Building Cross-Cluster Security Models for Edge-Core Environments Involving Multiple Kubernetes Clusters

Mahmood GholipourChoubeh

A Thesis
in
The Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Master of Applied Science (Information Systems Security) at
Concordia University
Montréal, Québec, Canada

August 2023

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Mahmood GholipourChoubeh**
Entitled: **Building Cross-Cluster Security Models for Edge-Core Environments Involving Multiple Kubernetes Clusters**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Information Systems Security)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

________________________________ Chair and Examiner
Dr. Mohsen Ghafouri

________________________________ Examiner
Dr. Jun Yan

________________________________ Co-supervisor
Dr. Lingyu Wang

________________________________ Co-supervisor
Dr. Suryadipta Majumdar

Approved by ________________________________
Dr. Jun Yan, Graduate Program Director

__________ ________________________________
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Building Cross-Cluster Security Models for Edge-Core Environments Involving Multiple Kubernetes Clusters

Mahmood GholipourChoubeh

With the emergence of 5G networks and their large scale applications such as IoT and autonomous vehicles, telecom operators are increasingly offloading the computation closer to customers (i.e., on the edge). Such edge-core environments usually involve multiple Kubernetes clusters potentially owned by different providers. Confidentiality and privacy concerns could prevent those providers from sharing data freely with each other, which makes it challenging to perform common security tasks such as security verification and attack/anomaly detection across different clusters. In this work, we propose CCSM, a solution for building cross-cluster security models to enable various security analyses, while preserving confidentiality and privacy for each cluster. We design a six-step methodology to model both the cross-cluster communication and cross-cluster event dependency, and we apply those models to different security use cases. We implement our solution based on a 5G edge-core environment that involves multiple Kubernetes clusters, and our experimental results demonstrate its efficiency (e.g., less than 8 s of processing time for a model with 3,600 edges and nodes) and accuracy (e.g., more than 96% for cross-cluster event prediction).

Acknowledgments

I would like to express my deepest gratitude to my thesis co-supervisors, Dr. Lingyu Wang and Dr. Suryadipta Majumdar. Their endless guidance and assistance significantly contributed to this journey, and their support was indispensable for its accomplishment.

I would like to extend my special thanks to my friend and fellow labmate Hugo Kermabon-Bobinnec for his invaluable assistance and support throughout this journey. Special appreciation to Dr. Yosr Jarraya for her invaluable guidance and advice.

As well as that, I would like to extend profound gratitude to my parents, brother, and sister and my dear friends, Sajjad and Behrooz for their unwavering support and love, shaping my path with their presence and encouragement.

I dedicate this thesis to my wife, Rokhsar, my unwavering source of love and inspiration. This thesis symbolizes our shared dreams and your constant support. Grateful for your presence in every step of this journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Problem Statement

Cloud and Kubernetes clusters have become standard and commonplace solutions to enable more cost-effective deployment of 5G applications. At the same time, to support large-scale applications such as IoT and autonomous vehicles, telecom operators are increasingly offloading the computation closer to customers (i.e., on the edge) in order to satisfy the latency and throughput requirements [30, 16]. While these strategies improve the overall performance and quality of service, security is often an afterthought: the distribution of workload among multiple clusters may increase the attack surface, the offloading or extending of the cloud may potentially involve less trusted or less protected edge providers. Moreover, confidentiality and privacy concerns of the providers may prevent them from sharing data freely with each other [29], particularly in scenarios where data sovereignty and cross-border data transfers are involved or when no prior trust relationship has been established.

1.2 Research Gap

There exist various security solutions for clouds and Kubernetes clusters, such as security verification [33], security impact prediction [48], and attack/breach detection [4] (a more detailed review of related work is given in Chapter 3). However, most such solutions are designed for enforcing security locally at each cluster, and they cannot be easily extended across multiple (edge and core) clusters. Furthermore, as mentioned above, the providers of those clusters would be reluctant to disclose confidential or private information about their infrastructure and users. This makes it infeasible to apply those existing security solutions on a central copy of data from all the clusters, which is necessary for many security analyses, such as verifying cross-cluster security breaches, detecting cross-cluster attacks, or predicting events across multiple clusters. To make things worse, as containers and cloud-native computing [15] are widely adopted due to their clear advantages in terms of less overhead and better performance, these also suffer from buggy images and weaker isolation compared to full-fledged VMs. For this reason, container environments and container orchestrators (such as Kubernetes) have become attractive targets of various security attacks [47]. Considering that more and more providers of critical services, including 5G core network functions [1], are moving to the cloud, addressing those limitations becomes a pressing concern.

1.3 Motivating Example

To make our discussions more concrete, we present a motivating example in the context of 5G networks. Specifically, Fig. 1 depicts a typical 5G edge-core environment, where the core cluster is owned by a mobile network operator who provides private 5G services [39, 42] to two different vertical industries, *Company 1* and *Company 2*, who own the two edge clusters. The two GDPR [41] icons attached to the two edge clusters indicate privacy concerns about leaking user information, which prevent *Company 1* and *Company 2* from sharing their data freely with the mobile network operator. On the other hand, they have to rely on the operator to provide necessary 5G core services and functionalities, including security solutions.

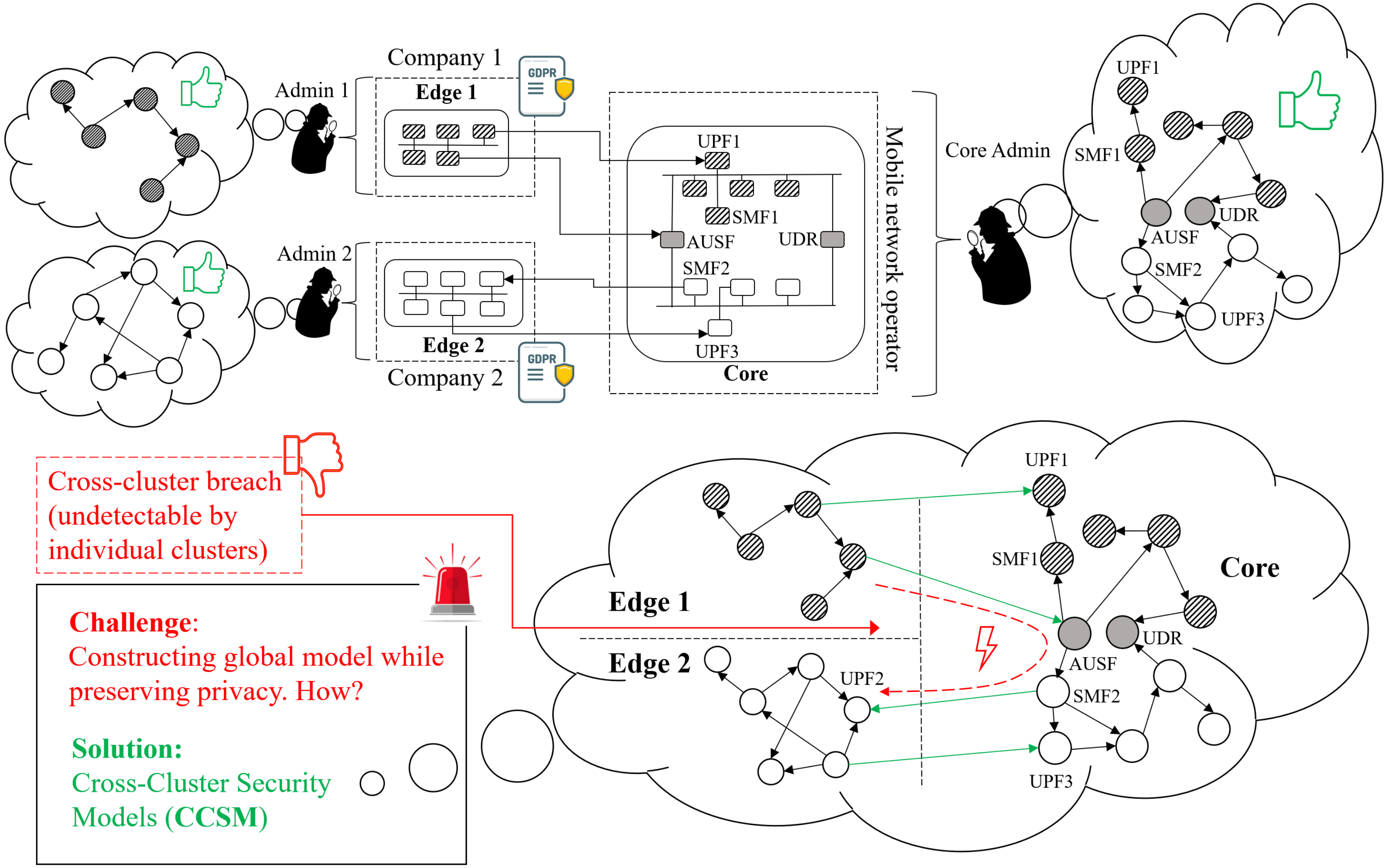


Figure 1: Motivating example.

In particular, suppose the administrators of the two edge clusters would like to verify a given security policy that their services are completely isolated from each other. For this purpose, as demonstrated by the call-outs, the administrators build graphic models about communications inside each cluster. Looking at each such local model alone, each administrator is convinced that the policy is satisfied since all the communications are limited to be between network functions inside the cluster. Similarly, the core administrator also sees nothing wrong in his/her local model (note the two network functions shown in gray color are not a concern as it is quite normal for some resources in the core to be shared [25]).

However, as demonstrated in the call-out at the bottom, relying solely on local models is insufficient to capture potential cross-cluster security breaches. Specifically, both edge administrators fail to see the fact that one of the network functions in *Edge 1* can actually reach another network function in *Edge 2* through the core (e.g., due to a misconfiguration in the AUSF and SMF2 network functions). This represents a violation of the given security policy, which cannot be identified based on the local models alone, and is only visible in a global model comprising information across all three clusters, as shown in the figure. Nonetheless, the aforementioned privacy concerns mean that such a global model cannot be trivially built by collecting data from all clusters. Instead, a solution must carefully balance between the need for building the global model to enable security solutions and the need for preserving confidentiality and privacy for each cluster.

1.4 Our solution

We propose an approach to build a Cross-Cluster Security Models (CCSM) to address the aforementioned challenge. Specifically, we design a methodology to (i) construct local security models at each edge cluster, and extend them to establish external links to other clusters, (ii) prune and anonymize the local models to avoid unnecessary information sharing before sending them to the core cluster, and (iii) construct and prune the global model at the core cluster before sending it back to each edge cluster. While this general methodology can potentially be applied to many different security models (encompass a range of models designed to portray a system's behavior from a particular perspective, intended for utilization by security mechanisms), we demonstrate such applicability through instantiating it to capture two complementary 5G security perspectives, namely, the communication among 5G network functions and the dependency among 5G events. Furthermore, we show how those security models may enable verifying cross-cluster security breaches, detecting cross-cluster attacks/anomalies, and predicting cross-cluster security impacts. As a proof of concept, we implement the solution and integrate it with Kubernetes and Free5GC (a popular open-source 5G network implementation [23]), and evaluate its effectiveness and efficiency through experiments.

1.5 Thesis Contribution

In summary, our main contributions are as follows:

- This work could enable a wide range of cross-cluster security solutions. Specifically, the proposed methodology can construct a global view of multi-cluster environments while preserving the confidentiality and privacy of each cluster. Such a global view contains important cross-cluster information (which are invisible at each cluster) that could enable various security solutions to identify cross-cluster security breaches, cross-cluster attacks/anomalies, and cross-cluster security impacts, which would otherwise be hidden if examined locally at each individual cluster.

- The proposed methodology consists of six steps for integrating clusters' local views in a confidential/private and scalable manner, and distributing the constructed global view back to each cluster on a "need to know" basis. This general methodology is instantiated to capture both the communication among 5G network functions and the dependency among 5G events, which complement each other to provide more security perspectives. To demonstrate the applicability of our solution, we describe three potential applications of our solution including security verification, attack/anomaly detection, and security impact prediction.

- To evaluate our solution, we generate the first multi-cluster dataset covering both communications among 5G core network functions and 5G control plane events in an edge-core testbed environment involving multiple inter-connected Kubernetes clusters. We implement our solution based on Privacy Set Intersection (PSI), anonymization (Crypto-PAn for communications and FF3 for events), and custom pruning techniques. We integrate our solution into Free5GC, an open-source 5G core implementation, and Kubernetes clusters. We evaluate our solution through experiments, and and our experimental results demonstrate its efficiency (e.g., less than 8 s of processing time for a model with 3,600 edges and nodes) and accuracy (e.g., more than 96% for cross-cluster event prediction).

1.6 Related Publications

Conference Paper. Our work for proactive security policy enforcement for containers has been published as an article in a peer-reviewed conference's proceedings:

ProSPEC: Proactive Security Policy Enforcement for Containers. Hugo Kermabon-Bobinnec, **Mahmood GholipourChoubeh**, Sima Bagheri, Suryadipta Majumdar, Yosr Jarraya, Makan Pourzandi and Lingyu Wang. *Proceedings of the*

Twelveth ACM Conference on Data and Application Security and Privacy (CODASPY'22), Apr 25-27, 2022, Baltimore-Washington DC Area, USA. (Acceptance ratio 30/111≈27%)

Conference Paper. Our work related to building cross-cluster security models for edge-core environments will shortly be submitted to a peer-reviewed conference:

> Building Cross-Cluster Security Models for Edge-Core Environments Involving Multiple Kubernetes Clusters. **Mahmood GholipourChoubeh**, Hugo Kermabon-Bobinnec, Suryadipta Majumdar, Yosr Jarraya and Lingyu Wang. *To be submitted at ACM ASIACCS 2024.*

1.7 Authors' Contribution

The student co-authors' contributions to the aforementioned articles are as follows:

ProSPEC. Mahmood GholipourChoubeh contributed to the experiments on learning time and model accuracy. Hugo Kermabon-Bobinnec contributed to the motivation, approach and design, implementation as well as experiments on the impact of cluster size, threshold and predictions on response time (whose corresponding sections have been excluded from this thesis).

Cross-Cluster Security Models. Mahmood GholipourChoubeh contributed to the motivation, approach and design, implementation as well as experiments on execution time, efficiency of pruning modules and performance in different use-cases. Mahmood GholipourChoubeh and Hugo Kermabon-Bobinnec contributed to the deployment of the testbed.

1.8 Outline

The remainder of the thesis is organized as follows. Preliminaries, background, and threat model are presented in Chapter 2. Chapter 4 gives an overview of our solution in terms of methodology and security benefits. Chapter 5 and 6 present the proposed methodology to build a global model for a communication graph and for an event dependency graph, respectively. Chapter 7 presents the implementation, and Chapter 8 discusses our evaluation results. In Chapter 9, we demonstrate different use-cases and applications of CCSM. Chapter 10 presents additional contribution that were made during this work. Finally, we conclude in Chapter 11.

Chapter 2

Background and Preliminaries

In this chapter, we provide necessary background and define our threat model.

2.1 5G Edge-Core Model

In the context of 5G networks, the edge-core model is a distributed architecture allowing a provider to move some resources and services closer to the user (i.e., at the *edge* of the network) while keeping core functions in a central place (i.e., at the *core*). This allows better throughput and latency for certain applications, and can also address privacy and confidentiality concerns by allowing companies to deploy some part of the network on their own premises.

For instance, the Mobile Edge Computing (MEC) standard [27] suggests placing the User Plane Function (UPF) at the edge and the control plane network functions (e.g., Access and Mobility Management Function (AMF) and Session Management Function (SMF)) on the core, which can improve the response time and performance. Other use cases such as private 5G [39, 42] and inter-operator roaming [17]) require more flexibility in distributing network functions between the core and the edge. Therefore, recent works (e.g, [16, 39, 42]) propose different edge-core architectures, such as deploying AMF and

SMF on the edge next to the UPF [16] (as depicted in Fig. 2), while the core provides authentication, authorization, and other control plane functionalities. Such an approach can considerably reduce signaling between the edge and core. Our work does not assume a specific 5G edge-core splitting configuration and instead can support different edge-core architectures.

