CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Securing Communication and Identifying Threats in Cloud Microservices

An Analysis on Zeek and Effectiveness of Traditional Security Methods.

Master's thesis in Computer Systems and Networks

TEJASWINI PRIYANKA R

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Securing Communication and Identifying Threats in Cloud Microservices

An Analysis on Zeek and Effectiveness of Traditional Security Methods.

TEJASWINI PRIYANKA R

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Securing Communication and Identifying Threats in Cloud Microservices
An Analysis on Zeek and Effectiveness of Traditional Security Methods
TEJASWINI PRIYANKA R

Academic Examiner:
Philippas Tsigas
Professor, Networks and Systems division
Department of Computer Science and Engineering
Chalmers University of Technology

Academic Supervisor:
Ismail Butun
Postdoc, Networks and Systems division
Department of Computer Science and Engineering
Chalmers University of Technology

Industrial Supervisor:
Mazen Harake
Manager, Software Factory
CEVT AB

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of an Intrusion Detection system

Typeset in LaTeX
Gothenburg, Sweden 2023

Securing Communication and Identifying Threats in Cloud Microservices
An Analysis on Zeek and Effectiveness of Traditional Security Methods.
TEJASWINI PRIYANKA R
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The automotive industry has grown from making simple passenger vehicles to vehicles which are completely aware of its surroundings. Every car which is being manufactured now promises intelligent functionalities and the cars keep getting smarter every year.

As vehicles become part of the IoT cyberspace, similar to smart home assistants, televisions etc., they are exposed to many dangerous threats by default. Hence, an automobile manufacturer's idea of safety and security needs to evolve from seatbelts and airbags to include the cybersecurity factor. The vehicle should be completely secure in all aspects, which certainly includes its network infrastructure to avoid any attempts to cause mishaps or accidents.

Newer technology is being adopted for providing several advantages and one such technology is Microservices Architecture. In simpler words, Microservices are a method of developing software applications which are made up of independently deployable, modular services. Each microservice runs a unique process and communicates through a well-defined, lightweight mechanism such as a container, to serve a business goal. However, one must also keep in mind that microservices are unmonitored, work on zero trust and they have complex communication.

The crux of this thesis involves investigating about how to secure these microservices, if a traditional Intrusion Detection System works effectively in the Microservice environment considering the drastic change in architecture. An open source IDS Zeek, which is acting as the traditional security method is used to demonstrate the process for monolithic and microservice based attacks. The corresponding results are thoroughly analyzed and suitable conclusions are drawn.

Keywords: cybersecurity, Zeek, Intrusion detection system, Microservice architecture, Monolithic architecture

Acknowledgements

I would like to express my deep sense of gratitude to my academic supervisor, Ismail Butun and thesis examiner Philippas Tsigas for their continuous guidance and support throughout this thesis. Their knowledge and subject matter expertise is invaluable and helped me overcome the difficulties faced during the course of this thesis work.

I thank Yusuf Kürşat Tuncel for his kind assistance and expert feedback in strengthening my thesis.

I would also like to thank my Industrial supervisor Mazen Harake from CEVT AB, for his time and devotion to this thesis. It was a distinct pleasure working with him on this thesis topic.

My special thanks to all other colleagues at CEVT AB who helped me during this thesis work. I thank them for their support in providing the necessary data and also guiding me during the course of this thesis.

I thank Edit Kárász for her guidance and support throughout the thesis.

Finally, I would like to thank all my friends and family for their constant love and support.

Tejaswini Priyanka, Gothenburg, June 2023

Abbreviations

API - Application Program Interface
BSD - Berkeley Software Distribution
CEVT AB - China Euro Vehicle Technology Aktiebolag
CIA - Confidentiality, Integrity and Availability
CIC - Canadian Institute of Cybersecurity
CSRF - Cross-Site Request Forgery
DDOS - Distributed Denial-of-Service
DHCP - Dynamic Host Configuration Protocol
DNS - Domain Name System
DoS - Denial of Service attack
ECU - Electronic Control Unit
ELK - Elasticsearch, Logstash and Kibana
FTP - File Transfer Protocol
GUI - Graphical User Interface
HTTP - Hypertext Transfer Protocol
HTTPS - Hyper Text Transfer Protocol Secure
IBM - International Business Machines Corporation
IDS - Intrusion Detection System
IP - Internet Protocol
IoT - Internet of Things
LDAP - Lightweight Directory Access Protocol
MITRE ATT&CK - MITRE Adversarial Tactics, Techniques Common Knowledge
ML - Machine Learning
MS - Microservices
MSA - Microservices Architecture
MTLS - Mutual Transport Layer Security
MiTM - Man-in-the-middle attack
NSM - Network and Systems Management
OAIT - Open Artificial Intelligence Technologies
OS - Operating System
OWASP - Open Web Application Security Project
PPS - Packets Per Second
RBAC - Role Based Access Control
RPC - Remote Procedure Calls
SMTP - Simple Mail Transfer Protocol
SQL - Structured Query Language
SSH - Secure Shell protocol
SSL - Secure Sockets Layer
TCP - Transmission Control Protocol
TLS - Transport Layer Security
UDP - User Datagram Protocol
URI - Uniform Resource Identifier
XSS - Cross site scripting

Contents

List of Figures **xiv**

List of Tables **xvi**

1 Introduction **1**
1.1 Problem Statement . . . 1
1.2 Aim and Purpose . . . 2
1.3 Hypothesis . . . 2
1.4 Limitations . . . 3
1.5 Methods . . . 3

2 Technical Background and Theory **5**
2.1 Automotive cybersecurity . . . 5
2.2 Defense In Depth . . . 6
2.3 Intrusion Detection Systems(IDS) . . . 6
2.3.1 Overview of Intrusion Detection Systems . . . 6
2.3.1.1 What is Intrusion Detection? . . . 6
2.3.1.2 Necessity for IDS . . . 7
2.4 Types and functionalities of IDS . . . 7
2.4.1 Anomaly based detection . . . 7
2.4.2 Signature based detection . . . 7
2.5 State of the art . . . 8
2.6 Docker . . . 8
2.7 Kubernetes . . . 9
2.8 Network Traffic Analysis . . . 9
2.9 Zeek . . . 9
2.9.1 Zeek Architecture . . . 10
2.9.2 Zeek scripting . . . 10
2.9.2.1 Part I . . . 10
2.9.2.2 Part II . . . 11
2.9.2.3 Part III . . . 11
2.9.2.4 Description of Part II and Part III . . . 11
2.10 Microservices Architecture (MSA) . . . 12
2.11 Security In General Computer Systems . . . 12
2.11.1 Security in Microservices . . . 13
2.12 Microservices security threats . . . 15

2.13 Security issues in Microservices . . . 16
2.13.1 Broader attack surface . . . 16
2.13.2 Poor performance due to distributed security screening . . . 16
2.13.3 Logging and Monitoring stress . . . 16
2.13.4 Compromised application security . . . 17
2.14 Web application security risks in Microservices . . . 17
2.14.1 Top countermeasures . . . 18
2.15 MITRE ATT&CK® framework (attacks) for Kubernetes platform . . . 19
2.16 Service Mesh Architecture . . . 20
2.16.1 Service mesh security in Microservices architecture . . . 21
2.16.1.1 Security features of service mesh . . . 21
2.16.2 Istio . . . 22
2.17 Comparison of IDS- Snort vs Suricata vs Zeek . . . 22

3 Related Work **26**
3.1 Literature summary and the road ahead . . . 27

4 Design and Implementation **28**
4.1 Phase One . . . 28
4.1.1 Environment setup . . . 29
4.1.2 Creating test setup and baseline . . . 31
4.1.3 Analysis of data post baseline creation . . . 31
4.2 Phase Two . . . 31
4.2.1 Introduction of Zeek into environment . . . 32
4.2.1.1 Monolithic architecture based attacks . . . 32
4.2.1.2 IDS evaluation dataset for monolithic architecture based attacks . . . 32
4.2.1.3 Monolithic based attacks based on CIC - IDS2017 . . . 33
4.2.1.4 Microservice attacks based on MITRE ATT&CK® . . . 33
4.2.2 Procedures applicable to both Monolithic and Microservices based attack scenarios . . . 35
4.2.2.1 Creating test scenario . . . 35
4.2.2.2 Simulating attacks . . . 35
4.2.2.3 Processing packet captures . . . 36
4.2.2.4 Visualizing logs using BRIM . . . 36
4.3 Implementation of solution . . . 36

5 Results **37**
5.1 Phase One Results . . . 37
5.1.1 Visualization of protocol data post baseline creation . . . 37
5.1.2 Visualization of Cumulative Network Protocols post baseline creation . . . 39
5.2 Phase Two Results . . . 40
5.2.1 Monolithic architecture based attacks . . . 40
5.2.2 Microservice based attacks . . . 43

6 Proposed Solution **45**

6.1 Istio service mesh to secure Microservice communication 45
6.1.1 Architecture of the application 45
6.1.2 Overview of Steps performed 46
6.1.2.1 Create local Kubernetes cluster 46
6.1.3 Testing Security features of Istio 47
6.1.4 Monitoring 48
6.1.5 Visualization 48
6.1.6 Mitigation of security issues by service mesh 51
6.1.6.1 Mitigation of MITRE ATT&CK using Service mesh . 52
6.2 Comparison of Service mesh vs Traditional Security 52

7 Conclusions **54**
7.1 Lessons learnt 54
7.2 Discussions 54
7.3 Future Directions 55

Bibliography **56**

List of Figures

1.1 Overview of Methods 4

2.1 Defense in Depth 6
2.2 Zeek Architecture 10
2.3 Perimeter Security 14
2.4 MITRE ATT&CK® Matrix [44] 19

4.1 Phase 1: Building the baseline 29
4.2 Hierarchcial view of Kubernetes architecture 30
4.3 View of single Micsoservice 30
4.4 Environmental setup 31
4.5 Phase 2: Introduction of Zeek - Monolithic Attacks 32
4.6 Phase 2: Introduction of Zeek - Microservices 34
4.7 Selected attacks from the "MITRE ATT&CK® Matrix for Kubernetes: Tactics & Techniques" [44] 34

5.1 Network Protocol Analysis - Kubernetes Pod A 37
5.2 Network Protocol Analysis - Kubernetes Pod B 37
5.3 Network Protocol Analysis - Kubernetes Pod C 38
5.4 Network Protocol Analysis - Kubernetes Pod D 38
5.5 Network Protocol Analysis - Kubernetes Pod E 38
5.6 Network Protocol Analysis - Kubernetes Pod F 38
5.7 Network Protocol Analysis - Kubernetes Pod G 38
5.8 Network Protocol Analysis - Kubernetes Pod H 38
5.9 Network Protocol Analysis - Kubernetes Pod I 39
5.10 Cumulative network protocols (Graph shown in logarithmic scale of base 10) 39
5.11 Graph representing attempts and detection of attacks 41
5.12 SQL Injection 42
5.13 FTP brute force 42
5.14 SSH Brute Force 42
5.15 Heartbleed and HTTP Stalling 43
5.16 Port scan 43
5.17 Port scan- Detailed view 43

6.1 Solution: Service Mesh 45
6.2 Architecture of the microservice application 46

6.3 Kubernetes cluster 47
6.4 Pods in the cluster 47
6.5 Book-info web application 47
6.6 Kubernetes dashboard 49
6.7 Istio dashboard 49
6.8 Service dashboard 50
6.9 Workload dashboard 50
6.10 Kiali dashboard 51

List of Tables

2.1 MSA threats and attacks 15
2.2 Comparison - Snort vs Suricata vs Zeek [36] 24
2.3 Performance metrics- Suricata vs Snort vs Zeek- CPU Usage [36] . . . 25

5.1 Number of packets per protocol in Network protocol analysis as conducted in Subsection5.1.1 40

6.1 Comparison - IDS & NSM Zeek vs Service mesh Istio 53

1
Introduction

The past decade has been a witness to outstanding developments in digital technologies which have eclipsed the decade of personal computers, revolutionary innovations like Open Artificial Intelligence Technologies (OAIT) like Machine Learning (ML), Internet of Things (IOT), Deep Learning, Big Data Analytics and several others. These cutting-edge technologies are quick in ways which they affect human lives and in the way they work. The companies that use these latest technology achieve significantly higher profit, increased productivity and performance. They aid in removing inefficiencies and understanding the customers requirements.

Cyber-Physical systems are now utilized to embed the driver assistance systems, safety and control systems into the automobiles of today. These automobiles depend on the complex software for multiple functionalities. These software evolve fast and increase the complexity, communication, computing and they control the infrastructure which make up the cyber and physical components. These cyber-physical devices of the mission critical components of a automobile are vulnerable to a list of security challenges, threats, intrusions and cyberattacks.

CEVT being a well-known innovation centre for the Zhejiang Geely Holding Group is making its safe cars safer by looking at the vulnerabilities in its network infrastructure which are the backbone of the connected cars. When broken down from a network infrastructure level, microservices form the basic and most complex entities of the connected cars. The microservices communicate in complicated fashion, work on zero trust, they are unmonitored and make a large attack surface. Microservices come with their own security challenges including establishing trust between microservices, containers, secret management, etc. Even the smallest vulnerabilities if found by wrong hands might be exploited, causing accidents or other serious incidents.

1.1 Problem Statement

The automotive sector is a newbie into using the internet for connected car services for autonomous driving cars. Cybersecurity needs to be added to their priority list to protect passenger lives. The usage of newer microservice architecture which has several security flaws puts the supporting infrastructure at risk of attack. Adoption of new technology considering only the advantages puts an organization into risk for

attacks. As said by Jim Manico (OWASP-Stammtisch München, 9th Nov 2018) – *"API security is at least a couple of years behind other types of web security" [5]*

Microservices Architecture uses container orchestration software for automating deployments and management of applications which run in containers and provide several other benefits. However, they introduce many complexities. It can be difficult to secure and are often exploited using their misconfigurations.
The thesis emphasizes the importance of security, for the microservices and that there needs to be quicker action from security software makers to adapt to the trend and cater to the new evolving architectures. The focus of this thesis is not on Zeek or Istio but on Intrusion Detection Systems and Service Meshes as a whole. Zeek and Istio are used as tools to assist with analysis.

1.2 Aim and Purpose

This thesis work is inspired from the research papers "Towards Effective Virtualization of Intrusion Detection Systems" [12] and "Security-as-a-Service for Microservices-Based Cloud Applications [8]" which present their solutions for solving the security requirements of microservices. The primary goal of this thesis is to analyse mitigation methods and find solutions that help in addressing the security concerns in Microservice communication. The thesis discusses the reasons why traditional security methods are not enough for microservices and the mitigation methods that help to protect the infrastructure by preventing attacks. The additional objectives are also to compare the IDS under investigation to other open source IDS's, to present visualization- monitoring logging options and presenting other work in this area.

1.3 Hypothesis

The research papers [8] and [12] discuss that regular Intrusion Detection Systems (IDS) which are used in the case of monolithic architecture-based applications do not apply effectively in the case of a microservice based environment. This is mainly due to the complexity and the security challenges of the microservices[30]. A signature-based IDS which works on detecting attack signatures might not detect a microservice-based attack and an anomaly-based IDS which works by detecting a difference in the regular functioning will have to be trained with a machine learning algorithm to detect anomalies in microservice. As previous research has been conducted on this topic [27], it will not be a part of the current thesis scope.

Security in microservices communication is a major concern as discussed by Dragoni et al.[29]. The security needs to be suited to the requirements of microservices. But current organizations deploy generic security measures used by Monolithic architecture and there is a risk of compromising services and data.

The thesis aims to investigate the security challenges in microservices, analysing if a traditional security method is effective, investigate and propose the use of service

meshes building on the work of "Building Secure Microservices-based Applications Using Service-Mesh Architecture"[38] as defense-in-depth methodology to protect communication in microservices.

1.4 Limitations

- Only a limited number of attacks will be chosen to demonstrate microservice based attacks on Microservices.
- Zeek will be used to study IDS and Istio will be used to demonstrate the security features of service mesh. Other contemporary tools such as Snort, Suricata, etc., will not be used in this thesis work, however a comparison study of these tools will be presented.
- Due to confidentiality, real world implementations of attacks and its consequences will not be shown. However, a generic method and it's implementation is presented.
- Due to the Non-Disclosure Agreement signed by the author, finding the right level of abstraction for the environment and scenarios for documentation purposes will be challenging.
- Real world implementation of comparison study between Zeek vs Snort vs Suricata will not be conducted. Only a theoretical comparison study is presented.
- The thesis work limits the focus to security features of service mesh only.

1.5 Methods

A brief overview of methodology adopted in this thesis work is shown in the Figure 1.1 with explanation of the process below.

