CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Designing a Proactive Security Solution for Kubernetes Clusters

Enabling Multi-Step Attack Detection with Machine Learning

Master's thesis in Computer science and engineering

HAMPUS DE FLON
OMAR SULAIMAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Designing a Proactive Security Solution for Kubernetes Clusters

Enabling Multi-Step Attack Detection with Machine Learning

HAMPUS DE FLON
OMAR SULAIMAN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Designing a Proactive Security Solution for Kubernetes Clusters
Enabling Multi-Step Attack Detection with Machine Learning
HAMPUS DE FLON
OMAR SULAIMAN

Supervisor: Eric Olsson, Department of Computer Science and Engineering
Advisors: Mikael Eriksson XA, Sathya Prakash, Ericsson
Examiner: Magnus Almgren, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Designing a Proactive Security Solution for Kubernetes Clusters
Enabling Multi-Step Attack Detection with Machine Learning
HAMPUS DE FLON
OMAR SULAIMAN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Kubernetes cloud environments often contain security vulnerabilities exploitable during runtime, but undetected by the scanning tools widely used during build or deployment phases, due to the distributed and dynamic nature of containerized applications. Traditional reactive security approaches for detecting runtime exploits usually depend on manual intervention, causing delayed responses and unnecessary service disruptions because of false positives.

This thesis develops and evaluates a proactive security solution for Kubernetes clusters, with particular focus on 5G core network environments. The proposed solution integrates multi-step attack detection with automated mitigation strategies to improve detection accuracy and response time. Using a labeled dataset of Falco alerts generated within a 5G core network by the ACE-WARP project, five statistical and machine learning models are developed and evaluated: Markov Chain, Hidden Markov Model, Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN), and Bayesian Network. With each model, the likelihood of an attack sequence belonging to the normal or attack distributions is used for the binary classification task of identifying sequences containing attacks.

All models are evaluated with both balanced (1:1) and imbalanced (1:50) class ratios. The LSTM and CNN models achieve the highest detection rate of 84.3 percent and the lowest false alarm rate of 0.02 percent under imbalanced settings. They maintain strong precision (average precision of 99.2 percent) and area under the curve (AUC-ROC of 96.2 percent), outperforming the baseline Markov Chain, other models, and prior work.

Beyond identifying the best-performing models, this thesis also proposes mitigation techniques such as pod deletion for immediate isolation of threats and sandboxing for forensic analysis. These components form a complete security solution designed as a containerized microservice using Docker and exposed through a RESTful API, that can be applied to large Kubernetes clusters. Therefore, this design supports scalability, modularity, and compatibility requirements with industrial 5G core network deployments.

Keywords: Kubernetes security solution, multi-step attack detection, attack mitigation

Acknowledgements

We want to thank Eric Olsson, who as our academic supervisor guided us tirelessly through our thesis and provided us with valuable insights and knowledge.

We also want to thank Mikael Eriksson XA, and Sathya Prakash from Ericsson for making this thesis possible and helping us solve different problems we encountered during our thesis.

Hampus de Flon, Omar Sulaiman, Gothenburg, 2025-07-18

Abbreviations

Cloud and Virtualization

VM Virtual Machine

OS Operating System

NF Network Function

NFV Network Function Virtualization

SDN Software-Defined Networking

eBPF Extended Berkeley Packet Filter

Cellular Networks

SBA Service-Based Architecture

AMF Access and Mobility Management Function

SMF Session Management Function

UPF User Plane Function

AUSF Authentication Server Function

UDM Unified Data Management

NSSF Network Slice Selection Function

NEF Network Exposure Function

UE User Equipment

QoS Quality of Service

Security

APT Advanced Persistent Threat

DoS Denial of Service

Machine Learning

LSTM Long Short-Term Memory

HMM Hidden Markov Model

CNN Convolutional Neural Network

Metrics and Detection

TP True Positive

FP False Positive

TN True Negative

FN False Negative

DR Detection Rate

FAR False Alarm Rate

AUC Area Under Curve

ROC Receiver Operating Characteristic

Other

ACE-WARP A Cost-Effective Approach to Proactive and Non-Disruptive Incident Response

Contents

List of Figures

List of Tables

1

Introduction

Modern society relies on applications and services including web services, email systems, cloud storage, video streaming, video conferencing, social media, and banking apps, all of which rely on low latency, high performance, and scalability to reach millions of users [41]. These applications and services often achieve this performance and scalability by utilizing a combination of cloud computing and 5G networks.

The underlying infrastructure of cloud computing is typically built using either containers or Virtual Machines (VMs). Containers are lightweight alternatives to VMs that enable faster deployment and scaling of applications by packaging application code together with dependencies [69]. Enterprise applications, Internet of Things (IoT) platforms, 5G networks, and edge computing are examples of applications built in cloud computing, enabled by the resource efficiency and scalability that cloud environments provide [20].

Managing containerized applications becomes challenging as these applications increase in both their scale and complexity. Kubernetes can automate away some of these challenges, as its open-source container orchestration system solves management challenges by handling the deployment, scaling, and management of containers. Kubernetes schedules containers across a cluster of nodes, resulting in efficient resource utilization and high availability [43].

However, the isolation of containers is weaker than that of VMs, making it more likely for an attacker to exploit a vulnerable container and escape to the underlying infrastructure. This can lead to compromise of both the host system and its hosted applications [28]. According to a Sysdig report [74], 87% of container images contain critical security vulnerabilities, primarily in the operating system packages or libraries and dependencies added to the container image.

To identify vulnerabilities, container scanning tools are commonly used during build or deployment. These tools statically analyze container images and their components, such as operating system packages and application dependencies, by comparing them against databases of known vulnerabilities. This allows developers to detect and remediate some security issues early in the container lifecycle. However, because these traditional container scanning tools are typically one-time services that analyze container images during build or deployment, comparing them against vulnerability databases, they do not generate alerts during runtime [72]. As a result, they may

miss vulnerabilities that are introduced or otherwise only detectable dynamically.

Runtime security tools such as Falco [5] can be used to detect attacks during runtime. Falco monitors system calls and behavior patterns in monitored containers. Custom rules can be developed to detect specific suspicious activities in certain applications, such as unauthorized file access to a specific file in a specific pod. When such behavior is detected, Falco raises alerts that a security administrator can act on in real time [5, 4]. However, Falco's rule-based approach will produce false positives, which can overwhelm administrators with excessive alerts, and complicate and delay a timely and accurate threat response [5].

Traditionally, responding to detected attacks relies on well-timed manual intervention by security administrators. However, acting too early can cause unnecessary service disruption if the attack turns out to be a false positive, while responding too late may allow the attacker to cause damage to the infrastructure [8]. This thesis offers a proactive security solution consisting of multi-step attack detection and proposed mitigation strategies.

This thesis develops and evaluates statistical models for performing binary classification, to classify sequences as either attack or normal. This enables multi-step attack detection based on Falco alerts. Specifically, we aim to identify which model achieves the best attack detection rates while also having low false alarm rates, under balanced (1:1) and imbalanced (1:50) class conditions.

The highest performing model is combined with a mitigation strategy to create an end-to-end security solution designed to be compatible with industrial use. This makes it possible to detect an attack before it is fully executed, and to initiate mitigation before the attacker can cause further damage, as seen in Figure 1.1. This combination of early detection and automated mitigation aims to reduce false positives and minimize service disruptions while enhancing security in Kubernetes clusters.

An earlier example of an approach with similar goals is the ACE-WARP project [8]. The ACE-WARP solution shares our objecives of early attack detection and automated mitigation. Compared to this prior work, this thesis instead focuses on large-scale Kubernetes clusters, such as those with thousands of pods, compared to ACE-WARP's smaller clusters. ACE-WARP's mitigation approach, which relocates pods via migration, is impractical for large clusters due to its high latency, which can otherwise overload the Kubernetes cluster [62, 10]. This work builds upon ACE-WARP's labeled Falco alert dataset and evaluates advanced models, including LSTMs and CNNs, to improve detection accuracy. It also proposes mitigation strategies, such as pod deletion or sandboxing, designed for industrial-scale applications.

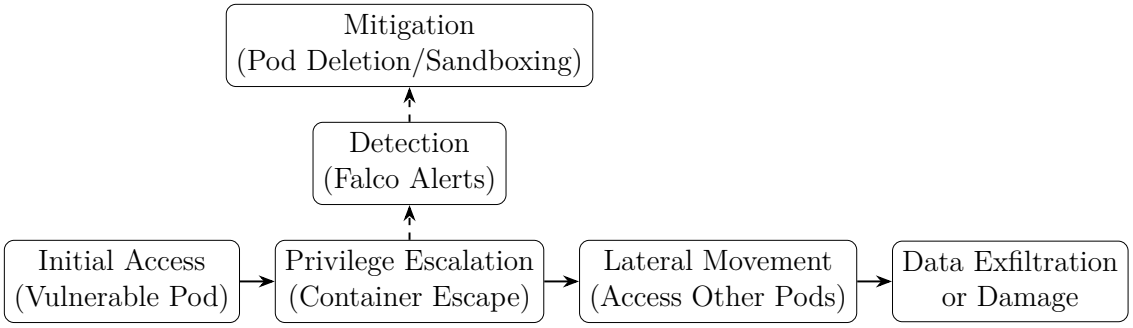

Figure 1.1: Illustration of the flow of a multi-step APT attack in a Kubernetes cluster, showing both the attack progression and where this thesis' solutions will detect and mitigate the attack.

While there are complementary methods to secure Kubernetes clusters in other ways, this thesis focuses on runtime security solutions. Other methods, such as container image scanning, are out of scope [5, 17, 39].

Runtime security tools are designed to detect threats while a container is running. Tools like Falco [5], Seccomp [17], and ProSPEC [39] base their analysis on data such as Linux system calls, Kubernetes metadata, or container behavior to monitor suspicious activity. When something unusual happens, these tools generate alerts.

Different tools have different strategies for responding to threats. Falco and Seccomp follow a reactive approach. While they automatically detect anomalies, such as unexpected process execution or file access, they rely on administrators to manually take action. This can lead to alert fatigue [54], where administrators become overwhelmed by frequent alerts and start ignoring them. Manual intervention can also be slow, giving attackers time to cause more damage [12].

Other tools employ a more proactive strategy. Instead of waiting for a human to respond, they automatically shut down or isolate potentially compromised containers. This can stop attacks quickly, but also comes with the risk of acting on false positives. If a normal process is incorrectly flagged, it can lead to unnecessary service disruptions and affect the user experience [8].

For example, ProSPEC [39] takes a proactive approach by triggering mitigation actions at runtime, after detecting critical events. This helps reduce further damage, but still carries risk. In some cases, the attacker may have already done serious harm before the response begins.

1.1 Problem Formulation

As noted previously, traditional incident response in Kubernetes clusters relies on a security administrator who manually reviews alerts generated by a runtime security tool such as Falco. This task is prone to human errors, not scalable, and finding the right balance in response timing can be difficult. If the administrator acts immediately on alerts that later turn out to be false positives, it can result in unwanted service

disruption. On the other hand, if the administrator delays the mitigation and the attack turns out to be real, it may allow the attacker to cause significant damage to the underlying infrastructure.

Furthermore, the number of alerts is often overwhelming, making it challenging for administrators to respond effectively without automation [8]. Achieving a high detection rate (to minimize the rate of missed attacks) in combination with a low false alarm rate (to avoid unnecessary interventions) is crucial for constructing a trustworthy and reliable security solution.

1.2 Aim

This thesis develops and evaluates multiple models for multi-step attack detection in Kubernetes clusters. These models perform binary classification to label sequences as either normal or attack sequences. An attack sequence is a series of system or application-level events that contain at least one step in a multi-step intrusion attempt. The evaluations are conducted under balanced (1:1) and imbalanced (1:50) class conditions to identify the model with the highest performance in realistic conditions.

1.3 Research Questions

This thesis aims to answer the following research questions:

1. Which model provides the highest detection rate for multi-step attacks?
2. Which model provides the lowest false alarm rate?
3. How does class balance affect the accuracy of the models?
4. Which model can be best used for multi-step attack detection?
5. Can the multi-step attack detection be integrated into a complete security solution for industrial use?

1.4 Challenges

Building a proactive security solution for Kubernetes clusters faces several challenges:

1. **Imbalanced Data**: As is typical in the security domain, the dataset has more normal sequences than attack ones (1:50 ratio). If not addressed, this imbalance makes it hard for models to learn attack patterns without missing attacks or raising false alarms [77].
2. **Complex Attack Patterns**: Multi-step attacks, like those conducted by APTs, use actions that blend with normal behavior, requiring models to spot long sequences accurately [15].

3. **Effective Mitigation**: Automatic mitigations to stop attacks also need to avoid harming services. Mitigations such as deleting or sandboxing pods need to be carefully designed to avoid downtime in large clusters [10].

4. **Limited Dataset**: The dataset used covers only specific 5G attacks, limiting models' ability to detect new threats in other setups [8].

Altogether, these challenges require careful model tuning, data handling, and mitigation planning to create a reliable solution for industrial use.

1.5 Comparison with ACE-WARP Project

This thesis and the ACE-WARP project share the goal of enhancing security in Kubernetes clusters based on Falco alerts by detecting and mitigating multi-step attacks. However, this thesis cannot adopt ACE-WARP's approach, due to broad differences in use cases and system requirements. These differences lead to specific variations in our design choices throughout data preprocessing, model selection, and mitigation strategies.