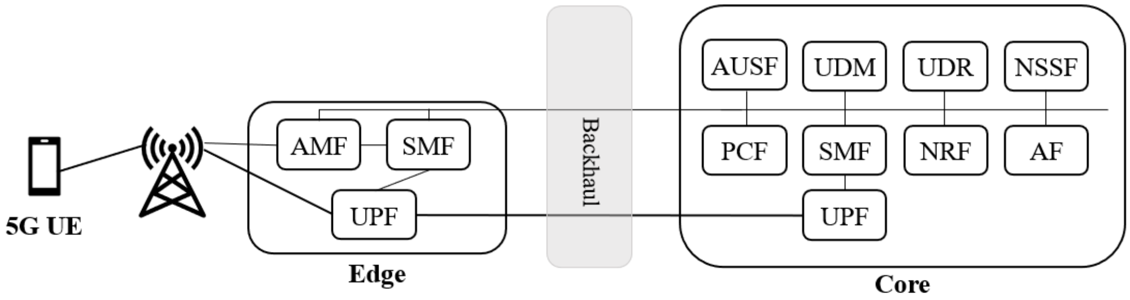


Figure 2: An example of 5G edge-core architecture [16].

2.2 Confidentiality in Private 5G

Private 5G is a concept encompassing the provision of tailor-made network applications and services to private businesses and third-party service providers [53]. Due to the high costs associated with deploying a standalone private 5G network on-premises for businesses, they can benefit from the private 5G services provided by mobile network operators. In such a scenario, the edge cluster is usually deployed on each business's own premise, where some of the operator's services and resources will be shared among tenants (e.g., Radio Access Network and control plane [3]). Consequently, confidentiality concerns may inevitably emerge due to potential conflict of interest between the tenants, and leakage of data, configurations, or network architecture [39] to third parties. Our work addresses such concerns by leveraging privacy-preserving, anonymization, and pruning mechanisms.

2.3 Communication and Event Models

We briefly review two existing models used to capture the communications and event dependencies, respectively. We will apply our methodology to build such models across multiple clusters later.

2.3.1 Communication Model

A communication model depicts the existence of communications between network functions (NFs) or between an NF and another network element (e.g., gateway). Communication models are usually represented as directed graphs including two types of nodes, namely, internal nodes representing NFs belonging to the local cluster, and external nodes representing nodes belonging to other clusters (which can be either the source or destination of communications with internal nodes). The presence of an edge between two nodes indicates a directed network communication between those two nodes, while no edge means no communication exists between them. Communication models capture the connectivity dependency between entities (e.g., reachability) and can be used to detect network breaches and anomalous communications [18, 50]. As an example, Fig. 3a shows the communication model (as a graph) for NFs inside one cluster. For instance, SMF initiates a connection with several other NFs, while it only accept a connection initiated by AMF. .

2.3.2 Event Dependency Model

An event dependency model depicts the occurrence dependency (e.g., precedence or succession) between two events. Event dependency models are represented with directed graphs where nodes represent events and directed edges represent dependencies between events. Event dependency models are well studied (e.g., [21, 26, 24]) and used in different security solutions, such as event prediction [28, 33] and anomaly detection [13].

The models can be constructed through gathering application events (e.g., from logs, API calls, and messages) and establishing dependencies between such events based on different metrics (e.g., frequency and closeness) or techniques (e.g., Bayesian learning, LSTM, and n-gram). Fig. 3b depicts an excerpt of event dependency model from a 5G core network. For instance, the event “AUSF-HandleUeAuthPostRequest” depends on the event “AMF-HandleRegistrationRequest”, with the latter further depending on “AMF-Handle-InitiateUEMessage”.

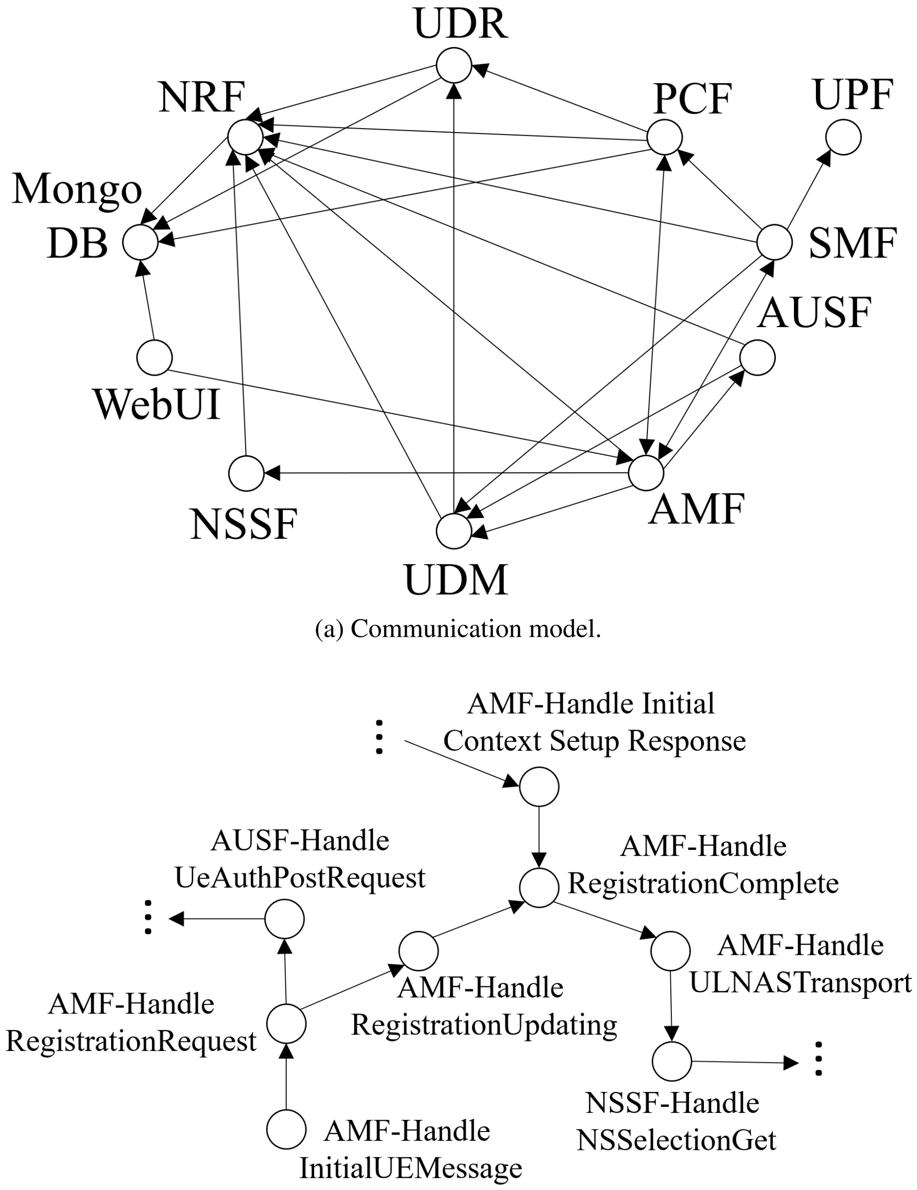
2.4 Threat Model

2.4.1 In-scope Threats

We focus on security breaches and attacks that involve multiple clusters (i.e., cross-cluster threats). Such breaches and attacks may originate from misconfigurations, user mistakes, or exploits of vulnerabilities by either insiders or external attackers. We assume the breaches and attacks may be identified based on the communications or event dependency models using existing security solutions including (but not limited to) offline security auditing, runtime security monitoring, and security impact prediction (as demonstrated later in Chapter 9). Like most existing privacy-preserving solutions, we assume honest-but-curious providers who follow the proposed methodology.

2.4.2 Out-of-scope Threats

We do not consider breaches or attacks that are contained inside a single cluster (as these can be tackled using existing solutions [28, 7, 18, 50, 13]) or those that cannot be identified based on the communications or event dependency models (e.g., container or OS-level attacks). We do not consider malicious providers who deviate from our methodology. In the other words, For correct deployment of CCSM and its integrity, we rely on the providers.



(a) Communication model.

(b) Event dependency model.

Figure 3: Excerpt of communication and event dependency models of a 5G core network.

Indeed, we assume the integrity of our solution itself, Kubernetes, and the underlying infrastructure (including the data sources, such as network traffic and application logs, used to be build our models), and hence attacks that can temper with these are out-of-scope.

Chapter 3

Related Work

In this Chapter, we review related work in the domain of 5G security as well as federated learning efforts that contribute to the same objective.

3.1 Cloud Security

There are several works addressing different aspects of cloud security, including security verification and attack detection.

The authors in [18] proposes to detect breaches in software-defined networks (SDN) based on network topology and forwarding rule observation and analysis. Unlike our solution, it is not designed for distributed and multi-tenant aspects (such as the edge-core model and private 5G) and does no take into account the privacy challenges brought by these. In [59], authors present a survey on network services anomaly in NFV environments. Our current solution can subscribe to the *Topology anomalies detection* category, but additionally encompasses topology anomalies happening over multiple infrastructures. There are several proactive security compliance verification works (e.g., [9, 37, 32, 31, 28]) for OpenStack clouds. For instance, Weatherman [9] and Congress [37] verify security policies in clouds using graph-based and Datalog-based models, respectively. *LeaPS* [32] and

Proactivizer [33] are proactive security auditing solutions for cloud environments, while *ProSPEC* [28]. They use management events to learn a management event dependency model and predict future management events. In contrast to our solution, those works are not designed for application-level events, and to tackle the complexity and privacy challenges of multi-cluster environments.

In [14], the authors propose a framework to predict advanced persistent threats (APT) in a distributed 5G environment, while ensuring privacy. Our approach additionally aims at creating global models for different purposes (including attack detection, security verification and security impact prediction). The authors in [52] propose a federated-learning approach to detect network attacks in 5G distributed systems using packet traffic analysis. Our approach additionally encompasses application events to model the cloud at a higher level and discover different, and potentially stealthier attacks. WARP [7] propose to proactively mitigate attacks in Kubernetes cluster by predicting the attacker's next more and migrating vulnerable resources accordingly. All these work could benefit for the cross-cluster dimension of our solution to extend their scope.

3.2 Event Dependency

In [60], the authors propose an advanced approach to correlate events. This approach can be used to increase the reliability of our event dependency model. In this work, we add a federated and distributed approach to correlate events from different sources and environment, potentially enabling the discovery of otherwise invisible breaches and attacks. [24, 21, 26] propose approaches to match events. [26, 24] follows a frequency based metric to assess the dependency of events.

3.3 Federated Learning

Federated learning (FL) is used to collaboratively train a global model. A server node orchestrates multiple client nodes by sending them a global model. Each client node locally trains the model using its local data without the need to share it with the server. Su *et al.* [45] applied edge federated learning in a smart grid environment to share the private energy data of users. The edge-cloud collaboration helps in training global models with minimum communication overhead while ensuring data privacy.

Multiple devices can collaboratively train a security model using FL to learn for the security appliances (e.g., anomaly detection). For instance, Rasha et al. [40] studied the application of federated learning in smart cities (e.g., transportation, healthcare, communication) aiming at improving security and privacy. The authors further provided a few research issues and future directions to improve data protection and performance. Although FL might train a model without sharing user's data, it might lead to multiple iterations that in consequence impact the network performance. FL solutions might also suffer from distributed data and model poisoning [44]. In addition, none of those efforts focus on addressing 5G-specific challenges that involve cross-cluster security verification.

Chapter 4

CCSM

This chapter presents an overview of the CCSM approach, along with its example and potential security applications.

4.1 Overview

Fig. 4 illustrates an overview of the CCSM approach to build cross-cluster security models for edge-core environments involving multiple Kubernetes clusters. The inputs to CCSM are network traffic and event logs collected from from the 5G edge and core networks hosted on multiple Kubernetes clusters. CCSM comprises two phases: *Building Local Models* and *Building Global Model*.

The *Building Local Models* phase is to create local models for each cluster including all the edges and core by following Steps 1 to 4. (1) CCSM constructs local models for both communication and event dependencies from network traffic and event logs, respectively. (2) It extends the local model by identifying external nodes that interacts with the core. (3) It prunes the other nodes in the local model that do not interact with the core. (4) It anonymizes all the sensitive information in the model (e.g., IP addresses, network function names) to protect the confidentiality of an edge (typically owned by different tenants) and share on a “need to know” principle. The *Building Global Models* phase is to combine local models to encompass a global view of the security posture across all clusters by following Steps 5 and 6. (5) CCSM constructs the global model for both communication and event dependencies by aggregating local models for all edges and core. (6) It prunes the global model from the edges’ perspectives to enable the creation of edge-wise global views that are sent back to respective edges. We further detail these steps in Sections 5 and 6.

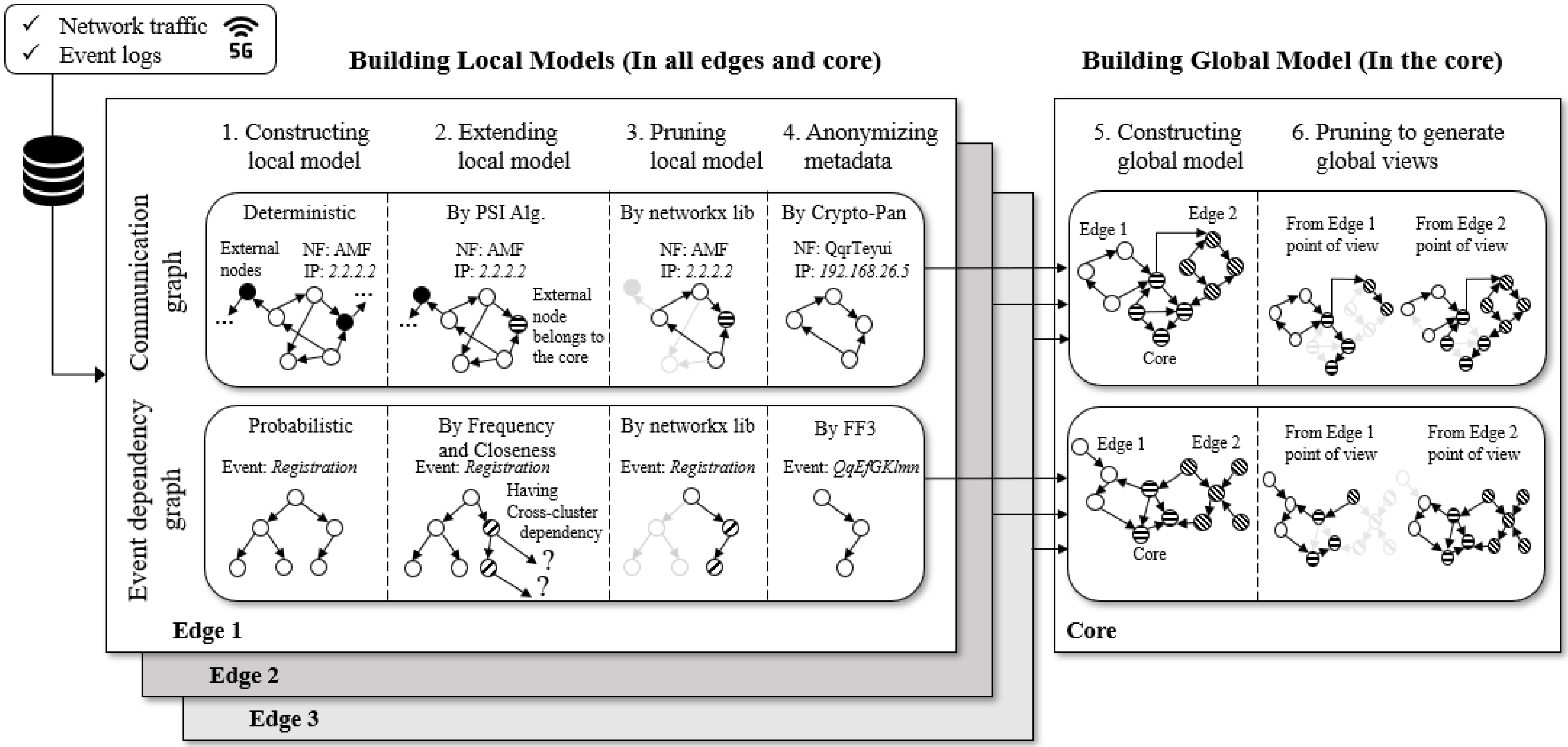


Figure 4: Overview of the CCSM approach.