- **Background work**
  - **Environment analysis**
    - Understanding the current microservices cloud architecture in the organization.
    - Identifying the communication protocols used.
    - Spotting areas in the environment which are viable to being compromised.
    - Identifying risks in environment and the reason for proposing an IDS solution.
  - **Creating test setup**
    - Communicating project requirements to operations team and acquiring replica environment.
    - Testing replica infrastructure.
- **Building a baseline**

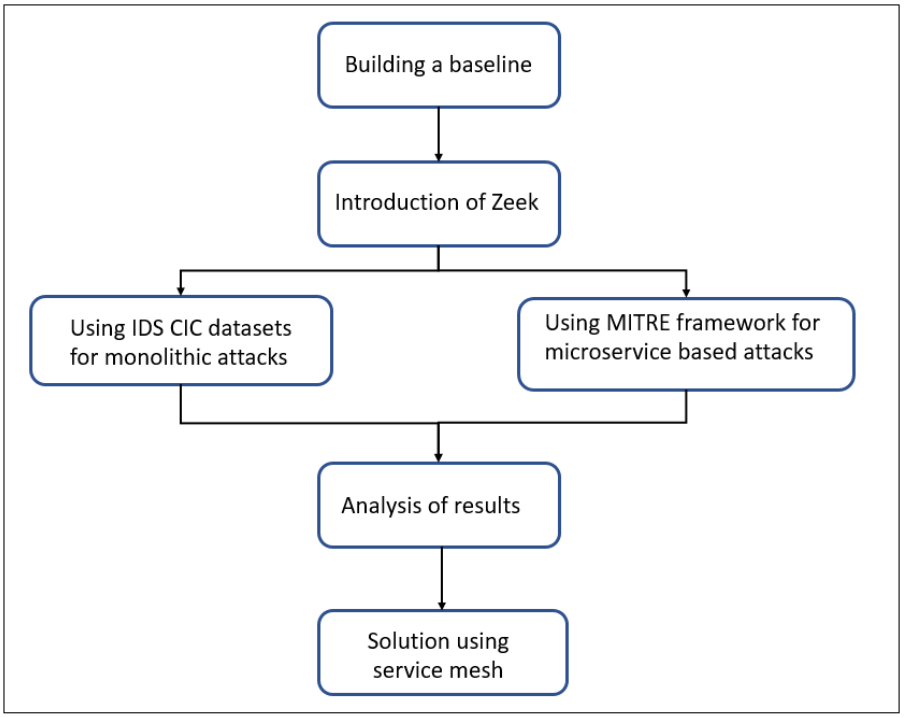

Figure 1.1: Overview of Methods

  - Creating a baseline of the current network traffic to record current state of system.
  - Using Wireshark[4] to identify protocols in the network activity.

- **Introduction of Zeek**
  - Using IDS CIC dataset for monolithic attacks
  - Using MITRE ATT&CK framework for microservice based attacks

- **Analysis of results**
  - Analysing the performance of Zeek for monolithic and microservice based attacks.

- **Solution using Service mesh**
  - Creating an application(microservices) for demonstrating security features of Service mesh Istio
  - Configuring the Monitoring for the microservices
  - Configuring the Logging and visualization for the microservices
  - Summarizing the findings in a comparison study on Zeek and Istio.

2

Technical Background and Theory

Recent times have seen a large section of the industry allotting budgets to cybersecurity infrastructure and research, making it one of the top priorities while building their network. There have been numerous attacks on many well known organizations like Citrix, U.S. Customs and Border Protection/Perceptics, Yahoo, GitHub and so on. Security vendors like Citrix have a hard time in establishing security internally. The intention behind these attacks could be financial gain, politics, competition. Organizations are investing their time and effort in keeping consumer data and their internal data safe from the eyes of attackers. The subsequent sections describe concepts that are relevant to this thesis work and will assist in giving a better understanding of the methods adopted.

2.1 Automotive cybersecurity

In the case of automobiles, the traditional security approach has always been to focus most of the resources on the most critical system components and to protect them from threats. This implies that there are some components or systems of lesser importance, which are unprotected and vulnerable to attacks. Present day automobiles are targets of cyberattacks mainly due to their complexity. A majority of high-end vehicles contain

- More than 100 Embedded Electronic Control Units (ECUs)
- More than three kilometers length of cables
- More than 100 million lines of software
- More than five in-vehicle networks

The widespread availability of the Internet allows cyber attackers to attack the cyber-physical systems from anywhere and at anytime. There can be multiple reasons for these cyberattacks such as Denial of Service (DOS), theft of sensitive data, to compromise functioning of components/ entire vehicle or harm passengers. The automobile industry is facing an era of digital transformation, mainly because of the shift from product-based to service-based business strategies, need for climate friendly alternatives, opportunities concerning security, connectivity and drastically changing customer behaviour.

2.2 Defense In Depth

Protecting a computer network is done with a series of defensive mechanisms. The intent of this mechanism is to provide redundancy in case a vulnerability is exploited or a particular security mechanism fails. There are many attackers who have a variety of attack methods available and hence, it is not possible to have a single method to protect a computer network.

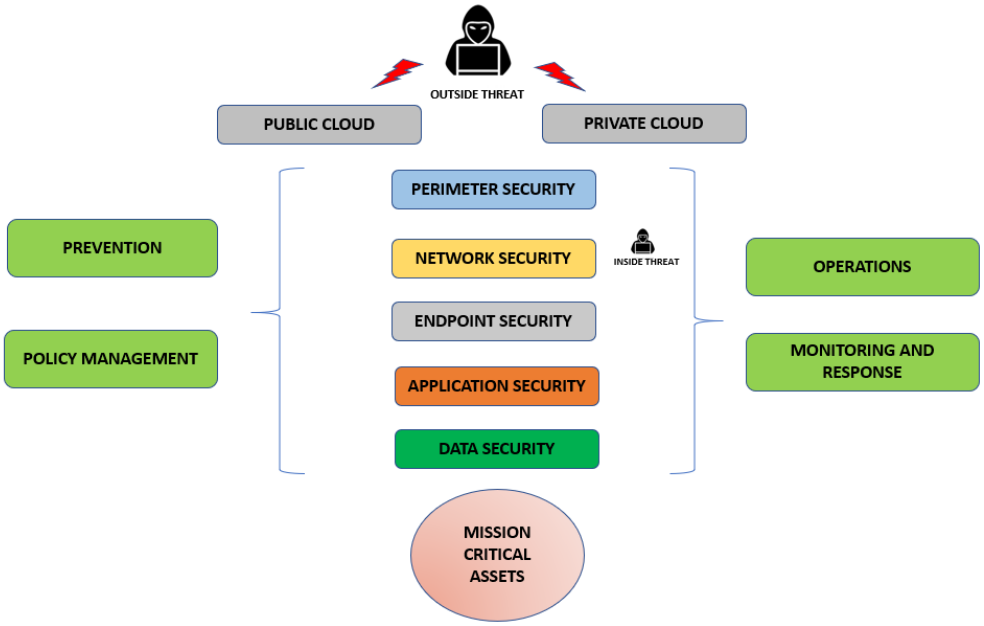

Figure 2.1: Defense in Depth

2.3 Intrusion Detection Systems(IDS)

IDS are software or hardware systems which automate the process of monitoring events which occur in a computer system or in a network and then analyze them for security issues. IDSs have become a necessity to the security infrastructure of any organization due to several networks and security breaches.

2.3.1 Overview of Intrusion Detection Systems

2.3.1.1 What is Intrusion Detection?

Intrusion detection is the process of monitoring events that occur in a computer system or network and analysing them for signs of *intrusion* which is defined as an attempt to compromise the confidentiality, integrity or availability. Attackers try to gain additional privileges or misuse the privileges by accessing systems from the Internet. IDSs automate this process of constantly monitoring the network for intrusions and assist with the analysis process.

2.3.1.2 Necessity for IDS

Organizations nowadays are heavily dependent on extensive network connectivity due to remote office sites. With intrusion detection, organizations can protect their systems from various threats. IDSs are used to detect attacks and security violations. IDS is used as a tool to document the existing threats to an organization. IDS provides improved diagnosis and also acts as a quality control agent for security design and administration for complex organizations.

2.4 Types and functionalities of IDS

IDSs are classified into two major types-

- Anomaly based detection
- Signature/knowledge or rule based detection

2.4.1 Anomaly based detection

Anomaly based detection identifies unusual behaviour which is on a host or a network. They work on an assumption that attacks differ from "normal" or legitimate activity and so they can be detected by systems which can compare these differences. Anomaly detection involves construction of profiles which represent the regular or normal behaviour of a user, host or a network. These profiles are often built from historical data which is collected over time showing normal operation. Following this, the detectors collect event data and use a variety of methods to identify when the activity of the "item" being monitored deviates from the normal.

Assuming that an anomaly based IDS is deployed on microservices, it will take a considerable amount of time for detecting the intrusion and it will also have a significant number of false positives initially. Considering that new services are spun-up very frequently, there is a high probability that these new services will be classified a threat. There are other studies which are deploying Machine Learning[27] [31] and Automatic Anomaly detection [28]. These studies are also discussed in the related work chapter.

2.4.2 Signature based detection

This method uses a pre-recorded traffic which contains known attacks and these are used as a baseline to identify future attacks. The advantage of this type is that it can accurately detect all the known attacks. The drawback is that new attacks that have not been profiled will not be detected. This type of IDS is more like an anti-virus software which detects a known attack pattern. Given the number of constantly scaling microservices in any environment, consolidating the communication logs from the microservices will be a huge task. Manually analysing logs is extremely cumbersome because of the large volume of logs and packets. Application security is critical, but this method will hinder progress.

2.5 State of the art

- **Snort** - An open source intrusion detection and intrusion prevention system which was developed in 1998. It has the ability for real time traffic analysis and packet logging. It uses rule-based pattern matching techniques in detecting intrusions. Snort rules require action (Deny/Accept), protocol, IP addresses and port numbers. Snort cannot detect zero-day attacks or attacks which are not recorded in its rules. Providing rules manually for microservices is extremely cumbersome because of the volume of microservices.

- **Suricata** - This is also a signature-based IDS like Snort which uses extensive rules and signature language. Suricata rules consist of an action (Deny/Accept), protocol, IP addresses and direction of rules. Suricata has a higher system overhead than Snort and is less accurate in stressed environments [36]. Suricata allows to create dynamic rules which is difficult to create in Snort.

- **Zeek** – A network security monitor which is used to detect suspicious activity and troubleshoot the network. Unlike Suricata and Snort, Zeek focuses majorly on network analysis. Zeek can log and store network activities for several protocols like DNS, FTP, HTTP, SSH and SSL. The logs can be processed with the help of external log searchers (for example BRIM). Zeek can also be integrated with ELK for further analysis. Zeek uses pattern-based intrusion detection and it uses an event-based programming model for anomaly detection. Zeek supports clustered deployment and many workers can process the traffic streams.
  Zeek is further explained in detail in section 2.9 with relevance to the current thesis scope.

2.6 Docker

Let's say there are three applications which are Python based and they need to be hosted on a single server. Each of these applications use different versions of Python and have different dependencies and required libraries, they are different from one another. There cannot be different versions of Python installed in the same machine. This issue can be solved by having three physical machines or with one physical machine which is powerful enough to host three virtual machines on it. Both the solutions are expensive as the right hardware is necessary.
However, using Docker in the above scenario makes the process less cumbersome and is less expensive as well.

Docker is a platform for developing, shipping and running applications[34]. Docker is independent of libraries or versions of software. Docker helps with **Task isolation**, **support for multiple coding languages**, **multi cloud support** and offers many other advantages.

2.7 Kubernetes

Kubernetes is an open source orchestrator used for deploying containerized applications. It was developed by Google. Since the introduction of microservices, many independent components which are individually scalable and upgradable are brought with it. There is now a huge number of services and they all need individual attention which is impossible to achieve. All these applications need a global management software and one such application is "Kubernetes, K8s or kube" which automates all Linux container operations. Kubernetes eliminates the need for manual processes in deploying and scaling containerized applications.

The basic terminology related to Kubernetes is explained below:

- **Pods** - It is the smallest and simplest Kubernetes object. A Pod represents a set of running containers on the cluster.
- **Node** - It is a worker machine in Kubernetes.
- **Cluster** - A set of worker machines(nodes) which run containerized applications.

2.8 Network Traffic Analysis

The process of Network Traffic Analysis involves recording, reading and analyzing communication patterns in Network traffic, this is done to detect and to respond to security threats[6]. This project uses a kubectl plugin called *ksniff*[5] which makes use of very popular packet capture tools -*tcpdump*[3] and *Wireshark*[4] for starting a remote packet capture on any pod in the Kubernetes cluster.

2.9 Zeek

Zeek is an open source security and network traffic analyzer. The important advantage one can gain from installing Zeek is that there is a detailed set of log files of the network activity. The inbuilt features of Zeek include a range of analysis and detection tasks which is useful in detecting malware, detecting SSH brute forcing, reporting older vulnerable versions of software which are seen on the network and so on. Zeek uses custom scripting and also has plenty of pre-built functionalities. Zeek does not require specific hardware and hence, it's a low-cost solution. Zeek is extremely customizable and configuring Zeek is done by adding scripts. The working of the SSH Brute force Zeek script used in this thesis work is explained in the Subsection 2.9.2.

Zeek is used for detection of malicious activity, detection of anomalies and for behavioral analysis. A huge number of organizations use Zeek for protecting their infrastructure. Zeek is used in this thesis to test against Monolithic based attacks based on Intrusion Detection Dataset CIC-IDS2017[22] and microservice based attacks using the MITRE ATT&CK.

2.9.1 Zeek Architecture

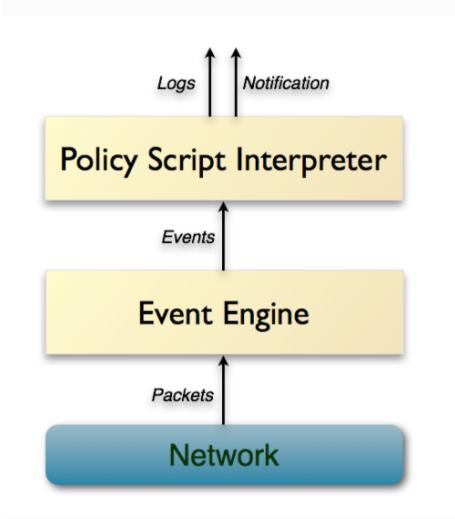

Figure 2.2: Zeek Architecture

Figure 2.2 explains the architecture of Zeek which consists of two major components. The *Event Engine* reduces an incoming packet stream into a stream of events. These events are further classified into layer based events by sub components. The second component is the *Script Interpreter* which executes the event handlers which are written in the custom scripting language. Scripts can be used to get desired parameters and statistics from the incoming traffic, generate real time alerts and execute external programs.

2.9.2 Zeek scripting

Zeek has its own event-driven scripting language, which makes it easy to extend and customize Zeek's capabilities. Zeek being an event-driven language, the events that need response require to be defined when it is encountered during network traffic analysis. A Zeek script has three distinct sections, Part I has a base level with no indentations. The libraries are included in the script using " **@load**". These directives ensure that files framework, and the notice framework has been loaded by Zeek. A namespace is defined using "**module**". Part II describes the export section. Part III is about the event queue and event handlers. The instance below explains how a Zeek script works in parts, using SSH brute forcing showing how Zeek captures(logs) a host which is trying to guess passwords. The code excerpts are explained later in detail in Subsubsection 2.9.2.4.

2.9.2.1 Part I

```
@load base/protocols/ssh
@load base/frameworks/sumstats
```

```
@load base/frameworks/notice
@load base/frameworks/intel

module SSH;
```

2.9.2.2 Part II

```
export {

        redef enum Notice::Type +=
        {
            Password_Guessing,     ...(i)
            Login_By_Password_Guesser,
        };
        redef enum Intel::Where +=
        {
            #An indicator for login
            SSH::SUCCESSFUL_LOGIN,  ...(ii)
        };
const password_guesses_limit: double = 30 &redef;  ...(iii)
const guessing_timeout = 30 mins &redef;     ...(iv)
const ignore_guessers: table[subnet] of subnet &redef;...(v)
```

2.9.2.3 Part III

```
event zeek_init()
        {
        /...
        NOTICE([$note=Password_Guessing,        ...(vi)

        $msg=fmt("%s appears to be guessing
        SSH passwords (seen in %d connections).",
        key$host, r$num),

        $sub=sub_msg,
        $src=key$host,
        $identifier=cat(key$host)]);
```

2.9.2.4 Description of Part II and Part III

Part II, the **export** section refines an enum which describes the type of notice which will be generated. The **Notice::Type** is extended and allows the Notice function to

generate notices. It generates extra notices other than the default log types, which can be an email notification which is sent to an email address. In this example shown-

- (i) indicates that a host which was identified as a password guesser has had a successful login attempt.
- (ii) is an indicator of the login for the intel framework.
- (iii) shows the number of failed SSH connections before a host is confirmed as a host which is guessing passwords.
- (iv) indicates the amount of time to remember the successful logins.
- (v) can be used to whitelist hosts or networks from being tracked.

Part III shows event handling as Zeek processes the network traffic, the events are handled in first-come-first-serve basis.

- Event (vi) generates the notice and displays the IP address of the host which is trying to guess passwords.

2.10 Microservices Architecture (MSA)

Microservice Architecture is a relatively new architectural style which is inspired by Service-Oriented Architecture. Microservices was first mentioned in 2011 and is a relatively new term. Hence, there is no agreement as to how it should be defined. Few of the definitions given by Dragoni [10] is:

Definition 1:*A microservice is a cohesive, independent process interacting via messages.*

Definition 2:*A microservice is a distributed application where all its modules are microservices.*

The definitions above meant that microservice architecture is composed of individual independent microservices. These individual components are responsible for their own functionality.
Since it is established that there are no formal definitions for microservices, there is a requirement to describe the characteristics of the architecture for a better understanding.

2.11 Security In General Computer Systems

There are three integral properties that need to be satisfied to ensure security. These properties are described in Williams and Stallings [17] are as follows-

- **Confidentiality**
  - **Data confidentiality:** Makes sure that confidential data is not available to unauthorized parties.
  - **Privacy:** Guarantees that individuals control on what information related to them may be collected and who else can access that information.
- **Integrity**
  - **Data integrity:** Assures that information is changed only in an authorized manner.
  - **System integrity:** Guarantees that the system performs ins functions without being manipulated by any other parties.
- **Availability:** Assures that the system is working and that the services are not denied to any authorized users.

The above listed concepts are referred to as **CIA triad**. However, there is also a need for additional concepts to support the triad. They are as follows

- **Authenticity:** The quality of being genuine and trusted.
- **Accountability:** The systems must be able to assure against non-repudiation. The systems must maintain records of activities to permit for later forensic analysis to trace security breaches and to aid in disputes.