The ACE-WARP project uses a Bayesian Network model to detect attacks in Kubernetes clusters by mapping variables and their relationships to predict attack risks. Comparing with this model directly is difficult, as the project lacks clear documentation of the model, including details such as accuracy, and rather focuses on the performance of the overall system [8]. Its risk calculation and mitigation strategy, which involve moving pods linked to an attacked pod, was demonstrated to work in small clusters, but our use case is for large applications with thousands of pods and nodes [62]. In this setting, the ACE-WARP approach takes too long and risks slowing down or otherwise disrupting services in large-scale setups [10]. In contrast to this prior work, this thesis is designed with large Kubernetes clusters in mind, leading us to explore models and mitigation strategies better suited for such environments.

1.6 Outline

In Chapter 1, we outline the motivation and challenges of securing Kubernetes clusters against multi-step attacks, emphasizing the limitations of static container scanning and the need for runtime security solutions like Falco. It introduces the problem of manual alert review, key challenges in attack detection, and the thesis's structure. In Chapter 2, we provide the necessary background for this thesis, including introducing the concepts of cloud computing, Kubernetes, Kubernetes security, Falco, and several machine learning models, while also surveying related work. In Chapter 3, we present our method for performing binary classification to detect multi-step attacks. In Chapter 4, we present the results of our evaluation of the performance of the different models. In Chapter 5, we discuss our results and offer possible interpretations. We also consider limitations of this thesis, potential ideas for future work, and ethical considerations. In Chapter 6, we lay out how the multi-step attack detection model

can integrate with a mitigation strategy to provide a complete security solution, suitable for industrial applications. Finally, in Chapter 7 we conclude the thesis.

2

Background

This chapter provides the necessary background for this thesis, and surveys related work.

2.1 Virtual Machines and Containers

VMs are built using hypervisors to simulate physical hardware. In this design, a guest operating system is run for each instance. VMs provide strong isolation between each instance, since each VM operates independently at the hardware level [18]. However, an increased resource overhead is the cost of this isolation, since each VM requires its own system resources and OS kernel. Furthermore, additional overhead is entailed by hardware components that also need to be virtualized.

Containers provide a lightweight and resource-efficient alternative to VMs. Containers share an OS kernel and package application code together with dependencies [69]. Containers also avoid hardware virtualization, allowing both faster deployment and scalability. Therefore, containers are suitable for dynamic workloads in cloud environments. However, containers have new security challenges, because of their reliance on the host OS kernel. For instance, if an attacker exploits a kernel vulnerability or misconfigurations, it can lead to container escape attacks that compromise the host or other containers [57].

2.2 Kubernetes

A Kubernetes cluster consists of worker nodes and a control plane, as shown in Figure 2.1. Each worker node runs one or more pods, which are the smallest units that hold containers. Pods are managed by the Kubelet (which handles execution), a container runtime (for handling images), and kube-proxy (for networking) [7]. The control plane, which includes the API server and scheduler, manages how workloads are assigned and how the cluster operates [6].

Since Kubernetes relies on containers, it inherits their isolation-related security risks. A pod can be compromised through either privilege escalation or misconfiguration. This could let attackers escape to the host, access the Kubelet API, or control the cluster through the API server [58, 45]. A real example of this happened in 2018,

when attackers exploited a misconfigured Kubernetes dashboard at Tesla, and wasted resources and exposed internal systems to run cryptocurrency mining containers [14].

To reduce these risks, container scanning tools are often used during the build or deployment phases. However, since these tools do not run during execution, they might miss threats that appear at runtime [72]. To have comprehensive security, it is important to also use runtime tools, enforce security policies, and apply network segmentation. These steps can help detect attacks in real time, limit how far an attacker can move, and protect sensitive data [10, 5].

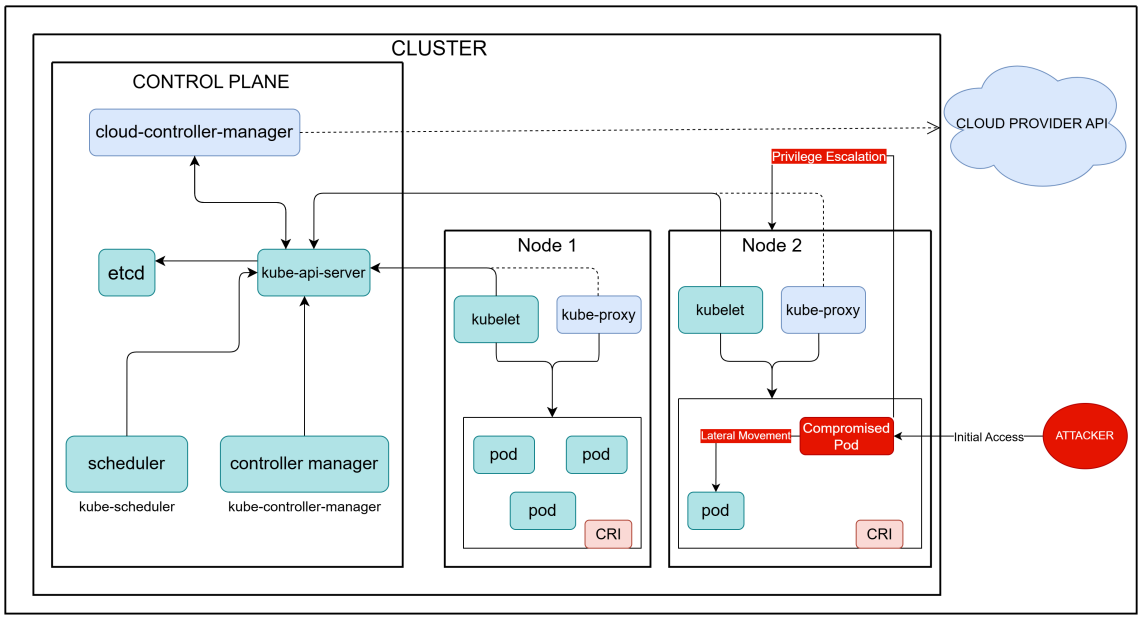

Figure 2.1: Illustration of how an attacker can perform privilege escalation in a Kubernetes cluster consisting of control plane, nodes, and pods.

2.3 Falco

Kubernetes runtime security can be improved through the use of tools like Falco [5], which enable monitoring Kubernetes clusters for potential threats during runtime. Falco can operate at different levels in containerized environments, including monitoring system activity at the kernel level. It accesses kernel system calls using, for example, eBPF (extended Berkeley Packet Filter) or a kernel module, allowing for it to oversee low-level events across nodes with low performance overhead. This system-level visibility allows Falco to detect behaviors that differ from expected workloads, such as spawning a shell inside a container or writing to sensitive directories.

Falco works by deploying its agents on worker nodes, where each agent is responsible for the evaluation of the system activity in real-time against a set of predefined rules. If a suspicious or unauthorized action is detected, an alert will be triggered and sent to a logging or monitoring system. Alerts are tagged with metadata, such as container name, process path, user identifier, and associated MITRE ATT&CK tactics. This makes it possible for a security administrator to act on alerts.

It is also possible to create custom rules, defining the specific conditions for an alert to be triggered. However, poorly defined rules can increase false positives or false negatives [4]. A rule with overly general conditions may trigger frequently on normal activity, leading to excessive false positives. Furthermore, false positive alerts are unavoidable, especially in complex environments. For example, consider the following Falco rule designed to detect an unauthorized shell spawning in a container:

```
rule: detect_shell_spawn
desc: Detects a shell process spawned inside a container
condition: container.id != host and proc.name in (bash, sh, zsh)
output: Shell spawned in container
(container=%container.name proc=%proc.name user=%user.name)
priority: WARNING
```

This rule triggers when a shell process is spawned inside a container. If an attack exploits a vulnerability to spawn a shell inside a container named, for instance, `web-app`, the rule would then be triggered to give the alert below:

> Shell spawned in container (container=web-app proc=bash user=attacker)

This alert contains metadata to help an administrator identify and mitigate the incident. However, if a script in a container named, for example, `admin-tool`, also spawns a shell process for routine tasks, the rule will still trigger. The rule's false positives could be limited by modifying its conditions, such as by excluding certain containers, but making the rule too specific can increase the risk of false negatives, allowing some malicious activity to go undetected.

2.4 Sequential Models for Attack Detection

Sequential data consists of data points arranged in a specific order, where each adjacent point may be dependent on its neighbors. Sequential data can be used for sequence classification. In this thesis, sequence classification is used to distinguish normal sequences from multi-step attack sequences. Several models can be used to *predict the next step in a sequence*. Choosing the right model is mostly a function of the complexity of the dependencies in the sequences. Simple statistical models like Markov Chains [67] can be used to identify simpler sequential patterns. However, in datasets with more complicated patterns, machine learning models with more parameters, such as LSTM, CNN, or HMM may perform better.

Sliding window techniques are often used on sequence data to improve predictive modeling. Long sequences are divided into smaller subsequences using a specific window size.

Models can also be trained to *perform binary classification* by distinguishing normal sequences from attack sequences, rather than being trained to predict the next step in a sequence. One way this can be done is by calculating the likelihood of a sequence under both normal and attack conditions. The difference in their log-likelihoods is passed through a sigmoid function to produce a probability between 0 and 1, where

a sequence scoring above 0.5 is classified as an attack, and otherwise classified as normal activity.

Two important metrics for multi-step attack detection are detection rate and false alarm rate. A high detection rate indicates that the model can detect most of the attacks that occur. Meanwhile, a low false alarm rate indicates that the model correctly identifies only the attacks, and doesn't flag other normal activity. This is crucial for creating a multi-step attack detection that is both reliable and trustworthy.

The proportion of attack sequences compared to normal sequences has large impact on the performance of models. Assessing the performance of models under different class balances, such as balanced (1:1) or imbalanced (1:50), can help to determine which distributions the model works with [37]. The imbalanced (1:50) ratio is used to simulate real-world scenarios, where attack sequences are rarer than normal sequences. Methods using neural network model typically learn the majority class more effectively, as they rely on gradient-based optimization. This can result in biased predictions and poorer results on the minority classes [77]. Underfitting the minority class occurs because the imbalance makes it harder for the model to learn. As a result, models often perform poorly on minority classes. However, in real-world applications in security, recognizing the minority class is crucial.

2.5 Markov Chain

Markov Chains are statistical models used to describe sequences of states, where the next upcoming state depends on the current state. This model is memory-less, and follows the Markov property [68]. In this design, there are fixed probabilistic transitions between states. The model has a set of states with a set of transitions that provide the probability of moving from one state to another. Formally, for a sequence of states $\{X_t\}_{t=0}^{\infty}$, where $X_t \in S$ and $S = \{s_1, s_2, \ldots, s_n\}$ is a finite or countable state space, the Markov property is defined as:

$$P(X_{t+1} = s_j \mid X_t = s_i, X_{t-1}, \ldots, X_0) = P(X_{t+1} = s_j \mid X_t = s_i) = p_{ij},$$

where p_{ij} is the transition probability from state s_i to state s_j.

Transition Matrix

The dynamics of a Markov Chain are fully defined by its transition matrix P, an $n \times n$ matrix where each entry p_{ij} represents the probability of moving from state s_i to state s_j. The matrix satisfies:

$$p_{ij} \geq 0 \quad \text{for all } i, j, \quad \text{and} \quad \sum_{j=1}^{n} p_{ij} = 1 \quad \text{for each } i,$$

ensuring that the probabilities are non-negative and each row sums to 1. For a Markov Chain with three states $S = \{s_1, s_2, s_3\}$, an example transition matrix is:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix},$$

where, for instance, p_{12} is the probability of transitioning from s_1 to s_2.

Graphical Representation

A Markov Chain can be visualized as a directed graph, where nodes represent states and edges are labeled with transition probabilities p_{ij}. Figure 2.2 illustrates a three-state Markov Chain with its transition probabilities.

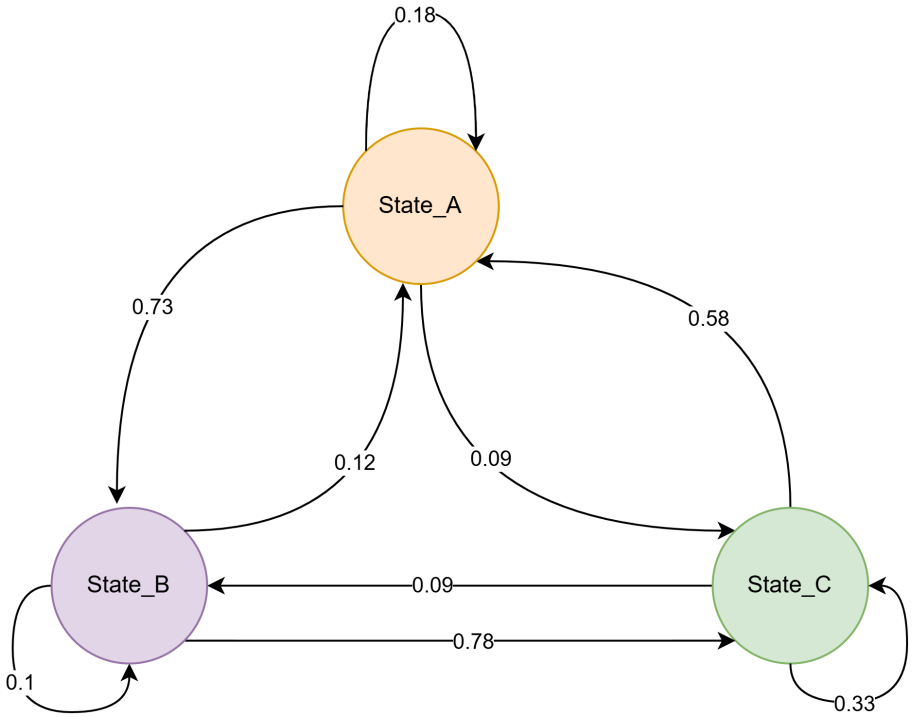

Figure 2.2: Example of a Markov Chain consisting of three states, along with the transitional probabilities.