4.2 Example

Using Fig. 4, we show a simplified example with the outputs of CCSM. (1) The constructing local model step forms the communication model with the IP addresses for network functions (NFs) (`2.2.2.2` for the NF `AMF`) as the nodes and their communication as the edges as well as the event dependency model with the event names as the nodes (`registration`) and their transitions as the edges; where internal nodes are indicated as unfilled and external nodes as filled. (2) The extending step identifies external nodes belonging to the core (indicated as horizontally stripped) that belongs to the core and (3) the pruning step removes the other external and internal nodes that are not directly connected with the core. (4) The anonymization step anonymizes the NF name (`AMF`), IP address (`2.2.2.2`), and event name (`registration`) to (`QqrTeyui`), (`192.168.26.5`) and (`QqEfGKlmn`), respectively, using privacy-preserving algorithms (e.g., Crypto-PAn [43]). (5) The constructing global model aggregates three local models for Edge 1, Edge 2 and the core (where nodes are indicated using no-pattern, angular strips, horizontal strips, respectively). (6) The final step prepares the global views specific to Edge 1 and Edge 2, while pruning the part of model that belongs to other edge. We further elaborate on this example in Examples 1-4.

4.3 Security Applications

By building cross-cluster security models and providing edge-specific global views, CCSM enables several security applications. First, constructing a global model allows individual clusters to detect and predict security issues in them that would not be completely possible using only local models. From the observation that clusters can interact with external entities but their vision is limited to their own scope, and that 5G environments are naturally multi-tenant and distributed, we claim that extending security model across clusters can help understanding, detecting, and predicting security issues. Second, global views offer the possibility to audit other (potentially untrusted) tenants of the cloud environments. By revealing the impact of local activities outside the cluster, one is able to assess whether security policies are enforced (e.g., if physical or logical boundaries isolation are effectively respected). Third, our global model and views can help administrators represent and assess the state of complex, distributed cloud environments. For instance, global model can be used to plan maintenance or change in the environment and prevent once unpredictable "cascading" effects (i.e., a small change in one cluster might have large, disastrous consequences in the other clusters that depend on it). We detail use-cases inspired from real-world scenarios in Chapter 9.

Chapter 5

CCSM Approach for Communication Model

This chapter presents our methodology for building local and global communication models.

5.1 Building Local Communication Models

For each edge as well as the core cluster, we detail in the following how CCSM constructs a local communication model.

Data Collection and Processing. To build communication models, our approach requires to capture network traffic between NFs. To later aggregate the local models, our approach relies on the external communications each model shares with other clusters. Therefore, network traffic logs are processed to identify two types of communications: internal communication which refers to communication between two internal IP addresses and external communication which refers to communication between an external and an internal IP addresses. A single communication entry comprises the IP addresses of the source and the destination of a communication, additionally it also stores the name of the corresponding

NF for internal IP addresses. The obtained communications logs are stored in a CSV file and serve as input for the next steps.

Constructing. This step consists of building the communication model from the collected data. For communication models, this process is deterministic, as we construct a directed graph with one node per unique IP address involved in the communication, and a directed edge between two given nodes based on the existence of at least one communication record between source and destination nodes in the network traffic/in the application log. Nodes are labeled with the associated IP address (and with the name of the corresponding NF, for internal IP addresses). Step **a** in Fig. 5 depicts a sample graph representing a communication model, with internal nodes (i.e., NFs belonging to the local cluster, in white) and external nodes (i.e., a source or destination node external to the cluster, in black).

Extending. This step is to distinguish those external nodes shared with the core cluster from other external nodes (e.g., public IP addresses on the Internet) in a confidential manner. To that end, our approach is to identify external nodes in the edges' local model that in fact belong to the core cluster (and therefore virtually *extend* the reach of the edges' local models to the core). Specifically, the edge and the core clusters share their respective external communications with each other and identify those in common without disclosing any sensitive data. More precisely, to verify common connections between a given edge cluster and the core cluster without disclosing any sensitive data (i.e., the NF name or IP address), we utilize a privacy-preserving intersection (PSI) algorithm [38], a cryptographic method for secure multiparty computation [12]. PSI enables two entities to compute and find the shared elements between their sets while keeping the non-shared items confidential. Once common communications are identified, each edge tags the corresponding external nodes as belonging to the core cluster. Step **b** in Fig. 5 depicts a communication model sharing three of its external nodes with the core cluster (striped diagonally).

Pruning. Local models can have a sheer size and therefore would require intensive computational resources during later steps. Additionally, the network architecture that can be inferred from the model may still reflects potentially sensitive information (e.g., single points of failure) that individual clusters might be reluctant to share. To address both these challenges, our idea is to prune the local models and only preserve nodes and edges that are relevant to the global model. Specifically, we remove nodes (and corresponding edges) that have no path from/to an external nodes belonging to the core cluster. Our pruning technique allows to greatly reduce the size of the local models (as evaluated in Chapter 8) while preserving confidentiality by removing nodes that are irrelevant for the security analysis (i.e., on a “need to know” basis).

Anonymizing. While confidentiality is partially achieved during *extending* (using PSI) and *pruning* (confidentiality of the network architecture) steps, anonymization aims at completely decorrelating the edge cluster from its communication model. Therefore, our approach anonymizes any sensitive data before sending the local model for aggregation in the core cluster. In fact, the NF’s names and IP addresses (which are sensitive data as they can be traced back to that particular edge cluster) are anonymized using format-preserving encryption (FPE) algorithms [8] so that the ciphertext is in the same format as corresponding plaintext. Particularly, for IP addresses, we leverage Crypto-PAn [56, 57], an FPE algorithms specifically designed for anonymizing IP addresses while preserving their subnet structures (which might be useful for later investigations). Table 1 shows several examples of IP addresses anonymized using Crypto-PAn in a prefix-preserving manner. For example, as first and second IPs in the table belong to the same subnet, after anonymization their corresponding anonymized IP addresses will belong to the same subnet again. In the same way, as the third one has 16 bits in common with two previous ones, after anonymization the corresponding anonymized IP address will have the same first 16 bits.

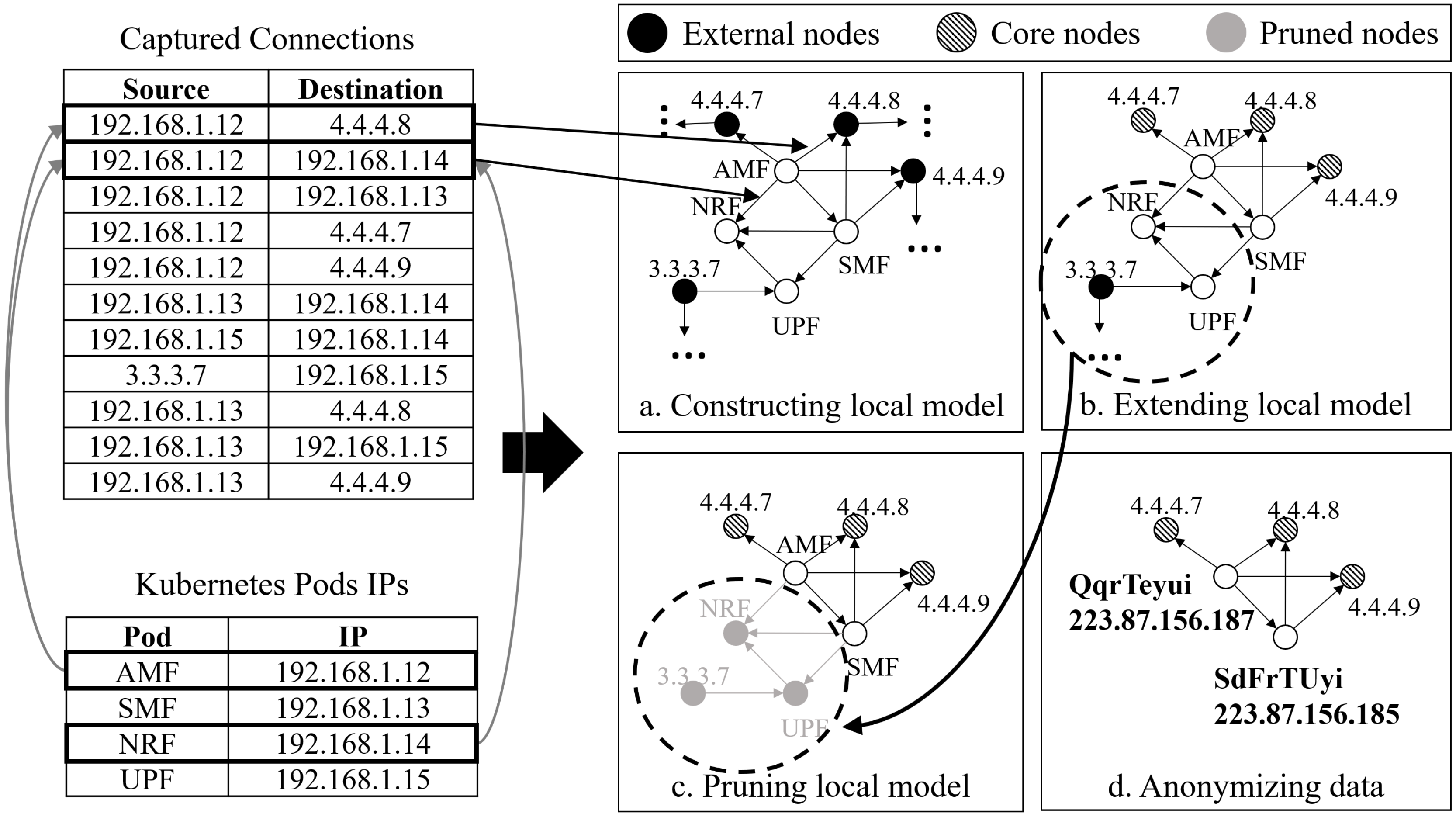
It is worth mentioning that, even though we utilized Crypto-PAn as an example here for

Table 1: An example of format-preserving encryption using Crypto-PAn.

Original IP	**IP anonymized by Crypto-PAn**
192.168.1.13	***223.87.156***.185
192.168.1.14	***223.87.156***.187
192.168.5.23	***223.87***.155.187
192.130.3.54	***223***.125.128.117
10.10.10.25	29.21.233.153

anonymization, other more secure alternatives such as the approach proposed by Mohammady et al. [34] can be used to strengthen potential weaknesses against inference attacks by fingerprinting and injection [11, 10, 58]. More generally, format-preserving encryption can be also used for other attributes based on necessity, type of application, and customer's requirements. For other applications, CCSM can anonymize different types of attributes (e.g., timestamps, owner, namespace, etc.) using more general methods such as a generalized framework which are provided by Xie *et al.* in [55]. Similarly, the FF3 standard [20] is used to anonymize NF names. By utilizing those anonymization algorithms, we allow local clusters to hide sensitive information before sending their local models for the centralized global model building. As well as that, For the confidentiality guarantee, we rely on the state-of-the-art approaches, such as PSI [38], Crypto-PAn [56, 57], and thus, ensure the same confidentiality level as those works or their alternatives based on the aforementioned discussion.

Example 1. Fig. 5 depicts an example of local communication model building. First, in Fig. 5.a, CCSM constructs the initial version of the local model (as a graph) based on the collected data, which represents communications between 4 NFs (e.g., AMF, NRF, SMF, and UPF) and 4 external IPs which are shown in the *Captured Connections* and *Kubernetes Pods IPs* tables. The first two entries of the captures connections are used to create an edge for the external communication between the AMF (address 192.168.1.12) and an external IP 4.4.4.8, and for the internal communication between AMF and the NRF. Then, in Fig. 5.b, CCSM identifies three of those external IPs (4.4.4.7, 4.4.4.8 and 4.4.4.9) as belonging to the core cluster (through the use of PSI, not depicted in the figure) and marks them accordingly. In Fig. 5.c, the nodes corresponding to NRF, UPF, and external IP $3.3.3.7$ are pruned because none of those can reach or can be reached from any nodes in the core. Conversely, SMF and AMF can reach the core IPs, therefore these nodes are kept. Finally, in Fig. 5.d, the name and IP addresses of remaining internal nodes such as AMF are replaced with the anonymized name *QqrTeyui*, and anonymized IP address $223.87.156.187$ by using FF3 and Cryto-PAn respectively.



Captured Connections

Source	Destination
192.168.1.12	4.4.4.8
192.168.1.12	192.168.1.14
192.168.1.12	192.168.1.13
192.168.1.12	4.4.4.7
192.168.1.12	4.4.4.9
192.168.1.13	192.168.1.14
192.168.1.15	192.168.1.14
3.3.3.7	192.168.1.15
192.168.1.13	4.4.4.8
192.168.1.13	192.168.1.15
192.168.1.13	4.4.4.9

Kubernetes Pods IPs

Pod	IP
AMF	192.168.1.12
SMF	192.168.1.13
NRF	192.168.1.14
UPF	192.168.1.15

a. Constructing local model

b. Extending local model

c. Pruning local model

d. Anonymizing data

Figure 5: Running example: illustrating steps of the *building local models* phase for edge 1.

Algorithm for Building Local Models. Algorithm 1 shows our pseudo-code for building local models. There are four functions for the four steps of the *Building local models* phase. First, the *Constructing local model* function (Line 4) parses a `connections.csv` file and converts its content in a graph. Then, this graph is processed by the second function *Extending local model* (Line 14) to extract external connections and find common connections with the core cluster (or with other edges, for the core). To do so, the core and edges use the PSI algorithm to find common connections without endangering their own privacy. Third, the graph is processed by the *Pruning local model* function (Line 31) to check the reachability of nodes and edges between the edges and the core cluster. This function keeps those that are reachable and remove others. Finally, the *Anonymization local model* function (Line 47) applies anonymization techniques (e.g., PSI and FF3) to prepare the local model for aggregation (in the core cluster) while preserving confidentiality.

5.2 Building Global Communication Model

During this phase, CCSM receives local models from all edges and the core clusters, then aggregates them to build a global model. The edge clusters communicate directly with the core cluster, but not with each other, thus this global model aims at revealing all those communications between edges which are not otherwise easily identifiable. The aggregation phase happens in a centralized manner (e.g., in the core cluster) in two steps, detailed as follows.

Constructing the Global Model. In this step, we collect and aggregate all local models (edges' and core's) based on their shared nodes (previously identified in the *extending* step). The core cluster takes local models (including its own) as graphs and computes their mathematical union. It is made possible to find common nodes between graphs since the external nodes (belonging to the core) are not anonymized in the previous step; instead, only internal nodes are. Because the graph union is commutative, aggregating local models can be done iteratively and in any order. The obtained global communication model reflects how the edge clusters potentially interact with each other through the shared nodes in the core.