2.11.1 Security in Microservices

Microservices being relatively new and extremely popular in the industry brings new security challenges which were not present in the monolithic applications. There is a need for effective microservice security solutions, which are easy to automate and also lightweight.

With organizations migrating services from monolithic to microservices architecture, code that was inaccessible through Web APIs are now unprotected and exposed. There is a demand for prioritizing microservices security which is unmet. Fetzer[7] explores on how microservices could be used to build critical systems if it has secure containers and compiler extensions. Sun[8] discusses an Intrusion Detection System which is based on Software Defined Network. Despite there being several research works which are being carried out, a structured understanding of microservice security is absent.

Tetiana[9] argues what the microservices security is a multifaceted problem and it is heavily dependent on the technology that is used and the environment. The microservice security issues can be broken down into following layers-

- **Hardware**
  Hardware security issues are hidden under abstraction. Nevertheless, they are still a security and reliability concern. Bugs in hardware are very dangerous as they chip away security mechanisms of the other layers. Backdoors in hardware can be introduced during manufacturing.
- **Virtualization**
  Operating System processes offer very little separation from other services

within the same system. Containers and virtual machines offer more protection against compromised services. Attacks against virtualization layer include hypervisor compromise, shared memory attacks and sandbox escape. Usage of malicious images are also a major security concern.

- **Cloud**
  Security issues with regard to cloud computing includes unlimited control of cloud provider over everything it operates.
- **Communication**
  This layer is affected by the classic attacks on network stack and protocols, attacks against SOAP, RESTful Web services etc. This layer is also vulnerable to attacks like eavesdropping, identity spoofing, session hijacking, Man-in-the-Middle and Denial of Service attacks. This layer can also attack TLS layer using Heartbleed.
- **Service or Application**
  This layer is mainly affected by SQL injection flaws, access control, broken authentication, exposure of sensitive data, Cross-Site Scripting(XSS).
- **Orchestration**
  The attacks concerning this layer include compromising discovery service, registering malicious nodes in the system and redirecting communication to them.

This thesis mainly focuses on the communication layer which consists of network stack, protocols, attacks against protocol.
Until recent times perimeter defense was widely used in the security of microservice-based systems. But, perimeter security as shown in Figure 2.3 is inadequate[32] because if an attacker were to breach the perimeter, they will be allowed free movement in the network. This is when zero trust security model assists[14]. There should be an assumption that other services in the system might be compromised and this may affect the state of the entire system. Defense in Depth as described in Figure 2.1, is a concept of placing overlapping security mechanisms.

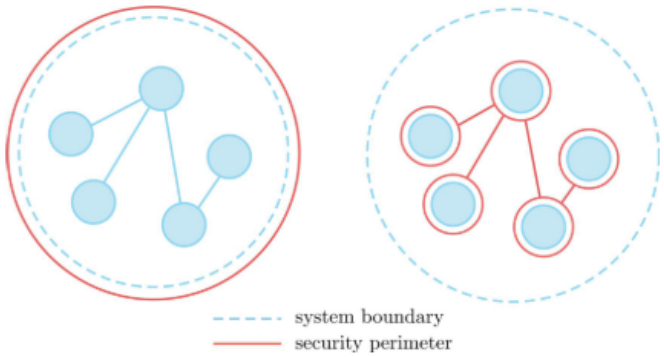

Figure 2.3: Perimeter Security

As mentioned by Dragoni[10], microservice pattern poses several trust and security challenges. This section highlights what the key security issues are-

- **Greater surface attack area** - Applications that utilize monolithic architectures communicate via internal data structures or via sockets. The attack

surface is also controlled by a single Operating System. On the otherhand, the feature of microservices is that the applications are broken down into services and they communicate with each other via APIs, which are exposed to the network. Hence, it is more exposed to attacks than the conventional methods.

- **Network complexity** - Microservices are created in such a fashion that there are several independent applications which interact with each other which results in complex network activity. There is an increased difficulty in debugging and to perform forensic analysis. This may be a vulnerability that attackers would exploit.
- **Trust** - Microservices are designed to trust each other. Hence when there is a compromised microservice which may be used to attack, this also brings down the whole application.

2.12 Microservices security threats

Microservices architecture bring new security threats and vulnerabilities. These threats may be internal attacks or external attacks. For securing microservices, both internal and external attacks need to be detected and prevented[33]. The classification of microservices attacks based on target attacks is shown below and the attacks based on the threat categories are presented in table 2.1

- **User-based attacks:** Attacks where the users are involved directly or indirectly.
- **Data attacks:** Attacks which target sensitive data which can be manipulated.
- **Infrastructure attacks:** Attacks which target the monitoring, load balancer etc. which are the architectural elements of the microservice architecture.
- **Software attacks:** These are attacks involving manipulation of code or injection of scripts for malicious purposes.

Threats	Attacks
User-based Attacks	Brute Force Attack, Spoofing, Unauthorized Access, MITRE ATT&CKS
Data Attacks	Heartbleed, MiTM, Sniffing attack, MITRE ATT&CKS
Infrastructure Attacks	Compromise containers, Port Scans, Misconfiguration, MITRE ATT&CKS
Software Attacks	DoS, SQL Injection, Cross site scripting,MITRE ATT&CKS

Table 2.1: MSA threats and attacks

2.13 Security issues in Microservices

Microservices have emerged as an architectural design pattern aiming to solve the problems of scalability, ease of maintenance of online services. However security breaches are also threatening the confidentiality, integrity and availability of microservices based systems.

An application which uses monolithic architecture has very few entry points and needs lesser security screening. Not every component of the application is displayed to the outside world, it accepts requests directly. The more the number of entry points, the broader the attack surface is for the application, this is explained in detail in section 2.13.1. Most monolithic applications have centrally enforced security and the individual components do not have to take carry out additional checks. The security model is much more uncomplicated and straightforward than that of an application built around microservices architecture. Due to the built-in nature of microservices, security is challenging.

2.13.1 Broader attack surface

Communication within internal components happens in a single process in monolithic applications as compared to microservices where these internal components are built to be independent and seperate microservices which uses remote calls to communicate. Each of these microservice independently accepts requests and they have their own entry points. As the number of entry points increases, the attack surface broadens. Each entry point needs to be protected equally without any weak links.

2.13.2 Poor performance due to distributed security screening

Individual microservices in a MS deployment need to carry out security screening independently. These recurring, distributed security checks could heavily impact latency and degrade the performance of the system. Few overcome this by simply trusting the network and avoid security checks at each individual microservice. Many industries are moving towards zero-trust networking principles and each carry out security closer to the resource in the network.

2.13.3 Logging and Monitoring stress

The microservice applications are distributed, the services are independent and hence will have more logs. There is a high chance of high importance issues being camouflaged among other logs. The distributed nature of microservices creates complexity with logging in a centralized location. As new services are built instantaneously, maintaining and configuring a monitoring system area new challenge. The constant interaction of microservices with each other makes them dependent on each other. It is not enough to have just logs of individual microservice, the logs need to record the interactions and dependencies too. Logging in microservices is a

completely different from logging for monolithic applications. There is more data, more logs and more complexity in microservices. The logs need to be managed in a more streamlined way.

2.13.4 Compromised application security

Applications are what make up the microservices. With every additional service that is created, there is a challenge of maintenance and configuring the monitoring. In the case that the environment employs a perimeter security, then all incoming traffic is trusted and is allowed free movement. This doesn't just affect one microservice, but also compromises the entire application.
The various threats to application security are discussed in the next section 2.14

2.14 Web application security risks in Microservices

The Open Web Application Security Project(OWASP) is a foundation dedicated to improving the security of software. The OWASP API security Top 10's [35] aims at preventing companies from using vulnerable API's. API's are used to expose microservice to the external world . Listed below is the top ten list of API security vulnerabilities:

- **API1:2019 Broken Object Level Authorization**- APIs often expose the endpoints which handle object identifiers. This creates a larger attack surface. The mitigation method requires authorization checks in functions which access data sources.
- **API2:2019 Broken User Authentication**- The implementation of authentication is often incorrect which allows attackers to compromise authentication tokens or impersonate other users. Eg. Brute force attacks. This can be resolved using mTLS for service to service communications.
- **API3:2019 Excessive Data Exposure**- Developers usually expose object properties despite their sensitive nature which can be exploited.
- **API4:2019 Lack of Resources & Rate Limiting**- APIs do not have any restrictions on the number of resources which can be requested by a client. This can severly affect server performance leading to Denial of Service and also brute force attacks
- **API5:2019 Broken Function Level Authorization**- Access control policies are complex with roles, groups and have unclear separation between administrative and regular functions. This leads to authorization flaws. Attackers can gain administrators access. This can be mitigated by securing the services based on mTLS
- **API6:2019 Mass Assignment**- Combining user provided data to data models without filtering based on an allow list leads to Mass Assignment. This

allows attackers to manipulate object properties which they are not authorized to access.

- **API7:2019 Security Misconfiguration**- Security misconfigurations are common in ad-hoc configurations, verbose error messages containing sensitive information etc. This can be solved by securing communications based on TLS.

- **API8:2019 Injection**- Injection flaws like SQL, Command injection occur when malicious data is sent to the interpreter as part of a command or query. The attacker can trick the interpreter into executing commands and accessing unauthorized data.

- **API9:2019 Improper Assets Management**- There are more endpoints exposed by an API than a traditional web application. Deprecated API versions and exposed endpoints need to be consistently updated in the documentation.

- **API10:2019 Insufficient Logging Monitoring**- Insufficient logging and monitoring allows attackers to attack systems, destroy data and infect other systems. This can be mitigated by setting up a centralized logging and monitoring system which can trace all events and logs across services.

2.14.1 Top countermeasures

The top countermeasures for the security risks mentioned in section 2.14 are discussed the research paper "Security in Microservices Architectures[19]" are below:

- **Authentication and Authorization**- The main objective of authentication and authorization of microservices is to deploy systems which can be trustworthy and have centralized authentication methods like RBAC

- **Securing Data at Rest**- Data breaches often take place in protected environments because it is reachable at a poorly secured location. Even though defense in depth methodology is deployed, it is necessary to ensure stored data is encrypted

- **Defense in Depth**- The architecture of an application should consider controlling the services of ports and incoming services. The defense should act in layers to protect the applications

- **Network and Subnet**- The services should be spread across different networks or subnets and control them with a firewall. Their connections and active ports and MAC addresses etc need to be defined

- **Logging and Monitoring**- The activities of the services need to be logged and monitored consistently. This needs to be a centralized activity and the interaction between the services also needs to be logged

The design and implementation work is motivated by the reasons mentioned in the above section to analyse the usage of traditional security methods like Intrusion Detection Systems for Microservices. The solution which is proposed to mitigate the risks in the OWASP Top 10 and the MITRE ATT&CK tries to encompass all these security requirements.

2.15 MITRE ATT&CK® framework (attacks) for Kubernetes platform

The MITRE ATT&CK® framework is a repository of information on the methods and strategies that attackers employ to breach and attack Kubernetes clusters. An attacker typically plans how to enter a cluster and cause damage by going through the steps of an attack lifecycle. For the attack to be successful, the attacker must move through each of these phases. The figure **??** below shows the attacks which are selected

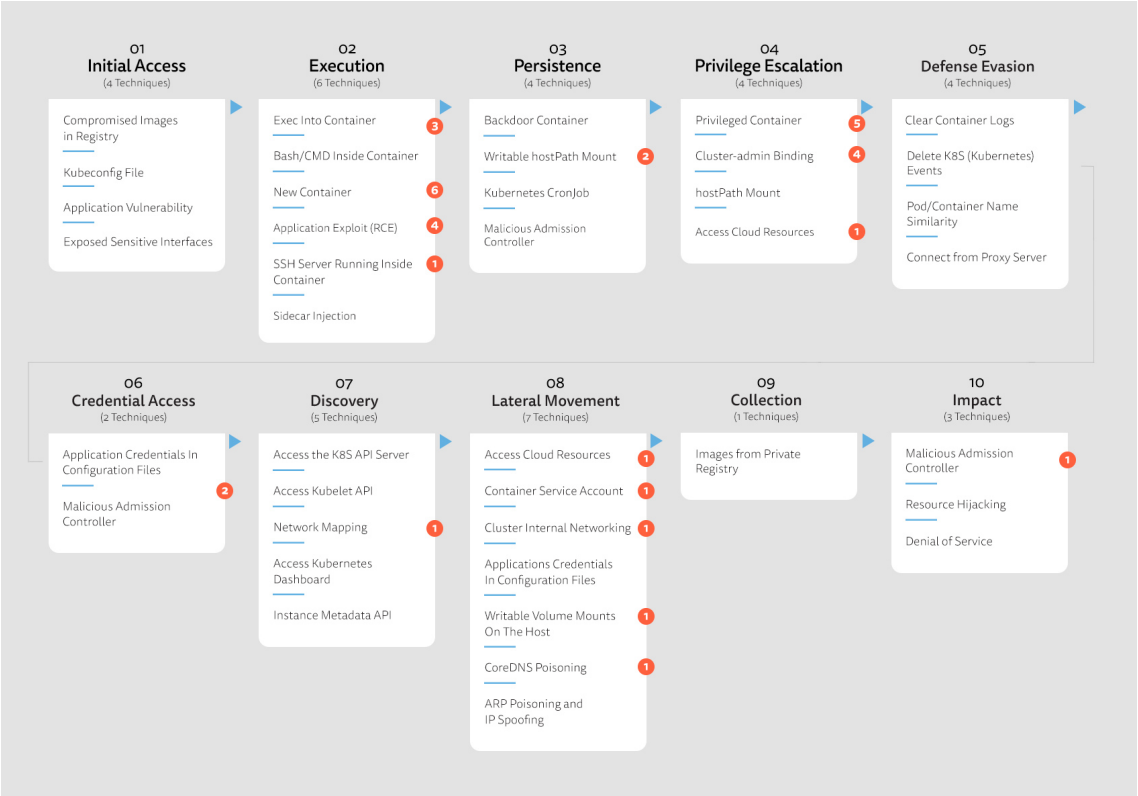

Figure 2.4: MITRE ATT&CK® Matrix [44]

MITRE ATT&CK framework tactics includes the following stages

- **Initial access**
  This is the first stage of attack. These techniques in early stage mostly rely on unpatched vulnerabilities and credential leaks. The attacker might perform this attack by compromising various resources which are deployed on the

cluster.

- **Execution**
  The attacker executes code inside a Kubernetes cluster to exploit a vulnerability.
- **Persistence**
  Maintaining a continuous connection to the compromised target is very important to have access to its resources. This step is mostly performed by exploiting a backdoor.
- **Privilege escalation**
  This stage is used to acquire higher privileges. Mostly done by accessing a node through a container or even using cloud resources.
- **Defense evasion**
  This technique is used to avoid detection, it includes tactics like deleting evidence of an attackers existence or hiding the directions to gain access to a resource.
- **Credential access**
  After gaining elevated privileges, set backdoors and dodged the defenses, during this stage the attacker now looks for data and credentials.
- **Discovery**
  The attacker uses this method of attack to attain information about the internal networking and the components of the cluster.
- **Lateral movement**
  With code execution complete and attacker traces removed, the adversary can now control and access all the nodes in the cluster. This technique allows an adversary to control and access the network and resources remotely.
- **Collection**
  This tactic includes techniques attackers use to gather information apart from the cluster which was compromised.
- **Impact**
  This tactic is the destruction or disruption of the resources in the compromised environment. This is the end goal for the adversary, destruction of data, hijacking of resources and denial of service are some of the techniques.

The thesis uses this framework to choose the attacks to be detected by Zeek IDS. This is further explained in detail in subsection 4.2.1.4 and illustrated with the diagram of the framework in Figure 4.7.

2.16 Service Mesh Architecture

The concept of service mesh was defined by William Morgan [26] in 2017 as follows:
A service mesh is a dedicated infrastructure layer for handling service-to-service communication. It's responsible for the reliable delivery of requests through the complex topology of services that comprise a modern, cloud native application. In practice, the service mesh is typically implemented as an array of lightweight network proxies that are deployed alongside application code, without the application needing to be aware.

2.16.1 Service mesh security in Microservices architecture

Regardless of the type of architecture used, security is very important. Microservices when compared to other architecture styles have additional security needs. Authentication, authorization, traffic-flow control, extensive logging of the communication. Traditional security involves having a strong perimeter for checking unauthorized access. Once this perimeter is passed, all the users inside this perimeter are considered trusted and are allowed to communicate without verifying their identity.

Zero trust is a security model that conveys the message of not trusting any system internal or external to your network. The system must be verified before establishing trust and then granted minimal access to resources that the system requires. The concept of *zero trust* [14] was very difficult to achieve before the arrival of service mesh. A traditional security model depends on a firewall to protect against incoming traffic. Although in a cloud-based environment the network perimeter still needs to be protected, a perimeter-based security model is no longer sufficient. With a zero-trust security model, the incoming traffic is not trusted by default. It requires other security controls like authentication and encryption. Service meshes provide authentication and authorization via a central certificate authority which provides a certificate for every service.

Service meshes provide platform developers the power to enforce policies like mutual TLS, encrypting traffic between services and to avert man-in-the-middle attacks. It is the complete responsibility of the service mesh to provide all these services.
A service mesh is a tool for adding observability, security and reliability features to applications by inserting these features at the platform layer than at the application layer. In a microservice based environment service meshes remove the burden on individual services for implementing these controls and it allows for simpler, centralized management accross all the microservices.

With the rise in usage of microservice based architectures for application building on Kubernetes, a service mesh provides the much needed observability, reliability and security features. The biggest advantage of using a service mesh is that the application does not need to implement these features.

2.16.1.1 Security features of service mesh

There are several features of service mesh architecture [15] [38] out of which, this thesis focuses only on the security features that service mesh offers.