2.6 Hidden Markov Model

Hidden Markov Models (HMMs) are probabilistic models used to model more ccomplex sequences of data, in tasks such as speech recognition and language processing [63]. In contrast to a normal Markov Model, an HMM assumes there is also a hidden sequence of states that cannot be directly seen, but each hidden state produces an output that can be analyzed to guess what the hidden state is. The system moves between these hidden states using the Markov property, meaning the next state will only depend on the current state.

An HMM consists of three parts: the probabilities of switching between hidden states, the probabilities of seeing certain outputs from each state, and the starting probabilities of the states. This model depends on two important ideas: that state transitions follow the Markov property, and that the visible outputs depend only on the current hidden state.

2.7 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are Recurrent Neural Networks (RNNs) designed to capture long-term dependence in sequential data [34]. While regular RNNs usually struggle to learn from long sequences during training, LSTM can learn long sequences by using their memory cells. An LSTM memory cell stores the state c_t at each time step t, to maintain some information from its previous input. The input gate controls how much new information should be added to the memory. The forget gate decides what old information to remove, and the output gate decides what information to pass on to the next time step or the final output. This makes LSTMs capable of modeling long-term dependencies in sequential data, making them suitable for predicting the next step in a sequence or for classifying sequential data.

2.8 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a type of deep learning model used for data that has a grid-like structure, such as images or time-series data [46]. They consist of convolutional layers that apply filters to recognize patterns in the training data. After these initial layers, pooling layers are used to decrease the size of the data, while keeping the most important data. This reduces the number of computations while also reducing overfitting. CNNs are well-suited for image classification, object detection, and facial recognition tasks because they learn complex features step-by-step [42]. However, their training requires large amounts of labeled data and is computationally heavy.

2.9 Bayesian Model

Bayesian models are designed to make decisions when there is uncertainty about the true state of the data or the environment. They use probabilities to represent the model's belief about possible outcomes based on its currently available information. When new data come in, the model's beliefs are updated using Bayes' theorem [26]. Variables in these models are defined as probability distributions, which allows the model to include prior knowledge and also give an estimate of the confidence in its predictions.

Bayesian methods are used in many areas, such as spam filters, medical diagnosis, and learning model parameters. Since exact calculations are often hard, methods like Markov Chain Monte Carlo or variational inference are used to estimate results [66]. Even though Bayesian models are powerful for handling uncertainty, they can be slow to compute and can depend a lot on their initial assumptions.

2.10 MITRE ATT&CK Tactics

MITRE ATT&CK techniques can be extracted from alerts that Falco generates. These tactics represent adversaries' strategies during cyber attacks [53].

Mitre Credential Access

An adversary attempts to steal credentials, such as account names and passwords, using keylogging or credential dumping. This allow the attacker to escalate privileges and make detection harder. For instance, Falco may detect an attempt at credential dumping when a process is accessing sensitive files.

Mitre Defense Evasion

An adversary attempts to evade detection and bypass security controls while operating in a compromised environment. This tactic allows attackers to be stealthy and persistent through obfuscating malicious code, disabling security tools, or imitating legitimate processes. For example, Falco may detect signs of evasion when a process disables a security daemon or renames a malicious binary to mimic a trusted utility.

Mitre Discovery

The adversary uses tools to gain knowledge about the environment before deciding how to act. This helps the adversary get information about their environment, like ports and device specifications. For example, Falco can detect discovery activity by detecting if a process runs a tool to display network connections inside a container, as this may be an attempt to enumerate network connections or running processes.

Mitre Execution

The adversary tries to run malicious code on a system after gaining initial access, which can result in data theft, persistence, lateral movement, or other attacks. This tactic includes methods like executing commands via command-line interfaces, running scripts, or deploying compiled binaries like malware executables. For example, Falco might trigger an alert when a shell is spawned inside a container to execute a malicious script.

Mitre Exfiltration

The adversary attempts to steal data from the environment. It is common for collected data to be stored with compression or encryption. This data, once acquired by the attackers, is then sent out of the environment to an attacker controlled endpoint. For example, Falco can detect exfiltration by identifying if a process runs tools transferring data over the internet, sending large amounts of data to an external IP address from a container.

Mitre Persistence

The adversary is trying to maintain their presence in the system. Different techniques are used to ensure that the adversary can maintain access to systems, such as changed credentials and other interruptions that could remove their access. For example,

Falco might alert when a process modifies a Linux file for scheduling recurring tasks to schedule a malicious script.

Mitre Lateral Movement

The adversaries can navigate through a network to expand their control and reach valuable targets after gaining initial access. Once inside, attackers can, for example, exploit remote services or use stolen credentials for authorized logins to move across systems undetected. The risk of lateral movement lies in its ability to transform a single breach into a widespread compromise. For example, Falco can detect lateral movement by flagging an unauthorized secure shell connection attempt from a compromised container to another node in the cluster.

Mitre Privilege Escalation

The adversary tries to gain higher privileges and permissions. It is usually possible for attackers to explore a network or system with even unprivileged access, but higher privileges make it much easier. Techniques include taking advantage of misconfigurations, system weaknesses, and vulnerabilities. For example, Falco might trigger an alert when a process exploits a kernel vulnerability to elevate its privileges within a container.

2.11 Advanced Persistent Threats

Advanced Persistent Threats (APTs) are targeted cyberattacks that are distinguished by their ability to stay hidden in a system for a long time. Unlike fast, random attacks, APTs are carefully set up by skilled groups, such as government-backed hackers or organized crime teams [80]. In Kubernetes clusters that handle container workloads for 5G networks, APTs can use outdated software, or bad access rules to steal data, get inside without permission, or otherwise interfere with services.

2.11.1 Characteristics of APTs

APTs stand out from regular cyberattacks in these ways:

- **Clear Targets**: APTs are aimed at important targets like government offices or large tech companies, not just any vulnerable system. For example, a 2013 report identifies education, telecom, and aerospace as key targets [23].
- **Organized Teams**: These attacks come from well-funded groups, often using previously unknown weaknesses (zero-day vulnerabilities) or special tools [71].
- **Long Duration**: APTs can endure for months, all the while evading detection. For example, the 2009 Operation Aurora attack on Google lasted for six months [15].
- **Hidden Tactics**: Attackers use further tricks such as encryption or trusted services to cover their tracks, unlike short attacks.

2.11.2 APT Attack Model: Six Steps

APTs follow a clear plan called the "intrusion kill chain" with six steps [35]:

1. **Research and Setup**: Attackers collect details using public info or tricks like fake emails and build custom attacks, such as harmful files.
2. **Sending the Attack**: Attacks are delivered through targeted emails or hacked websites.
3. **First Break-In**: Attackers get in through software weaknesses, such as multiple zero-day vulnerabilities in Internet Explorer during Operation Aurora (CVE-2010-0249) [15].
4. **Remote Control**: Attackers set up control from afar using tools like Remote Access Trojans (RATs).
5. **Spreading Inside**: Attackers move around the network using stolen passwords or admin tools.
6. **Stealing Data**: Sensitive information is gathered, locked with encryption, and sent to outside servers.

2.11.3 Relevance to the Case Study

This study looks at Falco alerts from simulated APT attacks in a 5G Kubernetes setup. This dataset includes several attacks that are based on real APT campaigns, such as APT3, APT29, and targeted phishing. These are shown in Table 3.1. These attacks follow common APT steps like lateral movement, persistence, and stealing data.

Knowing how APTs work helps detection models, such as LSTM and CNN, learn their patterns. This makes it easier to spot these hidden attacks and improve security in important systems including 5G networks.

2.12 Evaluation Metrics

This section introduces and explains the metrics we use to evaluate our models performance in detecting multi-step attacks. It is crucial to use metrics that both measure the model's efficiency as binary classifiers and their ability to detect attacks while also having a low false alarm rate.

2.12.1 AUC-ROC

The Receiver Operating Characteristic (ROC) curve evaluates a model's ability to distinguish between positive and negative classes. It plots the true positive rate (correct positive predictions divided by total positives) against the false positive rate (incorrect positive predictions divided by total negatives) at different thresholds. The ROC shows the trade-off between accurate positive predictions and false alarms.

The Area Under the Curve (AUC) of the ROC curve measures the overall performance of the model. AUC ranges from 0 to 1, where 0.5 means random guessing and 1.0 indicates perfect classification. A higher AUC suggests a better model, and is useful for comparing different classifiers, especially on imbalanced datasets [11, 37].

2.12.2 Detection Rate

The Detection Rate (DR) indicates the proportion of actual positive events that are classified correctly as positive, and the total number of positive events. The DR is calculated using the number of False Negatives (FNs) and the number of True Positives (TPs) using the formula [60]:

$$\mathrm{DR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

In attack detection systems, a higher DR indicates higher attack detection, which is crucial to not miss attacks [29].

2.12.3 False Alarm Rate

The False Alarm Rate (FAR) evaluates the proportion between the number of negative events incorrectly classified as positive, and the total number of actual negative events. The FAR is calculated using the number of FPs and the number of TNs using the formula [60]:

$$\mathrm{FAR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

In attack detection systems, keeping the FAR low is important because excessive false alerts can cause operators to ignore warnings and lose their trust in the system [40]. Having a low FAR in combination with a high DR is crucial to correctly identify a high percentage of attacks.

2.13 Related Work

Prior research on multi-step attack prediction and mitigation has been conducted across various domains, with a growing use of machine learning and statistical models for sequential data analysis. Relevant work, starting with multi-step attack detection models, followed by approaches specific to prediction, mitigation, and provenance analysis, is presented below.

Multi-Step Attack Prediction and Mitigation

Traditional detection and mitigation systems are reactive, and will only react to security breaches or violations of policy after damage has been inflicted. Tools such as Sysdig [73], Falco [5], Tetragon [75], and OPA/Gatekeeper [2] monitor containers at runtime. Sysdig monitors at the system level, and Falco detects malicious activities during runtime by deploying agents in each worker node and reporting suspicious

actions as alerts to security teams. Tetragon uses eBPF to watch what is happening inside the Linux kernel with low performance cost. It can spot suspicious actions, like running unknown programs or accessing files without permissions. OPA/Gatekeeper, on the other hand, enforces security policies at admission time (whenever a pod is created, updated, or deleted).

KubAnomaly [76] uses anomaly detection for Kubernetes security monitoring by identifying suspicious behaviors in the cluster. In contrast, Du et al. [19] propose an anomaly detection and diagnosis system for container-based microservices that monitors and analyzes real-time performance data using machine learning models trained with fault injection. Zou et al. [82] develop an online Docker container anomaly detection system based on an optimized isolation forest algorithm, which assigns weights to resource metrics to improve detection accuracy and analyzes container logs to locate the root causes of anomalies. However, these approaches primarily focus on detection and diagnosis of attacks without integrating mitigation mechanisms. Our solution complements these methods by providing predictive attack modeling that enables proactive mitigation to prevent attacks before they occur.

One option for attack mitigation is dynamic resource isolation or sandboxes, which isolate compromised pods by wrapping them with a lightweight VM to separate the pod from other workloads. Solutions such as Kata Containers [36] implement this technique to provide stronger security restrictions than typical containers. Sandboxing mitigates the attack surface by blocking lateral movement within the cluster, while remaining non-disruptive and computationally inexpensive.

Compared to this thesis, there is a certain resemblance, as we also work on Kubernetes security, detect suspicious behaviors, and mitigate attacks to prevent privilege escalation. However, our approach differs as we integrate predictive multi-step attack modeling to enable proactive mitigation by deleting suspicious pods before attacks inflict more damage, unlike the reactive detection and alerting tools mentioned before. Furthermore, while sandboxing, such as Kata Containers, isolates pods non-disruptively, our pod deletion strategy prioritizes fast threat containment.

2.13.1 Proactive Attack Detection and Mitigation

Other works have explored proactive security policy enforcement for containerized and cloud environments [13, 48, 49, 47]. ProSPEC [39] extends proactive auditing to Kubernetes, but it only initiates mitigation after attack detection, which is too late to prevent damage. Unlike these solutions, our approach employs a lightweight, predictive mitigation framework that minimizes service disruptions while detecting attacks at an early stage.

2.13.2 Provenance Analysis

Provenance-based security solutions, such as OmegaLog [32], focus on analyzing the causes of security breaches by tracking the origin and flow of data and events. While this seems to be effective for forensic analysis, these approaches lack mechanisms for active mitigation. For example, OmegaLog creates very detailed logs to help track

exactly what happened where, which makes it easier to understand the full history of events. Some solutions [31] use tactical analyses of security events, while others [79] provide a user-interface-driven scope for provenance investigation.

2.13.3 ACE-WARP

The ACE-WARP project [8] is an earlier complete proactive security solution consisting of early attack detection and mitigation. First, they generate a dataset [9] containing Falco alerts inside a 5G core [25]. The alerts come from both normal activity and sampled attacks, and each alert is labeled as either normal or attack. Each alert is then mapped to a sequence by extracting the alert's MITRE ATT&CK tactic, which are summarized in Table 3.1. A Bayesian Network model is trained to predict the next step for a sequence. The predicted next step is then used in a risk calculation formula, together with other factors. The mitigation phase is then initiated if the risk exceeds a specific threshold. Their mitigation works by relocating all pods connected to the attacked pod. In their evaluation, the overall security solution can mitigate up to 81% of the attacks with an average false positive rate of 39%.

This thesis also proposes an early attack detection and mitigation security solution. However, the use case differs slightly, so both the multi-step attack detection as well as the mitigation strategy differ to the design of Bagheri et al. [8]. This thesis still builds upon ACE-WARP dataset. However, some different preprocessing steps are taken, as specified in Section 3.2. To enable comparison, this thesis also trains a Bayesian Network, but for classifying sequences as either normal or an attack instead of predicting the next alert in a sequence. The mitigation strategy also differs due to different use cases and requirements. ACE-WARP evaluates its complete security solution of early attack detection and mitigation, using accuracy and the false positive rate. Rather than only using the false positive rate, this thesis uses FAR and DR to evaluate how well the models identify attacks.