Pruning to Generate Global Views. Once the global model is built, it captures all cross-cluster relationships. However, following the "need to know" principle, each edge cluster

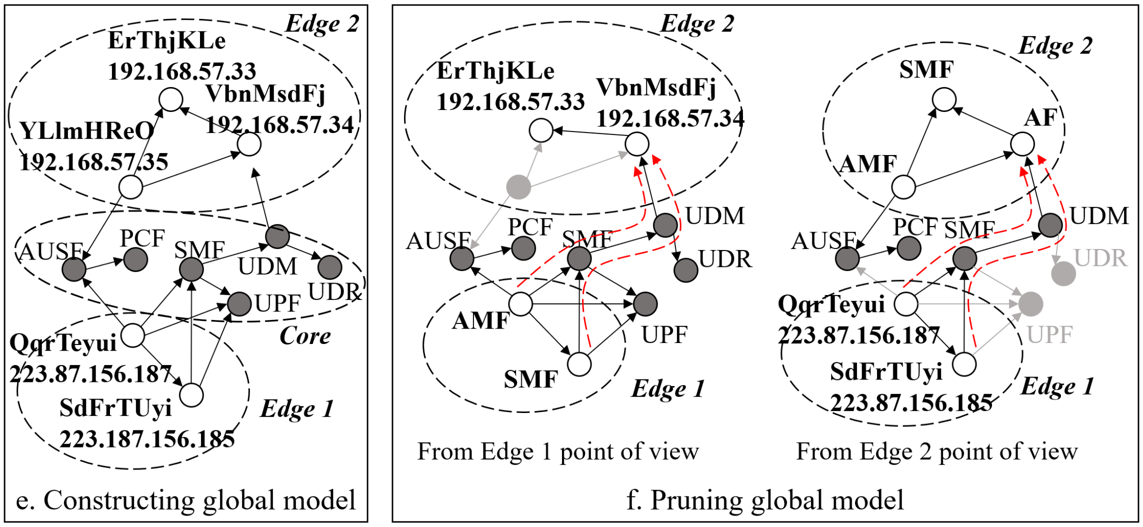
Algorithm 1 Building Local Model.

```
Input: Collected data $Connections.csv$ (SRC(IP, Name),DST(IP, Name))
Output: Local model $G$

function CONSTRUCTING_LOCAL_MODEL($Connections.csv$)
    //Build local model from input file (Deterministic)
    Temp Variable LocalModel($G = (E, V)$)
    for each (SRC, DST) in $Connection.csv$ do
        $G.V$.add(SRC)
        $G.V$.add(DST)
        $G.E$.add(SRC, DST)
    end for
    Return $G$
end function

function EXTENDING_LOCAL_MODEL($G$)
    for edge e in $G.E$ do
        if e is an external connection then
            ExternalEdges.add(e)
        end if
    end for
    //Find common edges between edge cluster and core
    CommonEdges = PSI(ExternalEdges)
    for e in CommonEdges do
        if  SRC is external IP  then
            $G.V$(SRC).isCommon = 1
        else
            $G.V$(DST).isCommon = 1
        end if
    end for
    Return $G$
end function

```

```
function PRUNING_LOCAL_MODEL($G$)
    Temp Variable LocalModel($G^{'} = (E, V)$)
    for node $n$ in $G.V$ do
        if $n$.isCommon == 1 then
            $G^{'}.V$.add(ancestors(n))
            $G^{'}.V$.add(descendants(n))
        end if
    end for
    for edge e in $G.E$ do
        if e.SRC is in $G^{'}.V$ and e.DST is in $G^{'}.V$ then
            $G^{'}.E$.add(e)
        end if
    end for
    Return $G^{'}$
end function

function ANONYMIZING_LOCAL_MODEL($G$)
    for node n in $G.V$ do
        n.SRC.IP = Crypto-PAn(n.SRC.IP)
        n.DST.IP = Crypto-PAn(n.DST.IP)
        n.SRC.Name = FF3(n.SRC.Name)
        n.DST.Name = FF3(n.DST.Name)
    end for
    Return $G$
end function
```

is only entitled to view the fraction of the global communication model that is related to it. Therefore, we apply multiple times the same pruning process as in the *pruning local model* step to the global model while considering a different edge's point of view at a time. Specifically, for each edge, we create a restricted view of the global communication model by removing any node that cannot be reached from, or cannot reach an internal node of the original local model. This ensures that each edge cluster receives only the relevant information needed to improve their local security view, based on sharing the least required information without unnecessary overhead. Finally, once the global views have been constructed, each global view is securely transmitted to the corresponding edge cluster. This may involve encrypting the model, use secure communication channels, to ensure that only authorized parties can access the global views (these are not studied in this work).

Example 2. Fig. 6 depicts an example of building a global model from local models from Edge 1 (at the bottom; following Example 1) and Edge 2 (at the top) and the Core (in the middle). First, in Fig. 6.e, three local models belonging to Edge 1, Edge 2, and Core respectively, are aggregated into a global model based on common external nodes previously identified (i.e., AUSF, SMF and UPF for Edge 1; UDM and AUSF for Edge 2). Then, in Fig. 6.f, the global model is pruned from each Edge's point of view to provide global views. From the point of view of Edge 1, its internal nodes AMF and SMF are related to the node (anonymized) *VbnMsdFj* in Edge 2 through the Core's SMF and UDM. Additionally they are related to the node *ErThjKLe* by transitivity, however the last node of Edge 2 *YLlmHReO* is not included as it has no relationship with Edge 1 (it cannot be reached, nor can reach any node from Edge 1). Similar logic can be applied from the point of view of Edge 2.



e. Constructing global model

f. Pruning global model

Figure 6: Running example: Output of global builder module on core cluster

Algorithm for Building Global Model. Algorithm 2 details the process of constructing the global model. First, the *Constructing global model* function (Line 4) merges all local models depending on their connections with the core. To do so, it constructs the set intersection between each edge's local model and its own local model using the PSI algorithm. Progressively, local models are merged with the core's local model to form a global model. Then, CCSM shares a version of the global model to each edge, containing only the information related to that edge. To that end, the *Pruning global model for edge* function (Line 21) removes all edges and nodes that are not reachable from that specific edge cluster or can not get reached by any one node from that edge cluster.

Algorithm 2 Building Global Model.

```
Input: $LocalModels$, Edge number $j$
Output: Global model $G_i$ pruned for Edge number $j$

function CONSTRUCTING_GLOBAL_MODEL($LocalModels$)
    Temp Variable GlobalModel($G = (E, V)$)
    for Graph $G_i$ in $LocalModels$ do
        for edge e in $G_i^{'}.E$ do
            if e.SRC.isCommon == 1 or e.DST.isCommon == 1 then
                if verify_by_PSI_result(e) then
                    Continue
                else
                    Alarm for anomaly
                end if
            end if
        end for
        G.merge($G_i^{'}$)
    end for
    Return $G$
end function

function PRUNING_GLOBAL_MODEL_FOR_EDGE($G$, $j$)
    Temp Variable GlobalModel($G^{'} = (E, V)$)
    for node $n$ in $G.V$ do
        if $n$ is a node from local model of edge $j$ then
            $G^{'}.V$.add(ancestors($n$))
            $G^{'}.V$.add(descendants($n$))
        end if
    end for
    for Edge $e$ in $G.E$ do
        if $e$.SRC is in $G^{'}.V$ and $e$.DST is in $G^{'}.V$ then
            $G^{'}.E$.add($e$)
        end if
    end for
    Return $G^{'}$
end function
```

Chapter 6

CCSM Approach for Event Dependency Model

In this chapter, we describe how CCSM constructs a local event dependency model for each cluster, and then aggregates them across multiple clusters. Our methodology can be seen similar to the one in the previous chapter however, it is more elaborative as there is no straightforward approach to identify dependency relationships between various application events. Therefore, we first describe our approach to determine such dependencies, and then explain how we utilize it to build local and the global models, and generate the global views subsequently.

6.1 Building Local Event Dependency Models

First, our solution builds local dependency models encompassing the relationship between local events (i.e., events generated by applications and NFs deployed in the edges and the core separately). Then, it identifies events which have a cross-cluster dependency and finally anonymizes the model before sending it for aggregation.

Data Collection. To build the event dependency model, we first collect 5G event logs at the application level. This collection process occurs at the NF level using Kubernetes' Pods and containers logging. Then, event logs from all 5G applications are subsequently merged and sorted based on timestamp. Following this, the logs are parsed to extract the event name (usually in the form of `<NF name>-<associated 5G procedure>`. The sorted events entries are stored and used to build the local model.

Identifying Event Dependencies. Unlike communication models, where dependency relationships can be assessed based on a clear factor (i.e., the presence or not of a network communication), the dependency between two application events cannot be identified straightforwardly. Instead, our key idea is to rely on two properties for ordered pair of events, namely, their *temporal closeness* and their *occurrence frequency*. We compute those properties for each ordered pair of events in the cluster and compare them to their respective user-defined thresholds to determine whether they have a dependency relationship or not. We present here our methodology to identify event dependencies, and utilize it to construct the local event dependency models and the global cross-cluster event dependencies models (implementation details in next chapter). More formally, given a set of collected events $E = \{e_1, e_2, \cdots, e_n\}$ where event e_i happens at time t_i, and $j > i$ means event e_i happens before event e_j, we define event dependency between two events in a given ordered pair (e_i,e_j) as a Boolean function that captures the existence (or not) of a dependency between those events as follows:

$$Dependency(e_i, e_j, T_c, T_f) = true \iff \frac{|\{(e_i, e_j) \mid (t_j - t_i) \leq T_c\}|}{NumberOf(e_i)} > T_f \qquad (1)$$

where $T_c \geq 0$ is a user-defined threshold for the temporal closeness, which is defined as $t_j - t_i$ and $T_f \in [0, 1]$ is the user-defined threshold for the occurrence frequency of ordered pair of (e_i,e_j) when its temporal closeness is less than or equal to T_c. The occurrence frequency is expressed using the left term of the inequality in the second term of the above equation. Note that |_| means the set size and $NumberOf(e_i)$ counts the total number of occurrences of the event e_i independently of the event e_j.

Thus, only a subset of the possible dependencies of the form e_i->e_j will be considered in the event dependency model, which satisfy the closeness threshold and frequency threshold chosen by the user. In Chapter 8, we show how the choice of these thresholds impacts the dependency model generated by CCSM and provide guidelines on how they can be chosen depending on the use case.

Constructing. As mentioned earlier, we use the aforementioned event dependencies identification method to build local dependency models. To do so, CCSM constructs a local model based on the two aforementioned parameters, namely, the *occurrence frequency* (named frequency in the remaining for brevity) and the *closeness* between ordered pairs of events. More specifically, for any given ordered pair of events (e_i, e_j), we count the number of times that event e_i is followed by event e_j within a time interval smaller than or equal a predefined closeness threshold. Subsequently, the frequency of occurrence of event e_j after event e_i satisfying the temporal closeness condition, is computed and compared with the frequency threshold. If the computed value is larger than the threshold the dependency is considered, otherwise it is discarded. This probabilistic approach provides flexibility to users in defining the dependency relationship between events and allows them to customize these two parameters to identify the best values for their specific application (i.e., anomaly detection applications might benefit from smaller frequency thresholds, as anomalies happen in a non-regular and local manner). This flexibility is later discussed in Chapter 8.

Extending. To extend the scope of the local models and further consider cross-cluster dependencies, we apply our dependency identification method between events from different models. First, event names are anonymized, then the events logs are shared with the core cluster. Event timestamps are not anonymized as their exact values are needed by the core cluster to establish further event dependencies. Unlike for communication models where

PSI is used to identify common nodes in a privacy-preserving manner, the same reasoning cannot be applied since cross-cluster event dependencies have to be assessed based on parameters, particularly occurrence frequency of ordered pairs consisting of both edge's events and core's events. Therefore, it is necessary that edge clusters share their entire anonymized event logs with the core for a centralized processing.

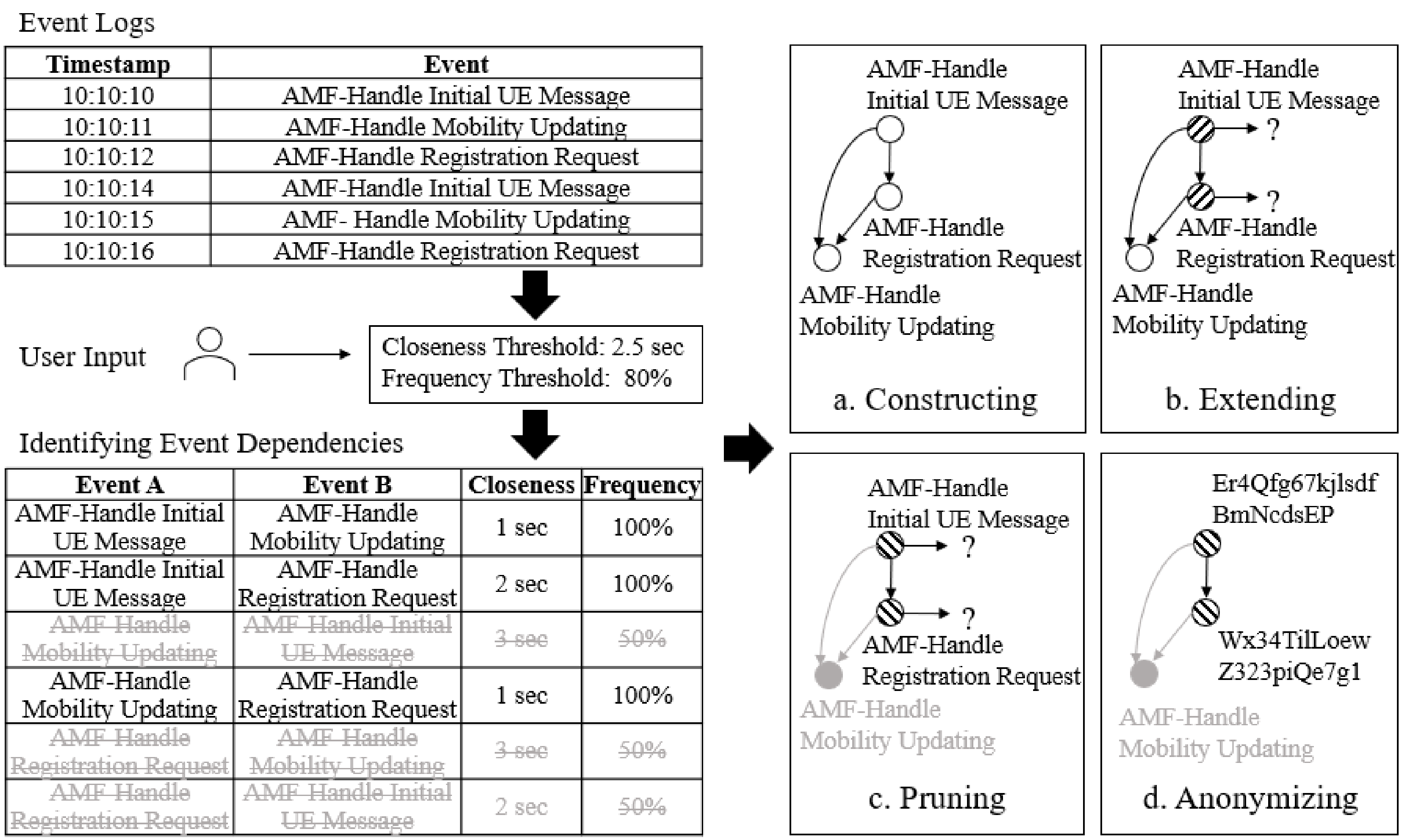
The core cluster employs our dependency identification approach based on user-defined thresholds, and the closeness and frequency for each pair of events A and B belonging to the edge and core clusters, respectively. Following this, the core cluster returns to the edge the list of events that are involved in cross-cluster dependencies (i.e., events that have a dependency with at least one event on the core cluster, striped diagonally in Fig 7).