- **Authentication and authorization**- Certificate generation, whitelist, blacklist, API keys, key management.
- **Secure communication**- Mutual Transport Layer Security (mTLS), encryption etc.
- **Traffic-flow control**- The service mesh controls the flow of traffic between the services, incoming traffic and outgoing traffic. The mesh maintains a registry of all services.
- **Observability and monitoring features**- These are a set of activities which

involve measuring, gathering and analyzing signals. Metrics, distributed tracing and access logs are the various kinds of telemetry provided.

2.16.2 Istio

Istio [16] is Google's open source project for connecting, securing and managing microservices. Istio is chosen as an instance for service mesh as it is open source and is readily available. The main features of Istio are as follows:

- Control flow of traffic between services
- Provide access control and policies for applications
- Provide security at both network and application layers by securing communication channel.
- Manages authentication, authorization and encryption of communication channels

The focus of this thesis is only on the security features of Istio. Istio service mesh has two parts, the control plane and data plane. The control plane has these components for security:

- **Mixer**: This component provides the logging and monitoring of all the traffic.
- **Pilot**: This component configures and deploys policies to the sidecar proxies.
- **Citadel**: This component manages keys and certificates for authorization.

The security features offered by Istio are listed in the following:

- **Certificate management**
  - Configured the Istio Certificate Authority with a root certificate, signing certificate and key.
  - Verifies the certificate chain from root to workload certificate.
- **Authentication with mTLS**
  - Enabled, configured Istio authentication policies.
  - Mitigates Man-in-the-Middle attacks and replay attacks.
- **Authorization**
  - RBAC to map users to roles and assigning permissions.
  - Setup authorization for HTTP traffic.
  - Setup authorization for TCP traffic.
  - Enforcing IP based access control on Ingress gateway (controls and monitors the incoming traffic).

2.17 Comparison of IDS- Snort vs Suricata vs Zeek

For the purpose of this thesis Zeek was chosen as the IDS. Suricata and Snort was chosen as the other open source IDS for comparison. The IDS Zeek is be compared with other open source IDS Snort and Suricata in Table 2.2. The parameters which are of interest are the CPU Usage, Packet loss and the IDS's support for microservices. The general capabilities of Snort and Suricata are presented in Section 2.5

The IDS Zeek was investigated under different traffic rates and attacks using the dataset, number of packets lost and CPU usage metrics. The results are presented in Subsection 5.2.1. Suricata and Snort are capable of multi-threading,this means that they can utilize hardware with multiple CPU's/cores to distribute the workload. Suricata and Snort uses rule based detection which are brittle and require constant updating to make sure they are effective. When ever a rule needs to be created or re-done it puts the infrastructure at risk. Event based detection however, requires lesser attention. Snort does not adapt well to high speed networks and experiences packet loss. Zeek is able to log and record the packets and events. Snort and Suricata have challenges to provide support to Microservices since manually making rules for a microservice environment is very complex and tedious. Table 2.2 provides a comparison between the three IDSs [36].

The Performance metrics mentioned in Table 2.2 can be further explained with the investigation carried out using the paper [36] where CPU usage, packet loss and number of alerts were recorded with varying traffic rates. The results in Table 2.3 show that the CPU usage of Suricata and Snort increased quickly with increasing traffic rates for TCP traffic, while Zeek flattened at higher rate. It was also noted that Suricata has the highest CPU usage for higher traffic rates for UDP traffic. Zeek performed consistently for both TCP and UDP traffic.

Parameter	Snort	Suricata	Zeek
Type	Signature based IDS	Signature based IDS	Network security monitor and logging tool.
Supports multi-threading?	Yes	Yes	Single thread and supports single core system
Detection method	Rule based detection	Rule based detection	Event based detection
Capabilities	Difficult to adapt to complex threats with high speed networks	Handles complex threats	Capable of recording/logging detailed network behaviour and triggering events
CPU usage for TCP traffic	Increases rapidly with increasing traffic rates	Increases rapidly with increasing traffic rates	Flatter at higher rates
CPU usage for UDP traffic	Increases rapidly with increasing traffic rates	Highest CPU usage for higher traffic rates	Flatter at higher rates
Packet loss (TCP & UDP)	Packet loss	Less TCP packet loss	Performs consistently for TCP and UDP
Microservice support	Manually adding rules will be a challenge	Manually adding rules will be a challenge	Event based log recording and alerting. Can be used for network logs in small microservice environments
Operating system	Linux, FreeBSD, OpenBSD	Linux, FreeBSD, OpenBSD, Windows	Linux, FreeBSD, MacOS X

Table 2.2: Comparison - Snort vs Suricata vs Zeek [36]

Traffic Rate	**TCP**			**CPU**		
	Suricata	**Snort**	**Zeek**	**Suricata**	**Snort**	**Zeek**
50	5.0	11.0	10	9.0	12.0	17
100	9.0	12.0	9.0	11.0	16.0	19.0
200	16.0	15.0	10.0	27.0	18.0	19.0
300	23.0	16.0	30.0	34.0	25.0	20.0
400	22.0	26.0	31.0	44.0	29.0	20.0
500	25.0	32.0	39.0	43.0	29.0	27.0
600	38.0	44.0	40.0	54.0	45.0	33.0
700	45.0	48.0	40.0	67.0	52.0	37.0
800	47.0	50.0	42.0	68.0	56.0	36.0
900	48.0	52.0	43.0	69.0	59.0	37.0
1000	54.0	60.0	43.0	73.0	68.0	43.0
2000	58.0	62.0	46.0	81.0	70.0	55.0

Table 2.3: Performance metrics- Suricata vs Snort vs Zeek- CPU Usage [36]

3
Related Work

This chapter briefly summarizes the scientific work that supports this thesis. These research works have contributed in development of ideas for testing and methods used in this thesis.
The main ideologies of this thesis were inspired from the academic works titled "Security-as-a-service for microservices-based cloud applications"[8], "Overcoming security challenges in microservice architectures"[9] and "Microservices: yesterday, today and tomorrow"[10].They explore the various security challenges in microservices. The work also describes the impacts of using microservices like increased difficulty in monitoring of applications, zero trust, compromise of multiple applications. The project builds a flexible monitoring and policy enforcement infrastructure for network traffic for securing cloud applications.

The authors in work [9] describe how current state of the art only provides a very narrow view into the security concerns of microservices. The work offers two main insights, one being that microservice security is a multi faceted problem which requires a layered security solution. The other being that when these security challenges are solved, it increases the robustness of a system.
The authors of [10] talk about the evolution of microservices starting with the history of software architecture, current problems and discussing the future challenges. This survey offers an academic viewpoint about microservices architecture and also discusses issues and solutions.

The National Security Agency's Cybersecurity and Infrastructure Security Agency's Cybersecurity Technical Report called "Kubernetes Hardening Guidance" [41] described the various security challenges associated with configuring and setting up a Kubernetes cluster. This report explains primary actions which include scanning containers and pods for vulnerabilities, misconfigurations, firewalls and log auditing.

The authors of the special publication "Building Secure Microservices-based Applications Using Service-Mesh Architecture"[38] have made a good effort to address security in all aspects of service to service communication. They discuss resiliency techniques, continuous monitoring and methods to solve the security issues in microservices. The work also describes service mesh as the best known method which can cater to these requirements.

The thesis work "Automatic Anomaly Detection and Root Cause Analysis for Microservice Clusters"[28] explains the difficulty in managing, monitoring and debug-

ging a large microservice cluster which is deployed in the cloud. Reporting anomalies should be the main priority in microservice environments. This thesis uses the service mesh *Istio*[16] as a proxy, Istio uses Envoy[24] as a proxy. A sidecar container with Envoy is added to each microservice pod. Istio adds some components which acts like a control plane enabling the traffic management, security and configuration for the proxies. The author describes the system having two phases, *learning phase* which observes the normal behaviour of the microservice application and a *detection phase* which tries to detect anomalies and find root causes. In order for this system to function it uses monitoring metrics gathered from a microservice cluster. Another thesis titled "Machine Learning for a Network-based Intrusion Detection System An application using Zeek and the CCIDS2017 dataset" [22] works by training a Machine Learning system with the CCIDS2017 dataset which contains both benign and malicious traffic.
This method can be used for real time ML-detection for new attacks which have not been recorded. The accuracy of the IDS after implementing the ML model is calculated using the Bayes theorem. This thesis serves as an inspiration for the future work in this area where Machine learning can be used to train a system to detect attacks.

3.1 Literature summary and the road ahead

The papers reviewed above have assisted in setting the theme for this thesis. They have described the various challenges which are part of building a secure infrastructure, they also discuss solutions to mitigate these issues. For instance using Machine learning in association with an IDS. They explain the importance of continuous monitoring and logging and addressing misconfigurations etc. This thesis uses the above papers as support and demonstrates how traditional security methods are unsuccessful in microservice based environments. The solution explores the idea of using a service mesh for securing the communication within microservices and addressing other security requirements of microservices. The thesis titled "Machine Learning for a Network-based Intrusion Detection System An application using Zeek and the CCIDS2017 dataset" [22] paves way for discussion and work around using Machine learning algorithms for detecting intrusions. Developing smarter infrastructure which do not require manual intervention becomes an absolute necessity.

4

Design and Implementation

This chapter includes the methodology adpoted for the thesis work and the implementation of the same. The methodology explained in the Figure 1.1 is further divided into two phases for ease of understanding.

Phase one involves observing the current state of the microservices and building a baseline. In Phase two, the IDS is introduced and is subject to attacks and the IDS is further evaluated. The below sections highlight the stages in the project and document the environment and implementation methods.

Phase One:
- Background work
- Building a baseline

Phase Two:
- Introduction of Zeek into environment.
- Analysis of data post Zeek installation.

Implementation of solution:
- Setting up service mesh.
- Analysis of service mesh solution.

4.1 Phase One

The Figure 4.1 shows the overall steps in Phase 1 of thesis for building a baseline. Each individual step is described in detail in the subsections.

The initial step is to perform the analysis of the current environment and prepare the software requirements if any and acquiring permissions from the organization. The next step is to setup a replica of the real environment. This is done to avoid any issues with disruption of production traffic. A detailed in-depth understanding of microservices functioning with Docker containers and Kubernetes is necessary. These steps complete the background work which is required.
Ksniff tool is used to capture packets, identify protocols and then to create a summary of the vulnerabilites that could exist. This replica setup will be used for the investigation.

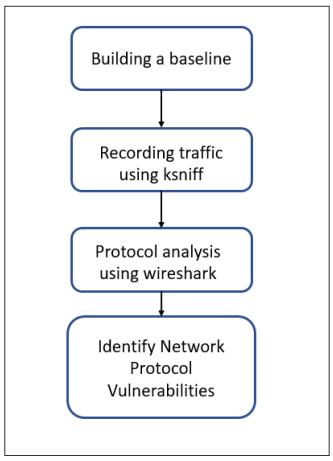

Figure 4.1: Phase 1: Building the baseline

4.1.1 Environment setup

This section focuses on the design and architecture of the virtual infrastructure and how the microservices are built. This segment also dives into how Microservices, Docker and Kubernetes collaborate in the environment.

Figure 4.2 shows the hierarchy in the Kubernetes architecture which is used in this thesis with multiple nodes and pods. The names of the pods, nodes and services are not added in this figure due to confidentiality. The environment is setup with docker containers, Kubernetes pods and Nodes which are all connected by a Master node. The formal definitions of these terms are given below-

- **Docker Container -** A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. [20]
- **Kubernetes Pod -** A Pod is a group of one or more application containers (such as Docker or rkt) and includes shared storage (volumes), IP address and information about how to run them.
- **Kubernetes Nodes -** A Pod will always run on a Node. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the Master.

The management of the Kubernetes cluster is done with the use of *kubectl* which is a command line tool. Kubectl uses Kubernetes API to interact with the cluster. Figure 4.3 is used to illustrate what a single microservice looks like. Multiple Pods may run on one Node and this figure shows a Pod running on one Node.

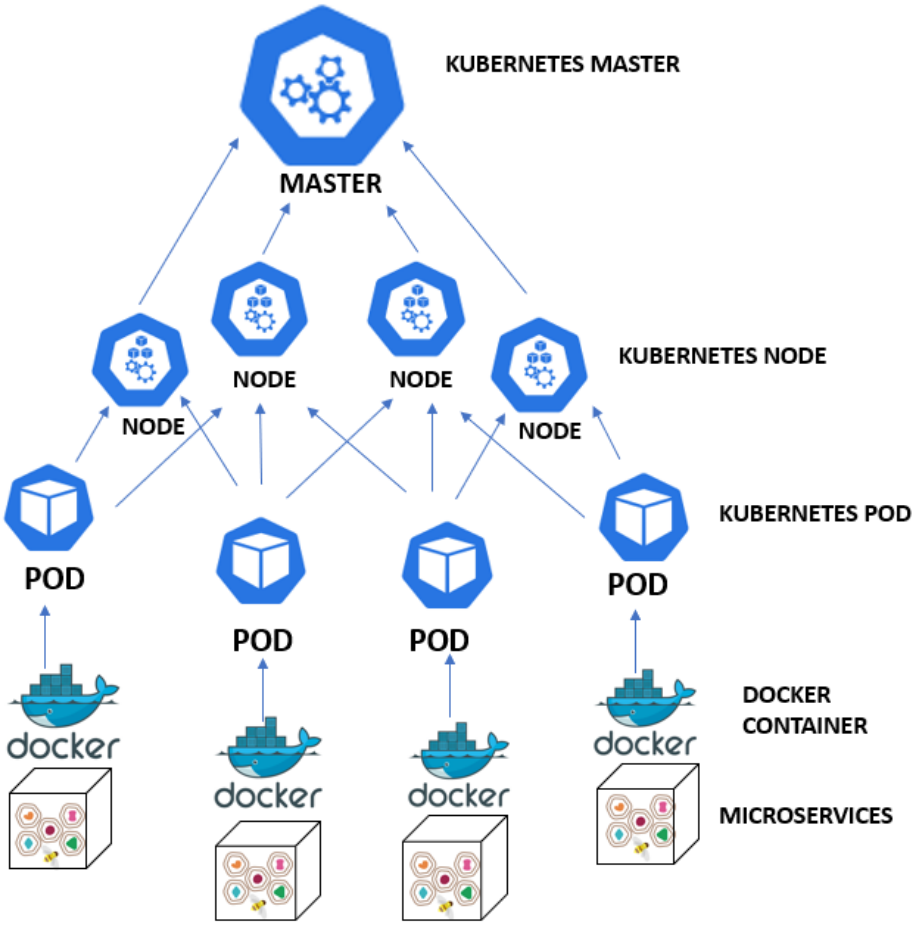

Figure 4.2: Hierarchcial view of Kubernetes architecture

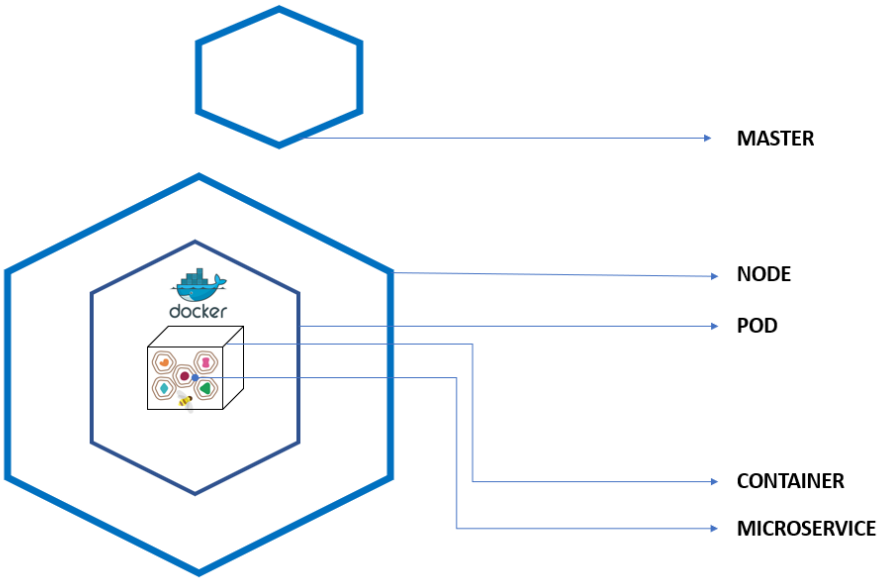

Figure 4.3: View of single Micsoservice

4.1.2 Creating test setup and baseline

This section highlights the procedures that were followed to setup the replica test environment for conducting the tests. The previous section 4.1.1 illustrated how the environments are setup in organizations, it must kept in mind that redundancy, high availability and also single point of failures are considered in production environment.
This project uses a replica of the setup as discussed in the previous section. There is a need to record outgoing and incoming traffic from the microservices. Since there are several layers of abstraction surrounding the microservices, traffic was recorded via *kubectl* plugin *ksniff* 2.8). *ksniff* is used for performing a remote capture on the pods, this plugin is useful to capture network activity between the microservices.
Using kubectl, ksniff is run on every Pod belonging to Nodes. This process needs to be carried out with caution since the network should be recorded for at least five hours. This might crash applications and make applications unresponsive.

4.1.3 Analysis of data post baseline creation

The incoming and outgoing traffic from the pods contains traffic between the microservices is captured. The output is directed to *Wireshark* where the .pcap files can be extracted. After extraction of the .pcap files, these files are converted to a user-friendly format like .csv and the patterns in the protocols are analyzed to find any vulnerabilities. The analysis of the results are done in the next chapter.

4.2 Phase Two

The previous section described in detail the components involved in construction of a microservice environment and how a test setup and a baseline was created. This section explains the steps which were performed after introducing the IDS in the environment.