2.13.4 Binary Classification of Normal and Attack Sequences

Kawawa-Beaudan et al. [37] use sequential models for binary classification on multiple datasets to distinguish attack sequences from normal sequences. They use models including LSTM, CNN, and different HMM models, under both balanced (1:1) and imbalanced (1:50) class conditions. The models are evaluated on several different datasets containing both attack and normal data. The models are built to enable binary classification of sequences, and identify if a sequence is either an attack or a normal sequence. The models are evaluated using AUC-ROC and average precision.

This thesis performs the same binary classification using statistical models to detect multi-step attack sequences and normal sequences. Similarly to this paper, our thesis also uses LSTM, CNN, and HMM models under balanced (1:1) and imbalanced (1:50) class conditions to evaluate how class balance affects the models' performance on the dataset. The same metrics, AUC-ROC and average precision, are used in this

thesis to assess the performance of the models. But in contrast to this paper, this thesis uses a Markov Chain as a baseline model.

3

Methodology

This chapter presents how we develop the multi-step attack detection models used in this thesis. We first overview the entire method, then introduce the dataset and our preprocessing, define the models used, and explain our evaluation metrics and use of cross-validation.

3.1 Overview

The ACE-WARP dataset [9] contains Falco alerts labeled as either normal (false alert, FP) or attack (true alert, TP) generated from a 5G Core [25]. The alerts labeled as normal originates from the application and the alerts labeled as attack originates from simulated attacks. This dataset can be used to perform multi-step attack detection with binary classification labeling sequences as either normal or attack. Alerts are mapped into sequences based on the container ID and the MITRE ATT&CK tactic. The tactics are then extracted from each alert, creating a sequence of MITRE ATT&CK tactics. Table 3.1 presents the eight different attacks from the dataset along with their features and sequences.

Table 3.2 and Table 3.3 show the resulting sequences. About 47% of the normal sequences have only one step, so an end token is added to clearly show where the sequence finishes. The average sequence length is 2.7 after adding the end token. Sequences are divided into subsequences of size 2 using a sliding window. This preprocessing largely follows the ACE-WARP preprocessing of the data.

A sequence is regarded as an attack sequence if any step in the sequence originates from an attack labeled alert, and otherwise as a normal sequence.The resulting dataset contains only 0.7% attack sequences. To handle this imbalance, the attack sequences are duplicated to create two versions of the dataset with class balances of 2% (1:50) and 50% (1:1). The dataset is then split into 70% for training, 15% for validation, and 15% for testing. After that, the sequences are tokenized into integers so they can be used as input for the model.

Markov Chain, HMM, LSTM, CNN, and Bayesian Network models are used for binary classification of sequences under both imbalanced (1:50) and balanced (1:1) class settings. The models are then evaluated using AUC, false alarm rate, detection rate, and average precision under both class balances to assess how class balance affects the model's performance.

Table 3.1: Overview of the attacks in the ACE-WARP dataset, adapted from [8]

Attack Campaign	**Attack Features**[a]					**MITRE ATT&CK Tactic Sequence**[b]
	PL	**PA**	**IN**	**IG**	**BD**	
Advanced Persistent Threat 3 [52]	✓	✓	✓	✓	✓	EXE, DE, DIS, DE, LM
Spam Campaign [65]		✓	✓	✓	✓	DIS, PER, EXE, DE, DE, LM, EXF
Advanced Persistent Threat 29 [51]	✓	✓	✓	✓	✓	PER, EXE, DE, PE, DE, DIS, LM, IA, PER, PE, DE
Escape Attack [81]				✓		PE, EXE, PER
Simulated Cryptominer Spread [64]	✓		✓	✓	✓	DIS, EXE, PER, DE, LM
Root Data Theft [16]			✓	✓	✓	DIS, PER, PE, EXF, PER, LM
Strategic Web Compromise [22]	✓		✓	✓	✓	DIS, EXE, DE
Targeted .gov Phishing [56]	✓		✓	✓	✓	DIS, PER, LM, EXF

[a] PL: Phishing email link, PA: Phishing email attachment, IN: Injection, IG: Information gathering, BD: Backdoor
[b] EXE: Execution, DE: Defense Evasion, DIS: Discovery, LM: Lateral Movement, PER: Persistence, PE: Privilege Escalation, IA: Initial Access, EXF: Exfiltration

3.2 Dataset

The ACE-WARP dataset [9], originates from the ACE-WARP project and contains 176,649 Falco alerts generated inside a 5G Core [25]. Table 3.1 details the eight attack features and MITRE ATT&CK tactic sequences. The alerts are labeled as either normal (FP) or attack (TP). The alerts are mapped into 52,957 sequences based on container ID using the alert's MITRE ATT&CK tactic. An end token is appended to each sequence to enable end-of-sequence prediction, but also to preserve the single-step sequence, which stands for 42.1% of the normal sequences. Table 3.2 displays the most common normal sequences, while Table 3.3 contains the most common attack sequences. The sequences are then split into subsequences of size two, using a sliding window. Since the average sequence length is 2.7, a smaller sliding window is chosen to retain most sequences, since a larger sliding window size would remove all sequences shorter than the selected size. All sequences with attack-labeled steps are considered attack sequences, and all others as normal sequences. The dataset contains 0.7% attack sequences, which are upsampled by duplication to preserve original patterns, to both imbalanced (1:50) and balanced (1:1) class conditions.

Five sequences with lengths between 150 and 100,000 steps are considered outliers and removed before applying the sliding window. They are much longer than the

Count	Sequence
46.6%	Lateral Movement, Privilege Escalation, End
42.1%	Lateral Movement, End
6.1%	Lateral Movement, Privilege Escalation, Lateral Movement, Privilege Escalation, End
5.2%	Lateral Movement, Lateral Movement, End

Table 3.2: Most common normal sequences observed in the dataset.

Count	Sequence
15.7%	Lateral Movement, Privilege Escalation, Persistence, End
12.8%	Lateral Movement, Persistence, End
8.0%	Lateral Movement, Privilege Escalation, Defense Evasion, End
6.7%	Lateral Movement, Privilege Escalation, End
5.7%	Lateral Movement, Execution, End

Table 3.3: Most common attack sequences observed in the dataset.

usual sequences, which normally start with the lateral movement tactic and are shorter than ten steps. These outliers likely occur because of problems during data generation, such as logging errors or loops in the simulation. For example, one sequence repeats the same "execution" and "persistence" steps hundreds of times. Figure 3.1 shows the sequence length distribution after removing these outliers.

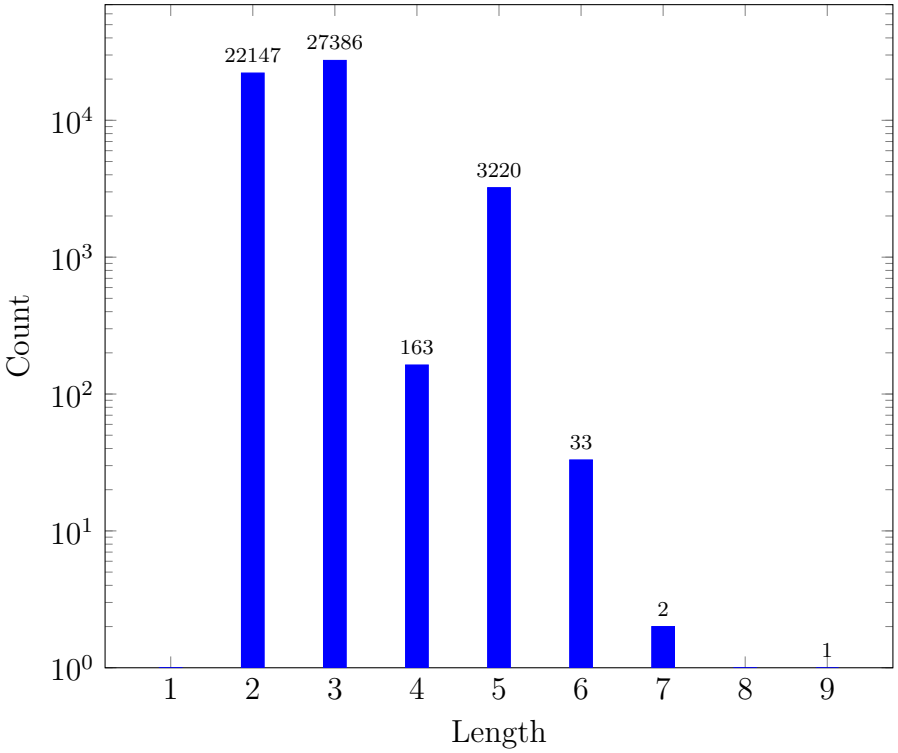

Figure 3.1: Sequence length distribution after removing outliers.

Table 3.4 displays the most common normal sequences after performing a sliding

window. 90% of the normal activity only contains three unique sequences. Meanwhile, Table 3.5, shows the most common attack sequences in a sliding window. In contrast, there are several unique attack sequences where the probability is more evenly distributed. While normal sequences are less varied, distinguishing them from diverse attack sequences is still challenging due to subtle attack behaviors and imbalanced data, where normal sequences far outnumber attacks. Therefore, both simpler models, such as statistical models, and also more complex machine learning models, are developed and evaluated. The ACE-WARP project performs the same data preprocessing procedure as this thesis, except for appending an end token to all sequences.

Count	**Sequence**
34.7%	Lateral Movement, Privilege Escalation
30.9%	Privilege Escalation, End
27.7%	Lateral Movement, End
3.6%	Privilege Escalation, Lateral Movement

Table 3.4: Common normal sequences observed in the dataset after applying the sliding window. Around 90% of normal activity is covered by just three patterns, highlighting the low variety of normal behavior in contrast to the more diverse attack sequences.

Count	**Sequence**
15.7%	Persistence, END
9.0%	Privilege Escalation, Persistence
7.8%	Defense Evasion, END
7.7%	Lateral Movement, Persistence
6.0%	Execution, END
5.6%	Discovery, END

Table 3.5: Most common attack sequences observed in the dataset after sliding window.

3.3 Multi-Step Attack Detection Models

This thesis compares five models: Markov Chain, HMM, LSTM, CNN, and Bayesian Network for binary classification of attack sequences under balanced (1:1) and imbalanced (1:50) class conditions. The models perform binary classification of sequences by classifying if a given sequence is either an attack or a normal sequence. The following subsections describe each model's architecture and classification strategy.

3.3.1 Common Classification Approach

All models perform binary classification by calculating the likelihood of a sequence under both normal and attack conditions. The difference in their log likelihoods is passed through a sigmoid function to produce a probability between 0 and 1, where

a sequence scoring above 0.5 is classified as an attack, and otherwise it is classified as normal.

3.3.2 Model Choices

Prior related work [37] perform binary classification under balanced (1:1) and imbalanced (1:50) class conditions. We include the models they used: CNN, LSTM, and HMM. The ACE-WARP project uses a Bayesian model, which we also include. Since the window size is two, a Markov Chain is used as a baseline model to determine whether the data can be captured by a simple model. If a model does not outperform the Markov Chain, then it is not useful for this task. The model choices and their performance are further discussed in Section 5.1.2.

While there are no parameters to configure for Markov Chain models, the other models are configured for this task.

Markov Chain

There are no parameters to configure for Markov Chain models.

Hidden Markov Model

HMMs are configured with hidden states of size two.

Long Short-Term Memory

The LSTM model's architecture starts with an embedding layer of dimension 100, which maps input tokens into dense vector representations. This is followed by an LSTM layer with 16 units, using both dropout and recurrent dropout rates of 0.4 to prevent overfitting. A fully connected dense layer with 32 units and ReLU activation and L2 regularization is then applied to improve generalization. The final output layer consists of a dense layer with a single unit and a sigmoid activation function to generate outputs between 0 and 1. The model is trained using a binary cross-entropy loss function and the Adam optimizer. Early stopping is used during training to prevent overfitting.

Convolutional Neural Network

The CNN model uses an embedding layer of dimension 100 to convert tokens into dense vectors. These embeddings are then inserted into a one-dimensional convolution layer with 128 filters, with a kernel size of 3, along with ReLU activation. A global max pooling layer is then used, followed by a dropout layer of size 0.4 to avoid overfitting. The final output layer consists of a fully connected dense layer with a single unit and sigmoid activation function to output a probability between 0 and 1. The model is trained using the binary cross-entropy loss function and the Adam optimizer. Early stopping is utilized during training to prevent overfitting.

Bayesian Network

A connection is made between the first and second token in each sequence to show how they depend on each other. Both Bayesian Network models are trained using a Bayesian Estimator with a Bayesian Dirichlet equivalent uniform setting, to score how good the Bayesian Network structure is.

3.3.3 Class Balances and Evaluation Metrics

The models are trained and evaluated under both imbalanced (1:50) and balanced (1:1) class conditions to assess how class imbalance affects the performance of these models.

The ROC curve is used to illustrate the balance between the rate of true positives and the rate of false positives at different classification thresholds. The AUC (Area Under Curve) makes it possible to compare ROC curves, and higher AUC values indicate better classifiers.

The DR of attacks is especially important, since it is crucial when finding the most reliable model. The FAR is also critically assessed to identify the model with the lowest FAR, which is important both for reliability and to prevent fatigue among security administrators. AP (Average Precision) is also used to assess the model's precision.

While AUC provides an overall performance measure, DR and FAR carry more practical significance in this context, as they directly impact the effectiveness and usability of the detection system.