Pruning. In this step, the event dependency model is pruned by removing non-essential nodes and edges while ensuring that important events (i.e., that play a role in the global model) are retained. This is achieved by pruning nodes that have no path from or to any nodes identified during the *extending* step.

Anonymizing. The final step in the *building local models* process is to anonymize the local model before sending it for aggregation in the core cluster. In this step, we utilize FF3, a Format-Preserving Encryption algorithm, to anonymize event names in the model, similarly to how we anonymize NF names for the communication model.

Example 3. Fig. 7 depicts an example of building a local event dependency graph model. Closeness and frequency threshold values are 2.5 s and 80%, respectively. First, in Fig. 7.a, CCSM constructs a local model as a graph by finding dependencies based on event logs and processing tables which are shown in the figure. For instance, there is no edge from *AMF-Handle Registration Request* to *AMF-Handle Initial UE Message* in the event dependency local model since its frequency (50%) is lower than threshold, even though it meets closeness threshold. Then, in Fig. 7.b, the graph is extended by tagging nodes which have cross-cluster dependency (shown with striped patterns). For instance, *AMF-Handle*

Initial UE Message and *AMF-Handle Registration Request*) are found to have dependencies with event in the core cluster based on the closeness and frequency they have with certain core events (not shown in this figure). After that, in Fig. 7.c, the extended model is pruned by removing irrelevant nodes and edges which are not essential for the analysis. For instance, the node *AMF-Handle Mobility Updating* is pruned as it cannot reach any node in the core, and cannot be reached from any node in the core. Finally, in Fig. 7.d, event names are anonymized by using FF3 in order to address privacy and confidentiality concerns. For example, *AMF-Handle Registration Request* is anonymized by FF3 to *Er4Qfg67kjlsdfBmNcdsEP*.



Event Logs

Timestamp	Event
10:10:10	AMF-Handle Initial UE Message
10:10:11	AMF-Handle Mobility Updating
10:10:12	AMF-Handle Registration Request
10:10:14	AMF-Handle Initial UE Message
10:10:15	AMF- Handle Mobility Updating
10:10:16	AMF-Handle Registration Request

User Input

Closeness Threshold: 2.5 sec
Frequency Threshold: 80%

Identifying Event Dependencies

Event A	Event B	Closeness	Frequency
AMF-Handle Initial UE Message	AMF-Handle Mobility Updating	1 sec	100%
AMF-Handle Initial UE Message	AMF-Handle Registration Request	2 sec	100%
~~AMF-Handle Mobility Updating~~	~~AMF-Handle Initial UE Message~~	~~3 sec~~	~~50%~~
AMF-Handle Mobility Updating	AMF-Handle Registration Request	1 sec	100%
~~AMF-Handle Registration Request~~	~~AMF-Handle Mobility Updating~~	~~3 sec~~	~~50%~~
~~AMF-Handle Registration Request~~	~~AMF-Handle Initial UE Message~~	2 sec	~~50%~~

AMF-Handle Initial UE Message
AMF-Handle Registration Request
AMF-Handle Mobility Updating
a. Constructing
AMF-Handle Initial UE Message
?
?
AMF-Handle Registration Request
AMF-Handle Mobility Updating
b. Extending
AMF-Handle Initial UE Message
?
?
AMF-Handle Registration Request
AMF-Handle Mobility Updating
c. Pruning
Er4Qfg67kjlsdf BmNcdsEP
Wx34TilLoew Z323piQe7g1
AMF-Handle Mobility Updating
d. Anonymizing

Figure 7: Running example: Output of local builder module for Edge 1 cluster

6.2 Building Global Event Dependency Model

In this phase, CCSM collects all local models from edges and core clusters and integrate them to construct global model. Then, it generates global views from global model. These two steps are explained in detail in the following.

Constructing the Global Model. In this step, we integrate all local models (edges' and core's) based on the cross-cluster dependencies previously identified. Since cross-cluster event dependencies are identified in the core only, a consequence of this is that edge clusters are not aware of the core-side of their dependencies. Based on the cross-cluster dependencies identified during the *extending* step, the core is able to connect the prune version of local models with its own model, effectively constructing a global model.

Pruning to Generate Global Views. Once the global model is built, we apply our pruning method from each edge's point of view. For each edge, we create a restricted view of the global model by removing any node that cannot be reached from, or cannot reach an internal node of the edge's original local model. At the end of this step, each edge receives only the necessary information to improve their local security (i.e., each edge sees a specific view from global model). This approach to generate and send a specific view each edge completely addresses privacy and confidentiality concerns of individual clusters.

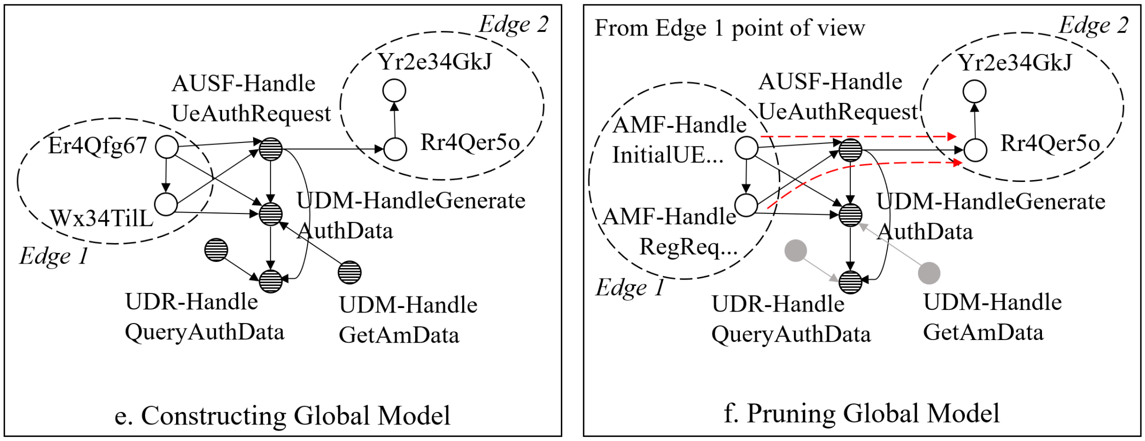


Figure 8: Running example: Output of global builder module on core cluster

Example 4. Fig. 8 depicts an example of building global event dependency model from three local models (Edge 1's and Edge 2's and Core's). In Fig. 8.e, the model of Edge 1 (left), Edge 2 (top right) and the Core (middle) are aggregated together. To that end, CCSM uses the closeness and frequency of events to find cross-cluster dependencies between edge and core nodes. From this step, it finds three new dependencies between events of Edge 1 and the Core: *Er4...→AUSF-HandleUeAuth...*, *Er4...→UDM-HandleGenerate ...*, and *Wx34...→UDM-HandleGenerate...* (the corresponding event logs are not shown). Similarly, it find one additional event dependency between *AUSF-HandleUeAuth...* (in the Core) and *Rr4Q...* (in Edge 2). Then, in step Fig. 8.f, CCSM prunes the global model from the point of view of Edge 1, by removing core events that cannot be reached from (or cannot reach) any event from Edge 1 (for instance, *UDM-HandleGetAmData*). As a result, the obtained model and view shows a potential event dependency from events in Edge 1 (*Er4...* and *Wx34...*) to an event in Edge 2 (*Rr4...*), indicating a potential breach which might be examined by experts. We will further demonstrate the applicability of such model in Chapter 9.

Chapter 7

Implementation

This chapter details the implementation of CCSM, then describes some challenges and technical issues we faced and how we overcame them.

7.1 Overview

We deploy a multi-cluster containerized 5G core network based on Towards5GS-Helm [49], an open-source initiative based on Free5GC [23]. We use Helm charts to automate the deployment of a 5G core network on Kubernetes. We use UERANSIM [5] to simulate the 5G Radio Access Network and generate events and traffic in the 5G core (e.g., register and de-register UEs). Fig. 9 describes the architecture of CCSM and the different technologies employed to realize each component. CCSM is implemented in a distributed manner including the *local model builder* module instances deployed on all individual clusters (i.e., edges and core), and the *global model builder* module deployed as a centralized manager on the core cluster. In the following, we detail on how we implement the local model builder and the global model builder.

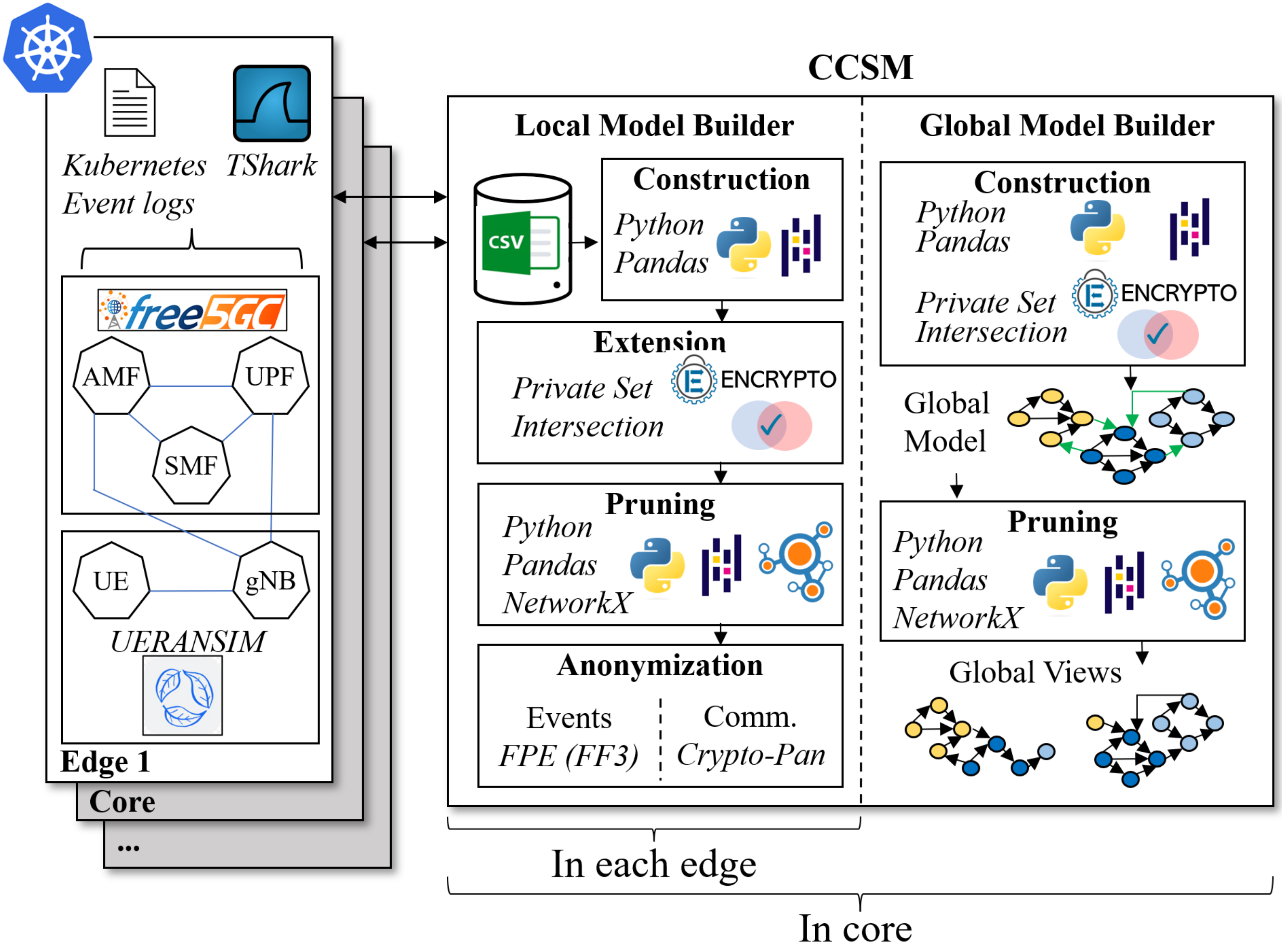


Figure 9: CCSM implementation

7.2 Local Model Builder

First, we collect and save data required to build local models on each edge cluster and in the core cluster. To build the communication graphs, we first collect the control plane traffic on each cluster by deploying a DaemonSet of Pods running TShark [51] to capture network trace on each node. We enable the Kubernetes `hostNetwork` option and run the pod as `privileged` to collect the traffic on the host interface. We send all collected traffic to the master Node of the cluster. To avoid using bandwidth, each collection agent filters out traffic unrelated to Kubernetes before sending it. A server on the master node receives all traffic logs as .pcap files and extracts the each connection (i.e., source and destination IP). Then, we map the IP addresses to Kubernetes Pods by querying the Kubernetes API, or marks the communication as external if no mapping is known (i.e., one of the communicants belongs to the core cluster). Communication are saved in a .csv file, where each entry is a communication between two NFs. On the other hand, to build the event dependency graphs, we first collect all logs using the Kubernetes API (`kubectl logs` command) directly from the master node, and keep only event logs. Then, we aggregate and sort all event logs by timestamp (regardless of their node or function of origin) and a custom Python script finds the event dependencies as described in Chapter 6. We save event dependencies in a .csv file, each entry being a relationship identified between two events according to user-defined closeness and frequency thresholds.

We use the Python library Pandas [54] to process the .csv file and generate the initial communication or event dependency graph. Then, we extend each local model and identify external connection or events belonging to the core cluster. Specifically, we first set up an open-source implementation of the PSI algorithm based on [38] as a server on the core cluster. For communication graph, we identify common IP addresses between each edge's communication and the core's NFs. For event dependency graph, dependencies between core and edge events are assessed using the same threshold parameters with the script previously mentioned.

To prune local models, we utilize the *Pandas* [54] and *NetworkX* [36] Python libraries. First, the graphs are converted to a *Network* object, then we call the *descendants* (or *ancestors*) functions of *NetworkX* to assess nodes that are reachable from (or can reach to, respectively) nodes marked as external (i.e., belonging to the core cluster), and remove others. Finally, we anonymize IP addresses using an implementation of Crypto-PAn [43]. On the other hand, events of NF names are anonymized with FF3 [22], a NIST-approved Format-Preserving Encryption (FPE) algorithm.

7.3 Global Model Builder

On the core cluster, we aggregate local models and build a global model. The anonymized local models are securely sent to the core cluster. Global models are constructed by identifying common nodes (for communication graph) or new event dependencies (for event dependency graph). For these, we employ PSI, the *Networkx* and the *Pandas* Python library.

7.4 Challenges

Log Precision. To precisely determine relationship and order between events logs, we rely on timestamp measurement output from each application's logger. However, using the default logs of Free5GC led to several inconsistencies as the default timestamp precision is at the second scale. Although this did not pose challenge within a single NF's logs (as logs are written sequentially to standard output, in the correct order), reliably determining the order of appearance of multiple events from different NFs was made impossible when all had the same timestamp. This behaviour is frequent since events in the 5G core happen at a rapid pace (e.g., a complete UE registration can take less one second and involve a dozen events).

Free5GC does not offer runtime options to modify and increase the precision of their logs, therefore, we modify the source code of the NF applications (in Go) to instead print logs at the nanosecond scale (i.e., changing `TimestampFormat` from `time.RFC3339` to `time.RFC3339Nano`). The change was officially merged into the vendor's official code base.