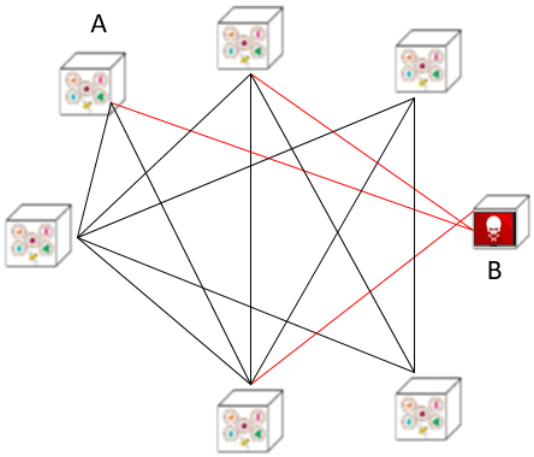

Figure 4.4: Environmental setup

4.2.1 Introduction of Zeek into environment

The Figure 4.4 provides a gist of the setup which has been created. The component A is a regular microservice which is communicating with other microservices. The component B is pod which uses TCP-replay to replay the attack dataset which is described in the Subsection 4.2.1.2. While the attacks are being sent via the communication channel, the network traffic is collected in format using ksniff. Since Zeek was not developed into a microservice, it is setup away from the cluster and the packet capture is analysed with it to produce zeek logs which is later analysed using BRIM. The setup described above is utilized for two scenarios, one for simulating monolithic based attacks and the other for simulating microservice based attacks. The figures in the respective sub sections provide an overview of the steps which have been performed.

4.2.1.1 Monolithic architecture based attacks

The test setup described in the previous section is used in this case to simulate monolithic based attacks with the help of the IDS Evaluation Dataset 4.2.1.2. The Figure 4.5 iterates briefly over the steps performed in this section of the thesis. The steps are explained in detail in further sub sections.

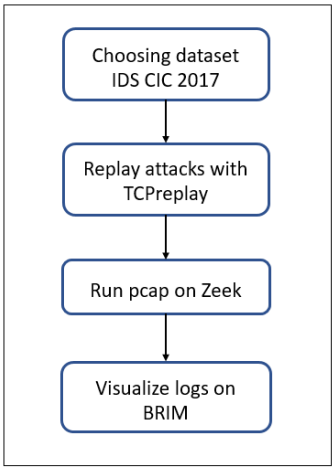

Figure 4.5: Phase 2: Introduction of Zeek - Monolithic Attacks

4.2.1.2 IDS evaluation dataset for monolithic architecture based attacks

The dataset utilized in this thesis is the **"Intrusion Detection Evaluation Dataset-CIC - IDS2017[22]"**.
The CICIDS2017 dataset comprises of the most recent common attacks and benign traffic which mimics real world data. This dataset has built the behaviours of 25 users based on HTTP, HTTPS, FTP, SSH and email protocols. To give a brief background, the data was captured during a period of five days from Monday, July 3, 2017 until 5 P.M Friday, July 7, 2017. The data captured on Mondays includes only benign traffic whereas the data captured on other days includes both benign traffic and attacks. The attacks that are implemented contain Brute force FTP, Brute

force SSH, DoS, Heartbleed and Web attack, all of which appear during the week. The creators of this dataset in their evaluation framework Gharib et al., 2016[25] have identified and satisfied eleven criteria that are necessary for making a reliable benchmark dataset.

CICIDS2017 dataset is chosen because, it is a combination of benign background traffic and attacks which simulates the regular traffic in microservices.

4.2.1.3 Monolithic based attacks based on CIC - IDS2017

This thesis focuses only on the attacks which are common and were identified in phase one as vulnerabilities. The scripts are created to examine if the attacks are logged and the administrator is notified. The attacks that were observed are

- **Brute force (SSH & FTP)** - One of the top twenty most common forms of attack to compromise servers. This attack tries to authenticate to remote SSH servers. The goal is simply to gain access to the remote SSH server.
- **Port scan** - This attack is performed by sending probe packets to a network or system and gather intelligence from the responses received. It can give information about the open ports and vulnerable services on the ports.
- **DDOS (slowloris & slowhttptest)** - This attack prevents the normal use of a network/system. It can be done by overloading the network with packets to degrade the performance or targeting specific hosts.
- **Heartbleed** - This is a bug in previous Open-SSL implementations of the Heartbleed protocol discovered in 2014. This bug leaves the memory on a host accessible to anyone who can exploit it over the network.
- **Cross site scripting (XSS)**- This attack is performed by inserting malicious scripts in HTML content of a web page. It can be used to steal cookies from a user for impersonation purposes.

These attacks are the ones which are most common to monolithic architecture. The security concerns of microservices are discussed in previous sections. Another set of tests are also performed to observe if microservice based attacks could be detected and the corresponding results were recorded.

4.2.1.4 Microservice attacks based on MITRE ATT&CK®

This section uses the setup described in 4.2.1 to simulate microservices based attacks with the help of the MITRE ATT&CK framework [43]. The Figure 4.6 provides a summary of the activities in this step. They are described in detail in further sub sections.

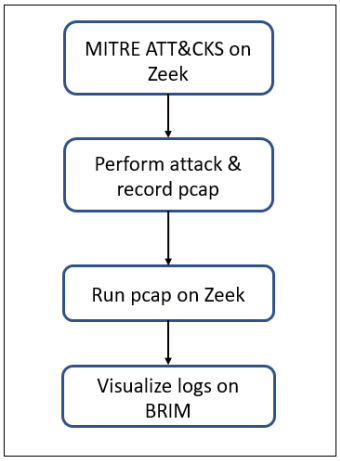

Figure 4.6: Phase 2: Introduction of Zeek - Microservices

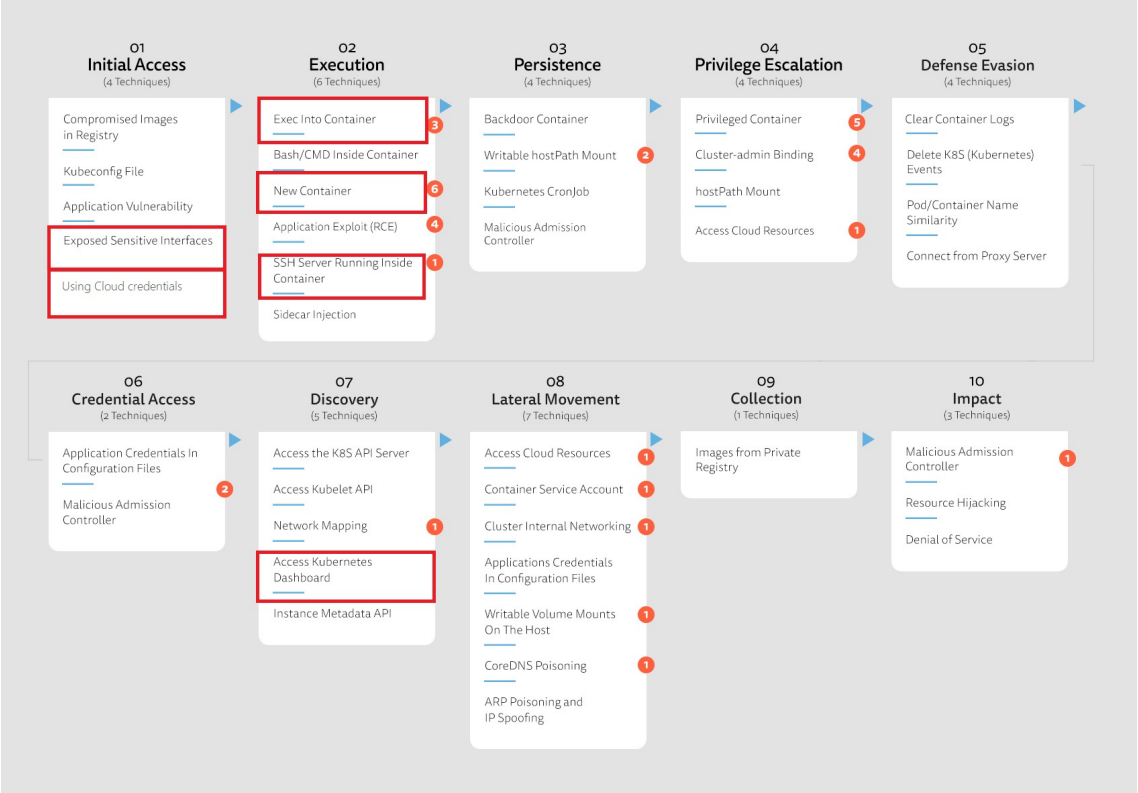

Figure 4.7: Selected attacks from the "MITRE ATT&CK® Matrix for Kubernetes: Tactics & Techniques" [44]

Referring to the Kubernetes attack matrix created by Microsoft[21], few attacks were tested to observe if they remain undetected by Zeek IDS. The attacks that are chosen for the demonstration of this process are highlighted in the Figure 4.7.
These attacks are based on container orchestration security and these parameters are not developed by Zeek. These configurations were left vulnerable to observe if it is detected by the IDS. A further analysis will be provided in the analysis chapter of

the thesis. This thesis focuses on the attacks below to demonstrate authentication, authorization, monitoring and logging based attacks:

- **Using cloud credentials** - An attacker who gains access to the cloud provider account credentials can use it to compromise the Kubernetes cluster.
- **Exposed dashboard** - The dashboard of Kubernetes is a web based interface which is used to manage the cluster resources. If exposed publicly it can allow unauthenticated cluster management. Certain versions deploy the dashboard by default.
- **Exec into containers** - Attackers with permission could run 'kubectl exec' and execute malicious code which could compromise the cluster.
- **New container** - This technique shows the potential of attackers with permission to launch new pods and deployments, replica sets etc., to execute malicious code and compromise the cluster.
- **SSH server running inside container** - This attack is when an SSH server acts as a pathway for an attacker who acquires credentials to the container via other methods. They can gain remote access to the container and execute malicious code or compromise resources.
- **Access Kubernetes dashboard** - Kubernetes, by default does not restrict network traffic between pods. When an attacker gains access to a single pod can further access the Kubernetes dashboard and get access to cluster information.

4.2.2 Procedures applicable to both Monolithic and Microservices based attack scenarios

4.2.2.1 Creating test scenario

The test setup in this thesis is as below:

- A regular functioning microservice environment
- IDS container built using the regular configuration method as described in IDS manual
- Setting up alerts and networks
- IDS configured to listen on the interface and capture logs and alerts when attacks/anomalies are encountered.

4.2.2.2 Simulating attacks

TCP- replay[23] is used to replay the attacks dataset to simulate incoming traffic from containers. The attacks in the dataset come split into five files each a combination of attacks and benign activity.
The files are replayed individually and the traffic is captured again to be fed into Zeek in the case of monolithic attacks. As in the case of microservice based attacks, the network activity is recorded and the attacks are performed. The logs are processed in the same way as monolithic attack logs.

4.2.2.3 Processing packet captures

The packet captures are stored in .pcap format and are run on Zeek command line to process the traffic. Zeek generates logs after processing the pcap files. These logs contain a record of every connection on the network. A Zeek log contains high-level entries which correspond to network activities, for instance, a login to SSH etc. Zeek generates a log which is determined by the protocol type. The most frequently used logs are listed below:

- **notice.log -** When an anomaly is detected by Zeek, a notice is raised in this file.
- **conn.log -** This file contains information relating to all TCP, UDP and ICMP connections.
- **weird.log -** This file contains data related to the packets which are not according to the standard protocols. It also contains packets which may be corrupted or damaged.
- **protocol logs - http.log, ssh.log etc. -** These files contain information regarding the packets which are found in these respective protocols.

4.2.2.4 Visualizing logs using BRIM

Zeek generates a large number of logs to process and is very difficult to find faults/anomalies. So, BRIM [37] is used in this thesis to visualize the Zeek logs and to enable easy filtering of data. BRIM is an open source tool for querying and analyzing Zeek logs.

4.3 Implementation of solution

The results for both Phase One and Phase Two for monolithic and microservice based architecture are explained in detail in the Results Chapter. After thorough analysis of the results for both monolithic and microservice based attacks, one can conclude that Zeek, as a traditional security method functions satisfactorily as intended for monolithic based attacks. However, there is a need to develop a solution for addressing the security concerns of Microservices which remain undetected by Zeek.

Figure 6.1 is used to show a high-level overview of the solution construction and the proposed solution is explained in detail in Chapter 5.

5
Results

This chapter summarizes the results of the data collection and analysis of the data. As mentioned earlier that the project is divided into two phases, correspondingly the results section is also better explained in two sections.

5.1 Phase One Results

5.1.1 Visualization of protocol data post baseline creation

The datasets collected were from a total of nine pods in Kubernetes. The traffic between these microservices was recorded using ksniff for five hours each. For better illustration, the data that was collected is represented as pie charts from Figures 5.1 to 5.9.

The pie charts shown below visualize the major protocols which are part of the network traffic. The main observations are that a major part of the traffic consists of TCP, TLS, HTTP and SSH traffic. The other protocols like NTP, ICMP, DHCP, ARP are not considered in this thesis as they form only a small portion of the network traffic. The combined network protocols is explained in the next section for cumulative network protocols.

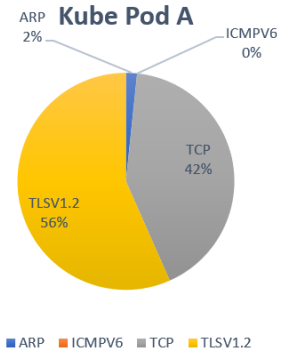

Figure 5.1: Network Protocol Analysis - Kubernetes Pod A

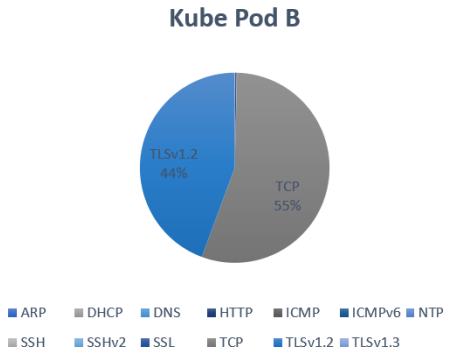

Figure 5.2: Network Protocol Analysis - Kubernetes Pod B

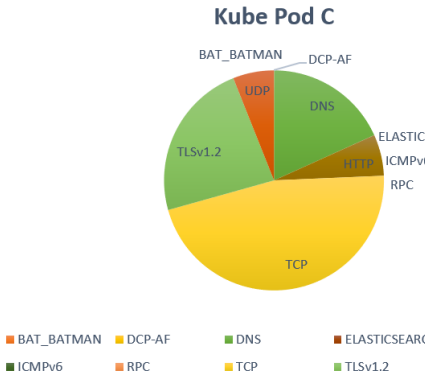

Figure 5.3: Network Protocol Analysis - Kubernetes Pod C

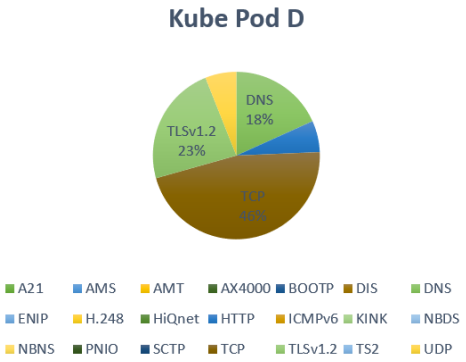

Figure 5.4: Network Protocol Analysis - Kubernetes Pod D

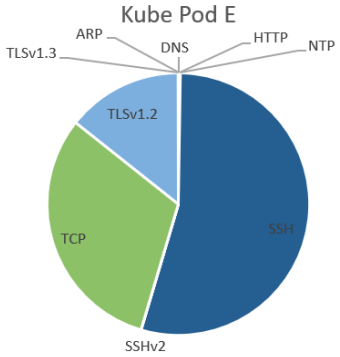

Figure 5.5: Network Protocol Analysis - Kubernetes Pod E

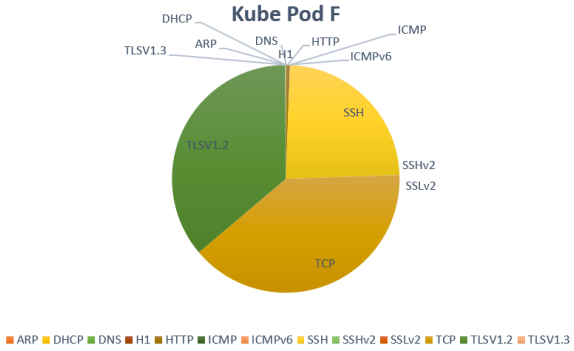

Figure 5.6: Network Protocol Analysis - Kubernetes Pod F

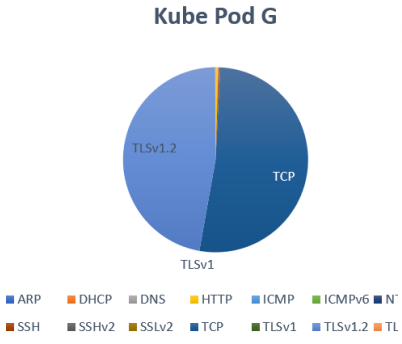

Figure 5.7: Network Protocol Analysis - Kubernetes Pod G

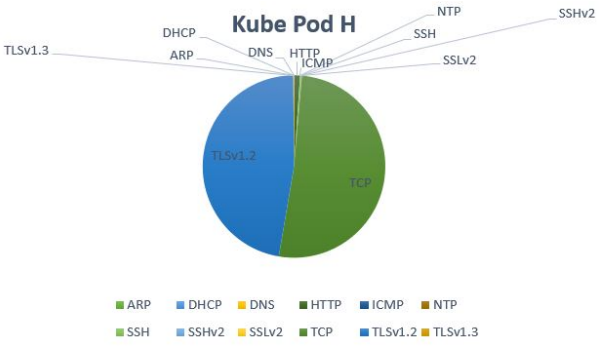

Figure 5.8: Network Protocol Analysis - Kubernetes Pod H

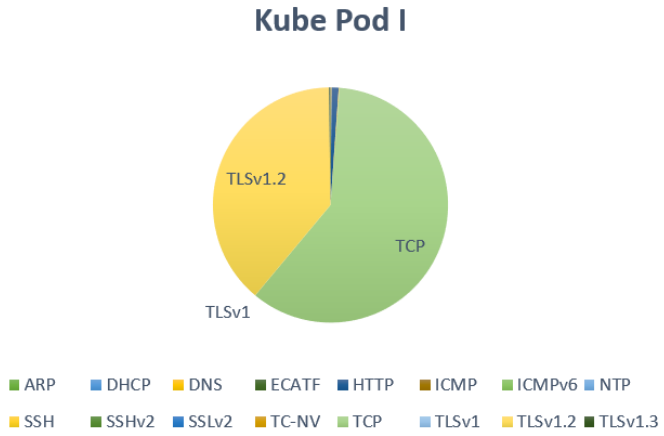

Figure 5.9: Network Protocol Analysis - Kubernetes Pod I

5.1.2 Visualization of Cumulative Network Protocols post baseline creation

The graph in Figure 5.10 (shown in logarithmic scale of base 10), indicates the aggregated network protocols which are used by the pods. This points out that the protocols that make up the major part of the network traffic stream are mainly HTTP, TCP and TLSv1.2. The table 5.1 gives the number of packets per network protocol which are selected. Considering the most frequently occurring protocols after baseline creation, the attacks for the next phase are chosen appropriately.