3.3.4 Training, Validation and Testing

The sequences are randomly shuffled and split into 70% training, 15% validation, and 15% testing sets using stratified sampling to maintain the proportion of attack and normal sequences in each split. The training set optimizes model parameters, while the validation set guides hyperparameter tuning. The validation data is not a subset of the testing data or the training data. The test set is held out until the final evaluation to assess generalization performance on unseen data. Hyperparameters are tuned to maximize average validation accuracy across five cross-validation folds, as described in Section 3.3.5.

3.3.5 Cross Validation

Using a validation set that is not a subset of the training set helps reduce bias in estimating how well the model generalizes to unseen sequences, and it also increases the variance of the validation accuracy. This is because the training and validation sets contain data with different sequences, making performance estimates less reliable. Cross-validation addresses this issue by training the models multiple times, each time using a different subset of the data as the validation set across five folds.

4

Evaluation

This chapter presents details about the implementation and evaluation data obtained from the multi-step attack detection models.

4.1 Implementation

A Python code template was developed in this thesis to implement these experiments. This code will be open-sourced, and can be reused for future experiments, as it includes data preprocessing and model evaluation tools. The data is tokenized using the Keras tokenizer [38]. Libraries like Pandas [50] and NumPy [30] were used. The LSTM models were built using Keras [38]. The Bayesian models originate from pgmpy [3]. Meanwhile, the categorical HMM is implemented using hmmlearn [33]. Cross-validation and data splits are implemented using scikit-learn [59]. The dataset used originates from the ACE-WARP project [9].

4.2 Multi-Step Attack Detection Models

In this section, we present the results obtained from the evaluation of the selected multi-step attack detection models. The presented evaluations are averages over all cross-validation folds, as previously mentioned in Section 3.3.5. The models were tuned using the hyperparameter combination that resulted in the highest average validation accuracy. The test accuracies were computed at the end, as explained in Section 3.3.4. The performance of each model is measured using AUC, AP, and FAR, for both the attack and normal classes under balanced and imbalanced conditions.

4.2.1 Summary

A summary of each model's performance can be seen in Table 4.1. The CNN and LSTM models had the highest performance across all metrics, especially under imbalanced settings with a FAR as low as 0.02%. The HMM model also had an equally good performance under balanced class conditions. The machine learning models outperform the baseline Markov Chain model on multi-step attack detection, by providing a substantially lower FAR. The overall lowest performing model is the Bayesian Network model, with both low AP and high FAR, indicating that it struggles with this binary classification task.

Table 4.1: Model performance comparison on test set under balanced and imbalanced conditions.

Model	Class Balance	Detection Rate	FAR	Average Precision
Markov Chain	1:1	90.0	6.7	91.7
	1:50	91.0	6.7	61.1
Hidden Markov Chain	1:1	88.2	3.4	92.7
	1:50	87.8	3.6	66.9
Long Short-Term Memory	1:1	89.4	3.4	93.2
	1:50	84.3	0.02	99.2
Convolutional Neural Network	1:1	89.4	3.4	93.2
	1:50	84.3	0.02	99.2
Bayesian Network	1:1	82.7	96.8	30.7
	1:50	70.0	33.7	51.6

4.2.2 Markov Chain

The Markov Chain model's performance for multi-step attack prediction is presented, with the model defined in Section 3.3.2. Table 4.2 shows the average accuracies across five cross-validation folds for training, validation, and testing under balanced (1:1) and imbalanced (1:50) conditions, achieving testing accuracies of 91.5% and 93.1%, respectively. Table 4.3 provides detailed performance metrics (AUC, DR, FAR, AP) for validation and testing stages.

Table 4.2: Mean accuracies of Markov Chain model across cross-validation folds.

Stage	Class Balance	Accuracy
Training	1:1	91.7
	1:50	93.3
Validation	1:1	91.7
	1:50	93.3
Testing	1:1	91.5
	1:50	93.1

Table 4.3: Performance metrics of Markov Chain model on validation and test sets.

Stage	Class Balance	AUC	Detection Rate	FAR	Average Precision
Validation	1:1	95.7	90.1	6.7	91.7
	1:50	95.9	90.4	6.7	61.1
Testing	1:1	95.8	90.0	6.7	91.7
	1:50	95.7	91.0	6.7	61.1

4.2.3 Hidden Markov Model

The HMM results for multi-step attack prediction, using the model described in Section 3.3.2, are presented. Table 4.4 shows mean accuracies over five cross-validation folds, achieving testing accuracies of 91.4% (1:1) and 96.2% (1:50). Table 4.5 details performance metrics (AUC, DR, FAR, AP) for validation and testing stages, using two hidden states to match a sliding window size of two.

Table 4.4: Mean accuracies of HMM across cross-validation folds.

Stage	Class Balance	Accuracy
Training	1:1	91.0
	1:50	96.1
Validation	1:1	91.0
	1:50	96.1
Testing	1:1	91.4
	1:50	96.2

Table 4.5: Performance metrics of HMM on validation and test sets.

Stage	Class Balance	AUC	Detection Rate	FAR	Average Precision
Validation	1:1	92.0	85.6	3.7	91.5
	1:50	92.2	85.4	3.7	66.4
Testing	1:1	94.4	88.2	3.4	92.7
	1:50	94.0	87.8	3.6	66.9

4.2.4 Convolutional Neural Network

The CNN results for multi-step attack prediction, for the model defined in Section 3.3.2, are presented. Table 4.6 outlines the model architecture againl; it includes 128 Conv1D filters, GlobalMaxPooling1D, and a 0.4 dropout rate. Table 4.7 shows mean accuracies over five cross-validation folds, with testing accuracies of 92.8% (1:1) and 99.7% (1:50). Table 4.8 provides performance metrics (AUC, DR, FAR, AP) for validation and testing stages.

Table 4.6: CNN model architecture and training configuration.

Parameter	Value
Embedding dimension	100
Conv1D filters	128
Conv1D kernel size	3
GlobalMaxPooling1D	
Dropout	0.4
Dense layer width	1
Dense layer activation	sigmoid

Table 4.7: Mean accuracies of CNN across cross-validation folds.

Stage	Class Balance	Accuracy
Training	1:1	92.5
	1:50	99.7
Validation	1:1	92.5
	1:50	99.7
Testing	1:1	92.8
	1:50	99.7

Table 4.8: Performance metrics of CNN on validation and test sets.

Stage	Class Balance	AUC	Detection Rate	FAR	Average Precision
Validation	1:1	96.6	88.8	3.6	92.8
	1:50	96.9	84.9	0.02	99.2
Testing	1:1	96.7	89.4	3.4	93.2
	1:50	96.2	84.3	0.02	99.2

4.2.5 Long Short-Term Memory

The LSTM model results for multi-step attack prediction, for the model defined in Section 3.3.2, are presented. Table 4.9 details the model architecture, including a 16-unit LSTM layer with 0.4 dropout and recurrent dropout. Table 4.10 shows mean accuracies over five cross-validation folds, with testing accuracies of 93.0% (1:1) and 99.7% (1:50). Table 4.11 provides performance metrics (AUC, DR, FAR, AP) for validation and testing stages.

Table 4.9: LSTM model architecture and hyperparameters.

Parameter	Value
Embedding dimension	100
LSTM width	16
LSTM dropout rate	0.4
LSTM recurrent dropout	0.4
Dense layer width	32
Dense layer regularization	L2 (0.04)
Dense layer 2 width	1
Dense layer 2 activation	sigmoid

Table 4.10: Mean accuracies of LSTM across cross-validation folds.

Stage	Class Balance	Accuracy
Training	1:1	92.5
	1:50	99.7
Validation	1:1	92.4
	1:50	99.7
Testing	1:1	93.0
	1:50	99.7

Table 4.11: Performance metrics of LSTM on validation and test sets.

Stage	Class Balance	AUC	Detection Rate	FAR	Average Precision
Validation	1:1	96.6	89.8	4.9	92.6
	1:50	96.9	84.9	0.02	99.2
Testing	1:1	96.7	89.4	3.4	93.2
	1:50	96.2	84.3	0.02	99.2

4.2.6 Bayesian Network

The Bayesian Network model results for multi-step attack prediction, for the model defined in Section 3.3.2, are presented. Table 4.12 shows mean accuracies over five cross-validation folds, with testing accuracies of 40.8% (1:1) and 65.7% (1:50). Table 4.13 provides performance metrics (AUC, DR, FAR, AP) for validation and testing stages.

Table 4.12: Mean accuracies of Bayesian Network across cross-validation folds.

Stage	Class Balance	Accuracy
Training	1:1	40.8
	1:50	65.7
Validation	1:1	40.8
	1:50	65.7
Testing	1:1	40.8
	1:50	65.7

Table 4.13: Performance metrics of Bayesian Network on validation and test sets.

Stage	**Class Balance**	**AUC**	**Detection Rate**	**FAR**	**Average Precision**
Validation	1:1	33.4	78.5	96.9	28.9
	1:50	64.2	71.0	34.4	51.6
Testing	1:1	35.4	82.7	96.8	30.7
	1:50	63.6	70.0	33.7	51.6

5

Discussion

This chapter compares and discusses the results presented in Chapter 4. It also discusses the limitations, future work, and ethical considerations related to this thesis.

5.1 Comparing Results

This section discusses the performance obtained from each model presented in Chapter 4 and discusses possible interpretations of these results.

5.1.1 Comparing model performance

The Markov Chain model was used as a baseline model to see if the data was simple enough for a basic Markov Chain model to match or outperform more complex machine learning models. Table 3.4 shows that the normal sequences consist of a few and relatively simple sequences, while Table 3.5 shows that the attack sequences are diverse and more uniformly distributed. The Bayesian Network model is used to enable comparison with the Bayesian Network model used in the ACE-WARP project [8]. HMM, LSTM, and CNN were chosen for their ability to capture complex sequential patterns, and due to their proven success in sequence classification tasks as mentioned in Section 2.13.4.

5.1.2 Model Architectures

Table 4.1 summarizes all models. Notably, all three machine learning models (HMM, LSTM, and CNN) outperform the baseline Markov Chain model, indicating that machine learning models perform better than simple Markov Chain models on this dataset. However, the Markov Chain still performs better than the Bayesian Network. If we look closer, the metrics for the Markov Chain are good for the balanced (1:1) class setting; however, the false alarm rate is higher than the machine learning models. In the imbalanced (1:50) class settings, the Markov Chain model has a low average precision compared to the machine learning models. The Markov Chain consistently outperforms the Bayesian model across all metrics.

The Bayesian Network's low performance on this dataset can be due to several factors. The dataset contains several loops, which are not compatible with Bayesian Networks,

and these loops decrease its accuracy compared to other models, that can model loops. The Bayesian Network also struggles to identify sequential dependencies since the sequences are of only size two, and these models are more suited for capturing dependencies in longer acyclic sequences. The Bayesian Network model has a low AUC, indicating that it cannot distinguish between the normal and the attack sequences.

Among the machine learning models, the HMM struggles the most with average precision under imbalanced (1:50) class conditions. However, for balanced (1:1) settings, it performs nearly as well as the LSTM and CNN. The HMM also outperforms the Markov Chain by achieving approximately half the already low false alarm rate. Overall, the LSTM and CNN models deliver the best performance across all models under imbalanced (1:50) class conditions, and maintain strong performance under balanced (1:1) class conditions, except for the higher false alarm rate. But their false alarm rate is still lower than the Markov Chains.

5.1.3 Class Balances

The Markov Chain and HMM models performed better under balanced (1:1) class conditions than under imbalanced (1:50) class conditions. Meanwhile, the LSTM, CNN, and Bayesian Network models had higher performance under imbalanced (1:50) class settings compared to the balanced (1:1) version.

The ratios are picked to represent one balanced and one imbalanced class ratio, but other ratios could also be explored to simulate other real-world settings, where attack sequences occur at different ratios to normal sequences. This thesis uses the same class balances as Kawawa-Beaudan et al. [37], but other ratios such as 1:100 or even 1:4 could be used. Recent analysis of real Security Operations Center (SOC) network logs from 2018-2022 indicates that, at least in this setting, alerts have a 1:100 balance of false alarms to true (successful) attacks, or 1:4 for attack attempts [78].

5.1.4 Preprocessing

The preprocessing of the data was designed to preserve the dataset's realism, meaning the data maintains its original characteristics and distributions as closely as possible, to make the models compatible with future applications.

Appending an end token to extend all sequences was preferred over removing all single-step sequences, which corresponds to 47% of the normal sequences. This was done before performing the sliding window. The sliding window was applied with a size of 2, since using a larger window size would remove all sequences shorter than the given window size, which corresponds to a significant portion of the dataset. Another advantage of using a small window size is that multi-step attacks can be detected from only two-step long sequences, making it possible to stop the attack before larger portions of the attack have been executed.

The dataset is upsampled by duplication to keep it as realistic as possible, meaning the original data distribution is preserved without introducing artificially generated

or altered sequences. Oversampling with randomness may create new sequences that are not realistic or representative of actual behaviors. Each unique sequence is upsampled equally to maintain the original proportions.

5.2 ACE-WARP Comparison

The ACE-WARP project has a different use case than this thesis, leading to different design choices when comparing this thesis to their project.

5.2.1 Preprocessing

The final dataset used differs from ACE-WARP's, since they instead downsample normal alerts by filtering out those with more than 80% similarity, and then upsample all attack alerts by duplication. This approach is similar to our approach, with two different class balances. However, this thesis does not downsample any normal sequences. Similarly to this thesis, they apply a sliding window of size two on all sequences.

5.2.2 Bayesian Network Model and Risk Calculation Formula

The accuracy of only the ACE-WARP's Bayesian Network model was never documented in their paper, and their model was also trained to predict the next step in a sequence, rather than performing binary classification. The model is not publicly available, and all of this makes comparisons challenging.

