting malicious traffic by analyzing metadata from captured traffic. Metadata does not contain the actual payload information and therefore the paper proposed a model where Intrusion can be detected even when the payload is encrypted by inspecting the metadata.

Barford and Plonka [5] wrote a paper about gathering and analyzing network flow data to detect traffic anomalies. They defined three anomaly groups and identified differences and similarities between these groups.

Tuck et al. [28] wrote a paper about Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection. The research delves into the possibilities of detecting intrusions by efficiently match strings. This results in a comparison of different algorithms and their performance.

2.1 Taxonomy of a Pentest

Hutchins et al. [18] defined an Intrusion Kill Chain based on the U.S. military targeting doctrine consisting of the following steps:

1. Find
2. Fix
3. Track
4. Target
5. Engage
6. Assess

According to Hutchins et al. [18] when creating a targeting doctrine for a computer network attack (CNA) or computer network espionage (CNE) it is closely related to military targeting doctrines. For this reason Hutchins et al. decided to create a Intrusion Kill Chain hereafter to be referred to as a Cyber Kill Chain based on the the U.S. military targeting doctrine from 2006. As stated by Hutchins et al. the intention of a Cyber Kill Chain is to gain a, for the attacker desired effect by making use of lateral movement or by tampering with the confidentiality, availability or the integrity of an information system.

We consider a pentest to be an organized and structured procedure. The goal of a pentest is to assess the possibilities to laterally move within a computer-infrastructure. Other goals are to tamper with the confidentiality, availability or the integrity of an information system [8]. A pentest is therefore, closely related to a CNA or CNE. The taxonomy of a pentest will be based on the Cyber Kill Chain defined by Hutchins et al., which consists of the following steps: "

1. **Reconnaissance**- Research, identification and selection of targets, often represented as crawling Internet websites such as conference proceedings and mailing lists for email addresses, social relationships, or information on specific technologies.

2. **Weaponization** - Coupling a remote access trojan with an exploit into a deliverable payload, typically by means of an automated tool (weaponizer). Increasingly, client application data files such as Adobe Portable Document Format (PDF) or Microsoft Office documents serve as the weaponized deliverable.

3. **Delivery** - Transmission of the weapon to the targeted environment. The three most prevalent delivery vectors for weaponized payloads by Advanced Persistant Threat (APT) actors, as observed by the Lockheed Martin Computer Incident Response Team (LM-CIRT) for the years 2004-2010, are email attachments, websites, and USB removable media.

4. **Exploitation** - After the weapon is delivered to victim host, exploitation triggers intruders' code. Most often, exploitation targets an application or operating system vulnerability, but it could also more simply exploit the users themselves or leverage an operating system feature that auto-executes code.

5. **Installation** - Installation of a remote access trojan or backdoor on the victim system allows the adversary to maintain persistence inside the environment.

6. **Command and Control (C2)** - Typically, compromised hosts must beacon outbound to an Internet controller server to establish a C2 channel. APT malware especially requires manual interaction rather than conduct activity automatically. Once the C2 channel establishes, intruders have "hands on the keyboard" access inside the target environment.

7. **Actions on Objectives** – Only now, after progressing through the first six phases, can intruders take actions to achieve their original objectives. Typically, this objective is data exfiltration which involves collecting, encrypting and extracting information from the victim environment; violations of data integrity or availability are potential objectives as well. Alternatively, the intruders may only desire access to the initial victim box for use as a hop point to compromise additional systems and move laterally inside the network." [18]

To efficiently execute a pentest, the steps of the Cyber Kill Chain have to be performed on the different layers of Cyber Space. According to the Joint Publication 3-12 (R) of the United States Department of Defense in Cyberspace three layers exist:

1. Physical Network Layer

2. Logical Network Layer

3. Cyber-Persona Layer

The physical layer consist of hardware that is used to build up a computer infrastructure or to cross geographical boundaries. A list could consist of the following hardware:

- Wired
- Wireless
- Optical
- Satellite

The logical network layer consists of elements allows for communication between systems. The Cyber-Persona Layer consists of the natural persons who use the physical or logical network layers. For example a user that is on Facebook is an element of the Cyber-Persona Layer. Clark extended on this model by adding the information layer. The information layer consists of the data that traverses the Logical Network Layer [14]. When performing a pentest all of the steps defined by the Cyber Kill Chain should be divided across the layers defined by the U.S. Department of Defense and by Clark whom tailored this to the needs of the pentest.

3 Methods

In order to research accountability in pentesting we have created a controlled experimental setup. In this setup we are able to run pentests and capture network traffic with 802.3 network loggers. The setup is an isolated network which enables us to store data traffic without any legal infringements or infringe upon other systems.

The experimental setup as explained in the next section, is used in two methods. Full network traffic capture and metadata network traffic capture. Since both methods use an duplicate of the experimental setup we are able to compare results from both methods.

3.1 Experiment Setup

The experiment is set up as depicted in figure 1.

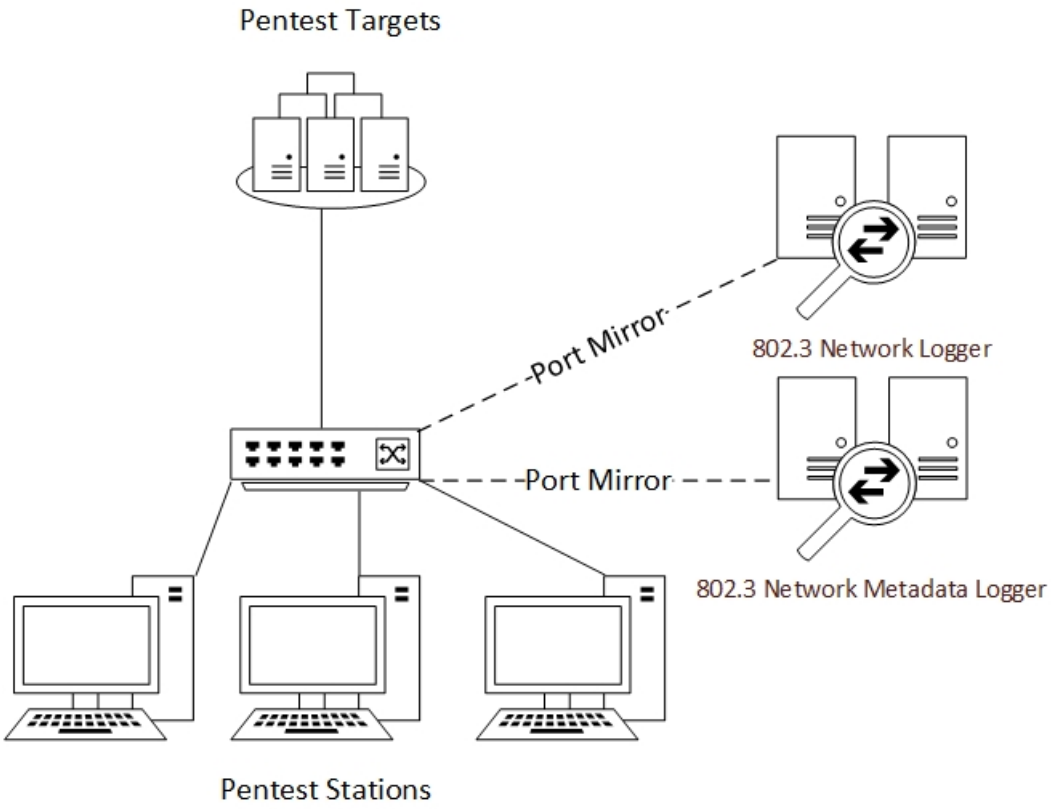

Figure 1: Experiment Setup

As depicted in figure 1 we have a controlled environment with two 802.3 loggers; one for full network captures and one for metadata network captures. We make use of both of the 802.3 loggers at the same time to be able to perform both experiments simultaneously. All of the traffic from the Pentest Stations to the Pentest Targets will be port mirrored to both the 802.3 loggers.

3.2 Full Network Traffic Capture

According to Lee and Lee capturing traffic requires a scalable infrastructure to cope with the large amount of data transmitted during a pentest[20]. Furthermore they state that it is likely that the amount of data during a pentest will increase overtime.

The amount of traffic, which will be captured requires a scalable infrastructure. In principle the amount of data will only grow, which brings challenges to storage and analyses. Apart from storage traditional relational database are unlike to be capable to scale and handle the dynamic tables needed for storing network traffic [4]. Therefore MongoDB was chosen, since according to Buck et al. [11] MongoDB is a scalable database capable of handling non-relational data [11].

Figure 2 depicts the setup for researching the feasibility of capturing the complete network stream. An analyst can either query MongoDB directly or use tools like MapReduce and Pig to verify accountability.

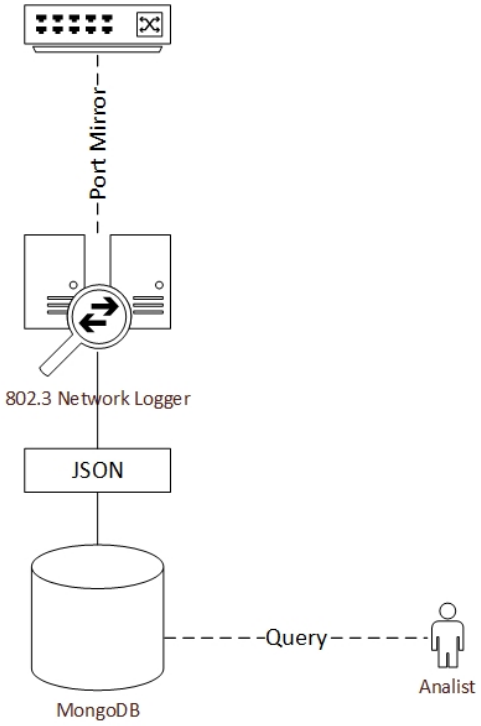

Figure 2: Setup to research feasibility to capture complete traffic stream

As depicted in figure 1 the switch is configured with a span/mirror port forwarded to 802.3 logger. The 802.3 logger writes the data to a pcap file. The pcap files are rotated every minute, resulting in the creation of a new pcap file every minute. The pcap file is converted, at one-minute intervals, to a json file. The json format is necessary for storage in MongoDB.

According to Huang et al. [17] MongoDB is capable of coping with the storing the generated data and is a scalable solution, if more capacity is required analysis can be done directly on the information stored in MongoDB. Initial capturing will be done with tcpdump 4.9.0 using libpcap 1.5.3 on Centos-release 7-4.1708. Pcap files are binary blobs, which cannot be imported by Mongodb or any other database directly. Mongodb requires newline delimited JSON format files. Tshark 2.4.4 will be used to convert PCAP to JSON.

To avoid file conflicts Tcpdump is configured to rotate the pcap file every minute, meaning that every minute a new pcap file was created. Due to the sequential nature of the pcap format conversion to json can only utilize one CPU thread [4]. A bash script will be used to check what the oldest pcap file is and if tcpdump completed writing network traffic to the file. If tcpdump is not writing to the file (e.g writing to a newer pcap) then tshark will convert the pcap to json and remove the pcap when conversion has been completed.

As mentioned in this section, pcap files have to be read sequentially from start to finish preventing multi-threading. This is due to the fact that pcap uses record pointers instead of delimiters or indexes [4]. This research will look into the restriction of this mechanism when capturing high-bandwidth traffic. The block structure is depicted in figure 3. The block type specifies the following section header.