Multi-cluster 5G Core Kubernetes. We implement CCSM in a realistic environment composed of multiple Kubernetes. Although default Kubernetes environment allow multiple users (tenants) to share one same cluster, tenants still share (e.g., the physical network links, the nodes). For this reason, using different user accounts to represent the core and the edges is not realistic. Instead, we deploy multiple real Kubernetes clusters, each independently managed by one entity (e.g., edge, core). Similarly to real MEC deployment, we connect those different clusters with a backhaul network and the corresponding network routing rules. To allow services and NFs from different clusters to communicate together, we leverage Submariner [46]. In particular, we deploy a service broker on the core cluster, a gateway on each cluster's master node, and a Submariner tunnel between each edge's gateway and the core cluster's gateway. For security reasons, we deploy our clusters in an internal network, using only private interfaces. However, Submariner requires public interfaces (specifically, the interface for the default route) to automatically discover gateway. To solve this issue, we temporary trick Submariner into using our private interfaces for the gateway discovery by setting them as default for the time of the discovery only. To avoid overcomplexity, we ensure that Pods and Services CIDR do not overlap between each clusters, even though Submariner offers a solution (GlobalNet) to overcome this issue. Additionally, we set up Free5GC to work across our multi-cluster Kubernetes environment. We export all Kubernetes services involved in cross-cluster communication depending in the core-edge configuration user (i.e., if UPF is deployed in Cluster A and SMF is in Cluster B, we export both UPF-PFCP, and SMF services IP address, so those can communicate according the 3GPP standard). To that end, we use Submariner `subctl export service` command.

Chapter 8

Evaluation

In this chapter, we present our experimental setup and dataset, then we evaluate CCSM execution time, the efficiency of its pruning module. Finally, we present an evaluation of our event dependency graph through different performance metrics.

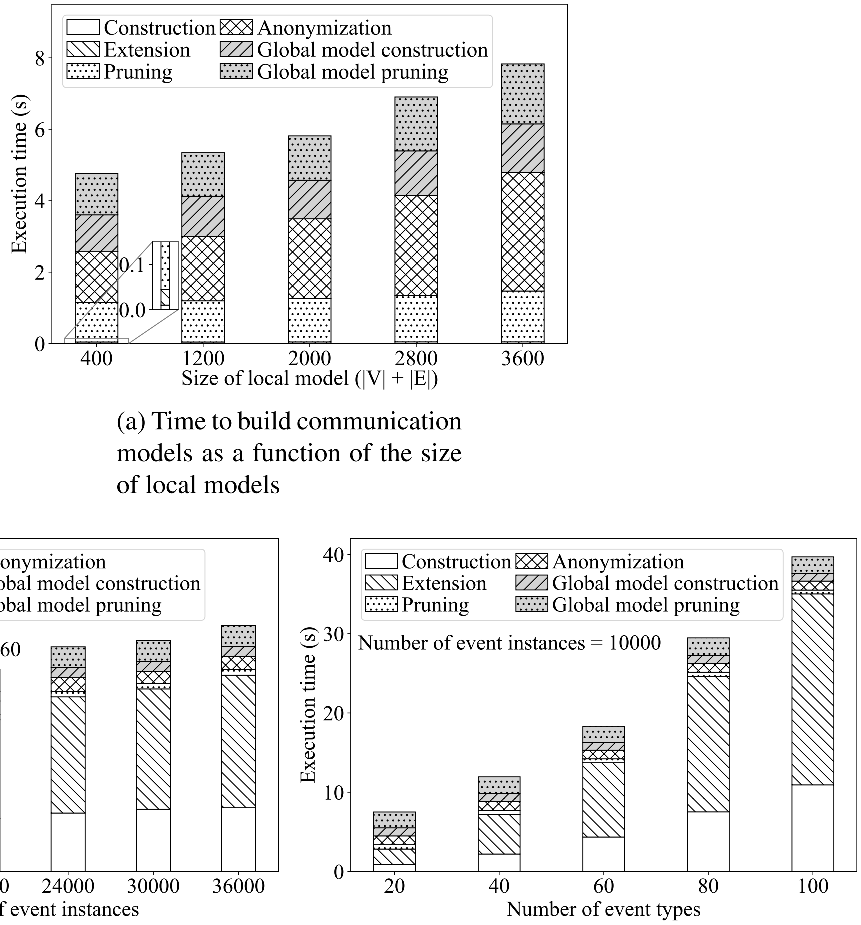
8.1 Experimental Settings and Datasets

We first describe the implementation of the testbed in which we deploy and evaluate our solution, and then we describe the dataset we collected and used as part of our evaluation.

Experimental Settings. CCSM is deployed across two Kubernetes clusters, each composed of one master Node and two worker Nodes. Each Node is a virtual machine with 4 vCPUs and 8GB memory, running Ubuntu 20.04. We use VirtualBox as hypervisor, and we create an internal network for each cluster and an internal network for the communication between clusters (backhaul). We deploy Kubernetes v1.23.14 on top of Containerd v1.4.6 using `Kubeadm` and the recommended installation steps. For the deployment of 5G core, we use a version of Free5GC for Kubernetes [49].

Dataset. We collect data in a real-world 5G core network deployed in Kubernetes using Free5GC. To obtain communication graphs, we use a local model as obtained in Fig. 3 as a seed to generate multiple clusters configuration, each with different NF splitting between edge and core. For the event dependency graphs, we deploy a multi-cluster 5G core network in Kubernetes and add several UEs to trigger 5G network procedures (e.g., registration, de-registration, mobility), then we collect the Free5GC application level logs by running `Kubectl logs` command for each Pods.

8.2 Experiment Results



(a) Time to build communication models as a function of the size of local models

(b) Time to build event dependency models as a function of the total number of events

(c) Time to build event dependency models as a function of the number of unique events

Figure 10: CCSM Execution time

Execution Time. In this experiment, we measure the execution time of our solution. Specifically, we measure the time needed by CCSM to perform each steps depending on parameters that are representative. For these experiments, we only consider one core cluster and one edge cluster.

Fig. 10a shows the time required for building communication models depending on the size of the local models (i.e., the sum of edges and vertices: |V| + |E|), ranging from 400 to 3600 (before pruning). Overall, the execution time of CCSM including *local model building* and *global model building* appears to grow linearly. Some specific steps (e.g., pruning) are almost not impacted by the size of the local models, while others (e.g., anonymization, performed by a third party tools) are more impacted, however all steps are performed in complexity at most linear.

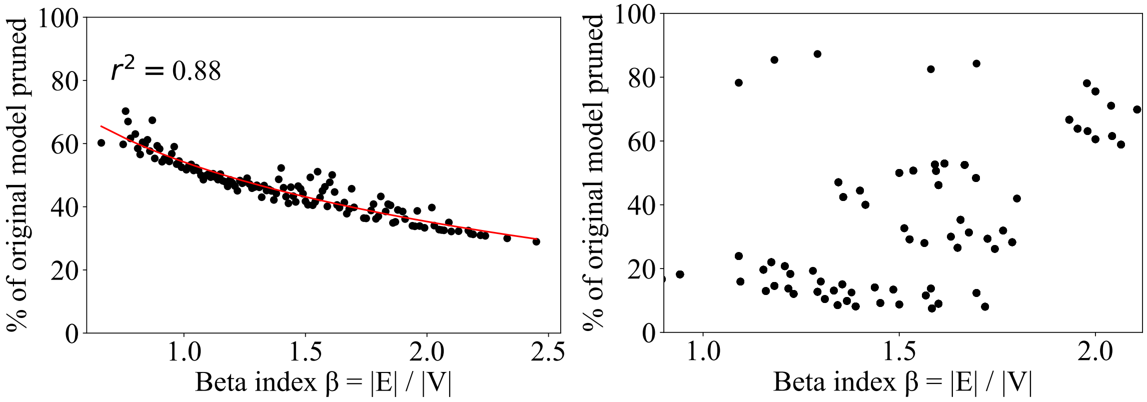
On the other hand, Fig. 10b and 10c show the time required by CCSM to build local and global event dependency models. Because the size of the model is not representative of different types of clusters (i.e., a large cluster can generate very few events, or a small cluster can generate a vast amount of events), we instead study the impact of two other parameters specific to the event dependency models, namely, the number of event instances and the number of unique event types. Fig. 10b presents the execution time of each step for event instances ranging from 6,000 to 36,000 while the number of unique event type is fixed to 60 (This value is number of event types we collected in on our testbed for a common scenario). It can be observed that the trend is linear and the local model extension step if the most time-consuming, as it requires to find cross-cluster dependencies between the edge and core. Additionally, Fig. 10c shows the effect of increasing the number of unique event types, while the number of event instances is fixed to 10,000. The execution time of CCSM appears quadratic, ranging from around 8 s to almost 40 s when the number of event types grows from 20 to 100. This is natural since each new unique event type might form dependency with the n existing event types, therefore going from n to $2n$ event types creates at most n^2 new dependencies.

We conclude that using CCSM in a real-world environment is possible and scalable since our approach is offline and local models are to be built only periodically (i.e., one model can be constructed offline and used at runtime for various use cases). Therefore, we consider an execution time in the range of seconds to be acceptable. It is to be noted that CCSM is by design scalable across multiple clusters, since, the *building local models* steps can be executed in parallel on each individual clusters. On the other hand, although the steps to build global models cannot be executed in parallel, our results show that these scale linearly, thus ensuring our approach is practical even for large clusters.

Efficiency of Pruning. In this experiment, we evaluate the efficiency of pruning techniques utilized to remove the non-necessary nodes and edges. To that end, we measure the size of models before and after pruning, and compare them. Fig. 11 depicts the percentage of original model that was pruned as a function of the local models' *beta* index [19]. The *beta* index is a measure of a graph's connectivity and is calculated as the ratio between the number of edges (|E|) and the number of nodes (|V|) in the graph.

Fig. 11a depicts our results for communication model. As *beta* indices ranging from 0.7 to 2.5, the efficiency of pruning diminishes exponentially. For a *beta* indices of 1 and lower, models are sparse, and pruning efficiently removes nodes and edges that do not have an impact on the global model. Therefore, the complexity of local models is greatly reduced, and the final size of local models average only 50% of their initial sizes. On the other hand, as models become more and more connected (*beta* index > 1.5), more nodes and edges are taken into account, and the pruning is less efficient. It is to be noted that this is not a limitation of our model, but rather a consequence of the complexity of models: CCSM carefully considers all possible dependency between clusters and only prunes nodes that won't have an impact on the final results at all. As a guideline, one can skip the pruning steps if dealing with complex, highly connected models (high *beta* index).

Fig. 11b shows our results with event dependency models. Even though no clear trend can be identified, results show that our approach usually prunes between 10% and 60% of the local model, regardless of its connectivity.



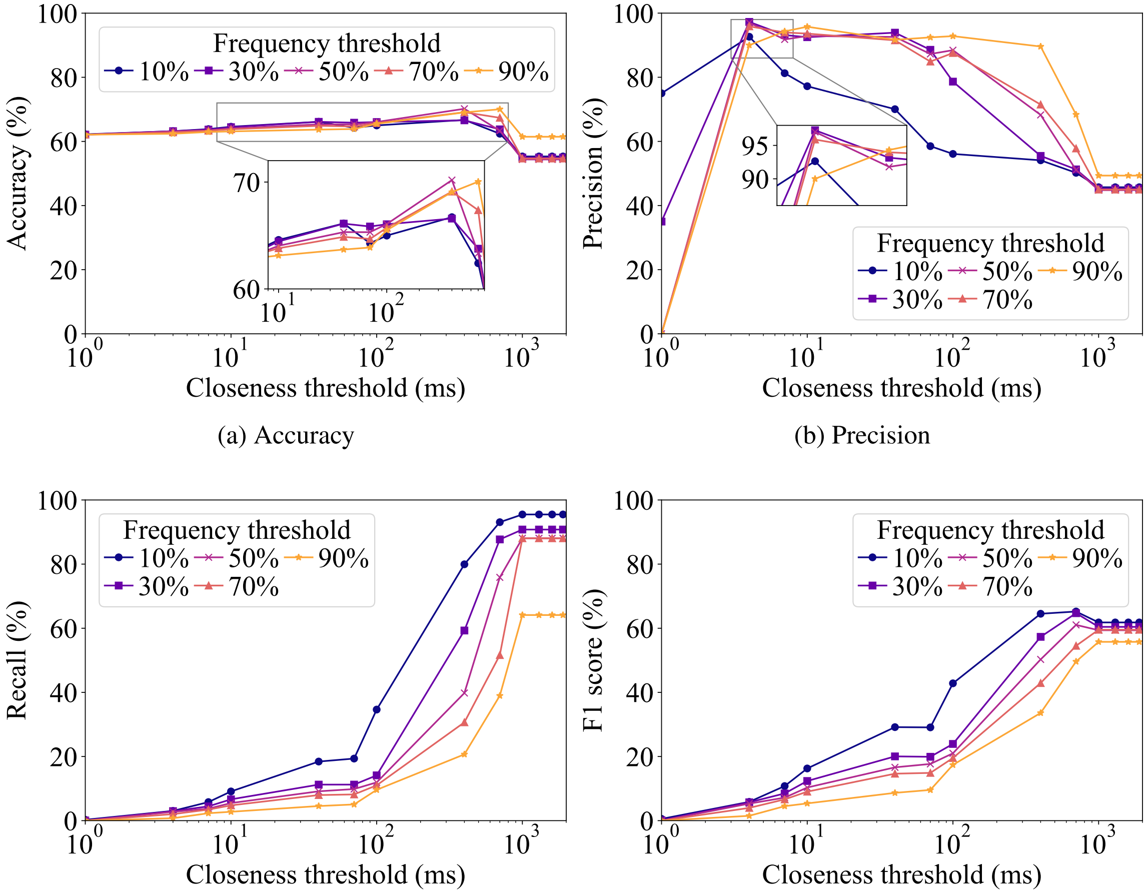
(a) Communication

(b) Event

Figure 11: Pruning efficiency evaluation

Performances of Event Model. In this experiment, we measure the accuracy of our event dependency model. Because these are built in a non-deterministic manner (i.e., based on temporal closeness and frequency of observation), we assess the quality of our method by comparing our event dependency models to a ground truth of event dependency in a 5G core environment (i.e., following the 3GPP standard [2]). For generating the ground truth manually, we analyze and compare the sequence of event logs which was collected on testbed with sequence diagrams, as well as other process interactions and activities outlined in the technical specifications provided by 3GPP in order to find event dependencies. It's important to note that while this approach isn't flawless and perfect but it is the best possible option, considering that practical implementations might deviate from the defined standards or even extend beyond them.

Our goal is to measure to what extent CCSM is able to reconstruct the event dependencies across multiple clusters using closeness and frequency as a way to extend and aggregate local models (as detailed in Chapter 6). Fig. 12 shows the accuracy, the precision, the recall and the F1 score of global model built by CCSM for different closeness and frequency threshold values. We perform these measurements on the cross-cluster event dependencies identified by our solution (i.e., we ignore the dependencies happening within the same cluster as they are trivial to identify, even though such dependencies are also useful for CCSM to extend local models). We measure accuracy, precision and recall as standard criterion's to measure performance of models.