By developing and evaluating a Bayesian Network Model, this thesis presents evidence that it performed poorly at binary classification on this dataset, compared to the LSTM and CNN models. This might suggest that using an LSTM or CNN could improve next-step prediction in ACE-WARP's system; however, this cannot be confirmed without further testing. The ACE-WARP project integrates its Bayesian Network model into a risk calculation formula, with additional parameters, before initiating mitigation. In comparison, this thesis shows that using an LSTM or CNN model for binary classification of sequences could be a simpler alternative to next-step prediction and risk calculation. But it also depends on the use case, since ACE-WARP's solution also takes the asset value of the pod into the risk calculation formula, something which is not needed (or well-defined) in our use case.

5.2.3 Different Mitigation

In Kubernetes environments, pod compromise poses a significant security risk, requiring swift and effective mitigation strategies. ACE-WARP's mitigation strategy is to reallocate all pods (migrate) connected to the pod being compromised pod. The process of performing this action is time-consuming in a pod cluster with thousands of pods. First, ACE-WARP's algorithm must identify viable options for moving the affected pods without disrupting service availability. Then the pods, which will be

moved, have to be terminated and scheduled at other locations. The moved pods then receive new IP addresses, and all service endpoints need to be updated to map to the new pods. Given these steps, this mitigation solution is likely costly and complex to implement at scale, and will introduce delays and operational overhead that could impact cluster performance and availability.

5.2.4 The overall ACE-WARP System

ACE-WARP's overall system can successfully detect and mitigate up to 81% of the attacks with an average of 39% false positives. In the context of their project, false positives do not matter in their use case because their solution does not cause service disruption on mitigation. However, in the use case in this thesis, a low false positive and a low false alarm rate is crucial, as the mitigation approach is not suitable for our use case, since the target consists of a significantly larger Kubernetes cluster.

5.3 Mitigation Strategy

This thesis has shown that LSTM and CNN models deliver a high attack detection rate and a low false alarm rate, which is crucial for multi-step attack detection. After an attack has been identified by the model, a mitigation can be initiated to stop the attack. These mitigation strategies should be suitable for a system with a high detection rate and a low false alarm rate. Two alternatives are pod deletion and pod sandboxing.

Pod Deletion

To prevent attackers from escalating privileges or moving laterally within the Kubernetes cluster, the system mitigates threats by deleting the compromised pod. This ensures immediate isolation of the attack and prevents further infiltration within the cluster. However, this approach disrupts the service running in the pod, requiring additional recovery steps.

Although effective, deleting pods can introduce service availability issues. Loss of ongoing requests may occur if the deleted pod was handling active requests, causing failures for users. Additionally, if critical services are terminated, system performance might decrease. False positives may also result in unnecessary pod deletion, disrupting legitimate services.

To mitigate these risks, the system needs to ensure graceful recovery by automatically spawning new pods after deletion to minimize downtime. Kubernetes service discovery redirects traffic to healthy pods, and a threshold-based mitigation approach is applied to reduce false positives by only triggering deletion for sequences exceeding a high-risk threshold.

Pod Sandboxing

To prevent lateral movement while allowing further investigation, an alternative mitigation strategy is sandboxing the compromised pod. In this approach, the pod is disconnected from the cluster, effectively isolating it from other services while maintaining its internal state. This prevents attackers from interacting with other pods, limiting the spread of an attack while enabling forensic analysis of the compromised instance.

However, unlike deletion, sandboxing preserves the pod's state, allowing security teams to analyze the vulnerability, investigate the attack, and develop targeted mitigation strategies. This approach prevents further lateral movement while providing valuable insights into the attack, whereas outright deletion only stops the attacker temporarily, which means that the attacker can target the vulnerability again if the pod is available.

Repeated Pod Deletion Monitoring Strategy

Attackers might try to trick the system by causing repeated alerts that lead to continuous pod deletions. This can hurt service availability or overload the system. For example, an attacker might compromise a pod, such as a User Plane Function in a 5G core network, and start processes that break Falco rules, like opening an unauthorized shell. Each violation creates an alert. If the alert is classified as an attack, the pod gets deleted. Kubernetes will then automatically respawn the pod through its ReplicaSet feature [21].

However, if the original problem, such as a misconfiguration, is still present, the attacker can use the same trick on the new pod again. This creates a loop where the attacker keeps triggering alerts and the system keeps deleting and respawning pods.

Monitoring deletions is one way to stop this. The Threat Moderator can keep track of how many times each pod has been deleted over a set time period, like 10 minutes. If a pod is deleted too many times, the system can raise a flag and send the issue to a security administrator. Since the LSTM and CNN models have a very low false alarm rate (0.02%, as shown in Table 4.1), this kind of alert should only happen in serious cases.

Extra information such as the pod's container ID, namespace, and which Falco rule was triggered, can be saved to help with investigation. The system can also note which MITRE ATT&CK tactic was matched. This makes it easier to figure out what went wrong and how to fix it.

Tradeoffs Between Pod Deletion and Pod Sandboxing

Both pod deletion and pod sandboxing aim to stop attackers by isolating compromised pods from the rest of the Kubernetes cluster. However, they differ in how they affect service availability, investigation capabilities, and system resources. Below is a comparison of their main tradeoffs.

Pod Deletion immediately removes the compromised pod, cutting off attacker

access. This is especially important for pods handling sensitive data, such as the Authentication Server Function. In large 5G systems, where services are often replicated across many pods, deleting one pod has minimal impact since Kubernetes quickly redirects traffic to healthy pods and respawns replacements. Deletion also frees up system resources, which helps during attacks that aim to exhaust CPU or memory.

However, deletion comes with drawbacks. It can disrupt services, especially if the pod was actively handling requests or running a unique function. Although the false alarm rate is low, rare false positives can still affect legitimate users. Another downside is the loss of forensic data when the pod is removed. Without data on what happened, it becomes very difficult to investigate what went wrong or how the attacker gained access.

Pod Sandboxing offers an alternative by isolating the pod from the network while keeping it alive. This allows security teams to analyze logs, processes, or system calls, which can help improve rules and patch vulnerabilities. It also avoids immediate service disruption, giving time for other healthy pods to take over.

The downside of sandboxing is that the pod remains in memory. If it contains sensitive information, this could pose a risk if the isolation isn't complete or if the response is delayed. Sandboxed pods also continue using cluster resources, which can become problematic at scale.

5.4 Industrial Use

A complete security solution is sought after in the industry to enhance runtime security in Kubernetes clusters. LSTM or CNN models can be used for binary classification of sequences and complemented with a suitable automatic mitigation strategy. Having a low false alarm rate is crucial for an attack-detecting system to prevent fatigue among security administrators. Intrusion detection systems usually aim for a false alarm rate below 3% which is accomplished in this thesis for both LSTM and CNN under the imbalanced data scenario (1:50), as displayed in Table 4.1. The detection rate is also important for a reliable system, and both the LSTM and CNN models offer a high detection rate under imbalanced conditions (1:50). While ACE-WARP's mitigation strategy may not cause service disruption in a small cluster, reallocating pods in a large cluster can often cause some temporary service disruption, both due to the difficulty of the algorithm finding where to move the affected pods, and all connections that have to be re-established after pods have been moved. Therefore, their mitigation strategy is not suitable for our industrial use case, and that is why this thesis proposes either deleting the pod or sandboxing it as two alternatives for further analysis.

The models developed in this thesis are designed to be as compatible as possible for future applications, but further research needs to be conducted to see how well the models perform in real applications. The evaluated false alarm rate and detection rate are promising, but will most likely perform differently when integrated into a real application, due to new sequences they will observe. Design choices such as

appending END to each sequence may cause the model to have lower precision, but further research needs to be done to draw any conclusions.

5.5 Limitations

To scope this thesis, multiple limitations had to be set. First of all, it would be unfeasible to make a multi-step attack prediction model for all possible types of attacks. Instead, efforts are made through predicting specific multi-step attacks within a Kubernetes environment, and only attacks that Falco can detect the individual steps of.

The machine learning models this thesis develops are restricted to 5G core applications with the same Falco ruleset that ACE-WARP used during dataset generation. The model can only identify the multi-step attacks it has been trained on, which are displayed in Table 3.1.

5.6 Future work

Expanding the Dataset

The dataset that the thesis uses is limited to specific attack types (see Table 3.1) and normal behaviors. Future work could expand on this dataset by adding data from different Kubernetes deployments, such as those running different applications or deployed in different cloud environments. Additionally, adding a wider range of attacks to the dataset would enable a more complete training and evaluation of the models, which will enhance their ability to generalize if implemented in new environments. With an expanded dataset, other models, such as transformer models, could also be explored.

Expanding Attack Detection

To enhance the system's proactive mitigation capabilities, we propose developing a tool that analyzes system and network data to generate predictive Falco rules. These rules can then be integrated directly into the system to protect against possible undetected vulnerabilities.

Intrusion Detection System

Our solution currently detects attacks by analyzing alerts fed into our models, focusing on predefined attack patterns and Falco rules. To enhance detection capabilities, we propose developing an intrusion detection system that establishes a baseline of normal activity within Kubernetes environments. This system will identify attacks as activity deviating from normal activity. By following this approach, the system could detect previously unseen or zero-day attacks that do not match known attack signatures.

Mitigation Stratigies

The mitigation strategies proposed in this thesis have not yet been empirically tested. Nonetheless, an analysis of their expected behavior introduces certain limitations. For instance, the deletion of a compromised pod may temporarily disrupt the attack, but it does not mitigate it completely, especially when the pod is automatically respawned by the Kubernetes controller. This delays the adversary's activity without addressing the root cause, allowing the attacker to persist or re-initiate the attack again once the pod is restored. Future work could explore alternatives such as isolating compromised pods and redirecting traffic to backup instances.

Enterprise Evaluation

The models and mitigation strategies proposed in this thesis have only been tested with simulated data and have not yet been fully implemented in an enterprise setting. Future work could involve evaluating these models in simulated test environments to analyze their performance across various workloads and attack scenarios. Deploying them in enterprise Kubernetes clusters with diverse configurations would offer insights into resource efficiency and compatibility, supporting further development for real-world applications.

5.7 Ethical Considerations

Training and testing data from Falco logs may include sensitive operational details about the Kubernetes cluster, such as system calls or pod interactions. Mishandling this data could expose important information or user activities. The ethical obligation remains to ensure that deployment in industrial settings continues to protect privacy through secure data pipelines and following local standards.

5.8 Sustainability

The proactive security solution developed in this thesis for Kubernetes clusters improves the reliability, efficiency, and ultimately longevity of cloud-native systems. This provides better structuring for critical applications like 5G core networks and increases security.

Environmental Sustainability

Cloud computing and 5G networks are heavy consumers of computational power, given that Kubernetes is used for container orchestration within these. In 2024, data centers consumed approximately 1.5% of global electricity, and this number is projected to double by 2030, largely driven by AI activities [70]. The proposed approach minimizes risks arising from prolonged malicious activities by early detection and mitigation of multistep attacks. These malicious activities may include cryptojacking or denial-of-service attacks that may unnecessarily drain computing

resources and consume more energy. As an example, preventing the exploitation of cluster resources by deleting or sandboxing compromised pods (see Section 5.3).

Operational Sustainability

The proposed solution offers a sustained operational environment by preventing service interruptions and increasing cluster security in Kubernetes. A false positive, for example, may lead to pod deletion, which causes temporary disruptions in the services (see Section 5.3). However, considering the very low FAR achieved by the LSTM and CNN models, such as 0.02% for imbalanced 1:50 cases (see Table 4.1), false positives are low enough to ensure fewer disruptions, thus maintaining the availability of the service. Such matters are critical in 5G networks, which need low latency and high availability for applications such as autonomous vehicles and industrial automation.

6

Industrial Implementation

In this section, a complete security solution within the 5G core network is proposed, to integrate both the multi-step attack detection model and its mitigation strategies. The proposed architecture is cloud-native and microservice-based to ensure modularity, scalability, and industry compatibility. It aligns with current industry standards, protocols, and best practices to facilitate integration into real-world operational environments. Further, a test environment is described for validating the solution in the simulated 5G network to ensure the applicability to various industrial use cases.

6.1 Security Solution for Industrial Use

The integration of models for detecting multi-step attacks with associated mitigation strategies will be done in a microservice so that it may be compatible within 5G core networks. A microservice is a lightweight software component, isolated in terms of responsibility, and communicating with others via a network interface, thereby being modular and scalable in large distributed systems [55]. Since 5G Core deployments consist mainly of large Kubernetes clusters with several pods, such microservices must be capable of reliably scaling and processing data with minimal latency to ensure real-time attack detection and mitigation along a variety of network functions, some of which are described in Appendix A.1 and Figure A.1.

The microservice package is containerized using Docker to ensure that it works across different environments, from on-premises data centers to public or private clouds. Inside the container, the detection model will be deployed, along with its dependencies and a lightweight runtime environment to reduce overhead costs and provide a consistent performance across different platforms.

A RESTful API will be implemented to act as an interface towards the actual microservice built on the Flask framework for integration with 5G core components [61]. Through the interface, the API will provide endpoints for receiving security events (such as those raised by Falco), processing them through the attack detection model, and then triggering mitigation actions such as pod isolation or pod sandboxing. For example, an endpoint (e.g., `/detect`) will accept JSON payloads containing alert data, detect if it is an attack, and then start mitigation actions if needed, see Figure 6.1.