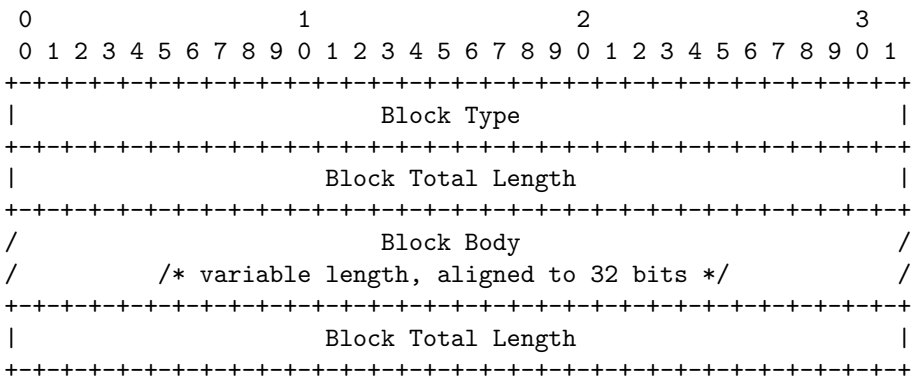

Figure 3: General Block Structure of a PCAP file [29]

PCAP files can consist of many blocks describing different kind of packets and traffic. Regarding this research the general flexible structure is important to understand the possible issues when reading PCAP files. The PCAP headers do not specify their length, which means that they have to be read sequentially thus preventing multithreading. An example of the PCAP header structure is depicted in figure 4.

```
Section Header
|
+- Interface Description
|  +- Simple Packet
|  +- Enhanced Packet
|  +- Interface Statistics
|
+- Name Resolution
```

Figure 4: PCAP header structure [29]

Pcap to JSON conversion

When running tcpdump a script was used to convert pcap to json files which can be inserted into mongodb. The script prevents file read conflicts by checking if the pcap file is in use by tcpdump. If the pcap file is in use then the script retries after 10 seconds.

Importing to MongoDB

After converting to JSON the information is inserted into MongoDB using Mongoimport. This tool supports multithreading, the wiredtiger storage engine compresses the data to preserve storage. While importing the data into MongoDB the system will be monitored to check the CPU, memory and disk performance to assess possible bottlenecks.

Performance

Iperf will be used as a tool to determine the performance of the setup. Although not representative for simulating pentesting traffic, it gives a baseline indicator of performance and possible bottlenecks. The setup is depicted in figure 1 where one pentest station will be an iperf server and one a iperf client. The traffic is mirrored to the 802.3 network logger to be stored in MongoDB as mentioned earlier in section 3.2 During the experiment the CPU, memory and disk load will be monitored using python scripts. To measure the time it takes to convert to json, a python script will be called every 60 seconds, which measures the number of pcap file in the directory. This is an effective way to measure if conversion is lagging behind the network capture since the pcap file(s) are sequentially converted starting with the oldest pcap file. If conversion speed is lower than the amount of captured data then the number of pcap files will be greater than 2 during an active capture since pcap files are rotated by tcpdump and only converted when not written to.

3.3 Metadata Network Traffic Capture

For the metadata network traffic capture we make use of a Python script with the Scapy library. We will perform pentests on a subnet that is created and controlled for this research as depicted in figure 1, while performing live captures of network traffic and network metadata using the Python Scapy script during the pentests. The metadata that will be captured is explained in section 3.3.1.

Because we expect that we will see and have to inspect large amounts of network traffic, we believe that hardware restraints such as a limited amount of memory (RAM) will make pattern recognition in metadata, during live captures, quite challenging. Therefore, for the experiments we will look into the feasibility of recognizing pentest patterns based on metadata during the live capture and writing only the

detected patterns into a MongoDB database as depicted in figure 5. Because we expect that the hardware will have difficulty logging and analyzing everything simultaneously, we will also propose an experiment, which first writes all the captured metadata to a MongoDB database, which then will be processed by an 802.3 data analyzer as depicted in figure 6.

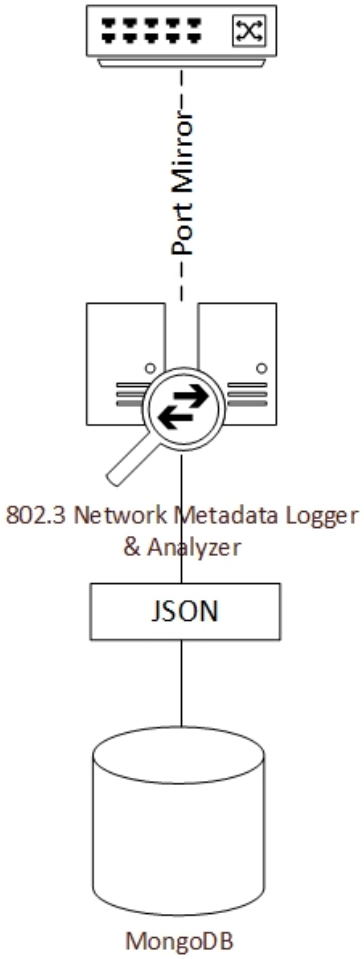

Figure 5: Experiment Metadata Analyzer-1

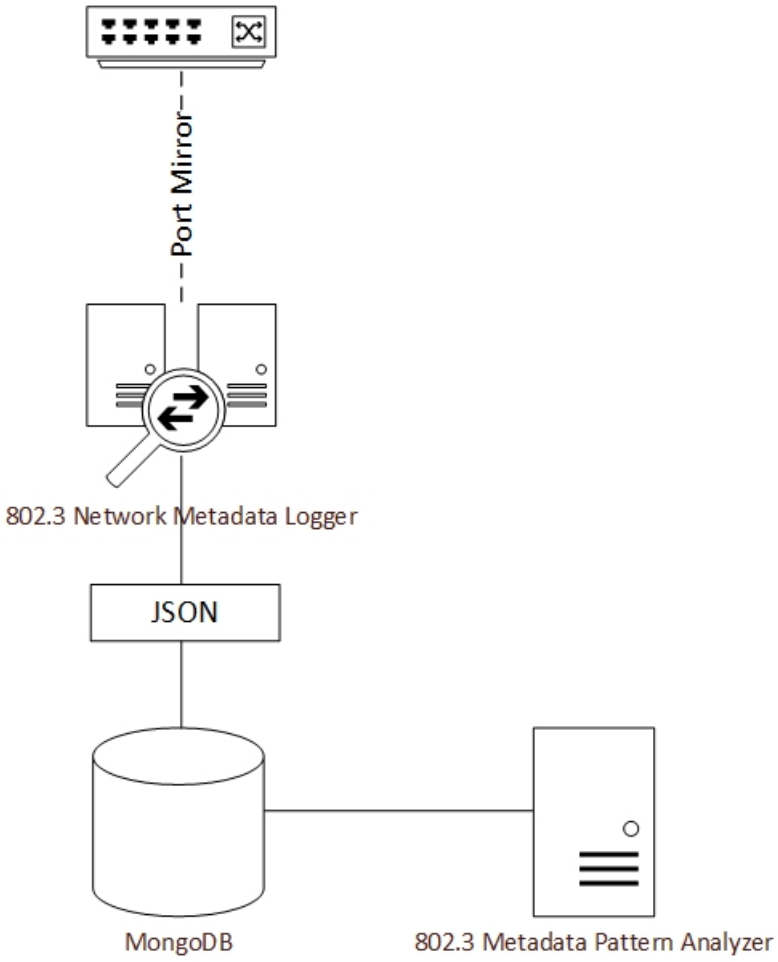

Figure 6: Experiment Metadata Analyzer-2

3.3.1 Network Metadata

To capture the possible relevant network metadata for our experiments, we first have to define the possible relevant network metadata. To define the possible relevant network metadata we refer to the Open System Interconnection Reference Model commonly known as the OSI model. As written by Briscoe [10], the OSI model consists of seven layers:

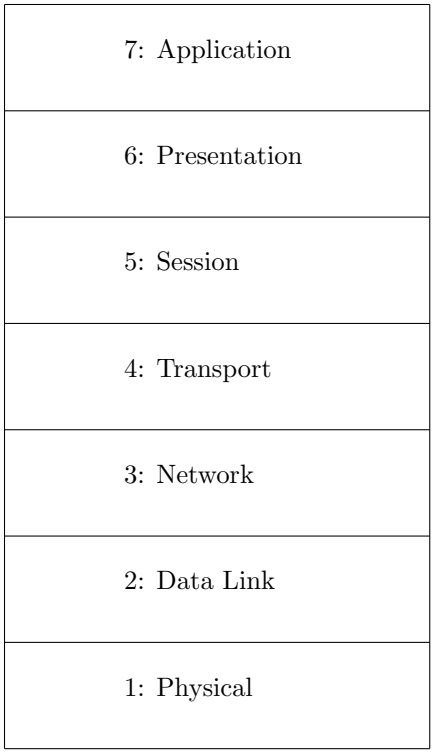

Figure 7: Seven Layer OSI Model [10]

According to Briscoe the layers one till five as depicted in figure 7 are used to transport the data to its destination, or present it to the user depending on the direction the stack is traversed [10]. Layer six is used to pack the data for transport or unpack the data to present it to the application layer. The application layer is used to present the data to the the user.

One of the most commonly used descriptions for metadata is that it is "Data About Data" [21], but as described by Lubas et al. there is a variety of ways to describe metadata. For this paper we will refer to the most commonly known description for metadata as "Data About Data". If we look at the OSI reference model again as depicted in figure 7 we know that the layers one till five are used to transport the data to its destination. This means that the layers one till five will create data about the data that is traversed via the stack. Because the layers one till five are most likely to create metadata we will focus our research into these five layers. When inspecting the OSI model as depicted in figure 7 we expect that we will be able to obtain metadata from the different layers as explained below:

- Physical Layer, The method used to access the physical network
- Data Link Layer, The access strategy used for the network and MAC Address identifiers
- Network Layer, The IP addresses used to communicate between networks
- Transport Layer, The transport layer protocol that is being used
- Session Layer, Session information used by different protocols

Because we know, which layers are most likely to create metadata. We examined, which metadata could be retrieved from the different packet headers used within the 802.3 protocol.

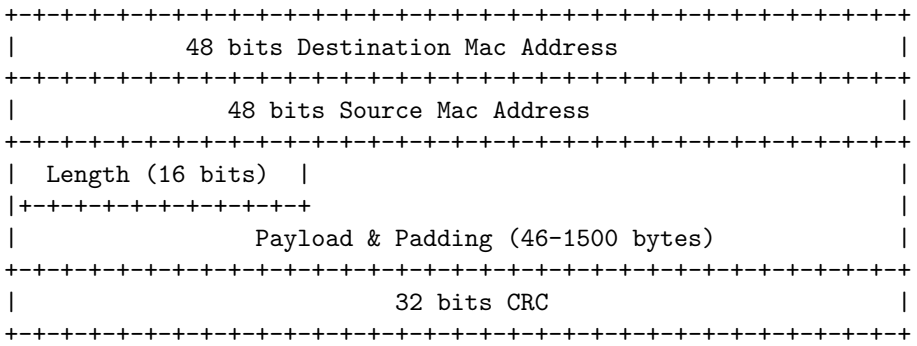

Figure 8: Ethernet Packet Header [9]

When examining the Ethernet Packet Header as depicted in figure 8. It can be seen that an Ethernet Packet Header contain a Destination Address and a Source Address. In the ethernet layer this means that it will contain the ethernet source and destination address, or so called Mac Addresses. To achieve accountability it is important to know how the packets flow between endpoints. Because of the DHCP protocol the chance exists that a system will have a different IP Address after the pentest than that it had during the pentest. The Mac Address however is less likely to be changed, which makes a mac address ideal to achieve accountability of, which systems have been targeted during the execution of a pentest. In addition to the chance of dynamically assigned IP addresses the chance exists that a system or device is behind a NAT of a corporate infrastructure. This means that the traffic can only be traced to a publicly available IP Address and not to the targeted endpoint.