(a) Accuracy

(b) Precision

(c) Recall

(d) F1

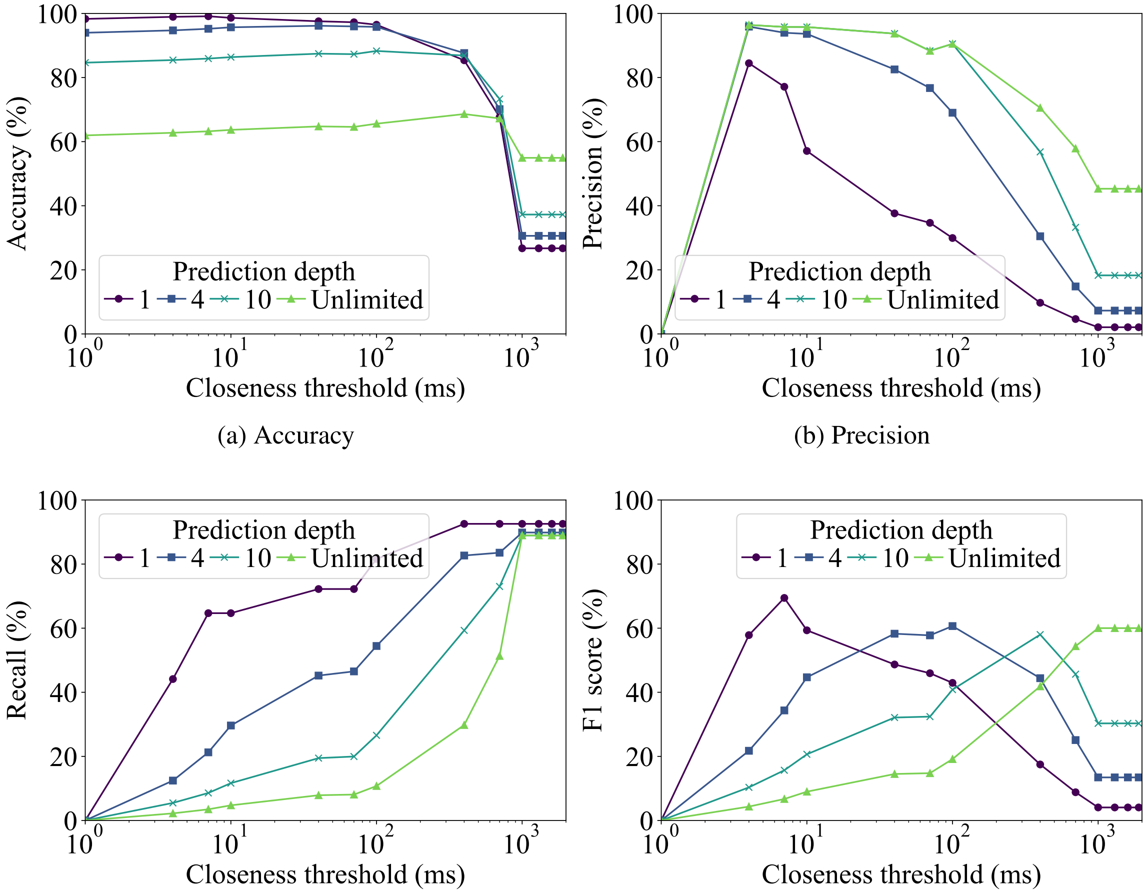
Figure 12: Performance metrics measurement in comparison with the ground truth

Results show that the overall accuracy of CCSM reaches a peak at slightly over 70% with a frequency threshold of 50% and a closeness threshold of 500 ms (i.e., two events are considered dependent if and only if they happen within less than 500 ms at least 50% of the time). Decreasing the frequency threshold overall results in a lower precision (i.e., higher false positive, as we consider two close events to be dependent even if they happen only sparsely over time) but higher recall (i.e., lower false negatives, as we do not miss dependent but rare events). Conversely, increasing the frequency threshold results in higher precision of our approach, but lower recall.

We observe similar behaviour when varying the closeness threshold: increasing it past 1,000 ms suddenly drops the accuracy of our solution since Free5GC procedure (e.g., UE registration/de-registration) typically happen within one second according to our observations. Considering low closeness threshold generally increases the precision (i.e., we solely consider events closely related in time, thus more likely to be really related) but decrease the recall (i.e., events indirectly related are missed if they happen on the long run), whereas high closeness threshold results in more false positives but less false negatives.

We conclude that there exist optimal closeness and frequency threshold values that maximize accuracy, precision or recall. Users can choose the closeness and threshold values that fits the best their requirements. For instance, users interested in event prediction might benefit more from higher recall (i.e., less false negatives) since the increased rate of false positives (i.e., wrong predictions) would only add delay, and might therefore be acceptable. On the other hand, anomaly detection applications might benefit from higher precision (i.e., less false positives) to reduce alert fatigue. We present use cases in more details in Chapter 9.

Event Prediction Evaluation. We extend the evaluation of CCSM event dependency graph model to predict events more or less close in the future. Specifically, we fix the frequency threshold of our solution at 70%, and evaluate the performance of our learned model to predict events at depth 1 (i.e., the immediate next event), 4 (i.e, the next four events), 10, and without prediction depth. Fig. 13 shows that the peak accuracy of 99% is reached when predicting the immediate next event with a short closeness threshold (10 ms). However, in similar conditions, our solution correctly predicts the four next events with 96% accuracy, the 10 next events with 88% accuracy, and finally all future events with 68% accuracy (similarly to Fig. 12). We can observe that CCSM has better precision at predicting events to unlimited depths (i.e., predicted events are more likely to effectively happen within a very large sequence of events, reducing the false positive rate) at the cost of recall (i.e., predicting more events results in larger false negatives).



(a) Accuracy

(b) Precision

(c) Recall

(d) F1

Figure 13: Performance metrics measurement in comparison with the ground truth for various prediction depths (frequency threshold = 70%)

Overall, results emphasizes the impact of user parameters on the performance of our models, depending on the use case. For instance, users willing to predict immediate next events with high accuracy would prefer a closeness threshold comprised between 10 ms and 100 ms, whereas users wanting a sensitive solution over long period (i.e., predict events in the far future with high recall) would prefer a closeness threshold higher than 1,000 ms.

Chapter 9

Use Cases

There are numerous security applications that can benefit from accessing a global view to achieve their objectives. In this chapter, we present major three use cases related to security (namely, cross-cluster security verification, cross-cluster security impact prediction and attack detection) and how CCSM can help addressing them.

9.1 Cross-Cluster Security Verification

Security verification refers to the process of verifying if the application deployed across distributed clusters under different domains complies with policies or predefined rules using the global models. The latter allows validation of properties impacting network communication, and the communication patterns across different NFs in the same or different clusters in different domains while preserving the confidentiality and privacy of information shared between domains. An example of properties could be cross-cluster network isolation, cross-cluster communication pattern, and/or cross-cluster event-based properties. An illustrative example which is shown in Fig. 14 is the verification of isolation of 5G network slices at the cross-cluster level. it shows how CCSM is able to address this challenge to verify network slice isolation at the cross-cluster level.

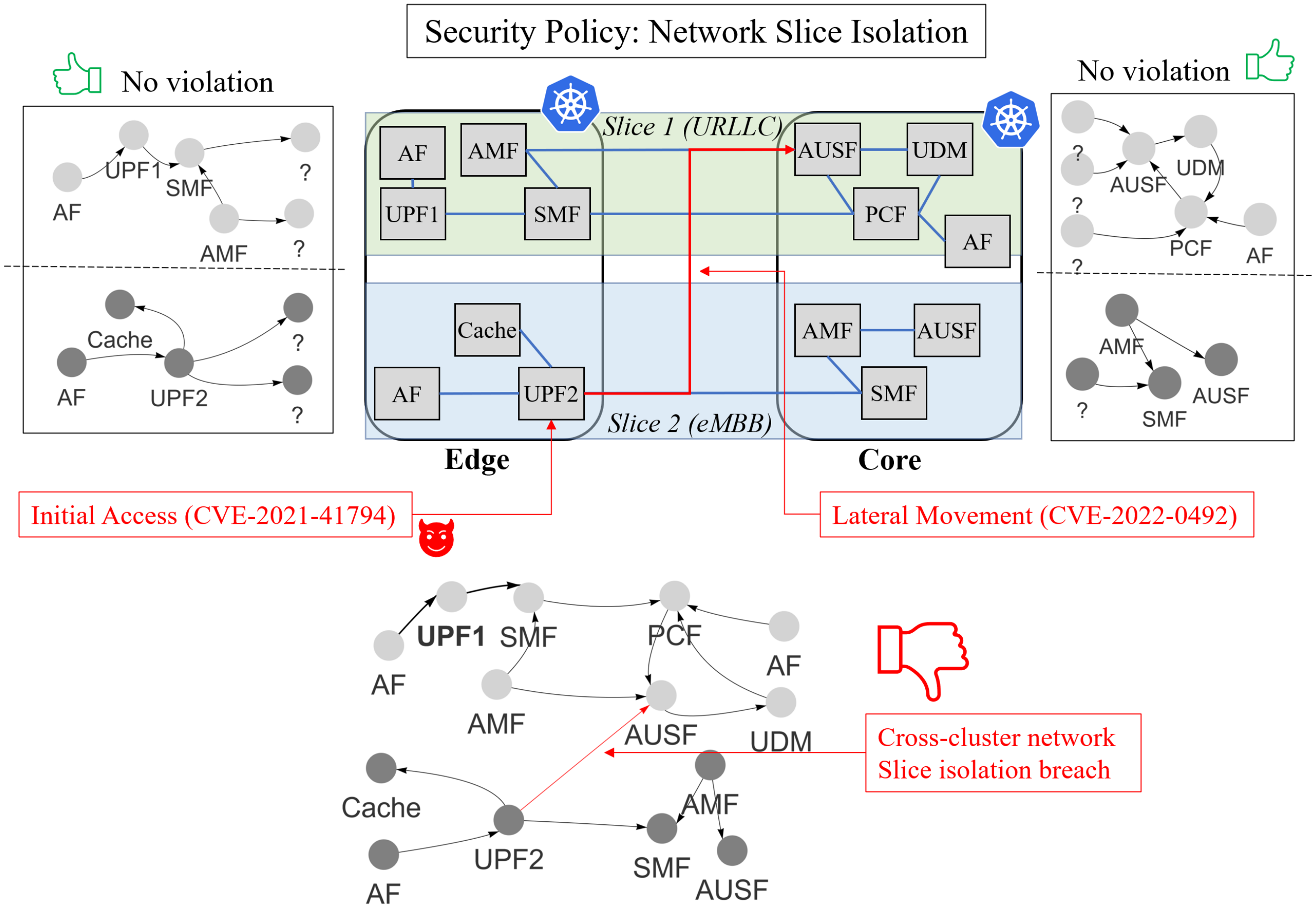


Figure 14: Cross-cluster network slice isolation by CCSM

9.2 Anomaly/Attack Detection

CCSM can also be used to detect anomalies across multiple clusters, that would otherwise go undetected. Fig. 15 depicts an example of scenario inspired from [42] involving a central cloud and two private 5G networks. The private cloud provides connectivity to user on-site, while central cloud (e.g., mobile network operator) provides roaming for private cloud users off-site. Following the authentication procedure in the private cloud, the authentication function (AUSF) of each private cloud transmits cryptographic keys to the management function (AMF) at the public cloud in order to secure the UE's communications while roaming. These extended authentication protocol (EAP)-based procedures are detailed in the 3GPP specifications and can be represented as a global model depicting the ground truth (at the top right corner). CCSM can be used to detect anomalies (e.g., a potential isolation breach) by first constructing the local models in each core cluster, then assembling the global view (depicted in the cloud callout). By observing the difference between the ground truth and the effective models, the three tenants can identifies anomalies. In this example, the collected global model shows that the `AMF UpdateKeys` and `UDR-QuerySMFRegList` events are unexpectedly related to UDR and AUSF events in the second private cloud, potentially indicating a leak of data from the public cloud to private cloud 2 (e.g., cryptographic keys).

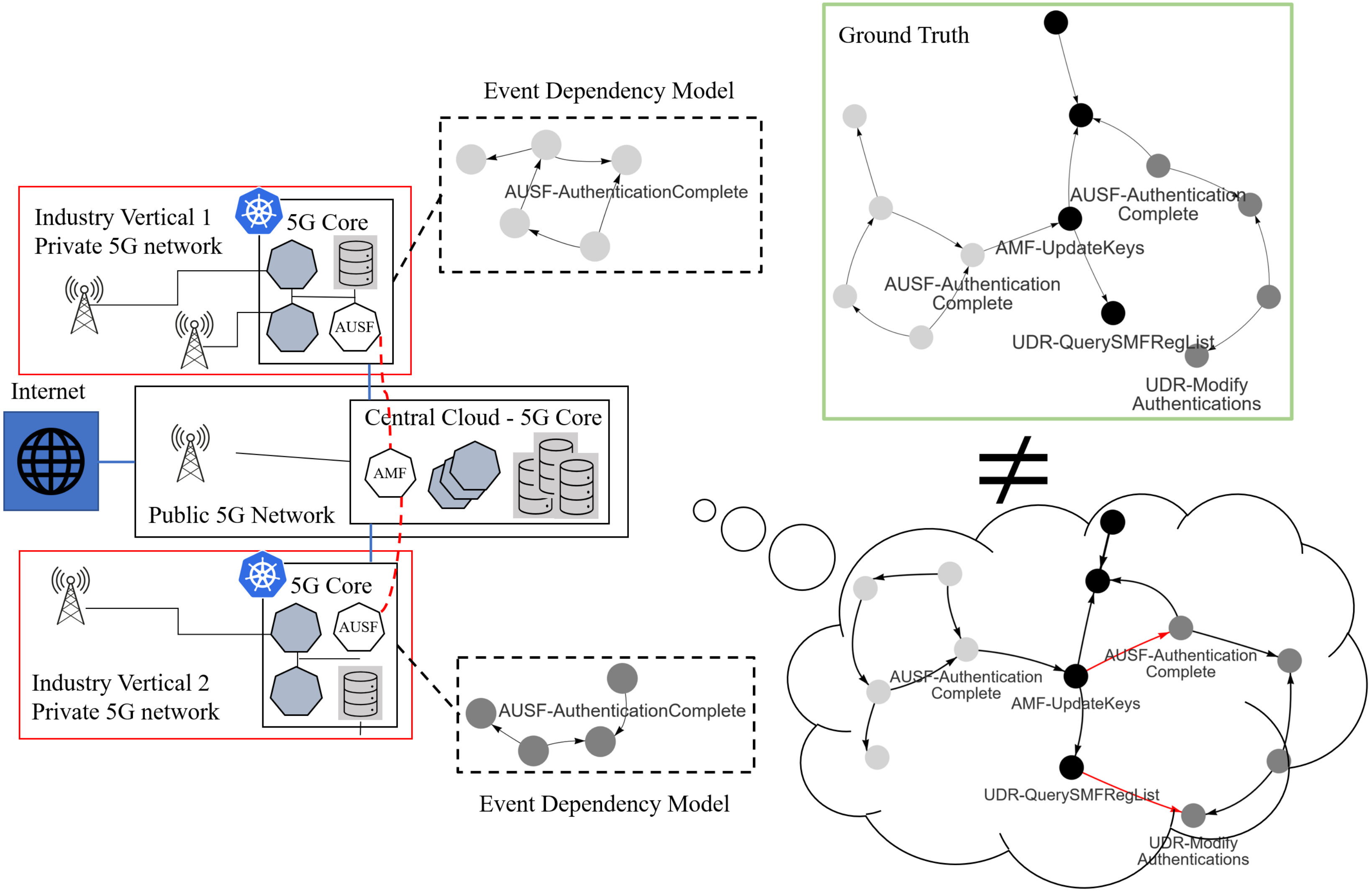


Figure 15: CCSM can help detect anomalies in cross-cluster level

Chapter 10

Other Contributions

.

10.1 ProSPEC

This section presents earlier contributions made to the paper “Proactive Security Policy Enforcement for Containers” (ACM CODASPY 2022), including collecting audit logs and building an event predictive model in a single Kubernetes cluster. These contributions were an inspiration towards building the cross-cluster dependency models that were presented in the previous chapters. In the following, we quickly introduce such predictive models and present experiment results regarding their learning time and accuracy. More details about how such models are built and used can be found in [28].