Retraining of the detection model with a custom Falco ruleset tailored for the target

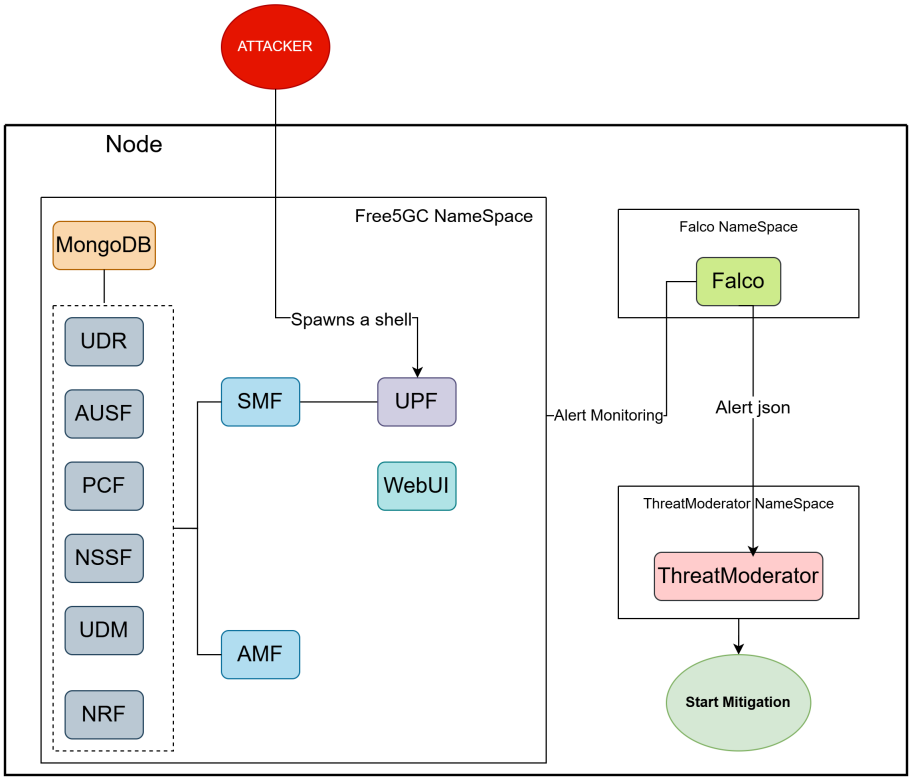

Figure 6.1: An Industrial Microservice, depicted as "Threat Moderator" in the figure and implemented with Free5GC network. This example illustrates how an attacker spawns a shell in a pod, how Falco detects the suspicious activity, and how the Threat Moderator acts upon it.

environment will be necessary to allow adaptation for a particular 5G deployment. The ruleset will detect suspicious behaviors such as unauthorized system calls or illegal network traffic specific to 5G network functions. With retraining, the model will be able to detect threats with fewer false positives and become more accurate overall.

6.2 Test Environment

To validate the service functionality, a test environment replicating the 5G core architecture will be deployed with Minikube, a lightweight tool that runs Kubernetes clusters on a single machine [44]. The environment is set up using the Free5GC Helm chart [24], which includes several important 5G core network functions such as the Access and Mobility Management Function (AMF), Session Management Function (SMF), as well as User Plane Function (UPF), as described in Appendix A.1. This allows the description of the setup to be a functional simulation of an actual 5G network as it includes a set of pods representing the network functions and their interactions, presenting a real testing environment for the solution.

Cloud-native runtime security tool Falco will be deployed with its official Helm chart, and monitor system calls and container events within the Minikube cluster [5]. Falco

will then issue alerts in real time whenever suspicious behavior occurs, like escalating privileges, accessing files without being authorized, which the microservice decides if it is an attack. The test environment will execute simulations of attack scenarios like lateral movements or exfiltration to determine the solutions' detection accuracy and mitigation response time.

The test environment will simulate high workloads to test scalability, so the microservice is capable of handling large-scale 5G core deployments. It will collect performance metrics such as alert processing latency and mitigation response time to verify the requirements of the ultra-low latency 5G core.

7

Conclusion

This Master's thesis develops and evaluates a proactive security solution for Kubernetes clusters with detection mechanisms and automated mitigation methods against multi-step attacks. This method can better protect cloud-native environments, especially 5G core network environments. Using a labeled dataset of Falco alerts generated by the ACE-WARP project, we perform comparative evaluations of five statistical and machine learning models. Comparisons were performed between Markov chain, HMM, LSTM, CNN, and Bayesian network for binary classification of alert sequences into normal or attack. The following conclusions address the research questions posed in Section 1.3:

1. **Which model provides the highest detection rate?**

   The LSTM and CNN models achieved the highest detection rate of 84.3% under imbalanced (1:50) conditions, as shown in Table 4.1, effectively identifying multi-step attacks in Kubernetes clusters.

2. **Which model provides the lowest false alarm rate ?**

   The LSTM and CNN models demonstrated the lowest FAR of 0.02% under imbalanced (1:50) conditions, minimizing unnecessary service disruptions.

3. **How does class balance affect the accuracy of the models?**

   Imbalanced settings (1:50) affected the models' performances negatively, but LSTM and CNN maintained high accuracy (AUC-ROC of 96.2%) and precision (AP of 99.2%), while simpler models like Markov Chain and HMM showed reduced performance in imbalanced settings.

4. **Which model can be best used for multi-step attack detection?**

   The LSTM and CNN models provide the most suited balance, with a detection rate of 84.3% and a FAR of 0.02% in imbalanced settings, outperforming Markov Chain, HMM, and Bayesian Network models.

5. **Can the multi-step attack detection be integrated into a complete security solution for industrial use?**

   Yes. The LSTM and CNN models, along with mitigation strategies such as pod deletion or sandboxing, can be containerized into a microservice using Docker and exposed using a RESTful API. As explained in Chapter 6, this design enables scalability and ensures compatibility with 5G core networks.

The results of this thesis demonstrate the effectivity of using LSTM and CNN models for binary classification to detect multi-step attacks on Kubernetes clusters, achieving high detection rates and low false alarm rates despite imbalanced conditions. With these models, an approach to security has been suggested such that after attack detection, immediate automatic mitigations can be executed. These mitigation strategies include pod deletion for immediate isolation of the threat and sandboxing for forensic analysis, which aims to minimize manual intervention and service disruption, while also further hardening the cluster. The microservice-based implementation, designed for industrial 5G deployments, offers scalability and adaptability, and aligns with cloud-native principles.

Bibliography

[1] 3GPP. *5G System Overview*. URL: https://www.3gpp.org/technologies/5g-system-overview.

[2] Open Policy Agent. "GateKeeper". In: *Open Policy Agent Docs* (2025). URL: https://open-policy-agent.github.io/gatekeeper/.

[3] Johannes Textor Ankur Ankan. *pgmpy: A Python Toolkit for Bayesian Networks*. 2024. URL: https://pgmpy.org.

[4] The Falco Authors. *Custom Ruleset*. Falco. Jan. 2025. URL: https://falco.org/docs/concepts/rules/custom-ruleset/.

[5] The Falco Authors. "Detect security threats in real time". In: *Falco Project* (2025). URL: https://falco.org/.

[6] The Kubernetes Authors. *Cluster Architecture*. Oct. 2024. URL: https://kubernetes.io/docs/concepts/architecture/.

[7] The Kubernetes Authors. *Viewing Pods and Nodes*. Nov. 2024. URL: https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/.

[8] Sima Bagheri, Hugo Kermabon-Bobinnec, Mohammad Ekramul Kabir, Suryadipta Majumdar, Lingyu Wang, Yosr Jarraya, Boubakr Nour, and Makan Pourzandi. "ACE-WARP: A Cost-Effective Approach to Proactive and Non-Disruptive Incident Response in Kubernetes Clusters". In: *IEEE Trans. Inf. Forensics Secur.* 19 (2024), pp. 8204–8219. DOI: 10.1109/TIFS.2024.3449038. URL: https://doi.org/10.1109/TIFS.2024.3449038.

[9] Sima Bagheri, Hugo Kermabon-Bobinnec, Suryadipta Majumdar, Yosr Jarraya, Lingyu Wang, and Makan Pourzandi. "Warping the Defence Timeline: Non-disruptive Proactive Attack Mitigation for Kubernetes Clusters". In: *ICC 2023-IEEE International Conference on Communications*. IEEE. 2023, pp. 777–782.

[10] David Bernstein and Stephen Ludwig. *Container Security: Fundamental Technology Concepts that Protect Containerized Applications*. Sebastopol, CA: O'Reilly Media, 2024. ISBN: 978-1098118211.

[11] Andrew P. Bradley. "The use of the area under the ROC curve in the evaluation of machine learning algorithms". In: *Pattern Recognit.* 30.7 (1997), pp. 1145–1159. DOI: 10.1016/S0031-3203(96)00142-2. URL: https://doi.org/10.1016/S0031-3203(96)00142-2.

[12] Vaughan Carey. *Manual vs Automated Alert Triage In Security Operations*. CloudGuard. June 2024. URL: https://cloudguard.ai/resources/automated-alert-triage/.

[13] OpenStack Congress. *The Most Widely Deployed Open Source Cloud Software in the World*. 2025. URL: http://www.openstack.org/.

[14] Wei Lien Dang. *Cryptojacking Attacks in Kubernetes: How to Stop Them*. Red Hat, June 2020. URL: https://www.redhat.com/zh-tw/blog/cryptojacking-attacks-in-kubernetes-how-to-stop-them.

[15] Daniel E. Capano. *Throwback Attack: Lessons from the Aurora vulnerability*. Apr. 2021. URL: https://www.controleng.com/throwback-attack-lessons-from-the-aurora-vulnerability/.

[16] National Vulnerability Database. *CVE-2020-14386*. Nov. 2024. URL: https://nvd.nist.gov/vuln/detail/CVE-2020-14386/.

[17] Docker. *Docker Engine Security: seccomp*. 2023. URL: https://docs.docker.com/engine/security/seccomp/.

[18] Docker. *Use containers to Build, Share and Run your applications*. 2025. URL: https://www.docker.com/resources/what-container/.

[19] Qingfeng Du, Tiandi Xie, and Yu He. "Anomaly Detection and Diagnosis for Container-Based Microservices with Performance Monitoring". In: *Algorithms and Architectures for Parallel Processing - 18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part IV*. Ed. by Jaideep Vaidya and Jin Li. Vol. 11337. Lecture Notes in Computer Science. Springer, 2018, pp. 560–572. DOI: 10.1007/978-3-030-05063-4_42. URL: https://doi.org/10.1007/978-3-030-05063-4%5C_42.

[20] Ericsson. *Unleash the full potential of 5G with cloud native*. 2025. URL: https://www.ericsson.com/en/cloud-native.

[21] Harold Finch. *Kubernetes Pod Gets Recreated When Deleted*. Accessed: 2025-06-16. July 2024. URL: https://medium.com/@haroldfinch01/kubernetes-pod-gets-recreated-when-deleted-a7a8eea832c8.

[22] FireEye. *APT37 (REAPER), The Overlooked North Korean Actor*. Jan. 2018. URL: https://services.google.com/fh/files/misc/apt37-reaper-the-overlooked-north-korean-actor.pdf.

[23] FireEye. *FireEye Advanced Threat Report: 2013*. 2014. URL: https://www.ismsforum.es/ficheros/descargas/fireeye-advanced-threat-report-20131395657540.pdf.

[24] Free5GC. *Free5GC Helm Chart for 5G Core Deployment*. 2023. URL: https://github.com/free5gc/free5gc-helm.

[25] Free5GC. *Open Source Core Network Implementation*. May 2025. URL: https://free5gc.org/.

[26] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. 3rd. Chapman and Hall/CRC, 2013.

[27] Grandmetric. *5G Core Network (5GC) Functions*. URL: https://www.grandmetric.com/5g-core-network-functions/.

[28] S. Ben Hai. "Climbing the Ladder: Kubernetes Privilege Escalation (Part 1)". In: *SentinelOne Blog* (Oct. 2024). URL: https://www.sentinelone.com/blog/climbing-the-ladder-kubernetes-privilege-escalation-part-1/.

[29] Neeraja Hariharasubramanian. *What is Anomaly Based Detection System*. Jan. 2025. URL: https://fidelissecurity.com/cybersecurity-101/learn/anomaly-based-detection-system/.

[30] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy". In: *Nature* (Sept. 2020). DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[31] Wajih Ul Hassan, Adam Bates, and Daniel Marino. "Tactical Provenance Analysis for Endpoint Detection and Response Systems". In: *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 1172–1189. DOI: 10.1109/SP40000.2020.00096. URL: https://doi.org/10.1109/SP40000.2020.00096.

[32] Wajih Ul Hassan, Mohammad A. Noureddine, Pubali Datta, and Adam Bates. "OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis". In: *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020. URL: https://www.ndss-symposium.org/ndss-paper/omegalog-high-fidelity-attack-investigation-via-transparent-multi-layer-log-analysis/.

[33] hmmlearn. *hmmlearn: Hidden Markov Models in Python*. Jan. 2010. URL: https://hmmlearn.readthedocs.io.

[34] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[35] E. M. Hutchins, M. J. Cloppert, and R. M. Amin. *Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains*. Lockheed Martin, Jan. 2011. URL: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf.

[36] Kata Containers. *The speed of containers, the security of VMs*. 2024. URL: https://katacontainers.io/.

[37] Maxime Kawawa-Beaudan, Srijan Sood, Soham Palande, Ganapathy Mani, Tucker Balch, and Manuela Veloso. "Ensemble Methods for Sequence Classification with Hidden Markov Models". In: *CoRR* abs/2409.07619 (2024). DOI: 10.48550/ARXIV.2409.07619. URL: https://doi.org/10.48550/arXiv.2409.07619.

[38] Keras. *Keras: Deep Learning for Humans*. June 2025. URL: https://keras.io.

[39] Hugo Kermabon-Bobinnec, Mahmood Gholipourchoubeh, Sima Bagheri, Suryadipta Majumdar, Yosr Jarraya, Makan Pourzandi, and Lingyu Wang. "ProSPEC: Proactive Security Policy Enforcement for Containers". In: *CODASPY '22: Twelveth ACM Conference on Data and Application Security and Privacy, Baltimore, MD, USA, April 24 - 27, 2022*. ACM, 2022, pp. 155–166. DOI: 10.1145/3508398.3511515. URL: https://doi.org/10.1145/3508398.3511515.