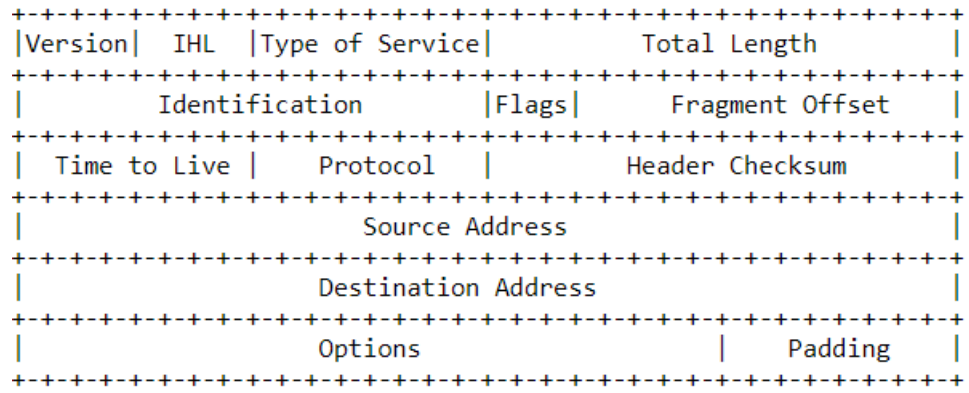

Figure 9: IPV4 Packet Header [25]

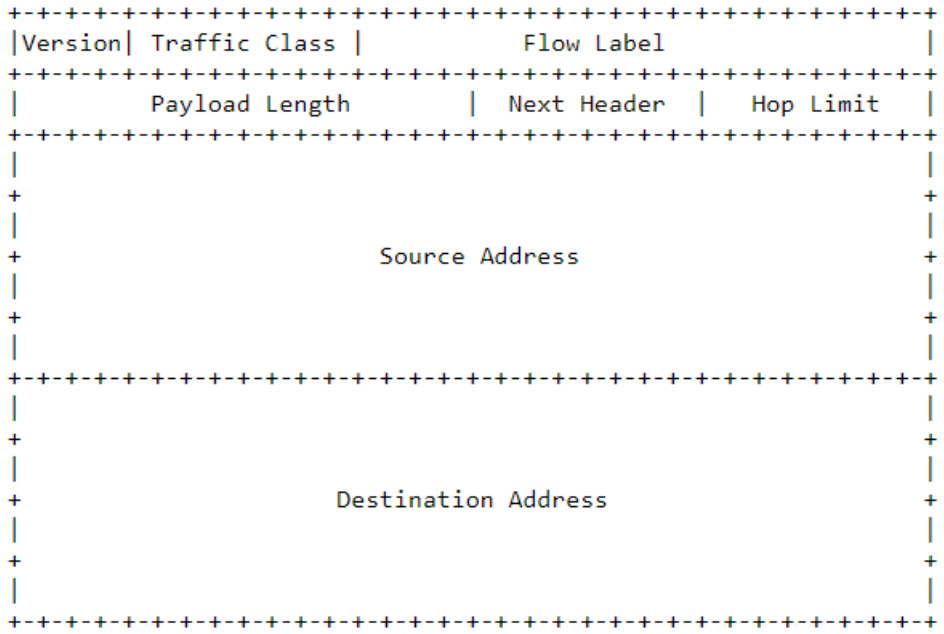

Figure 10: IPV6 Packet Header [15]

When examining the IPV4 header depicted in figure 9 and the IPV6 header depicted in figure 10 we can deduct that the IPV4 and IPV6 protocols make use of different IP headers. Because we want to achieve accountability for the IPV4 and IPV6 protocol we will extract the metadata from the IPV4 and IPV6 header, which they have in common. This means that we will use the IP Version number to determine if it is IPv4 or IPV6 traffic and the source and destination address to determine the IPV4 or IPV6 source and destination address.

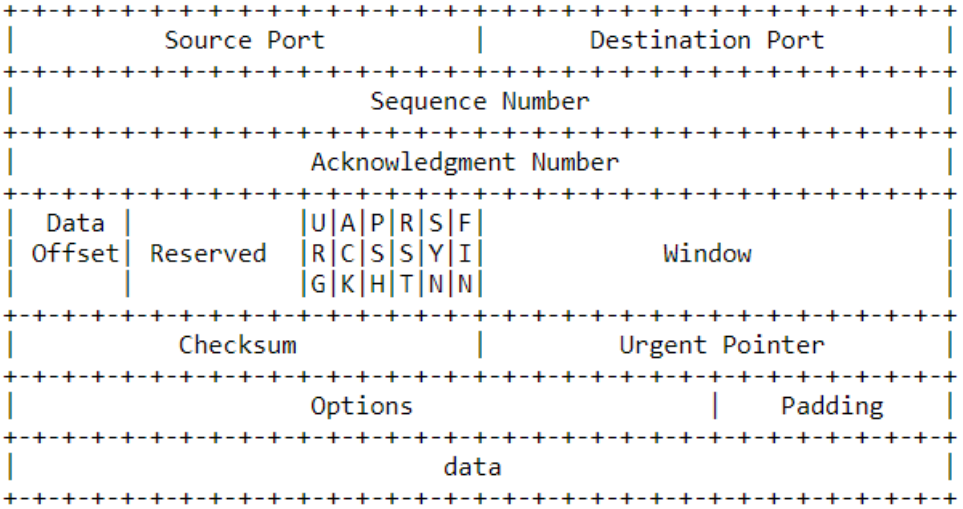

Figure 11: TCP-Header [26]

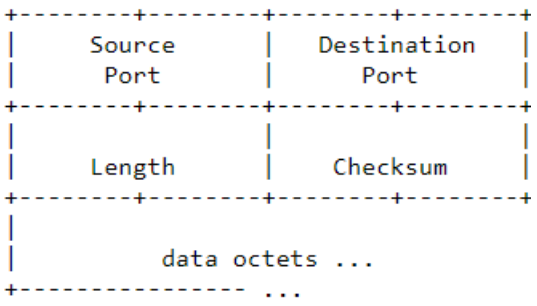

Figure 12: UDP-Header [24]

When examining the TCP Header depicted in figure 11 and the UDP Header depicted in figure 12 we can see that the UDP header requires less information than the TCP Header. This is due to the the difference between the two protocols. The TCP protocol is a statefull protocol [26] and therefore, needs more information about the connection than the stateless UDP protocol needs [24]. Because the the TCP protocol requires more information about the connection than the UDP protocol needs the TCP protocol creates additional metadata on the data that is actually being transmitted. As a result we can obtain more metadata from the TCP protocol than from the UDP protocol. Therefore from the TCP protocol we will log the Source Port, Destination Port, Sequence Number, TCP Flags and the Window Size. From the UDP Protocol we will only log the Source and Destination port.

When we combine all of the headers and the data that we will log we will get the following structure for the the metadata that we will log as depicted in figure 13

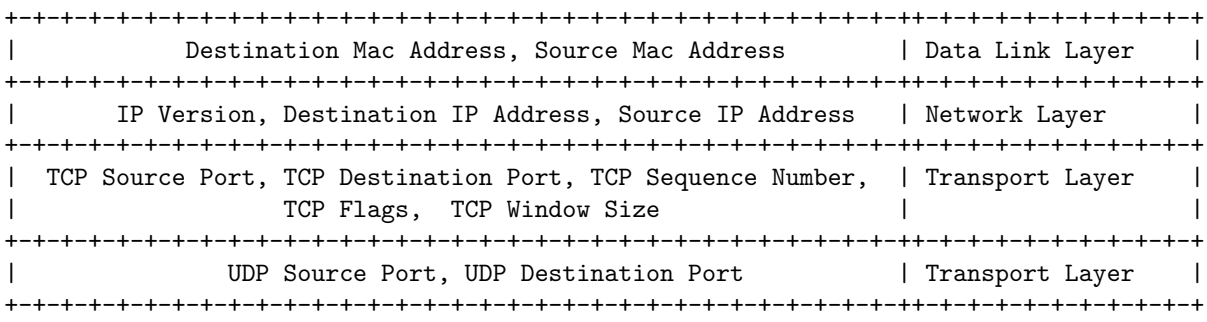

Figure 13: Metadata To Log

3.3.2 Detecting Centralized NMAP TCP Port Scans Based on TCP Metadata

To be able to achieve accountability of actions during the execution of a pentest one must be able to recognize the actions done during the execution of a pentest. As explained in section 2.1 the first step of the Cyber Killchain is the Reconnaissance phase. During the Reconnaissance phase a commonly used method is port scanning to identify possible interesting or vulnerable protocols on an computer infrastructure. Because NMAP is widely adopted as the de facto standard for portscanners we will look into the detection possibilities of the NMAP port scanner on TCP Metadata level. We consider NMAP as the de facto standard because of its integration with Kali Linux and as a commonly used tool by Red Teams [6]. As explained by Bennieston an NMAP

port scan is most commonly done by one of two type of scans the TCP connect scan or the TCP syn scan commonly referred to as the (Syn) Stealth Scan [7]. Based on the work of Jung et al. [19] in, which they have created an port scan detection algorithm called the Threshold Random Walk. We want to create a detection method based on TCP metadata and specifically the TCP flags. For this we will create an experiment using a python script, which will inspect TCP flags and raise an alarm at A during the experiment to determine threshold, for packets that have the characteristics of A NMAP port scan.

After having created a method to detect Nmap port scans we will test this method by performing 100 Nmap port scans with default settings to determine how accurate the method created by us is in detecting Nmap port scans.

3.3.3 Detecting TCP (Reverse) Shell Sessions Based On TCP Metadata

Because most of the information about an exploit is found in the higher layers of the OSI model, it is difficult to achieve accountability of actions during the execution of an exploit in a pentest procedure based on Metadata. We will look into the possibilities of detecting the Command and Control sessions, which will most likely be created after the execution of an exploit of a system as described by Hutchins et al. [18] in step 6 of the Cyber Killchain "Command and Control". To detect possible TCP (Reverse) Shells we will examine the TCP metadata to detect patterns in he TCP (Reverse) Shell connections. We expect, that as described in section 3.3.2, by making use of the TCP Flags and pre determined thresholds, we will be able to detect TCP (Reverse) Shells.

After having created a method to detect TCP (Reverse) Shells we will test this method by making use of a Python script for a TCP Shell Server and a TCP Shell Client. We will set the client and the server in a loop, which will first build up and then terminate 100 TCP Reverse Shells concurrently. To determine how accurate our method is in detecting TCP (Reverse) Shells it has to detect a percentage of the 100 sessions build up with the python scripts.

4 Results

4.1 Network Metadata

To examine the feasibility of logging all network traffic metadata we made use of python scripts, as described in section 3.3 to log network traffic metadata and to log system performance. To examine the feasibility of capturing network metadata, we have conducted the following experiments:

1. Capture Metadata For 12+ Hours a Python 2.7 script with the Pymongo and Scapy Module.
2. Capture Metadata For 12+ Hours a Python 2.7 script with the Pymongo and Socket Module.
3. Detect 100 centralized NMAP Scans Using a Python 2.7 script with the Pymongo and Socket Module
4. Detect 100 TCP Reverse Shells created with python 2.7 Scans Using a Python 2.7 script with the Pymongo and Socket Module

4.1.1 Experiments 1 & 2

During experiment 1 we performed a 12 hour network capture using Python with the Scapy and Pymongo module. During the capture we specifically captured metadata of the TCP and UDP protocol on our local subnet, as depicted in figure 1. During the process of capturing the metadata we monitored system performance and measured statistics of the CPU, Memory, Network, Disk IO and the total amount of Disk Space used every minute for the duration of the experiment.

During experiment 1 we observed that the system did not make efficient use of resources. In addition to this problem we found that the Python script in combination with Scapy was unable to process all packets in realtime that were received on the network interface. Because of this reason we created and designed experiment 2. In this experiment we created a Python script to perform the UDP and TCP metadata capture in combination with the Socket module. Because the Scapy module for Python is a packet analyzer and packet manipulation module we thought the performance of the script should improve when receiving the data directly from the Socket and processing it directly.

In the upcoming sections we will explain the observed differences in the results of the performance statistics for both performed experiments.

CPU

The statistics of the CPU performance for both experiments are included in the Appendix. For Metadata Experiment 1 in figure 25 on page 39 and in table 3 on page 40 and for Metadata Experiment 2 in figure 26 on page 41 and in table 4 on page 42.