Several other protocols are also part of the stream, but are omitted from the graph owing to confidentiality and their insignificance.

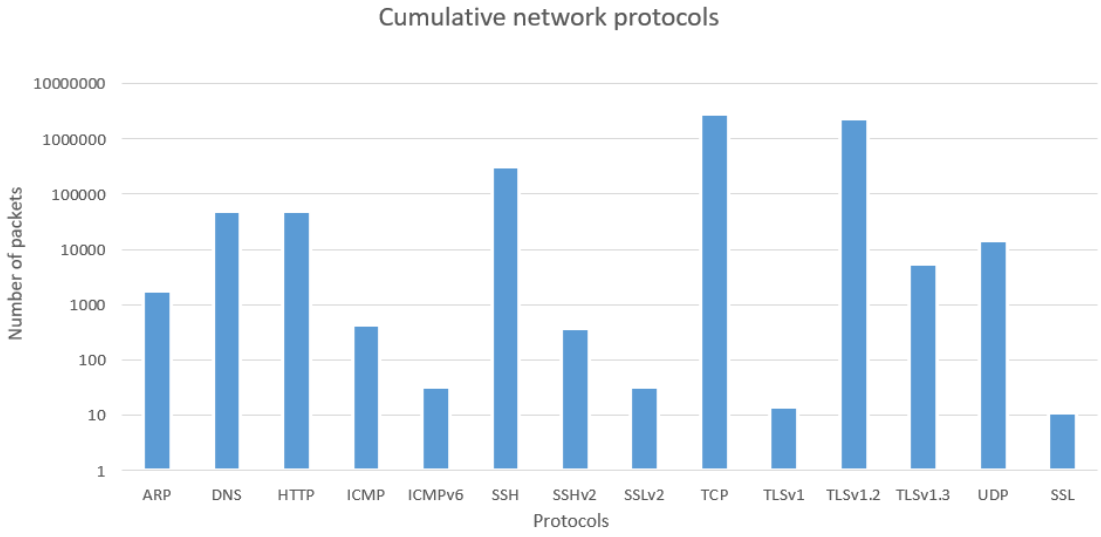

Figure 5.10: Cumulative network protocols (Graph shown in logarithmic scale of base 10)

Network Protocols	Number of packets
ARP	1,803
DNS	48,318
HTTP	48,178
ICMP	426
ICMPv6	32
SSH	310,483
SSHv2	365
SSLv2	33
TCP	2,861,079
TLSv1	14
TLSv1.2	2,330,671
TLSv1.3	5,526
UDP	14,799
SSL	11

Table 5.1: Number of packets per protocol in Network protocol analysis as conducted in Subsection5.1.1

5.2 Phase Two Results

5.2.1 Monolithic architecture based attacks

After the introduction of Zeek into the environment, the attacks in the Intrusion Detection Evaluation Dataset-CIC-IDS2017 were replayed and the traffic is recorded. To test the effectiveness of Zeek in logging and alerting for monolithic based attacks, the tests have been performed with the IDS evaluation dataset by replaying the traffic five times at different traffic rates from 50 to 2,000 packets per second(pps) and observing the logs. There was no packet loss at 50 pps and all the attacks were logged. As the pps increased, there was some packet loss and few attacks were not logged.

The graph in Figure 5.11 shows the number of times a particular attack was not logged (in red) and the number of times it was logged (in green), out of the total 5 attempts to detect the attacks. SQL injection and Port scanning attacks were detected in all five of the attempts and also alerts were received and logged.

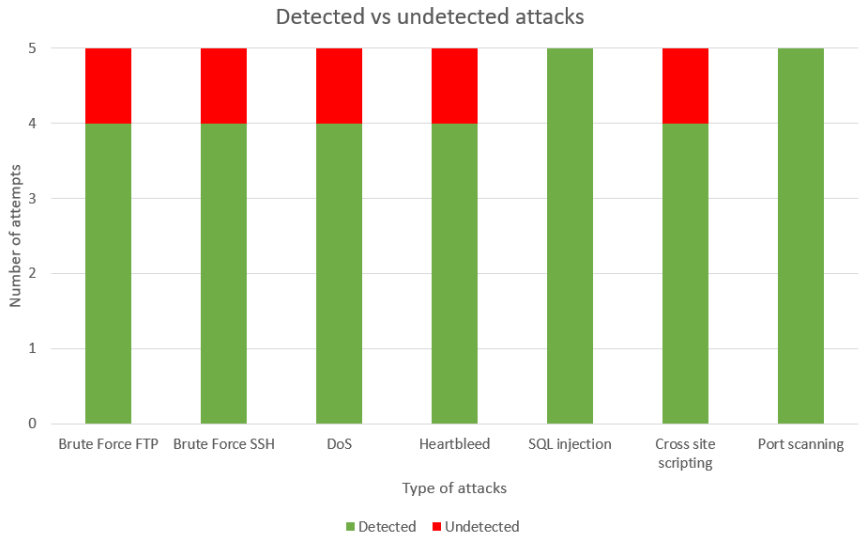

Figure 5.11: Graph representing attempts and detection of attacks

The logs received from Zeek are uploaded to BRIM for visualization. The attacks generate notice logs and alerts. The packets displaying the attacks are in frame for easier viewing.

The collection of results as shown in Figures 5.12 to 5.16, display the attacks as visualized in BRIM. The parameters shown in BRIM are Timestamp, Origin IP address, Destination IP address, port information, path, protocol, message etc., The results can be filtered to display only the parameters that need to be displayed.

Figure 5.12 shows the logs of a SQL injection attack, where the Zeek script was triggered when the URI in the attack matches to the regular expression in the script. The log displays the attackers IP address (id.orig_h) and the victim IP address (id.resp_h). It also shows the URI in the attack.

Figure 5.13 displays timestamp of the attack and logged time. The attacker and victim's IP addresses are recorded along with the note which displays "FTP:Bruteforce" and the message parameter shows the number of failed login attempts by the attacker.

Figure 5.14 displays the timestamp, attacker's and victim's IP addresses, the number of attempts at authorization(auth_attempts) and if the authorization was a success.

Figure 5.15 shows the attacker's and victim's IP addresses, the protocol, type of attack with a note and a message showing if an attacker or victim was discovered.

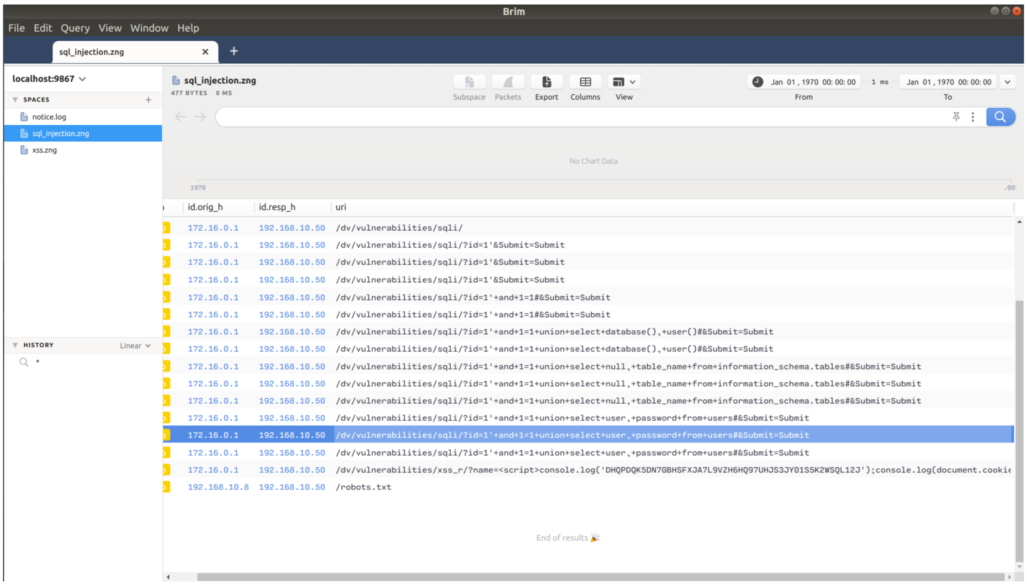

Figure 5.12: SQL Injection

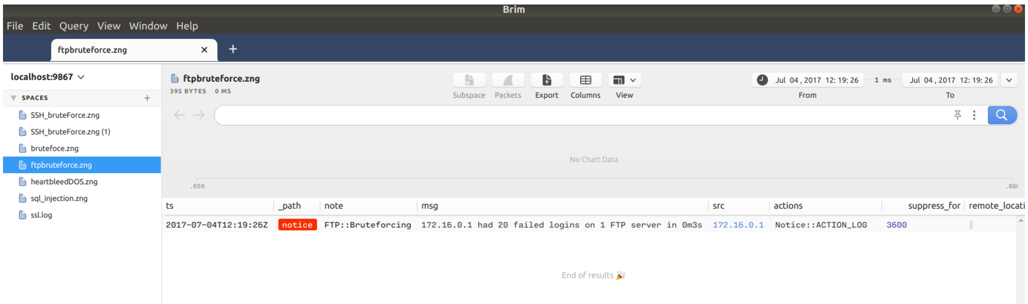

Figure 5.13: FTP brute force

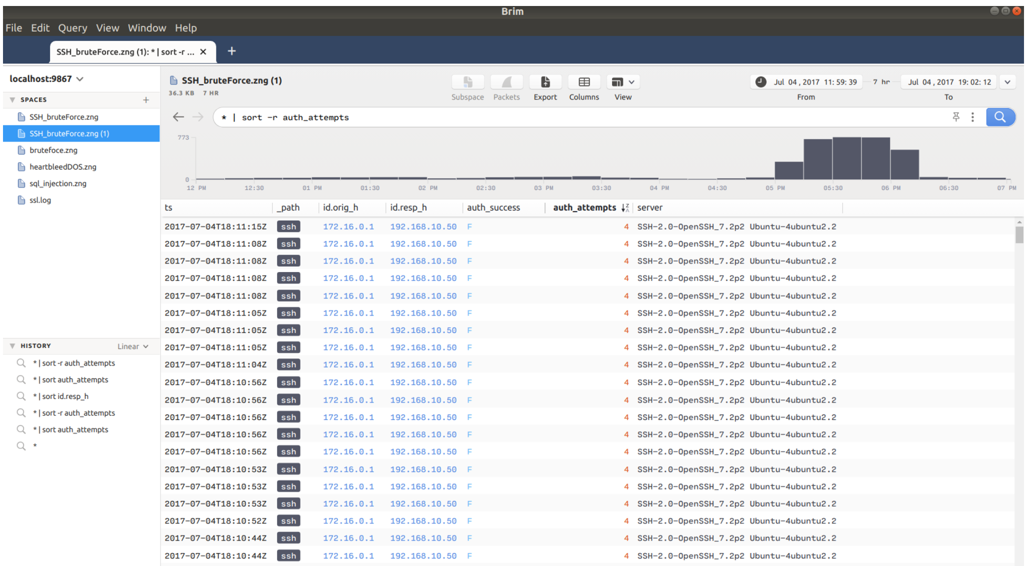

Figure 5.14: SSH Brute Force

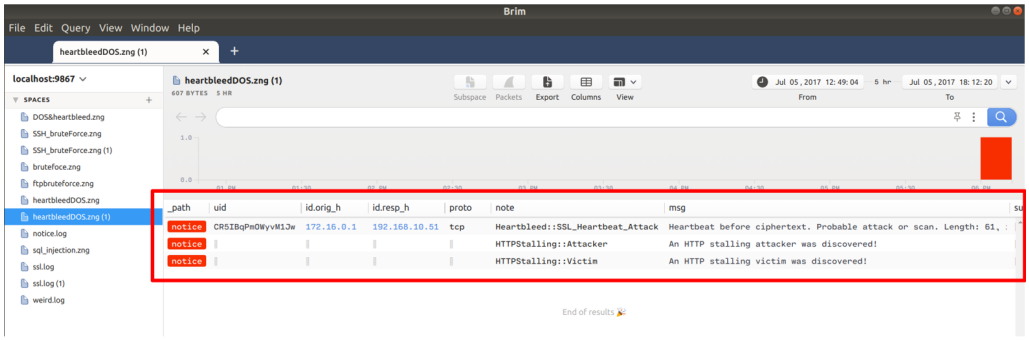

Figure 5.15: Heartbleed and HTTP Stalling

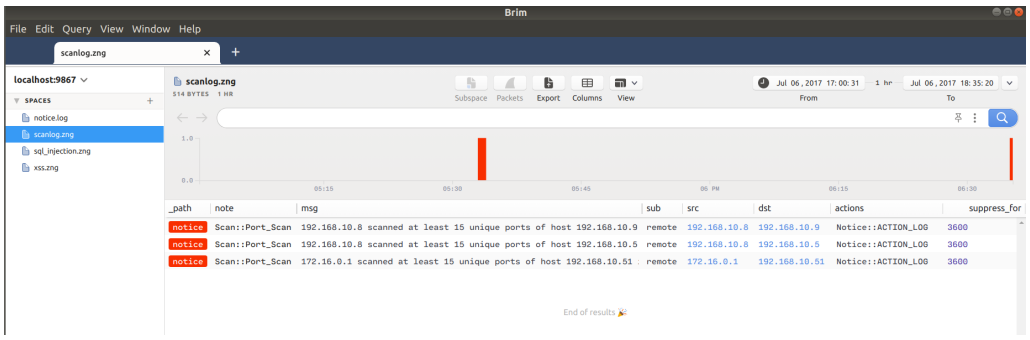

Figure 5.16: Port scan

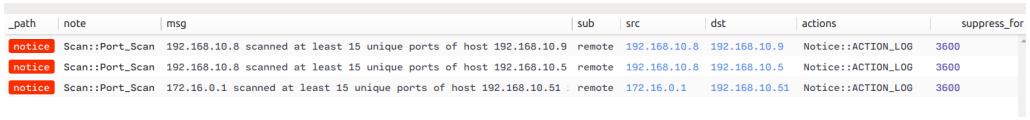

Figure 5.17: Port scan- Detailed view

Figure 5.17 shows a focused view of the logs for the TCP port scanning attack. The parameters are explained below -

- **note** - "Scan::Port_Scan": Port scans detect that an attacking host appears to be scanning a single victim on several ports
- **msg** - This displays the custom message set by the script in Zeek.
- **src** - Displays the source IP address (attacker's IP address)
- **dst** -Displays the destination IP address (victim's IP address)
- **actions** - Shows if an alert was issued or if it was logged.

The above results captures the monolithic based attacks on IDS Zeek. Monolithic based attacks are successfully detected and logged by Zeek as shown by the logs above.

5.2.2 Microservice based attacks

This section demonstrates how a traditional IDS like Zeek works in a microservice based environment to identify attacks which are more specific to microservices. The

attacks have been chosen from Figure 4.7 which is the MITRE ATT&CK framework. The images of the attacks are omitted for confidentiality reasons, however the steps in implementation of the attacks are described below:

- **Using cloud credentials** - This discusses the first threat vector-*Initial access* from the MITRE ATT&CK framework. The initial step is to gain access to the account of the cloud provider many of which are shared credentials(common password), and these details are used to manipulate the cluster.
- **Exposed dashboard** - This discusses the first threat vector-*Initial access* from the MITRE ATT&CK framework. The Kubernetes dashboard is a web-based user interface which is used for managing the cluster. This UI might expose an internal endpoint publicly, which allows for unauthenticated management of the cluster.
- **Exec into containers** - This attack is part of the second tactic in the attack matrix- *Execution*, which involves the attacker running code from within the Kubernetes cluster. Kubectl is a command line tool for managing the Kubernetes cluster. The command *'kubectl exec'* is used by a user to execute a command in a container. This command can be used to execute malicious code and compromise resources in the cluster.
- **New container** - This attack is performed by attackers with permission to launch new pods to execute the malicious code and manipulate clusters.
- **SSH server running inside container** - This attack is executed using an SSH server which is running inside a container, which allows to obtain credentials to that container and access it remotely to run malicious code.
- **Access Kubernetes dashboard** - The initial step is to gain access to a single pod which further provides access to the dashboard, retrieving information regarding the cluster.

These attacks do not generate any alerts or logs in Zeek. These attacks cause a lot of impact to the applications but are not detected by the IDS. The proposed solution is used to address these security requirements of microservices.
The mitigation of these microservice based attacks using the proposed solution is explained in section 6.1.6.