[40] Medha Khenwar and Meenakshi Nawal. "Challenges and Limitations of IDS: A Comprehensive Assessment and Future Perspectives". In: *SKIT Research Journal* Vol 14 (July 2024), pp. 35–39. DOI: 10.47904/IJSKIT.14.1.2024.35-39.

[41] Wesley Chai Kinza Yasar and Stephen J. Bigelow. *What is cloud computing? Types, examples and benefits*. Jan. 2025. URL: https://www.techtarget.com/searchcloudcomputing/definition/cloud-computing.

[42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[43] Kubernetes. *Kubernetes Documentation*. 2024. URL: https://kubernetes.io/.

[44] Kubernetes. *Minikube: Run Kubernetes Locally*. 2023. URL: https://minikube.sigs.k8s.io/docs.

[45] Kubernetes Authors. *Kubernetes Documentation: Security*. Feb. 2025. URL: https://kubernetes.io/docs/concepts/security/overview/.

[46] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proc. IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791. URL: https://doi.org/10.1109/5.726791.

[47] Suryadipta Majumdar, Gagandeep Singh Chawla, Amir Alimohammadifar, Taous Madi, Yosr Jarraya, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. "ProSAS: Proactive Security Auditing System for Clouds". In: *IEEE Trans. Dependable Secur. Comput.* 19.4 (2022), pp. 2517–2534. DOI: 10.1109/TDSC.2021.3062204. URL: https://doi.org/10.1109/TDSC.2021.3062204.

[48] Suryadipta Majumdar, Yosr Jarraya, Taous Madi, Amir Alimohammadifar, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. "Proactive Verification of Security Compliance for Clouds Through Pre-computation: Application to OpenStack". In: *Computer Security – ESORICS 2016*. Cham: Springer International Publishing, 2016, pp. 47–66. ISBN: 978-3-319-45744-4.

[49] Suryadipta Majumdar, Yosr Jarraya, Momen Oqaily, Amir Alimohammadifar, Makan Pourzandi, Lingyu Wang, and Mourad Debbabi. "LeaPS: Learning-Based Proactive Security Auditing for Clouds". In: *Computer Security – ESORICS 2017*. Ed. by Simon N. Foley, Dieter Gollmann, and Einar Snekkenes. Cham: Springer International Publishing, 2017, pp. 265–285. ISBN: 978-3-319-66399-9.

[50] Wes Mckinney. *pandas: a Foundational Python Library for Data Analysis and Statistics*. Jan. 2011. URL: https://www.researchgate.net/publication/265194455_pandas_a_Foundational_Python_Library_for_Data_Analysis_and_Statistics.

[51] MITRE. *APT 29*. Apr. 2025. URL: https://attack.mitre.org/groups/G0016/.

[52] MITRE. *APT 3*. Nov. 2024. URL: https://attack.mitre.org/groups/G0022/.

[53] MITRE. *Enterprise Tactics - MITRE ATT&CK*. 2025. URL: https://attack.mitre.org/tactics/enterprise/.

[54] Stamus Networks. *Cybersecurity Alert Fatigue*. URL: https://www.stamus-networks.com/cybersecurity-alert-fatigue.

[55] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015. ISBN: 9781491950357.

[56] Pierluigi Paganini. *Phishing campaigns target US government agencies exploiting Hacking Team flaw CVE-2015-5119*. July 2015. URL: https://securityaffairs.com/38707/cyber-crime/phishing-cve-2015-5119.html.

[57] Krishna G Pai. *Container Escapes: An Executive Guide to Mitigating Container Security Risks*. 2025. URL: https://krishnag.ceo/blog/container-escapes-an-executive-guide-to-mitigating-container-security-risks/.

[58] Advait Patel. *Docker Security in 2025: Best Practices to Protect Your Containers From Cyberthreats*. Feb. 2025. URL: https://cloudnativenow.com/features/docker-security-in-2025-best-practices-to-protect-your-containers-from-cyberthreats/.

[59] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (Jan. 2011), pp. 2825–2830. DOI: 10.5555/1953048.2078195. URL: https://dl.acm.org/doi/10.5555/1953048.2078195.

[60] David M. W. Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: *CoRR* abs/2010.16061 (2020). URL: https://arxiv.org/abs/2010.16061.

[61] Pallets Projects. *Flask Documentation*. Accessed: March 26, 2025. 2025. URL: https://flask.palletsprojects.com/en/stable/.

[62] Abdul Qadeer. *Scaling Kubernetes to Over 4k Nodes and 200k Pods*. Jan. 2022. URL: https://medium.com/paypal-tech/scaling-kubernetes-to-over-4k-nodes-and-200k-pods-29988fad6ed.

[63] L.R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286. DOI: 10.1109/5.18626. URL: https://doi.org/10.1109/5.18626.

[64] Kimberly Goody Rakesh Sharma Akhil Reddy. *CVE-2017-10271 Used to Deliver CryptoMiners: An Overview of Techniques Used Post-Exploitation and Pre-Mining*. Feb. 2018. URL: https://cloud.google.com/blog/topics/threat-intelligence/cve-2017-10271-used-deliver-cryptominers-overview-techniques-used-post-exploitation-and-pre-mining/.

[65] Cedrick Ramos. *Spam Campaigns with Malware Exploiting CVE-2017-11882 Spread in Australia and Japan*. Trend Micro. Oct. 2017. URL: https://www.trendmicro.com/vinfo/us/threat-encyclopedia/spam/3655/spam-campaigns-with-malware-exploiting-cve201711882-spread-in-australia-and-japan/.

[66] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. 2nd. Springer, 2004. ISBN: 978-1-4419-1939-7.

[67] Sheldon Ross. "Markov Chains". In: *Introduction to Probability Models (Eleventh Edition)*. Ed. by Sheldon Ross. Eleventh Edition. Academic Press, 2014, pp. 183–276. ISBN: 978-0-12-407948-9. DOI: https://doi.org/10.1016/B978-0-12-407948-9.00004-9. URL: https://www.sciencedirect.com/science/article/pii/B9780124079489000049.

[68] Thomas J. Sargent and John Stachurski. *Continuous Time Markov Chains: The Markov Property*. QuantEcon. 2025. URL: https://continuous-time-mcs.quantecon.org/markov_prop.html.

[69] Scale Computing. *Container Virtualization Explained: The Future of Scalable IT Infrastructure*. Dec. 2024. URL: https://www.scalecomputing.com/resources/container-virtualization-explained.

[70] Scientific American. "Data Centers Will Use Twice as Much Energy by 2030—Driven by AI". In: (Apr. 2025). URL: https://www.scientificamerican.com/article/ai-will-drive-doubling-of-data-center-energy-demand-by-2030/.

[71] Symantec. *The Nitro Attacks: Stealing Secrets from the Chemical Industry*. 2011. URL: https://scadahacker.com/library/Documents/Cyber_Events/Symantec%20-%20The%20Nitro%20Attacks.pdf.

[72] Sysdig. *12 container image scanning best practices*. 2025. URL: https://sysdig.com/learn-cloud-native/12-container-image-scanning-best-practices/.

[73] Sysdig. "Sysdig". In: *Sysdig Blog* (2025). URL: https://sysdig.com.

[74] Sysdig. *Sysdig 2023 Cloud-Native Security and Usage Report*. 2023. URL: https://sysdig.com/2023-cloud-native-security-and-usage-report/.

[75] The Tetragon Authors. *Tetragon: eBPF-based Security Observability and Runtime Enforcement*. 2024. URL: https://tetragon.io/.

[76] Chin-Wei Tien, Tse-Yung Huang, Chia-Wei Tien, Ting-Chun Huang, and Sy-Yen Kuo. "KubAnomaly: Anomaly detection for the Docker orchestration platform with neural network approaches". In: *Engineering Reports* 1.5 (2019), e12080. DOI: https://doi.org/10.1002/eng2.12080. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/eng2.12080.

[77] Shoujin Wang, Wei Liu, Jia Wu, Longbing Cao, Qinxue Meng, and Paul J. Kennedy. "Training deep neural networks on imbalanced data sets". In: *2016 International Joint Conference on Neural Networks, IJCNN 2016, Vancouver, BC, Canada, July 24-29, 2016*. IEEE, July 2016, pp. 4368–4374. DOI: 10.1109/IJCNN.2016.7727770. URL: https://doi.org/10.1109/IJCNN.2016.7727770.

[78] Limin Yang, Zhi Chen, Chenkai Wang, Zhenning Zhang, Sushruth Booma, Phuong Cao, Constantin Adam, Alexander Withers, Zbigniew Kalbarczyk, Ravishankar K. Iyer, and Gang Wang. *True Attacks, Attack Attempts, or Benign Triggers? An Empirical Measurement of Network Alerts in a Security Operations Center*. Ed. by Davide Balzarotti and Wenyuan Xu. 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/yang-limin.

[79] Runqing Yang, Shiqing Ma, Haitao Xu, Xiangyu Zhang, and Yan Chen. "UIScope: Accurate, Instrumentation-free, and Visible Attack Investigation for GUI Applications". In: *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020. URL: https://www.ndss-symposium.org/ndss-paper/uiscope-accurate-instrumentation-free-and-visible-attack-investigation-for-gui-applications/.

[80] Kinza Yasar and Linda Rosencrance. "advanced persistent threat (APT)". In: *TechTarget* (Dec. 2023). URL: https://www.techtarget.com/searchsecurity/definition/advanced-persistent-threat-APT.

[81] Zeljka Zorz. *Sudo vulnerability allows attackers to gain root privileges on Linux systems (CVE-2021-3156)*. Jan. 2021. URL: https://www.helpnetsecurity.com/2021/01/27/cve-2021-3156/.

[82] Zhuping Zou, Yulai Xie, Kai Huang, Gongming Xu, Dan Feng, and Darrell D. E. Long. "A Docker Container Anomaly Monitoring System Based on Optimized Isolation Forest". In: *IEEE Trans. Cloud Comput.* 10.1 (2022), pp. 134–145. DOI: 10.1109/TCC.2019.2935724. URL: https://doi.org/10.1109/TCC.2019.2935724.

A

Appendix 1

A.1 Key Components in 5G Core

The 5GC consists of modular Network Functions (NFs) that cooperate to provide connectivity, mobility, and data services. Figure A.1 illustrates these functions and their reference points, providing a visual overview of their interactions [27, 1].

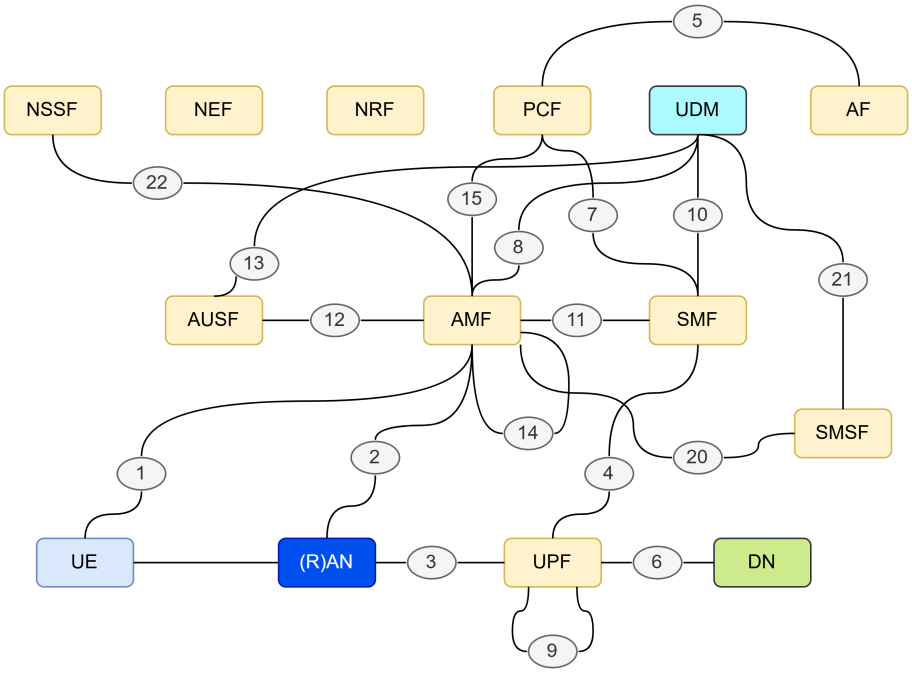

Figure A.1: 5G Core network architecture and reference points.

The 5GC relies on several NFs that collaborate to manage connectivity, mobility, and data traffic, as illustrated in Figure A.1. Key functions critical to this project are detailed below, with others summarized for context [27].

Access and Mobility Management Function. The AMF handles device registration, connection establishment, and mobility management for UE (User Equipment). It stops Non-Access Stratum signaling, performs authentication with the AUSF and UDM, and manages security during handovers, ensuring secure and seamless connectivity [27].

Session Management Function. The SMF manages data sessions, IP address allocation, QoS (Quality of Service) enforcement, and configurations for traffic steering. This functional unit thus ensures efficient data connectivity with support for downlink data notifications [27].

User Plane Function. The UPF is responsible for routing and forwarding data packets between UE and external networks, while packet inspection and QoS enforcement are performed at the UPF. It acts as a mobility anchor, but with options for low latency and high throughput data transfer [27].

Other Network Functions. The Policy Control Function offers policy settings for QoS and traffic management. The AUSF is responsible for the authentication of the UE's credentials; while the UDM is used for user profile management and authentication. As for allowing third-party integrations, that is the purpose of the Network Exposure Function; while the Network Repository Function offers support for service discovery. Together, these functions enable the security, scalability, and flexibility of 5G services [27].