When examining and comparing the results of Metadata Experiment 1 and Metadata Experiment 2 one can observe that there is a significant difference in CPU utilization. The major difference is caused by the fact that the Scapy Module when imported might cause interference with the core affinity when imported [3]. We used the solution proposed on Stackoverflow to reset the core affinity. After this and by replacing the Scapy module with the Socket module to directly read the network data from the interface performance improved significantly, as can be observed in figure 25 and in table 3 and figure 26 and in table 4.

Disk Performance

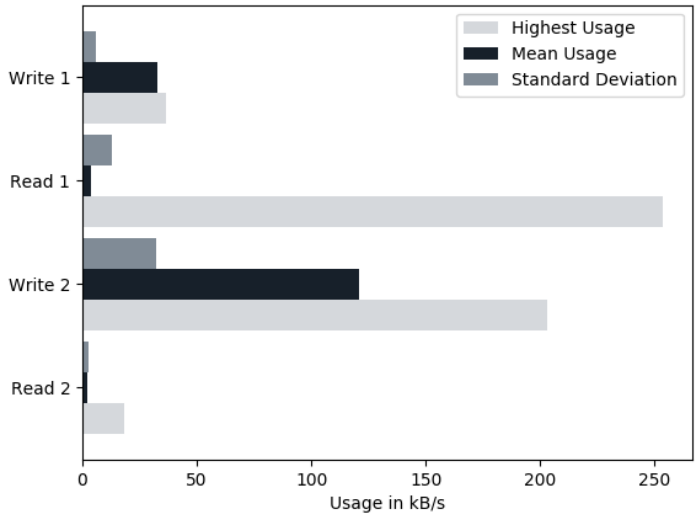

Figure 14: Results Of Disk Performance During Experiment

When comparing the results of write 1 and read 1 (experiment 1) and write 2 and read 2 (experiment 2) depicted in figure 14 one can observe that the read values have the same mean usage. The spike in the read usage in experiment 1 can be explained by a random process accessing the /dev/sdb disk during experiment 1. During experiment 2, one can observe that the average write value is higher than during experiment 1. This difference is caused by the fact that during experiment 2 we used the Socket module, which is able to process about 4 times the amount of packets as will be explained in 4.1.1.

The measured values of figure 14 are included in the Appendix in table 5 on page 42 and table 6 on page 42.

Memory

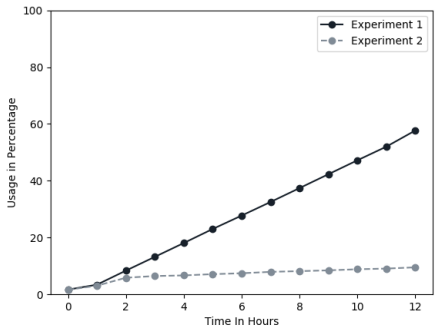

Figure 15: Results Memory Usage During Experiment 1

When comparing the results of both the experiments in figure 15 one can observe a significant difference in memory usage during the 12 hour experiment. From this experiment we can conclude that the Socket module makes more efficient use of memory than the Scapy module. It seems that the Scapy module stores more into memory than the Socket module of Python.

The measured values of figure 15 are included in the Appendix in table 9 on page 43 and 10 on page 43

Network

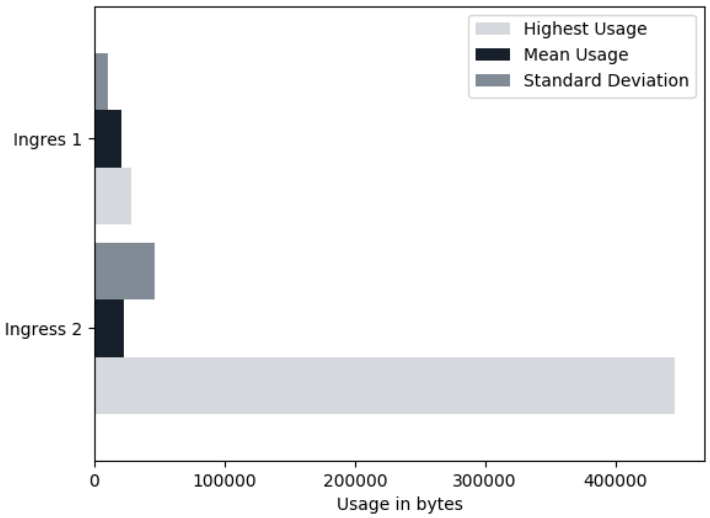

Figure 16: Results Network Performance During Experiments

When comparing the results of the both experiments in figure 16, one can observe that the mean usage is about the same. The highest usage spike for ingress 2 was a one time measured value, as a result of random network jitter. Most likely to be caused by the different systems on the local subnet. One can observe that the mean utilization of the network stays the same during both experiments.

The measured values of figure 16 are included in the Appendix in table 7 on page 42 and 8 on page 42.

Storage

Experiment Number	Disk Space Used in MB	Mean Item Size in Bytes	Total Packets Captured
1	71.5	383	1,323,088
2	261	351	4,341,085

Table 1: Mongo DB Statistics At The End Of The Experiments

When comparing the results of the both experiments in table 1, one can observe that we have processed more data in the seconds experiment. The mean object size

however remains almost the same during both experiments.

4.1.2 Experiment 3

As described in section 3.3.2 during this experiment we will examine the feasibility of detecting Nmap port scans. During the analysis of 100 nmap port scans with default settings we found that Nmap has the following characteristics:

Mean Completion Time	Packets Send	Packets Send To Target	Flags Set	Packets Send From Target	Flags Set	Mean Time Between Packets
13.302947s	3428	1714	Syn	1713	Rst	0.281ms

Table 2: Nmap Calculated Statistics After 100 Default Portscans

Based on the work of Jung et al. [19] we want to define a threshold at, which the system registers the possible portscan. When combining this with the work of Roesch et al. [27] we wanted to look at a threshold based on time and TCP flags. During the examination of the port scans we executed. We found that when using the Nmap tool, with exception of the full connect method of Nmap, every syn packet uses the same TCP sequence number as depicted in figure 17.

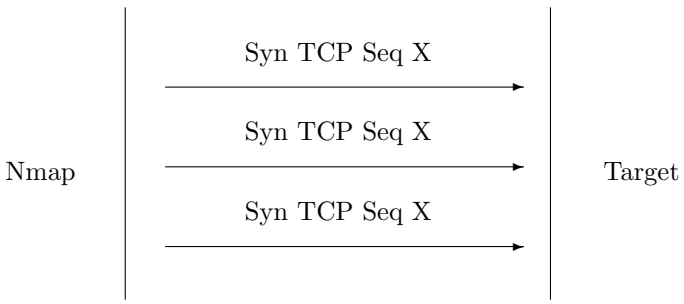

Figure 17: Nmap Detection Pattern

After having found that a Nmap portscan uses the same TCP sequence number for every portscan we started filtering for syn packets that make use of the same TCP Sequence number. Based on the mean time between packets that are being sent to the target host we defined a threshold of 1 second in, which a syn packet with the same TCP sequence number has to be received or else the portscan has ended. We have tested this method to detect 100 Nmap default portscans in, which we have achieved 100% accuracy. We have combined this with timestamps and using this method we are able to achieve 100% realtime accuracy for Nmap scans performed from host x to y within timeframe z.

4.1.3 Experiment 4

Based on the data and experience we obtained during the execution of experiment 3, explained in section 4.1.2, we want to detect a TCP reverse shell, which is often the result of an exploit performed on a system. To detect a TCP reverse shell we defined a method based on the work of Roesch et al. [27] When examining the characteristics

of a TCP shell we found that when sending data using the SSH protocol after the TCP 3 Way Handshake has been performed [13]. As mentioned by Roesch et al. [27] "almost all requests to web servers have their TCP PUSH and ACK flags set" [27] Based on this information we inspected established SSH sessions and found that we were able to detect the pattern as depicted in figure 18

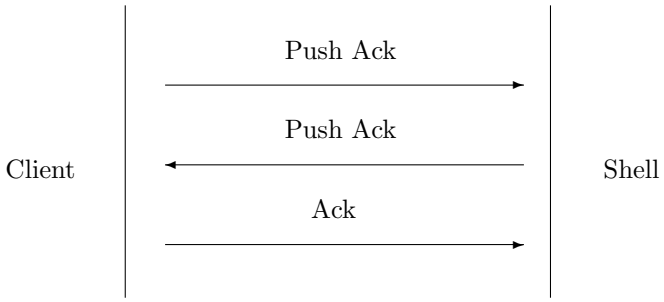

Figure 18: TCP Shell Detection Pattern

The pattern depicted in figure 18 enables us to look for TCP shells within the network. Using this method and combining this with the TCP end sequence of a TCP shell depicted in figure 19.

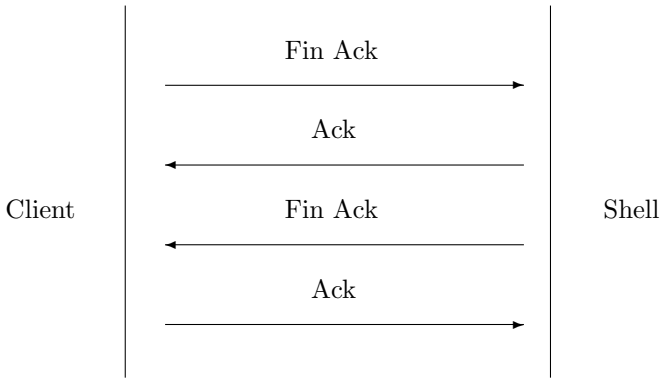

Figure 19: TCP Shell End Pattern

We can observe a detectable pattern within a TCP stream, which can be used to detect shells within a TCP stream. To test the patterns as depicted in figure 18 and figure 19. We scripted this into a filter using python 2.7 first looking in the TCP streams for the pattern depicted in figure 18 after having detected this pattern it will then look for the end pattern depicted in figure 19. We combine these patterns with time to achieve accountability for the moment a TCP shell is sending data till the moment a TCP shell is stopped. We have performed experiments with 100 in a loop created Python TCP Reverse shells on a host machine while trying to detect the TCP reverse shells with our custom build Python Filter. Using our Python filter we were able to detect 100% of the started TCP reverse shells in real time. By combining this with timestamps we were able to achieve accountability for TCP reverse shells based on our custom made filter.

4.2 Full Capture

Examining the feasibility of logging all network traffic, similar test where performed as in section 1. The following experiments have been conducted. This section will mainly focus on capturing and storing typical pentest traffic by simulating portscans and using pentest suites. Results give an indication of system performance while capturing and storing all network traffic. Furthermore the results indicate if accountability can be achieved when using this method.

1. Capture all traffic for 12 hours simulating traffic by executing a continuous portscan
2. Evaluating the usability for accountability

Memory

In the following expirement we measured memory usage during the experiment. The results are depicted in figure 20.

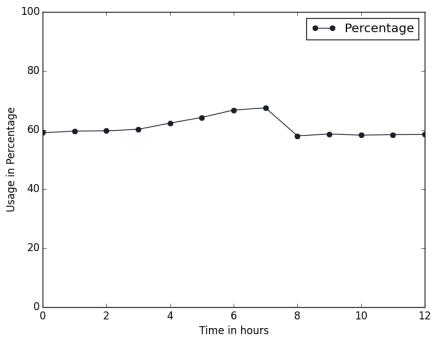

Figure 20: Memory usage in percentage

From the results above a stable trend in memory usage can be observed. However, the utilization is quite high, possibly MongoDB caches or reserves memory in order to serve queries.

Processor

This experiment measured the system performance while executing a default NMAP scan continuously for 12 hours. The results are depicted in the table 11.

What can be seen from the results in table 11 is that all cores at some point in time reach maximal utilization and that there is a high standard deviation since performance is measured in a percentage. To examine this further, we look into the specific programs being run, which are:

1. Tcpdump : Capture traffic and write to a pcap file
2. Tshark : Convert the pcap to JSON
3. Mongoimport: Import the JSON file to MongoDB

Follow-up experiments were conducted as described in section 3.2 focusing on system utilization caused by in the above enumerated programs.