6
Proposed Solution

6.1 Istio service mesh to secure Microservice communication

As discussed in previous chapters, conventional network security does not address security issues in distributed applications which are deployed in fast growing organizations. Service mesh and Istio are explained in detail in section 2.16. This solution describes how service meshes can be used to monitor and secure communications in cloud environments. A local microservice application was built as an example to demonstrate the security features of service mesh Istio.
The Figure 6.1 is used to show a high-level overview of the solution construction. The further sections explain them in detail.

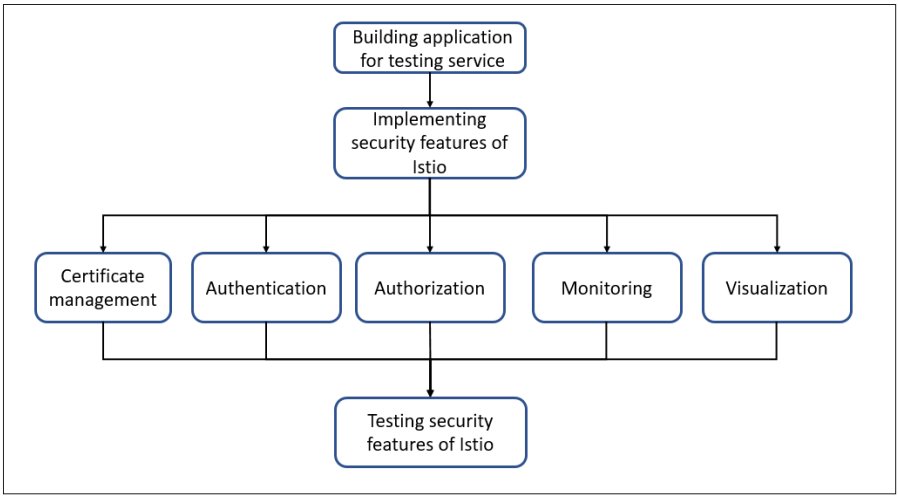

Figure 6.1: Solution: Service Mesh

6.1.1 Architecture of the application

The application is called BookInfo shown in Figure 6.2, which is broken into four separate microservices:

- **Productpage** - The productpage microservice calls the details and reviews microservices to populate the page.
- **Details** - The details microservice contains book information.

- **Reviews** - The reviews microservice contains book reviews. It also calls the ratings microservice.
- **Ratings** - The ratings microservice contains book ranking information that accompanies a book review.

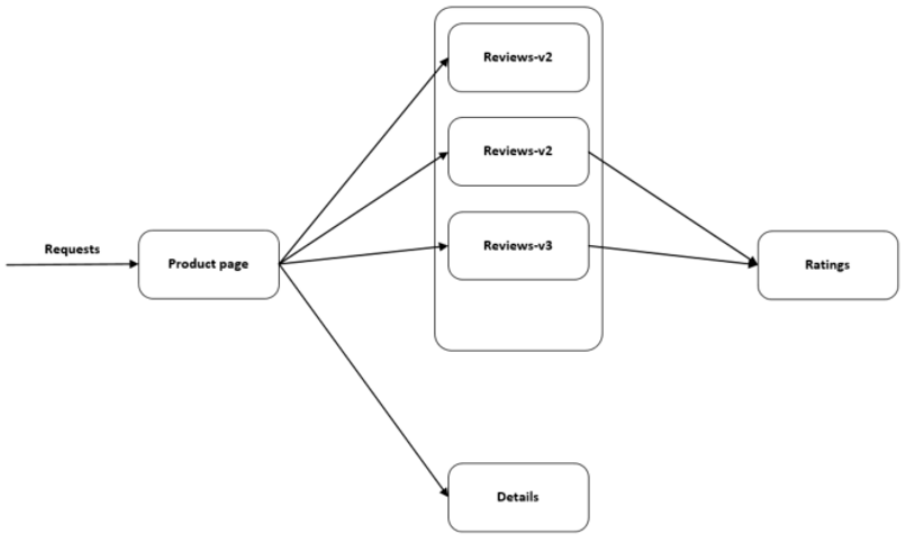

Figure 6.2: Architecture of the microservice application

6.1.2 Overview of Steps performed

- Used *kind* (tool for running Kubernetes local clusters and docker container nodes) to create a local Kubernetes cluster with pods(applications)
- Running a microservice locally
- Running the book-info application in Kubernetes
- Configuring Istio on all the microservices
- Testing security features
- Monitoring
- Setting up dashboards for visualization

6.1.2.1 Create local Kubernetes cluster

Kind is the tool which is used to run a local Kubernetes clusters using Docker. The Figure 6.3 displays the information of the cluster and the IP addresses that Kubernetes master and KubeDNS is running on. Figure 6.4 displays the pods (applications) which are deployed in the cluster. For instance, the pods "grafana", "prometheus" are used for visualization and collecting metrics respectively. The "productpage", "reviews-v1, v2, v3" are the application pods for the book-info pages. The webpage of book info application can be accessed after the cluster is created and the application is enabled for external access, which is shown in the Figure 6.5.

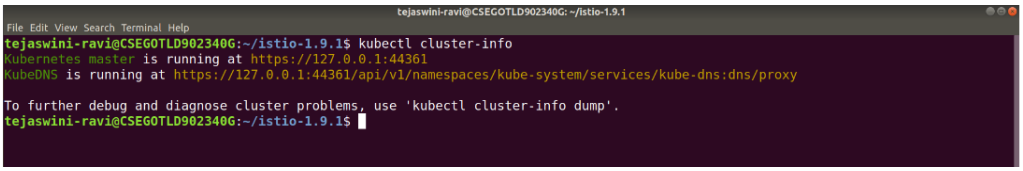

Figure 6.3: Kubernetes cluster

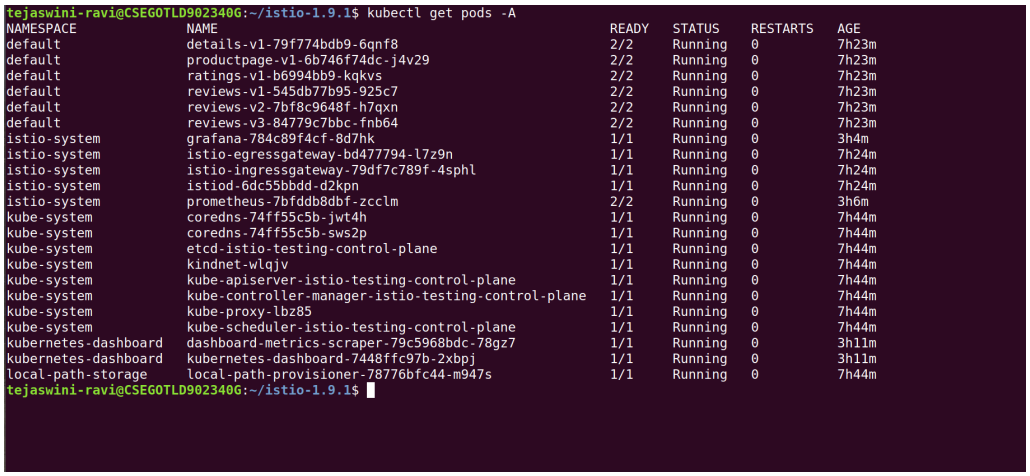

Figure 6.4: Pods in the cluster

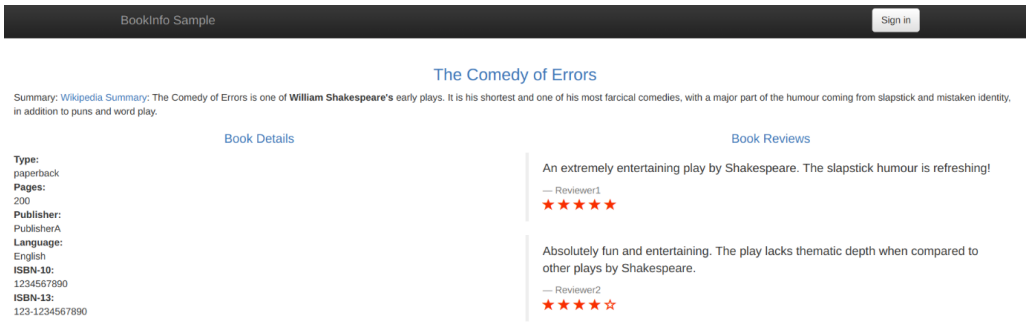

Figure 6.5: Book-info web application

6.1.3 Testing Security features of Istio

These security features of Istio were tested and verified in this thesis and are functional.

- **Certificate management**
  - Configured the Istio Certificate Authority with a root certificate, signing certificate and key.

  - Verified the certificate chain from root to workload certificate
- **Authentication**
  - Enabled, configured Istio authentication policies
- **Authorization**
  - Setup authorization for HTTP traffic
  - Setup authorization for TCP traffic
  - Enforcing IP based access control on Ingress gateway(controls and monitors the incoming traffic)

6.1.4 Monitoring

Monitoring is a very important support functionality which is offered by service mesh Istio. The Istio dashboard can be used to monitor the microservices in real time.

- Metrics - monitors latency, traffic, errors, and saturation these metrics are generated for all service traffic incoming and outgoing.
- Access logs - Record of each request including source and destination metadata. Assists in auditing service behaviour.

6.1.5 Visualization

Istio offers a pre-configured range of dashboards to choose from.

- **Kubernetes dashboard** - Created using kind (tool for running local Kubernetes cluster using docker) shows deployments, pods. This is shown in Figure 6.6 The Kubernetes dashboard shows the Deployments, Pods and the Replica Sets The Deployments pane provides information about the namespace, created time and the name of the docker image. The pane on the left side of the image provides more information regarding Workloads, Services, Configuration, Storage and Cluster information.
- **Istio dashboard** - Figure 6.7 gives an overview of all services in the mesh, this dashboard shows the number of virtual services, peer authentication policies, authorization policies and the latency rates in each service
- **Service dashboard** - Figure 6.8 gives a detailed breakdown of the metrics for a service. It displays metrics about requests and responses for every service in the mesh, HTTP and TCP. The figure shows client requests and server requests volumes, the duration taken for request to be serviced. The dashboard also gives information about the client workloads.
- **Workload dashboard** - Figure 6.9 shows metrics for each workload, inbound workloads and outbound workloads. This section provides metrics about requests and responses for each workload, also inbound and outbound workloads. The figure shows the incoming volume of requests(operations per second), the duration which the request takes to be serviced, inbound workloads shows the incoming requests.
- **Performance dashboard** - Gives the resource usage information of the mesh, vCPU usage, Memory and Data rates, Disk are few of the parameters which are displayed in this dashboard.

- **Control Plane dashboard** - Monitors performance and health of the control plane. Istio's control plane supplies a collection of self-monitoring metrics which allow for the monitoring of Istio's behavior itself.
- **Kiali dashboard**[40] - This is an add-on to Istio. It is a Web based GUI to view service graphs of the mesh etc., as shown in Figure 6.10. This enables tracking of a request in the mesh, which will in turn provides deeper understanding. The figure shows a Versioned App graph representing the traffic flowing through the service mesh refreshing every fifteen seconds. There are also several types of graphs to choose from: App, Workload, Service, Versioned App.

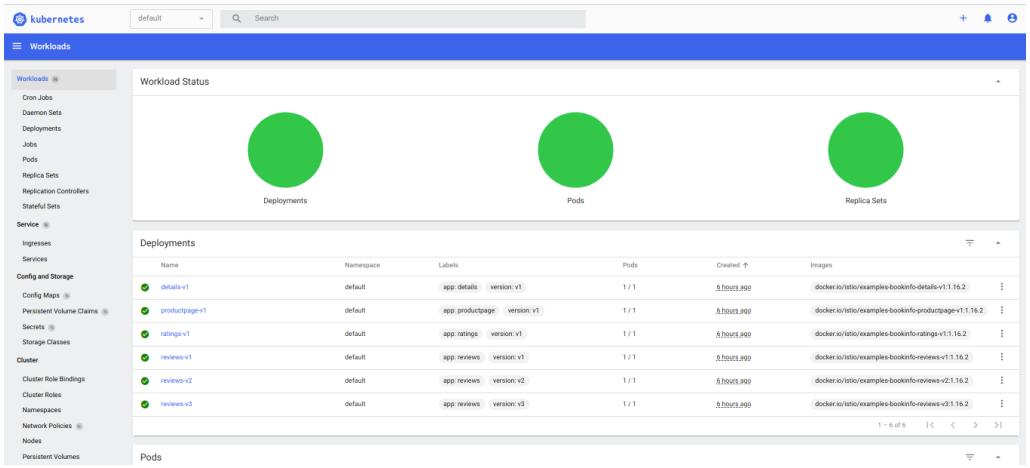

Figure 6.6: Kubernetes dashboard

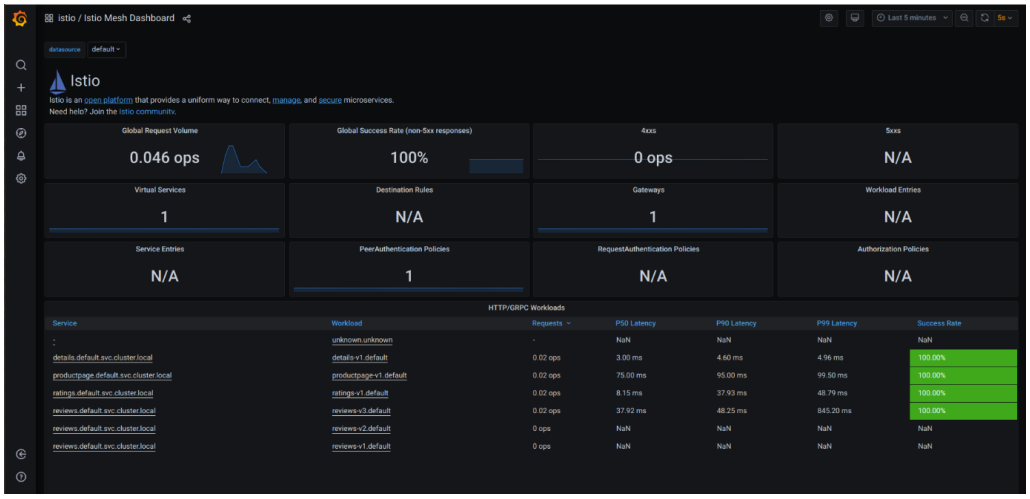

Figure 6.7: Istio dashboard

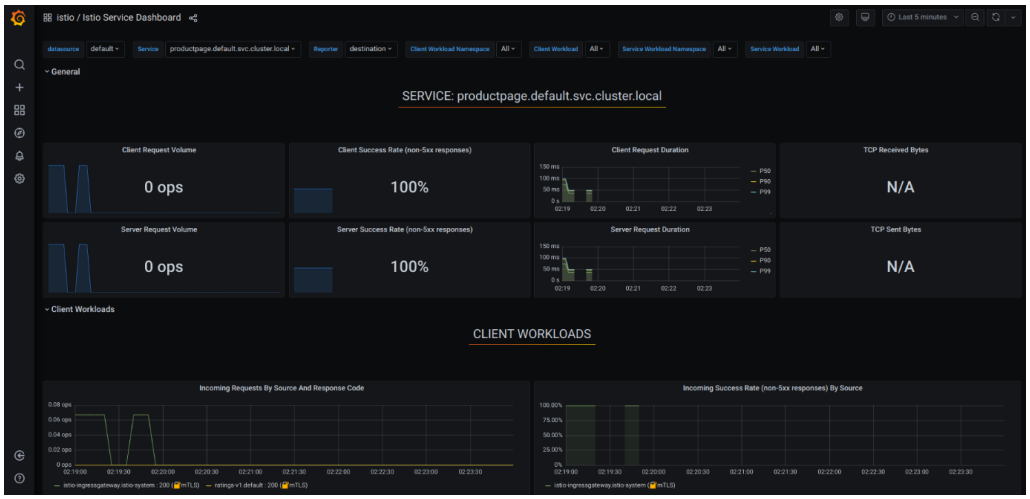

Figure 6.8: Service dashboard

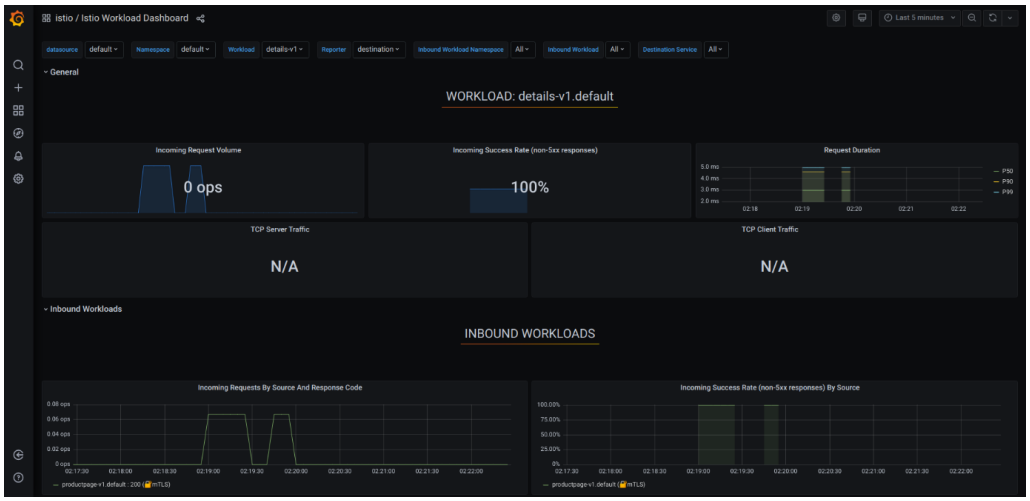

Figure 6.9: Workload dashboard

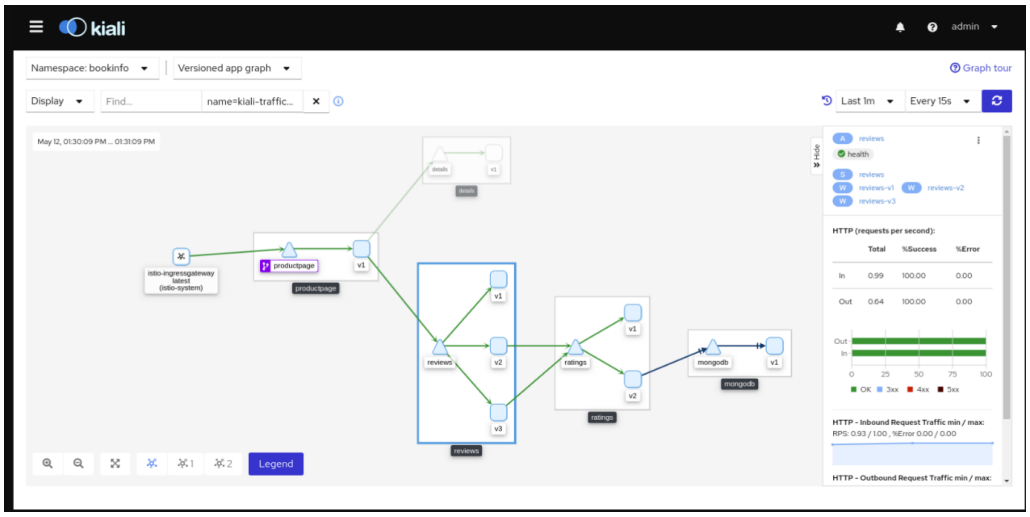

Figure 6.10: Kiali dashboard

6.1.6 Mitigation of security issues by service mesh

Traditional security approaches are ineffective in microservice environments, it is critical that a solution which offers deep monitoring and which has advanced capabilities is used. Istio is a service mesh (network of microservices) which helps to reduce the complexity of Microservices. There are several examples of service meshes like Linkerd, HashiCorp Consul, Aspen mesh and Istio. Istio was chosen because it is open source developed by Google, Lyft and IBM. Service meshes help to improve security in the following ways:

- Security by default - No changes are made to application
- Defense in depth - Provides multiple layers of defense
- Zero - trust network - Build security solutions on zero trust networks.
- Traffic observability - Contains many add-ons for investigation of network behaviour anomalies.
- Service meshes can be automatically encrypted with mTLS
- Role based access control
- Authentication of workload, manages certificates and authentication policies
- Monitors and control of ingress and egress traffic

Service meshes like Istio offer encryption, authentication, monitoring, logging and access control to protect from compromise and exfiltration. It monitors and controls the incoming and outgoing traffic using ingress and egress gateways. Secures communication between microservices.Performs authentication using certificates. Authorization policies for HTTP and TCP traffic. It also provides overall monitoring and logging at various levels and there is room for customization. The dashboards are built using Grafana for visualization and the metrics are scraped using Prometheus. The logs and dashboards are queryable. This thesis focuses only on the security features of Service mesh Istio. These factors are addressed by service mesh to secure communication between microservices. The service mesh methodology should be adopted over traditional IDS or firewalls which don't offer defense in layers and the advantages as mentioned above.

6.1.6.1 Mitigation of MITRE ATT&CK using Service mesh

The MITRE ATT&CK framework for Kubernetes and the attacks which are chosen are explained in detail in section 4.2.1.4. In this section it can be observed how these attacks are mitigated/avoided by service meshes.

- **Using cloud credentials** - This issue is mitigated with RBAC and authorization of user credentials.
- **Exposed dashboard** - This issue is mitigated using account privileges in RBAC and blocking ingress traffic from certain entities.
- **Exec into containers** - This issue is mitigated by restricting RBAC access to the pods
- **New container** - Mitigated by restricting RBAC permissions to create pods or replica sets etc.
- **SSH server running inside container** - This is mitigated with ingress traffic control and restricting network traffic to prevent unwanted access to the pod.
- **Access Kubernetes dashboard** - This is mitigated by restricting account privileges and blocking ingress traffic from certain entities.

6.2 Comparison of Service mesh vs Traditional Security

The objective of this thesis work is mainly to understand the security requirements of Microservice architecture based application, and how it differs from the security requirements of a Monolithic architecture based application. Also, to understand why traditional security is ineffective in a microservices environment. Although service meshes have many other features than just security, for the purpose of this thesis work, a comparison between Service Mesh Istio and IDS Zeek is made and summarized in table 6.1[38].
Summarizing the table provides us with an idea of how a traditional security system like an IDS is lacking while protecting a Microservice infrastructure. Microservices require an active monitoring system which can isolate compromised areas in the infrastructure. While an IDS like Zeek will assist in capturing network traffic and visualize the data, it cannot process the workloads and volume of traffic in a Microservices based environment. Service meshes work better in securing the environment since they solve issues of monitoring, logging, isolation of compromised pods, RBAC and controlling incoming and outgoing traffic.

	IDS & NSM- Zeek	Service mesh -Istio
Definition	Comprehensive network logging tool and Signature based IDS	Dedicated infrastructure layer with set of infrastructure functions helping with service discovery, traffic management, configuration, authentication, authorization and metrics and monitoring
Features provided	• Extensive logging • Identifying popular attacks • Supports custom scripting • High-performance and load balancing • Alerting • Open source • Integration with ELK	• Authentication, authorization - RBAC • Secure communication-mTLS, encryption • Traffic management - regulating traffic • Observability/monitoring - Logging, metrics- Prometheus, Grafana
Active/Passive?	• Plays a passive part in logging and alerting. • The attack cannot be prevented	• Helps in securing the infrastructure to prevent attacks. • On compromise of the pod, the pod is isolated using ingress and egress traffic.
Interaction	Command line tools like zeekctl	Dashboards and GUI for easy access
Primary feature	Primary feature is to analyze and segregate logs and send alerts when any event is triggered	• Intercepts and routes cluster traffic and gathering health metrics. • Service mesh can be leveraged to monitor and secure communication.
Visualization	Logs need 3rd party tools to visualize	Inbuilt monitoring features using Prometheus and Grafana provides with customizable dashboards
Compatibility with microservices	• Capable in identifying attacks CIC-IDS2017 which are pre-recorded and patterns are available. • Not capable of preventing microservice based attacks	• Prevents the malicious attacks from entering the infrastructure. • Prevents misconfiguration, microservice based attacks from entering environment. • Access control policies can be configured in layer 4 and 7

Table 6.1: Comparison - IDS & NSM Zeek vs Service mesh Istio

7
Conclusions

As said by Jim Manico (OWASP-Stammtisch München, 9th Nov 2018) – "API security is at least a couple of years behind other types of web security"[39]. Considering the rapid growth in microservices, there needs to be proportional growth in technologies to secure microservices. The chapter is divided into subsections for better understanding. The knowledge acquired by conducting this thesis is explained in Section 7.1, Section 7.2 summarizes the main ideas discussed in the thesis and Section 7.3 sheds some light on the potential directions for the work in the future.

7.1 Lessons learnt

This thesis explores traditional security methods using an IDS like Zeek and also modern security measures using a Service Mesh to secure the microservice communication. Certain limitations were set to limit the scope of the thesis. Based on the results obtained, the following conclusions can be drawn:

- Monolithic systems are relatively easier to protect by managing firewall and IDS rules. But this changes drastically in the case of microservices when there are thousands of services to protect.
- Monolithic based attacks were detected by traditional security methods like Zeek successfully and effectively. Traditional security lacks the visibility needed to secure containers.
- Microservice based attacks were not detected by traditional security methods and this creates a need for development of better suited security models for microservice architecture.
- To mitigate the security concerns related to microservice based attacks, this thesis investigates the feasibility of a service mesh to mitigate security issues of microservice based attacks. Although service meshes are not purely made for the purpose of security, it covers a major area of security requirements in microservices.

7.2 Discussions

Microservice architecture is emerging as a big trend in the software industry. Fast moving organizations choose flexible and quickly scalable systems paying with security as the price. Security is a very important aspect of building an infrastructure,

but there is only limited insight into the security concerns of microservices. When broken down from a network infrastructure level, microservices form the basic and most complex entities of the connected cars. Even the smallest vulnerabilities if found by wrong hands might be disastrous causing accidents or other serious incidents. Hence more research needs to be done in this field since organization's involving mission critical applications are adopting the microservice architecture. Security in microservices also needs equal attention and security needs to be the top priority for protecting the organization's infrastructure. As demonstrated by this thesis, traditional security methods like firewalls and IDS's lack providing the additional security features which are required by microservices to have a secure infrastructure. The security features provided by service meshes like Istio can be summarized as encryption, authentication, management of incoming and outgoing traffic, central monitoring and visualization. These features meet the demands of the microservices and help in securing communication in microservices.

While active protection of the infrastructure is of primary importance, this thesis shows the significance of encryption, logging, monitoring and alerting for identifying flaws in configuration and detecting malicious activity.

7.3 Future Directions

With the arrival of 5G and IoT, the near future foresees addition of many more devices to the network. This also includes connected cars in the form of Vehicle-to-Vehicle(V2V) communication, Vehicle-to-Everything(V2X). ECU tuning, infotainment hacking, data theft, ransomware attacks are just the few instances of the attacks. Attackers are able to find vulnerabilities even under the tightest security. This demands multi-layered defenses, internal IT networks using cloud security to foil cyber-attacks. Consumers demand vehicles with with the latest technology, connectivity and amenities. Seldom aware of how easily the cars can be hacked. This could potentially cost automobile manufacturers billions in sales and loss of confidence within consumer circles.

The "UNECE World Forum for Harmonization of Vehicle Regulations (UNECE WP29)" [42] during the summer of 2020 saw the adoption of two regulations UN R155 and UN R156, this will set the structure for vehicle cybersecurity for many parts of the world. The UN R155 regulation needs operation of a certified "Cybersecurity Management System (CSMS)", UN R156 requires a "Software Update Management System (SUMS)" for approval of new vehicles.
There are many reactions from Governments worldwide concerning the risks involved with connected automobiles. The EU plans to make the UN R155 and UN R156 regulations mandatory for the approval of new vehicles by July 2022 and to push these regulations to existing vehicles by July 2024.
These policies and regulations from governments combined with vehicle manufacturer's efforts, making stringent guidelines for designing, implementing security features and also securing their vehicle's network infrastructure will contribute towards making connected cars much safer against internal and external threats and attacks.

Bibliography

[1] Frisk, D., A Chalmers University of Technology Master's thesis template for LaTeX, Unpublished,2016

[2] Zeek, The Zeek project, © 2020, https://www.zeek.org

[3] TCPdump & libpcap, The Tcpdump Group, https://www.tcpdump.org/, Accessed on 04-08-2020

[4] Wireshark, https://www.wireshark.org/, Accessed on 06-08-2020

[5] Remote packet capture tool https://github.com/eldadru/ksniff, Accessed on 18-08-2020

[6] Definition of NTA, https://awakesecurity.com/glossary/network-traffic-analysis/, Accessed on 01-06-2020

[7] C. Fetzer, Building critical applications using microservices, IEEE Security Privacy, Volume 14, no. 6, pp. 86–89, November 2016.

[8] Y. Sun, S. Nanda, and T. Jaeger, Security-as-a-service for microservices-based cloud applications, in Cloud Computing Technology and Science (CloudCom 2015), IEEE, 2015, pp. 50–57.

[9] Yarygina, Tetiana, and Anya Helene Bagge, Overcoming security challenges in microservice architectures, 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), IEEE, 2018.

[10] Nicola Dragoni, Saverio Giallorenzo, Alberto Lafuente, Manuel Mazzara, Fabrizio Montesi, et al.. Microservices: yesterday, today, and tomorrow. Manuel Mazzara; Bertrand Meyer. Present and Ulterior Software Engineering, Springer, 2017, 978-3-319-67425-4. hal-01631455.

[11] Xiong, G., Tong, J., Xu, Y., Yu, H., Zhao, Y. (2014, September). A survey of network attacks based on protocol vulnerabilities. In Asia-Pacific Web Conference (pp. 246-257). Springer, Cham.

[12] Zhang, N., Li, H., Hu, H., Park, Y. (2017, March). Towards Effective Virtualization of Intrusion Detection Systems, In Proceedings of the ACM International Workshop on Security in Software Defined Networks Network Function Virtualization (pp. 47-50), ACM.

[13] OWASP Top Ten, https://owasp.org/www-project-top-ten/, Accessed on 01-09-2020

[14] Zero Trust Security, https://www.cloudflare.com/en-gb/learning/, Accessed on 01-09-2020

[15] Ramaswamy Chandramouli, Zack Butcher, Building Secure Microservices-based Applications Using Service-Mesh Architecture, May 2020,

[16] Istio, https://istio.io/latest/docs/concepts/what-is-istio/, Accessed on 19-08-2020

[17] Stallings, W. Brown, L. Bauer, M. D., Howard, M. (2012). Computer security: principles and practice, second edition. Prentice Hall.

[18] Nikravesh, A., Guo, Y., Qian, F., Mao, Z. M., amp; Sen, S. (2016). An in-depth understanding of multipath TCP on mobile devices, Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking. https://doi.org/10.1145/2973750.2973769

[19] Mateus-Coelho, N., Cruz-Cunha, M., amp; Ferreira, L. G. (2021), Security in Microservices Architectures, Procedia Computer Science, 181, 1225–1236. https://doi.org/10.1016/j.procs.2021.01.320

[20] Docker, https://www.docker.com/resources/what-container, Accessed on 11-07-2020

[21] Threat matrix for Kubernetes, https://www.microsoft.com/security/blog/2020/ //04/02/ attack-matrix-kubernetes/, Accessed on 02-08-2020

[22] Intrusion Detection Evaluation Dataset (CIC-IDS2017), Canadian Institute for Cybersecurity, https://www.unb.ca/cic/datasets/ids-2017.html , Accessed on 18-06-2020

[23] Tcpreplay-Pcap editing and replaying utilities, https://tcpreplay.appneta.com/ ,Accessed on 19-05-2020

[24] Istio, https://istio.io/latest/docs/ops/deployment/architecture/ , Accessed on 18-09-2020

[25] A. Gharib, I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, "An Evaluation Framework for Intrusion Detection Dataset," 2016 International Conference on Information Science and Security (ICISS), pp. 1-6, doi: 10.1109/ICISSEC.2016.7885840, 2016.

[26] Service Mesh Architecture- Definition, https://buoyant.io/2020/10/12/what-is-a-service-mesh/, Accessed on 18-09-2020

[27] Singh, S., amp; Banerjee, S. (2020), Machine Learning Mechanisms for Network Anomaly Detection System: A Review, 2020 International Conference on Communication and Signal Processing (ICCSP), https://doi.org/10.1109/iccsp48568.2020.9182197

[28] Automatic Anomaly Detection and Root Cause Analysis for Microservice Clusters, Viktor Forsberg, Umeå University, Master's thesis 2019.

[29] Nicola Dragoni, Saverio Giallorenzo, Alberto Lafuente, Manuel Mazzara, Fabrizio Montesi, et al.. Microservices: yesterday, today, and tomorrow. Manuel Mazzara; Bertrand Meyer. Present and Ulterior Software Engineering, Springer, 2017, 978-3-319-67425-4. hal-01631455

[30] A. Nehme, V. Jesus, K. Mahbub and A. Abdallah, "Securing Microservices," in IT Professional, vol. 21, no. 1, pp. 42-49, Jan.-Feb. 2019, doi: 10.1109/MITP.2018.2876987.

[31] VILHELM GUSTAVSSON , "Machine Learning for a Network- based Intrusion Detection System", KTH SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP

[32] Perimeter Security ,https://cloud.google.com/security/beyondprodmotivation, Accessed on 18-09-2020

[33] Hannousse, Abdelhakim Salima, Yahiouche, Securing Microservices and Microservice Architectures: A Systematic Mapping Study, 2020

[34] Docker, https://docs.docker.com/get-started/overview/, Accessed on 2-09-2020
[35] API security Top 2019, https://owasp.org/www-project-api-security/, Accessed on 20-09-2020
[36] Thongkanchorn, K., Ngamsuriyaroj, S., amp; Visoottiviseth, V. (2013). Evaluation studies of three intrusion detection systems under various attacks and rule sets. 2013 IEEE International Conference of IEEE Region 10 (TENCON 2013). https://doi.org/10.1109/tencon.2013.6718975
[37] BRIM network analyser, https://www.brimsecurity.com/, Accessed on 20-10-2020
[38] Chandramouli, R., amp; Butcher, Z. (2020). Building Secure Microservices-based Applications Using Service-Mesh Architecture. https://doi.org/10.6028/nist.sp.800-204a-draft
[39] Manico, Jim. Webservice, Microservice and REST Security, owasp.org/www-pdf-archive//Webservice_and_Microservice_Security_-_Jim_Manico.pdf.
[40] Kiali dashboard, https://kiali.io/, Accessed on 22-10-2020
[41] National Security Agency Cybersecurity and Infrastructure Security Agency, "Kubernetes Hardening Guidance", National Security Agency (NSA) Cybersecurity Directorate Endpoint Security Cybersecurity and Infrastructure Security Agency (CISA), 2021.
[42] WP29 World Forum for Harmonization of Vehicle Regulations (WP.29), https://unece.org/transport/vehicle-regulations/wp29-world-forum-harmonization-vehicle-regulations-wp29, Accessed on 22-09-2021
[43] Threat Matrix for Kubernetes, https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/, Accessed on 30-06-2021
[44] Mitre Attack Matrix for Kubernetes, https://www.weave.works/blog/mitre-att-ck-matrix-for-kubernetes-tactics-techniques-explained-part-1, Accessed on 01-03-2023