4.2.1 Tcpdump

Tcpdump captures the traffic, which is mirrored from the switch and saves this to a pcap file every minute. Upon researching we came to the conclusion that Tcpdump uses little system resources. The results are depicted in table 12 on page 45. The results show that during the experiments Tcpdump used little CPU-resources and according to the standard deviation was stable in CPU-utilization. Memory and disk usage was negligible.

4.2.2 Tshark

As mentioned in section 3.2 the structure of pcap files requires to be read sequentially preventing multithreading. This behavior can be seen in table 11 on page 45. Since CPU is measured in utilization per core in percentage, these result show that Tshark is only able to use a single core while converting pcap to json. While converting the results show that the single core being used by Tshark is fully utilized.

During the experiments we also measured the disk utilization which can be seen in figure 21 What can be seen from this graph is the fact that the write speed is higher than the read speed. Also the disk performance shows a low standard variation and the mean speed in reading as well as in writing is nearly as high as the maximum speed. Therefore, it can be concluded that the read and write speeds are stable. It is likely that the disk performance could be higher since Tshark could process the data faster as seen in table 11 on page 45.

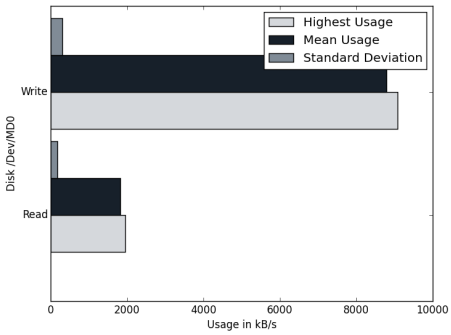

Figure 21: Disk usage during Tshark pcap to json conversion

The results show that Tshark write speed is consistently higher than the speed at which its reads. The result of the consistent higher write speed can be seen in the difference is size between the pcap and json file.

The results in table 8 where acquired by capturing a network stream generated using iperf. They show a growth in size of approximately 6.3 times the size of the pcap file.

4.2.3 Mongoimport

To maintain readability of this paper the graph depicting the CPU usage while importing JSON using mongoimport can be found in the figure 27 on page 44. The results show that the MongoDB tool mongoimport utilizes all available cores to import the JSON file into it's database. The results also show that the mean usage is nearly as high for all cores as the highest usage.

The feasibility of storing the captured data from the test can be seen in table 8. The storage size in GiB is 5,5 GiB, which is feasible given modern storage solutions. For example, our storage array has 22 TiB of storage which is several years old.

4.2.4 Usability in accountability

If a full capture were to be used for accountability there must be certainty that all packets where captured and stored. Therefore, we devised a simple but elegant test to verify the functionality. In our environment we setup one server to flood ping another server with one million packets. Since a successful ping consists of two packets (echo and reply) the total number of received packets must be two million.

Listing 1: Sending and recieving of one million ping packets

```
#!/bin/bash
sudo ping -f -c 1000000 192.168.1.107
1000000 packets transmitted, 1000000 received, 0% packet loss, time 151825ms
```

Listing 1 shows that one million ping reply packets were transmitted and one million replies were received withing tree minutes. Therefore the database should contain two million ping packets.

Listing 2: Counting the number of ICMP(ping) packets in the database

```
MongoDB Enterprise > db.ICMP.count({      "layers.icmp" : {"$exists" : true}});
2000000
```

The results in listing 2 show that the database contains two million ping packets. The figure 22 shows an example of a captured ICMP packet. Since all packets are capture fully all properties of a packet can be queried using the MongoDB query language. In the appendix the listing 4 can be found of a json formatted packet which can be imported an exported to/from the MongoDB database.

▼ (9) {_id : 5a71c61cafa670f72fb93e6c}	{ 3 fields }
_id	5a71c61cafa670f72fb93e6c
timestamp	1517405638697
▼ layers	{ 4 fields }
▶ frame	{ 13 fields }
▶ eth	{ 13 fields }
▶ ip	{ 22 fields }
▼ icmp	{ 10 fields }
icmp_icmp_type	8
icmp_icmp_code	0
icmp_icmp_checksum	0x00000ef2
icmp_icmp_checksum_status	1
icmp_icmp_ident	24659
icmp_icmp_seq	6
icmp_icmp_seq_le	1536
icmp_icmp_data_time	Jan 31, 2018 14:34:18.000000000 CET
icmp_icmp_data_time_relative	-19.302193000
▼ icmp_data	{ 2 fields }
data_data_data	8a:b4:00:00:00:00:00:00:10:11:12:13:14:15:16:17:18:19:1a:1b:1c:1d:1e:
data_data_len	48

Figure 22: Example data from a captured packet

5 Discussion

We are concerned with capturing large amounts of data due to the ethical and legal aspects. Bélanger and Crossler wrote a paper about privacy in the digital age. We were not able to assess the exact limitations resulting from the Dutch "Nieuwe Wet op de inlichtingen- en veiligheidsdiensten"(WIV) legislation. The European Union is also trying to catch up her legislation with the rapidly advancing digital possibilities, to prevent the invasion of the privacy of her inhabitants with the General Data Protection Regulation (GDPR)[1].

The methods proposed in this paper might intrude on the privacy of users. Therefore, it is needed to discuss the applicability of these methods when trying to achieve accountability.

6 Conclusion

6.1 Metadata

One can observe the following from the results explained in section 4.1:

1. Accountability based on Metadata capture is plausible
2. Port Scan Detection is based on TCP metadata is possible for NMAP
3. TCP shell detection based on TCP metadata is plausible
4. Python is not fast enough for realtime packet sniffing

Accountability of actions based on metadata capture is plausible. The difficulty however lies in the fact that some accountability of actions seem only achievable by inspecting the payload data of a packet. More research into this subject is needed to definitively state if full accountability can be achieved based on packet metadata.

Port scan detection based on TCP metadata is possible for NMAP. More research is still needed to look into the feasibility of detecting port scans when other tools are used and to detect UDP port scans.

TCP shell detection based on metadata is plausible. More research is needed to see if other TCP connection based protocol do not create false positives with the method used in this research.

During this experiment we found that the python method we used, was not as reliable as a real-time packet analyzer as methods written in the C programming language like TCPdump.

6.2 Full Capture

Given the results the following can be concluded.

1. Accountability based on full packet capture is plausible
2. MongoDB suitable storing packets and querying
3. The pcap file format possibly limits the amount of data which can be stored

Our results show that capturing network traffic for the purpose of accountability is plausible. The main limitation in our research is the libpcap library which stores captured traffic in pcap files. Due to the structure of these files they have to be read sequentially; this results into the fact that operation on these files cannot be multithreaded at the moment.

This introduces a possible limitation when converting to the json supported fileformat which is used when importing into MongoDB. Since only one core can be utilized when converting it can be concluded that operations on pcap files do not scale well.

What can be concluded is that our setup is suitable for analysis in regard of accountability. All packets are stored including a timestamp which can be used to verify actions during a pentest.

6.3 Comparison Of Experiments

In this section a comparison will be made between the results of the full capture experiment, explained in section 4.2, and the results of of the metadata capture experiment, explained in section 4.1

CPU Performance

From the CPU utilization measurements can be concluded that capturing of metadata is less CPU intensive than the full capture of metadata. The difference is especially apparent when comparing figure 26 on page 41 to figure 28 on page 26. When examining the results of the metadata experiment, one can observe that CPU utilization fluctuates little over the available cores, while CPU utilization when capturing all data has a high level of fluctuation.

The high fluctuation in the results of capturing all data, comes due to the fact that the conversion process from pcap to json is only able to use one thread and therefore, one core. This could be a limiting factor when scaling up.

Therefore, it can be concluded that full capture is feasible when a powerful CPU is available, preferably with a high clockrate. In comparison Metadata capturing requires far less computational power. Therefore, when looking at CPU utilization the most preferred method is making use of metadata capture method instead of the full capture method.

Disk Performance

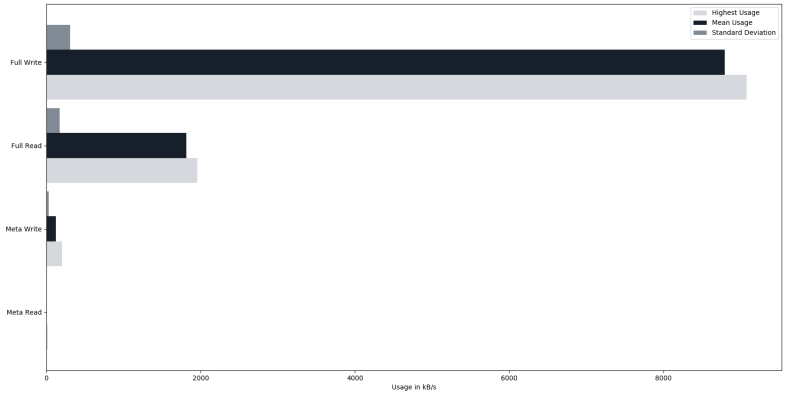

Figure 23: Comparison Of Disk Performance Between Full & Metadata Capture

When examining the difference in results of the full capture experiment and the metadata experiment in figure 23, one can observe a significant difference in read and write operations over the duration of 12 hours.

The difference between these experiments are caused by two different factors. The first factor is the amount of packets captured. During the full capture experiment 4,030,050 packets where captured, in comparison during the metadata capture experiment 4,341,085 packets were captured. The second factor is the mean item size when writing the packets into MongoDB, for the full capture experiment the mean item size was 3,8 KiB in comparison the mean item size for the metadata capture experiment was only 351 B. These results would explain the high difference in the mean write operations on the disk.

Based on the results of the both experiments, one can therefore conclude that the metadata method makes more efficient use of disk IO than the full capture method.

Memory Usage

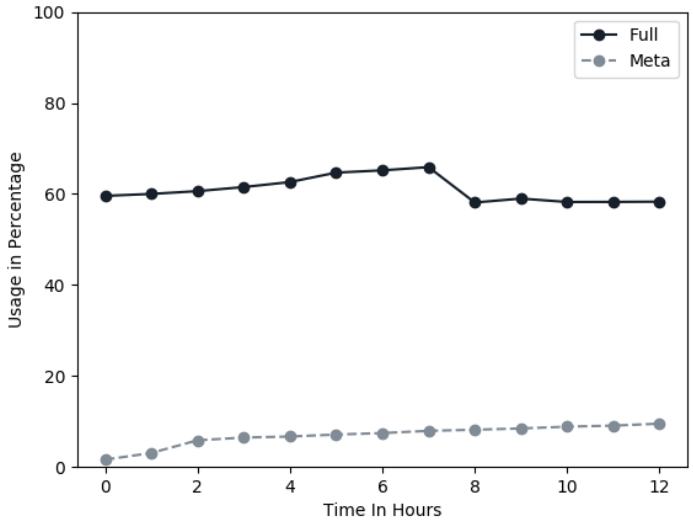

Figure 24: Comparison Of Memory Performance Between Full & Metadata Capture

When examining the difference in results of the full capture experiment and the metadata experiment in figure 24, we observed a significant difference in memory usage over the duration of 12 hours.

The full data capture experiment utilizes around 60% memory at any given moment in time during the experiment. MongoDB is responsible for this high memory usage, most likely because MongoDB makes use of some kind of caching.

When comparing the result of the full capture experiment to the result of the metadata capture experiment, we can observe that the memory usage of the metadata experiment gradually increases, but remains significantly lower than the full capture method. The difference in these results is most likely caused due to the fact that during the full capture experiment more data had to be processed as explained in the subsection Disk Performance of section 6.3.

From the comparison of the results in this section, we can therefore conclude that when looking at memory usage the metadata method would be the preferred method to achieve accountability.

6.4 Method Comparison

As can read be in section 6, the metadata capture method makes more efficient use of system resources than the full capture method. The downside of the metadata capture method in comparison with the full capture method exists in the fact that the Full capture method is more reliable in capturing packets at this moment in time. In addition to this it is harder to achieve full accountability of actions because during the capturing of metadata the data payload is not included. Exploits that will be performed during the execution of a pentest are most likely and easily to be detected in the payload of a packet. The downside of inspecting the packet payload exists in the privacy and legal aspects of this data as explained in section 5.

An own consideration has to be made when trying to achieve accountability of actions during a pentest. By making use of the full data capture method, accountability of actions are easier to achieve because the data payload is included in the capture. However when one wants to make efficient use of system resources and wants to avoid legal and privacy aspects than the metadata method should be the most likely choice.

7 Future Work

Data capture

Regarding full capture the main focus of future work is looking further into pcap structures to research the possibilities regarding multithreading. Operations on pcap files are currently done sequentially by one thread. This limits scalability since, adding more processors or machines will not results in better performance when handling pcap files. The libpcap and therefore, the pcap file format is the conventional way of capturing traffic. In this paper we discussed several limitations of the file format. Possible future work is therefore, looking into other libraries or methods to store captured traffic in a scalable database.

Port Scan Detection

More research is needed to look into the methods proposed in this paper to detect NMAP port scans. Future work should look into the applicability of this method on other port scan tools, the applicability of the proposed method on the UDP protocol based port scans and if this methods are known in other related work.

TCP Shell Detection

More research is needed to look into the methods proposed in this paper to detect TCP shells. In a more realistic network environment it might be that other statefull TCP based protocols cause false positives in the TCP shell detection rate.

Legal

In the last years the legislation of different regions tries to catch up with the fast moving digital trend. Further research is needed on the upcoming legislation in the area of digital privacy to prevent that the methods in this paper cross the boundary of illegality. Regarding our own work this is mainly relevant to Full capture since, all data is captured possibly containing privacy sensitive data.

Metadata Feasibility

More research needs to be done into the accountability of action in the other network protocols. The research in this paper namely focused on the TCP protocol. To achieve accountability of actions in the UDP protocol different thresholds needs to be defined and tested to look into the feasibility of accountability in these protocols.

Post Metadata Capture Analysis

Due to the fact that it took longer than expected to define the methods needed to achieve accountability in the TCP protocol. During this research we were unable to perform the experiment depicted in figure 6 on page 14. One can look into this method if realtime accountability of actions is not needed for a pentest procedure.

References

[1] Gpdr legislation, 2018. URL https://en.wikipedia.org/wiki/General_Data_Protection_Regulation.

[2] V. Ahlers, F. Heine, B. Hellmann, C. Kleiner, L. Renners, T. Rossow, and R. Steuerwald. Replicable security monitoring: Visualizing time-variant graphs of network metadata. *methods*, 6:9, 2014.

[3] ali m. Why does multiprocessing use only a single core after i import numpy?, 2013. URL https://stackoverflow.com/questions/15639779/why-does-multiprocessing-use-only-a-single-core-after-i-import-numpy.

[4] J. Anderson, C. Gropp, L. Ngo, and A. Apon. Random access in nondelimited variable-length record collections for parallel reading with hadoop. pages 965–970. IFIP, May 2017. ISBN 978-3-901882-89-0.

[5] P. Barford and D. Plonka. Characteristics of network traffic flow anomalies. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 69–73. ACM, 2001.

[6] B.Clark. *Rtfm: Red Team Field Manual*. CreateSpace Independent Publishing Platform, 2014.

[7] A. J. Bennieston. Nmap-a stealth port scanner, 2004.

[8] M. Bishop. About penetration testing. *IEEE Security Privacy*, 5(6):84–87, Nov 2007. ISSN 1540-7993. doi: 10.1109/MSP.2007.159.

[9] O. Bonaventure. Ethernet 802.3 frame format. URL http://cnp3book.info.ucl.ac.be/2nd/html/protocols/lan.html.

[10] N. Briscoe. Understanding the osi 7-layer model. *PC Network Advisor*, 120(2), 2000.

[11] J. Buck, N. Watkins, G. Levin, A. Crume, K. Ioannidou, S. Brandt, C. Maltzahn, N. Polyzotis, and A. Torres. Sidr: Structure-aware intelligent data routing in hadoop. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 73:1–73:12, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2378-9. doi: 10.1145/2503210.2503241. URL http://doi.acm.org.proxy.uba.uva.nl:2048/10.1145/2503210.2503241.

[12] F. Bélanger and R. E. Crossler. Privacy in the digital age: A review of information privacy research in information systems. *MIS Quarterly*, 35(4):1017–1041, 2011. ISSN 02767783. URL http://www.jstor.org/stable/41409971.

[13] L. Chappell. Inside the tcp handshake. *NetWare Connection*, 2000.

[14] D. Clark. Characterizing cyberspace: past, present and future. *MIT CSAIL, Version*, 1:2016–2028, 2010.

[15] S. E. Deering and R. M. Hinden. Internet protocol, version 6 (ipv6) specification. RFC 2460, RFC Editor, December 1998. URL http://www.rfc-editor.org/rfc/rfc2460.txt. http://www.rfc-editor.org/rfc/rfc2460.txt.

[16] N. Feamster. Outsourcing home network security. In *Proceedings of the 2010 ACM SIGCOMM workshop on Home networks*, pages 37–42. ACM, 2010.

[17] C. W. Huang, W. H. Hu, C.-C. Shih, B.-T. Lin, and C.-W. Cheng. The improvement of auto-scaling mechanism for distributed database - a case study for mongodb. In *2013 15th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–3, Sept 2013.

[18] E. M. Hutchins, M. J. Cloppert, and R. M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[19] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 211–225. IEEE, 2004.

[20] Y. Lee and Y. Lee. Toward scalable internet traffic measurement and analysis with hadoop. *SIGCOMM Comput. Commun. Rev.*, 43(1):5–13, Jan. 2012. ISSN 0146-4833. doi: 10.1145/2427036.2427038. URL `http://doi.acm.org.proxy.uba.uva.nl:2048/10.1145/2427036.2427038`.

[21] R. L. Lubas, A. S. Jackson, and I. Schneider. *The Metadata Manual : A Practical Workbook*. Chandos Information Professional Series. Chandos Publishing, 2013. ISBN 9781843347293. URL `http://search.ebscohost.com.proxy.uba.uva.nl:2048/login.aspx?direct=true&db=nlebk&AN=670969&site=ehost-live`.

[22] Marko Spithoff and Henk van Doorn. Pentest accountability scripts. `https://gitlab.os3.nl/mspithoff/RP-1_Scripts.git`, 2018.

[23] S. McLaughlin, D. Podkuiko, S. Miadzvezhanka, A. Delozier, and P. McDaniel. Multi-vendor penetration testing in the advanced metering infrastructure. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 107–116. ACM, 2010.

[24] J. Postel. User datagram protocol. STD 6, RFC Editor, August 1980. URL `http://www.rfc-editor.org/rfc/rfc768.txt`. `http://www.rfc-editor.org/rfc/rfc768.txt`.

[25] J. Postel. Internet protocol. STD 5, RFC Editor, September 1981. URL `http://www.rfc-editor.org/rfc/rfc791.txt`. `http://www.rfc-editor.org/rfc/rfc791.txt`.

[26] J. Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. URL `http://www.rfc-editor.org/rfc/rfc793.txt`. `http://www.rfc-editor.org/rfc/rfc793.txt`.

[27] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[28] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic memory-efficient string matching algorithms for intrusion detection. In *IEEE INFOCOM 2004*, volume 4, pages 2628–2639 vol.4, March 2004. doi: 10.1109/INFCOM.2004.1354682.

[29] M. Tuexen, F. Risso, J. Bongertz, and G. Harris. Pcap next generation (pcapng) dump file format. Internet-Draft draft-tuexen-opswg-pcapng-00, IETF Secretariat, June 2014. URL http://www.ietf.org/internet-drafts/draft-tuexen-opswg-pcapng-00.txt. http://www.ietf.org/internet-drafts/draft-tuexen-opswg-pcapng-00.txt.

[30] D. Worrall. Realtime sonification and visualisation of network metadata. Georgia Institute of Technology, 2015.

[31] A. Yasinsac and T. Leckie. Metadata for anomaly-based security protocol attack deduction. *IEEE Transactions on Knowledge & Data Engineering*, 16: 1157–1168, 09 2004. ISSN 1041-4347. doi: 10.1109/TKDE.2004.43. URL doi.ieeecomputersociety.org/10.1109/TKDE.2004.43.

8 Appendix

The scrips that are created for this project are published on the OS3 gitlab [22].

Listing 3: Converting pcap to json preventing read conflicts

```
#!/bin/bash
while true; do

s1="$(lsof -c tcpdump | grep /mongodb/performance/raw/ | awk '{print $9}')"
s2="$(ls /mongodb/performance/raw/ -t1 -d $PWD/*  | grep .pcap | tail -1)"


        while true; do
                if [ "$s1" == "$s2" ]
                then
                        echo "files equal"
                        sleep 10
                        break
                fi
                if [ "$s1" != "$s2" ]
                then
                        echo "files unequal"
                        tshark -T ek -r $s2 > ../json/capture.json
                        mongoimport ../json/capture.json -h 145.100.102.181 -d auto_capture -c BANDWITH
                        rm ../json/capture.json
                        rm $s2
                        break
                fi
        done
done
```

Listing 4: JSON structured captured packet


```
{
    "_id" : ObjectId("5a71c61cafa670f72fb93e6c"),
    "timestamp" : "1517405638697",
    "layers" : {
        "frame" : {
            "frame_frame_encap_type" : "1",
            "frame_frame_time" : "Jan 31, 2018 14:33:58.697807000 CET",
            "frame_frame_offset_shift" : "0.000000000",
            "frame_frame_time_epoch" : "1517405638.697807000",
            "frame_frame_time_delta" : "0.000085000",
            "frame_frame_time_delta_displayed" : "0.000085000",
            "frame_frame_time_relative" : "0.000876000",
            "frame_frame_number" : "11",
            "frame_frame_len" : "98",
            "frame_frame_cap_len" : "98",
            "frame_frame_marked" : "0",
            "frame_frame_ignored" : "0",
            "frame_frame_protocols" : "eth:ethertype:ip:icmp:data"
        },
        "eth" : {
            "eth_eth_dst" : "00:0c:29:48:2a:fc",
            "eth_dst_eth_dst_resolved" : "Vmware_48:2a:fc",
            "eth_dst_eth_addr" : "00:0c:29:48:2a:fc",
            "eth_dst_eth_addr_resolved" : "Vmware_48:2a:fc",
            "eth_dst_eth_lg" : "0",
            "eth_dst_eth_ig" : "0",
            "eth_eth_src" : "d4:ae:52:bf:e4:8a",
            "eth_src_eth_src_resolved" : "Dell_bf:e4:8a",
            "eth_src_eth_addr" : "d4:ae:52:bf:e4:8a",
            "eth_src_eth_addr_resolved" : "Dell_bf:e4:8a",
            "eth_src_eth_lg" : "0",
            "eth_src_eth_ig" : "0",
            "eth_eth_type" : "0x00000800"
        },
        "ip" : {
            "ip_ip_version" : "4",
            "ip_ip_hdr_len" : "20",
            "ip_ip_dsfield" : "0x00000000",
            "ip_dsfield_ip_dsfield_dscp" : "0",
            "ip_dsfield_ip_dsfield_ecn" : "0",
            "ip_ip_len" : "84",
            "ip_ip_id" : "0x00003ab3",
            "ip_ip_flags" : "0x00000002",
            "ip_flags_ip_flags_rb" : "0",
            "ip_flags_ip_flags_df" : "1",
            "ip_flags_ip_flags_mf" : "0",
            "ip_ip_frag_offset" : "0",
            "ip_ip_ttl" : "64",
            "ip_ip_proto" : "1",
            "ip_ip_checksum" : "0x00007c26",
            "ip_ip_checksum_status" : "2",
            "ip_ip_src" : "192.168.1.20",
            "ip_ip_addr" : "192.168.1.107",
            "ip_ip_src_host" : "192.168.1.20",
            "ip_ip_host" : "192.168.1.107",
            "ip_ip_dst" : "192.168.1.107",
            "ip_ip_dst_host" : "192.168.1.107"
        },
        "icmp" : {
            "icmp_icmp_type" : "8",
            "icmp_icmp_code" : "0",
            "icmp_icmp_checksum" : "0x00000ef2",
            "icmp_icmp_checksum_status" : "1",
```

```
                "icmp_icmp_ident" : "24659",
                "icmp_icmp_seq" : "6",
                "icmp_icmp_seq_le" : "1536",
                "icmp_icmp_data_time" : "Jan 31, 2018 14:34:18.000000000 CET",
                "icmp_icmp_data_time_relative" : "-19.302193000",
                "icmp_data" : {
                    "data_data_data" : "8a:b4:00:00:00:00:00:10:11:12:
                     13:14:15:16:17:18:19:1a:1b:1c:
                     1d:1e:1f:20:21:22:23:24:25:26:
                     27:28:29:2a:2b:2c:2d:2e:2f:30:
                     31:32:33:34:35:36:37",
                    "data_data_len" : "48"
                }
            }
        }
    }
```

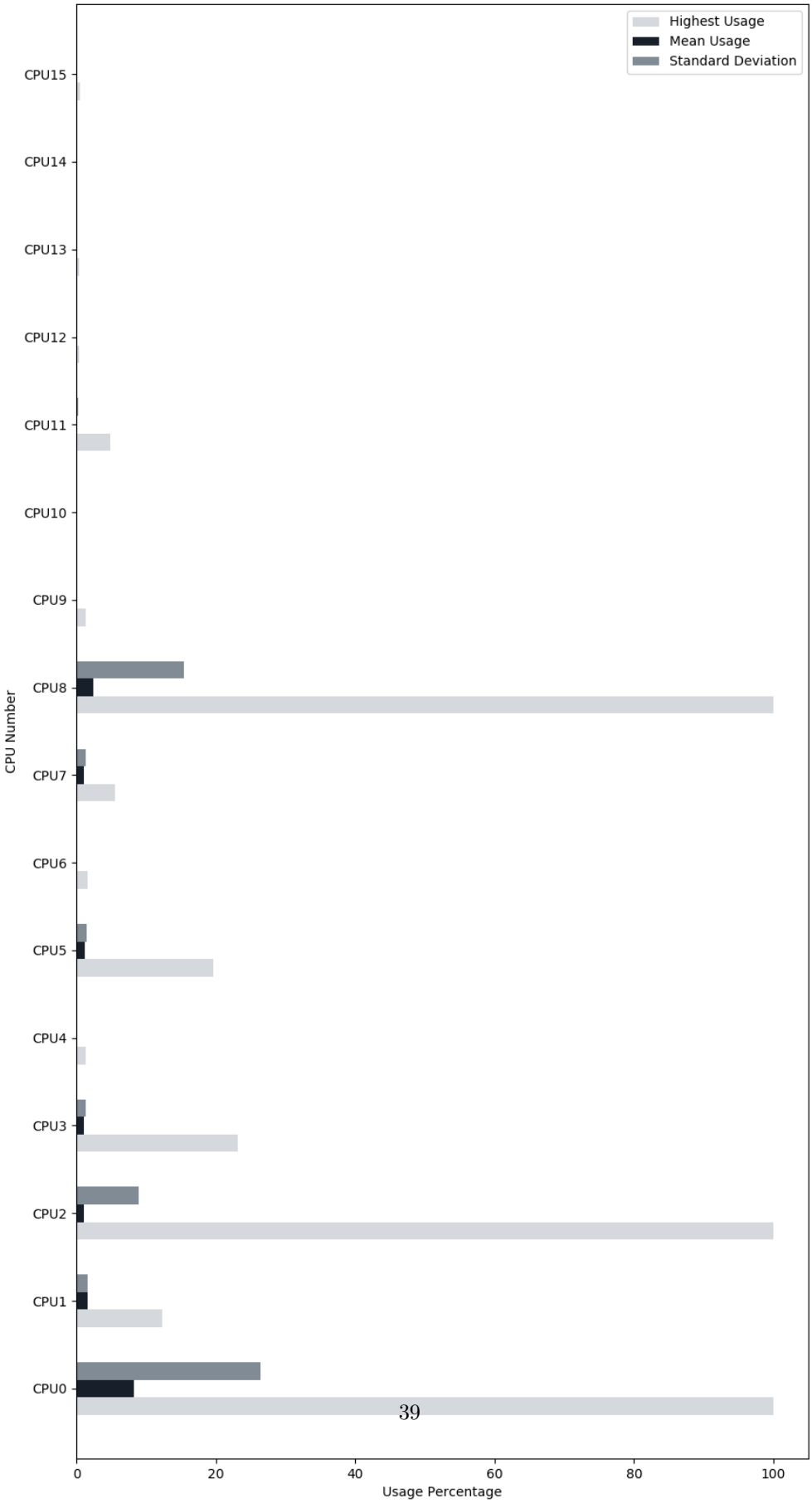

Figure 25: Graph CPU Performance Per CPU During Metadata Experiment 1

Type	Mean Usage in %	Highest Use in %	Standard Deviation
CPU 0	8.16288	100.0	26.4459
CPU 1	1.63648	12.295901366211263	1.60463
CPU 2	1.02486	100.0	8.9808
CPU 3	1.02294	23.130724396336387	1.4368
CPU 4	0.142207	1.3616738625041516	0.217878
CPU 5	1.21329	19.640479360852197	1.56345
CPU 6	0.103724	1.6491754122938531	0.183634
CPU 7	1.14579	5.6000000000000005	1.25056
CPU 8	2.53055	100.0	15.5338
CPU 9	0.0202621	1.2987012987012987	0.0680157
CPU 10	0.0083148	0.1665556295802798	0.0149853
CPU 11	0.0504778	4.945878434637802	0.230707
CPU 12	0.00242082	0.4660452729693742	0.018273
CPU 13	0.0249903	0.34970857618651124	0.0286254
CPU 14	0.00571036	0.14990006662225183	0.0114437
CPU 15	0.023802	0.5996002664890073	0.045158

Table 3: Metadata Experiment 1: Performance Statistics Per CPU Measured At 1 Minute Intervals For The Duration Of 12 Hours

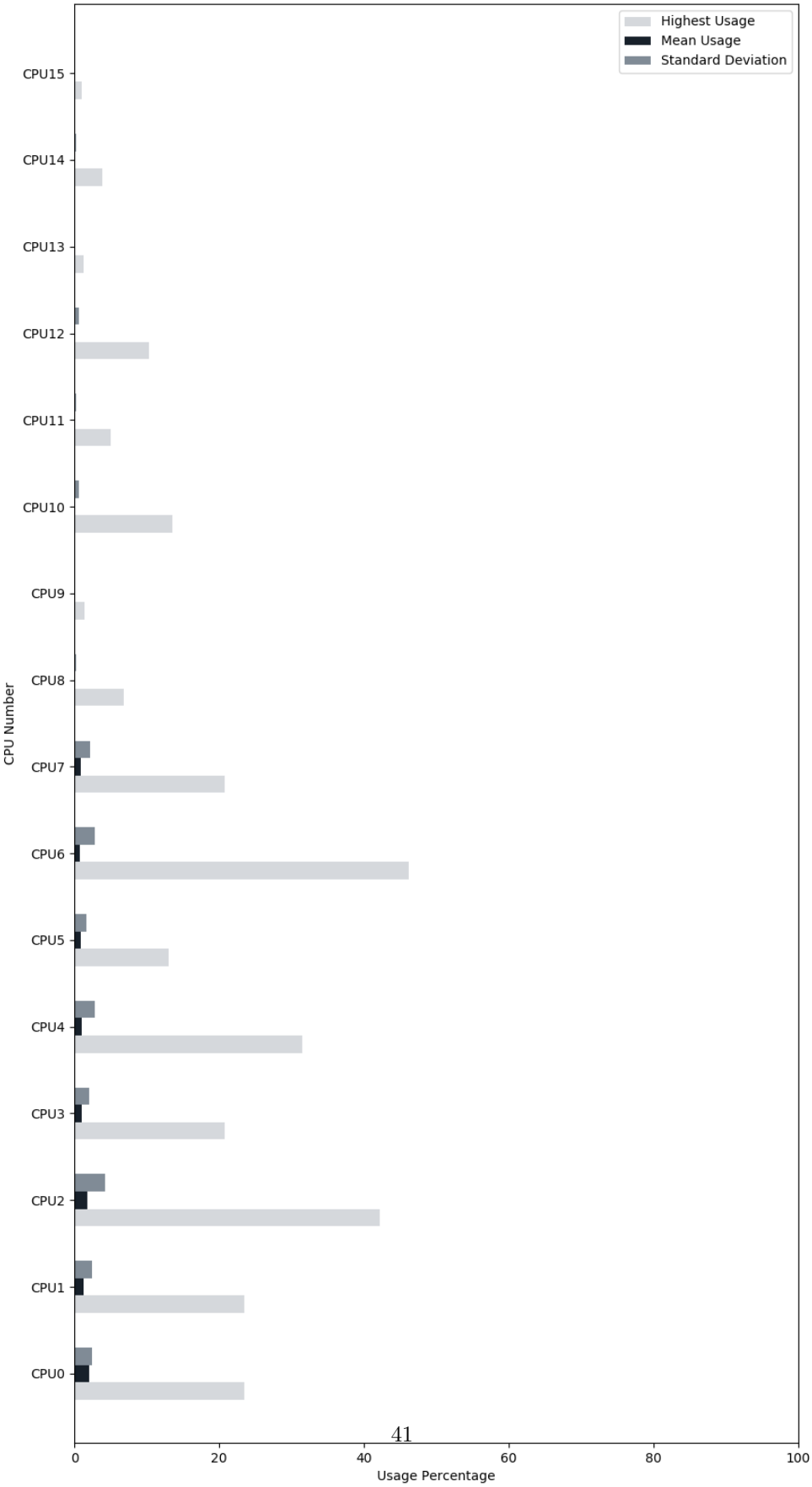

Figure 26: Graph CPU Performance Per CPU During Metadata Experiment 2

Type	Mean Usage in %	Highest Use in %	Standard Deviation
CPU 0	2.07909	23.44118638355241	2.51184
CPU 1	1.32737	23.54515050167224	2.46531
CPU 2	1.82097	42.23114233149356	4.3271
CPU 3	1.08944	20.807505444798124	2.03535
CPU 4	0.979611	31.490143668559973	2.80132
CPU 5	0.864488	13.05002509620211	1.70889
CPU 6	0.760897	46.200200870438564	2.89439
CPU 7	0.907568	20.783132530120483	2.21533
CPU 8	0.0563073	6.848171037247369	0.274931
CPU 9	0.0267433	1.4176117411607738	0.0803261
CPU 10	0.0884846	13.559605417154321	0.655206
CPU 11	0.0452862	5.037406483790523	0.259985
CPU 12	0.117944	10.364939176803865	0.72291
CPU 13	0.0378881	1.2504168056018674	0.0872348
CPU 14	0.0693562	3.8775170577467133	0.244411
CPU 15	0.0340973	0.9838252459563115	0.0739087

Table 4: Metadata Experiment 2: Performance Statistics Per CPU Measured At 1 Minute Intervals For The Duration Of 12 Hours

Type	Mean Usage in kB/s	Highest Use in kB/s	Standard Deviation
Read	3.71708	235.73	13.009
Write	32.5083	36.70	6.0984

Table 5: Metadata Experiment 1: Disk Performance Statistics Measured At 1 Minute Intervals For The Duration Of 12 Hours

Type	Mean Usage in kB/s	Highest Use in kB/s	Standard Deviation
Read	2.0307	18.53	2.54482
Write	121.914	203.21	32.1098

Table 6: Metadata Experiment 2: Disk Performance Statistics Measured At 1 Minute Intervals For The Duration Of 12 Hours

Max Network Usage in Bytes	Mean Network Usage in Bytes	Standard Deviation
27996	21160	10727.5

Table 7: Metadata Experiment 1: Network Performance Statistics Measured At 1 Minute Intervals For The Duration Of 12 Hours

Max Network Usage in Bytes	Mean Network Usage in Bytes	Standard Deviation
446092	22246	46269.1

Table 8: Metadata Experiment 2: Network Performance Statistics Measured At 1 Minute Intervals For The Duration Of 12 Hours

Time	Used Memory in %
2018-01-23 20:34	1.68947872739
2018-01-23 21:34	3.43643752462
2018-01-23 22:34	8.37820353806
2018-01-23 23:34	13.2206957435
2018-01-24 00:34	18.0983909958
2018-01-24 01:34	23.0326908017
2018-01-24 02:34	27.7590798325
2018-01-24 03:34	32.5438730891
2018-01-24 04:34	37.4290831886
2018-01-24 05:34	42.3212852578
2018-01-24 06:34	47.2197983394
2018-01-24 07:34	52.0113768465
2018-01-24 08:34	57.6948388321

Table 9: Metadata Experiment 1: Memory Usage Statistics Measured At 1 Minute Intervals Displayed At 1 Hour Intervals For The Duration Of 12 Hours

Time	Used Memory in %
2018-01-24 20:18	1.62377853332
2018-01-24 21:18	3.07685572952
2018-01-24 22:18	5.78469354501
2018-01-24 23:18	6.47722685111
2018-01-25 00:18	6.6960937386
2018-01-25 01:18	7.13698909982
2018-01-25 02:18	7.45291668491
2018-01-25 03:18	7.94564990053
2018-01-25 04:18	8.19286162467
2018-01-25 05:18	8.4779880638
2018-01-25 06:18	8.86795756665
2018-01-25 07:18	9.09236939001
2018-01-25 08:18	9.53972168313

Table 10: Metadata Experiment 2: Memory Usage Statistics Measured At 1 Minute Intervals Displayed At 1 Hour Intervals For The Duration Of 12 Hours

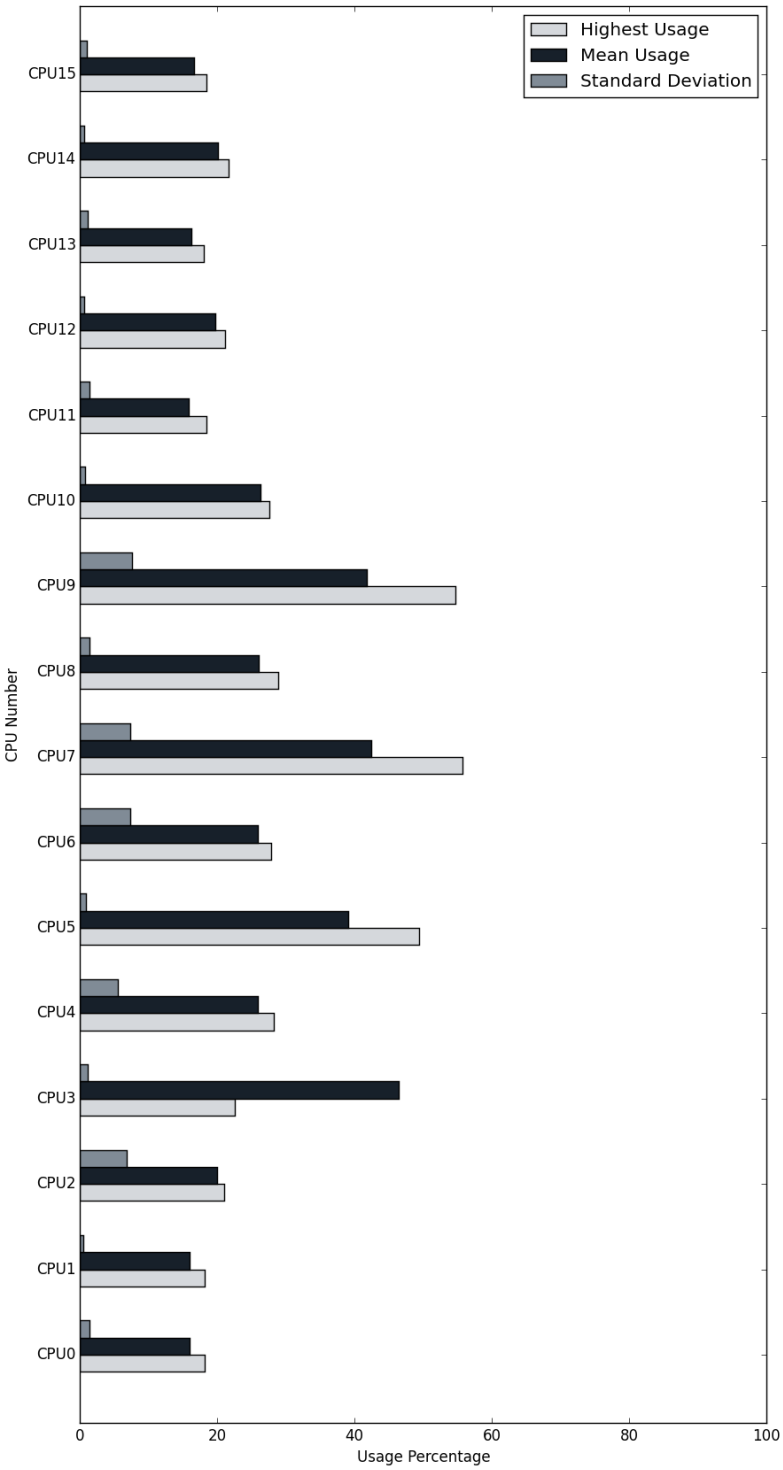

Figure 27: Graph CPU Performance Per CPU During Mongoimport

Type	Mean Usage in %	Highest Use in %	Standard Deviation
CPU 0	27.5671	99.833444	38.2188
CPU 1	27.5671	100.0	38.2188
CPU 2	38.2188	100.0	36.9437
CPU 3	9.99241	100.0	24.4719
CPU 4	16.3257	100.0	34.8598
CPU 5	7.4331	100.0	20.4161
CPU 6	12.0529	100.0	29.6994
CPU 7	12.6071	100.0	27.5199
CPU 8	31.2449	100.0	44.3446
CPU 9	37.6214	100.0	40.4369
CPU 10	23.6925	100.0	40.3025
CPU 11	7.47638	100.0	22.7909
CPU 12	1.19342	100.0	7.21468
CPU 13	4.96719	100.0	17.4896
CPU 14	1.82377	100.0	8.84589
CPU 15	3.24615	100.0	12.504

Table 11: Full Capture Experiment: Performance Statistics Per CPU Measured At 1 Minute Intervals During Mongo Import

Highest usage	Mean Usage	Standard Deviation
2.0	0.0263327	0.063251

Table 12: TCP Dump Utilization: Performance Statistics Per CPU Measured At 1 Minute Intervals For The Duration Of 12 Hours

Documents	Size Bytes	Storage Size Bytes	Avg Object Size	Index Size
4030050	14,5 GiB (15,533,405,143)	5,5 GiB (5,921,763,328)	3,8 KiB (3,854)	41,7 MiB (43,696,128)

Table 13: Full Capture Experiment: MongoDB Storage Space Used

Filename	Filesize
IPERF.pcap	14 GiB 14188688907 Bytes
IPERF.json	84 GiB 89862268165 Bytes

Table 14: Full Capture Experiment: Storage Space Used Per Capture File

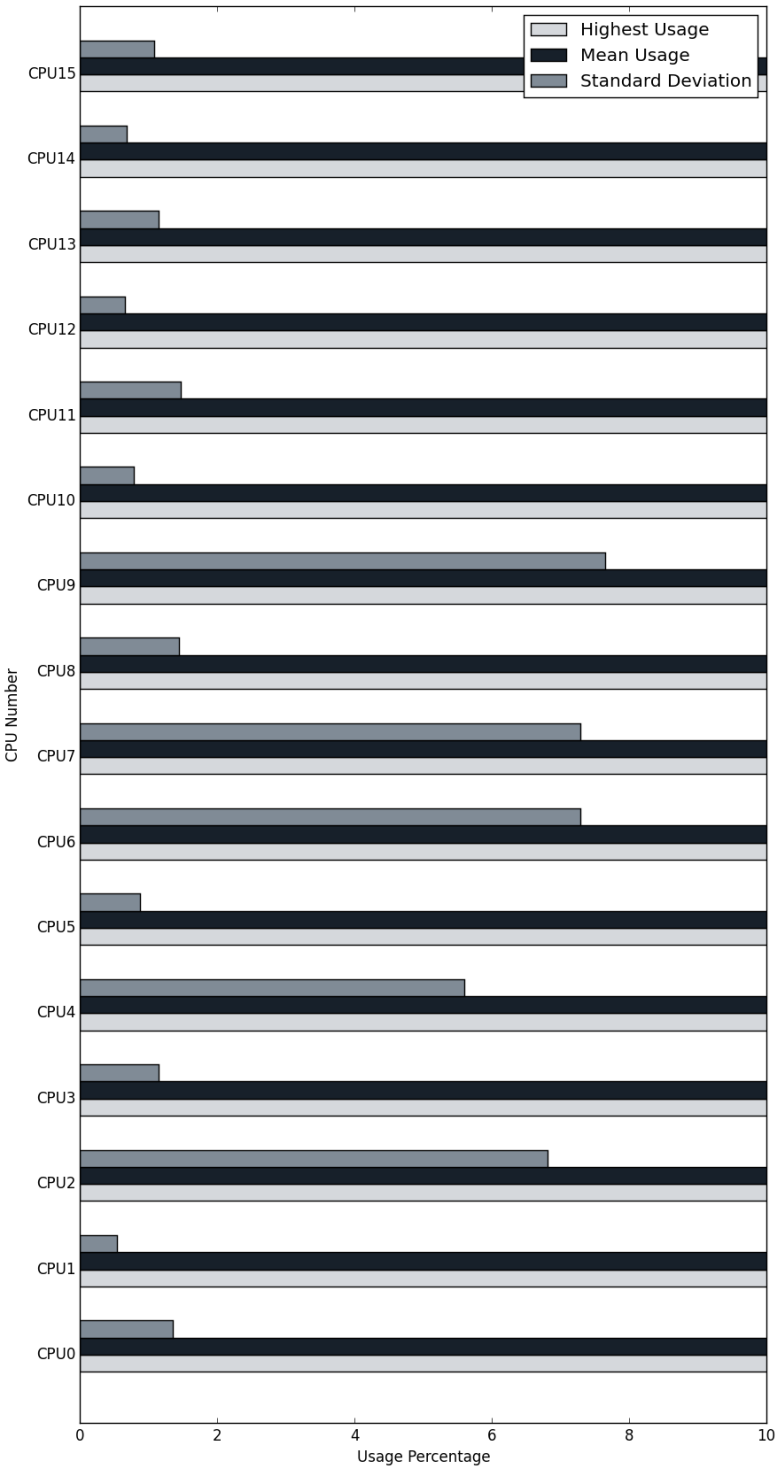

Figure 28: Full Capture Experiment: Performance Statistics Per CPU Measured At 1 Minute Intervals For The Duration Of 12 Hours

Time	Used Memory in %
2018-01-24 20:18	59.5816846516
2018-01-24 21:18	60.0025161816
2018-01-24 22:18	60.6165106165
2018-01-24 23:18	61.5161816681
2018-01-25 00:18	62.6118146848
2018-01-25 01:18	64.6881681616
2018-01-25 02:18	65.1841689468
2018-01-25 03:18	65.8963186981
2018-01-25 04:18	58.1238419846
2018-01-25 05:18	58.9816816818
2018-01-25 06:18	58.2346818132
2018-01-25 07:18	58.2515616816
2018-01-25 08:18	58.3026841848

Table 15: Full Capture Experiment: Memory Performance Statistics Measured At 1 Minute Intervals For The Duration Of 12 Hours

Type	Mean Usage in kB/s	Highest Use in kB/s	Standard Deviation
Read	1224.81	1244.67	12.0139
Write	5608.87	5695.28	48.92

Table 16: Full capture: Disk Performance Statistics Measured At 1 Minute Intervals For The Duration Of 12 Hours