10.2 Predictive Model

Fig. 16 shows an example of an event predictive model constructed from historical Kubernetes audit logs collected from our testbed. The model learning was done using a Bayesian network [35]. A predictive model is represented as a directed graph where nodes indicate

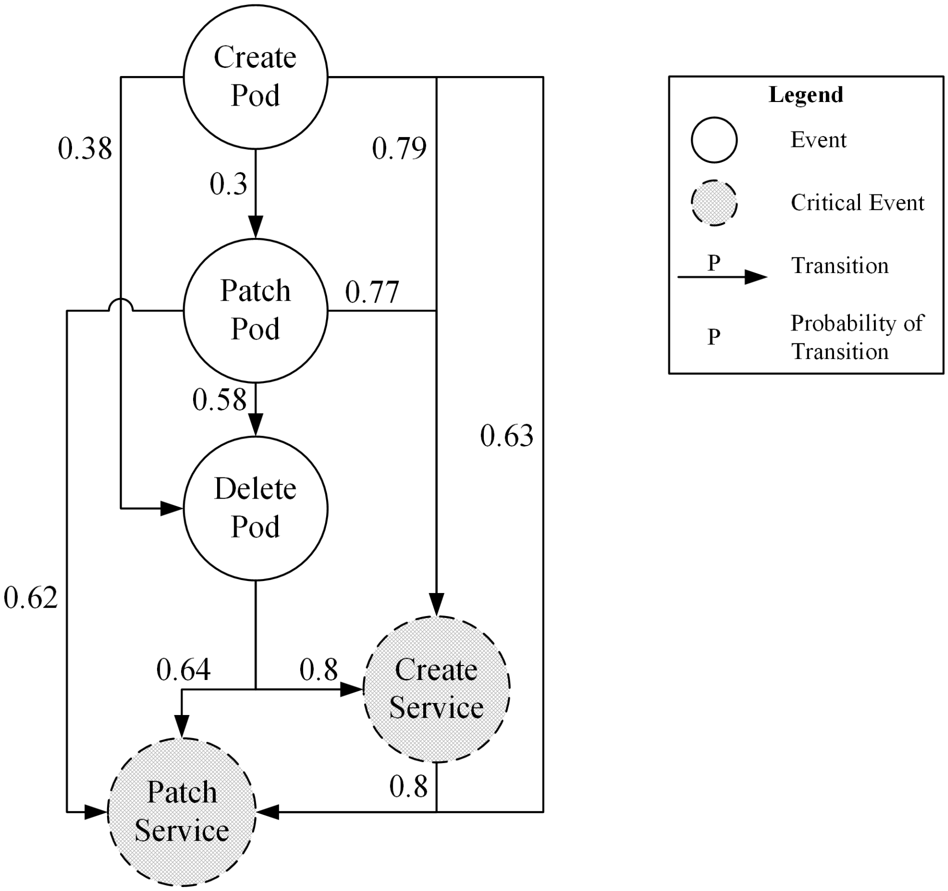


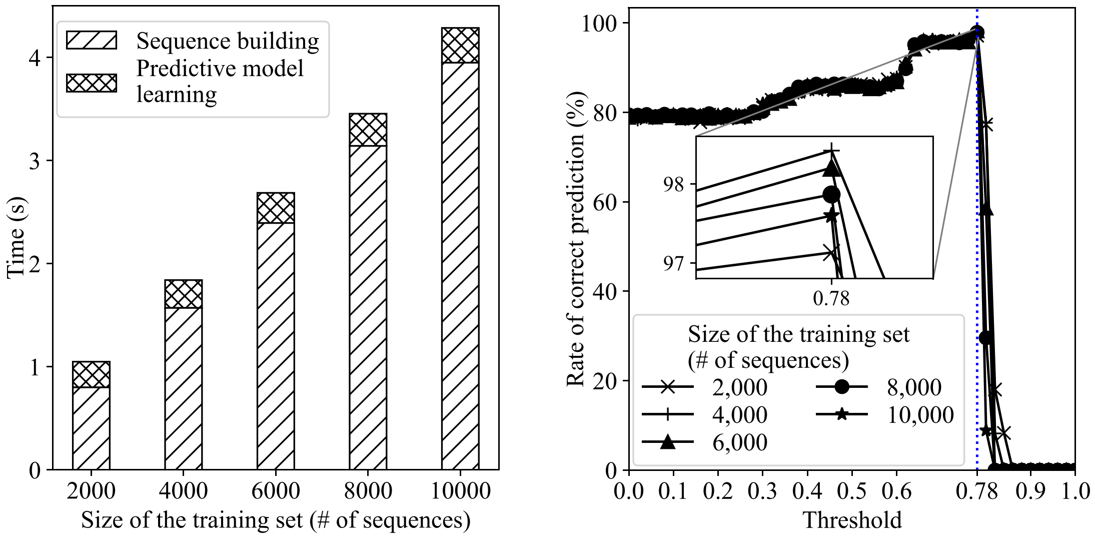
Figure 16: ProSPEC predictive models

container events, edges indicate their transitions, and labels on edges indicate the probabilities of a transition. For instance, a Kubernetes Service has 68% chance of being created after a Pod has been created.

10.3 Experimental Evaluation

The experimental environment is set up on our Kubernetes testbed described in [28]. For both experiments, ProSPEC is running on the master VM and OPA/Gatekeeper inside a container on a worker Node. In the following, we report the evaluation results of the offline learning time and rate of correct predictions of the predictive models.

Offline Learning Time. We measure the offline learning time, i.e., the time required for



(a) Measured time for different learning steps: sequence building and model learning

(b) The correct predictions rates of our model for different thresholds and # of sequences

Figure 17: Learning time and rate of correct predictions of our predictive model (dashed vertical line shows peak rate)

deriving the predictive model from event logs. The measured time in this experiment includes the time to build the event sequences as well as the time to learn the predictive model using the Bayesian network library pgmpy [6]. A preliminary log processing task, performed by Logstash, is not considered in this experiment as in practice it is supposed to be performed in parallel (i.e., while the audit logs are collected from the container environment) and therefore does not impact the learning time.

Fig. 17a shows the time required by the *predictive model building* module of ProSPEC to sequence the logs and build a predictive model while the number of event sequences varies from 2,000 to 10,000. We can see that the time required to perform both of those offline learning steps shows an upward linear trend. The linear trend is less pronounced for the predictive model learning than for the sequence building, as the time needed for the former is much less than that is needed for the latter.

For instance, the time required for model learning increases almost linearly from 248 ms to 337 ms with the increasing number of sequences, whereas the time required for sequence building is increasing from 801 ms to 3,950 ms under similar numbers of sequences. This has a practical implication since the more expensive sequence building only needs to be performed once for each event sequence, while the less expensive model learning may need to be repetitively performed (e.g., when new event sequences are added to the training data). Finally, the overall time reaches about 4 seconds for 10,000 event sequences, which is reasonable especially considering this is an offline step performed only periodically.

Rate of Correct Predictions at Runtime. This experiment is to assess the relation between the rate of correct predictions of our models, user-defined threshold values, and the size of the training set.

During the ProSPEC runtime phase, the chosen threshold for the critical events dictates if a pre-computation will be triggered or not. As the model accuracy will rely on whether we correctly predict critical events or not, it is thus important to show that the chosen threshold has an impact on the overall rate of correct predictions of the model. Note that the best accuracy and corresponding threshold may vary based on different predictive models. The rate of correct predictions is measured for different datasets by varying the number of event sequence from 2,000 to 10,000. 80% of each dataset is used during the training and the remaining is used for testing. For each threshold value, we define the rate of correct predictions as the number of successful predictions over the number of total predictions.

Fig. 17b shows the rate of correct predictions as a function of threshold values for different datasets. We find that the best rate for the used model (as in Fig. 16) reaches 98.4% for a threshold value of 0.78 and a training dataset of 4,000 sequences. However, small differences between different training sets are observed; specifically, it shows that a training set larger than 2,000 sequences does not significantly improve the rate of correct predictions.

Chapter 11

Conclusion

In this dissertation, we presented CCSM, a solution for building cross-cluster security models to enable various security analyses, while preserving confidentiality and privacy for each cluster. Our solution uses models that are employed for security purpose (e.g., communication model and event dependency model) and aggregates them into a global model and global views, respectively, in a confidentiality-preserving manner. The global models and views created can help detecting breaches or attacks happening across multiple clusters that would otherwise be invisible to local models. We apply our solution to a cloud-native 5G core in Kubernetes. Our experiments show that CCSM can learn and aggregate communication models rapidly (8 s for large clouds up to 3,600 edges and nodes), and can adapt to various use cases with high accuracy, precision and recall (up to 99% accuracy, 96% precision and 92.5% recall with different parameters).

Limitations and Future Work. There are a few limitations in our work as follows. First, although our evaluation shows that our approach is theoretically scalable, more experiments can be run to assess the performance of our solution over a large number of edge clusters. Additionally, a larger number of security models can be investigated to extend the security applications of our solution. For instance, communication models presented in this dissertation can be enriched (e.g., with the frequency, or the amount of data exchanged) in order to provide more information to the models’ users based on their requirements, and potentially increase their accuracy.

Bibliography

[1] 5G cloud native. https://www.ericsson.com/en/cloud-infrastructure. [Accessed 4-7-2023].

[2] 3GPP Technical Specification (TS) 23.502 version 16.7.0 Release 16. MITRE ATT&CK Framework. https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf. [Accessed 30-3-2023].

[3] 5G-ACIA. 5G non-public networks for industrial scenarios. *5G-ACIA white paper*, 2019.

[4] I. Ahmad, T. Kumar, M. Liyanage, J. Okwuibe, M. Ylianttila, and A. Gurtov. 5g security: Analysis of threats and solutions. In *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 193–199. IEEE, 2017.

[5] Ali Güngör. UERANSIM, 2023.

[6] A. Ankan and A. Panda. pgmpy: Probabilistic graphical models using python. In *SCIPY*. Citeseer, 2015.

[7] S. Bagheri, H. Kermabon-Bobinnec, S. Majumdar, Y. Jarraya, L. Wang, and M. Pourzandi. Warping the defence timeline: Non-disruptive proactive attack mitigation for kubernetes clusters. 2023.

[8] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-preserving encryption. In *Selected Areas in Cryptography: 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers 16*, pages 295–312. Springer, 2009.

[9] S. Bleikertz, C. Vogel, T. Groß, and S. Mödersheim. Proactive security analysis of changes in virtualized infrastructures. In *ACSAC*, 2015.

[10] T. Brekne and A. Årnes. Circumventing ip-address pseudonymization. In *CCN*, 2005.

[11] T. Brekne, A. Årnes, and A. Øslebø. Anonymization of ip traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies. In *Privacy Enhancing Technologies: 5th International Workshop, PET 2005, Cavtat,*

Croatia, May 30-June 1, 2005, Revised Selected Papers 5, pages 179–196. Springer, 2006.

[12] H. Chen, K. Laine, and P. Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1243–1255, 2017.

[13] L. Cheng, K. Tian, D. D. Yao, L. Sha, and R. A. Beyah. Checking is believing: Event-aware program anomaly detection in cyber-physical systems. *IEEE Transactions on Dependable and Secure Computing*, 18(2):825–842, 2019.

[14] X. Cheng, Q. Luo, Y. Pan, Z. Li, J. Zhang, and B. Chen. Predicting the apt for cyber situation comprehension in 5g-enabled iot scenarios based on differentially private federated learning. *Security and Communication Networks*, pages 1–14, 2021.

[15] CNCF. CNCF Annual Survey 2022., 2022.

[16] M. Corici, P. Chakraborty, and T. Magedanz. A study of 5g edge-central core network split options. *Network*, 1(3):354–368, 2021.

[17] M. Corici, P. Chakraborty, T. Magedanz, A. S. Gomes, L. Cordeiro, and K. Mahmood. 5g non-public-networks (npn) roaming architecture. In *2021 12th International Conference on Network of the Future (NoF)*, pages 1–5, 2021.

[18] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann. Sphinx: detecting security attacks in software-defined networks. In *Ndss*, volume 15, pages 8–11, 2015.

[19] C. Ducruet and J.-P. Rodrigue. Graph theory: Measures and indices. *The geography of transport systems*, 2013.

[20] M. Dworkin et al. Recommendation for block cipher modes of operation: methods for format-preserving encryption. *NIST Special Publication*, 800:38G, 2016.

[21] D. R. Ferreira and D. Gillblad. Discovering process models from unlabelled event logs. In *Business Process Management: 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings 7*, pages 143–158. Springer, 2009.

[22] FF3, 2023.

[23] Free5GC, 2023.

[24] Y. Gao, S. Song, X. Zhu, J. Wang, X. Lian, and L. Zou. Matching heterogeneous event data. *IEEE Transactions on Knowledge and Data Engineering*, 30(11):2157–2170, 2018.

[25] B. Han, A. DeDomenico, G. Dandachi, A. Drosou, D. Tzovaras, R. Querio, F. Moggio, O. Bulakci, and H. D. Schotten. Admission and congestion control for 5g network slicing. In *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6. IEEE, 2018.

[26] J. Kang and J. F. Naughton. On schema matching with opaque column names and data values. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 205–216, 2003.

[27] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, et al. Mec in 5g networks. *ETSI white paper*, 28(2018):1–28, 2018.

[28] H. Kermabon-Bobinnec, M. Gholipourchoubeh, S. Bagheri, S. Majumdar, Y. Jarraya, M. Pourzandi, and L. Wang. Prospec: Proactive security policy enforcement for containers. In *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*, pages 155–166, 2022.

[29] R. Khan, P. Kumar, D. N. K. Jayakody, and M. Liyanage. A survey on security and privacy of 5g technologies: Potential solutions, recent advancements, and future directions. *IEEE Communications Surveys & Tutorials*, 22(1):196–248, 2019.

[30] M. Laroui, B. Nour, H. Moungla, M. A. Cherif, H. Afifi, and M. Guizani. Edge and fog computing for iot: A survey on current research activities & future directions. *Computer Communications*, 180:210–231, 2021.

[31] S. Majumdar, Y. Jarraya, T. Madi, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi. Proactive verification of security compliance for clouds through pre-computation: Application to openstack. In *ESORICS*. Springer, 2016.

[32] S. Majumdar, Y. Jarraya, M. Oqaily, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi. LeaPS: Learning-based proactive security auditing for clouds. In *ESORICS*. Springer, 2017.

[33] S. Majumdar, A. Tabiban, M. Mohammady, A. Oqaily, Y. Jarraya, M. Pourzandi, L. Wang, and M. Debbabi. Proactivizer: Transforming existing verification tools into efficient solutions for runtime security enforcement. In *ESORICS*. Springer, 2019.

[34] M. Mohammady, L. Wang, Y. Hong, H. Louafi, M. Pourzandi, and M. Debbabi. Preserving both privacy and utility in network trace anonymization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 459–474, 2018.

[35] R. E. Neapolitan et al. *Learning Bayesian networks*, volume 38. Pearson Prentice Hall Upper Saddle River, NJ, 2004.

[36] NetworkX, 2023.

[37] OpenStack Congress, 2015. `https://wiki.openstack.org/wiki/Congress/`.

[38] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on {OT} extension. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 797–812, 2014.

[39] W. Y. Poe, J. Ordonez-Lucena, and K. Mahmood. Provisioning private 5g networks by means of network slicing: Architectures and challenges. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2020.

[40] A.-H. Rasha, T. Li, W. Huang, J. Gu, and C. Li. Federated learning in smart cities: Privacy and security survey. *Information Sciences*, 2023.

[41] G. D. P. Regulation. General data protection regulation (gdpr). *Intersoft Consulting, Accessed in October*, 24(1), 2018.

[42] P. Schneider, C. Mannweiler, and S. Kerboeuf. Providing strong 5g mobile network slice isolation for highly sensitive third-party services. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2018.

[43] K. SHIMA. Crypto-PAn, 2015.

[44] M. Song, Z. Wang, Z. Zhang, Y. Song, Q. Wang, J. Ren, and H. Qi. Analyzing user-level privacy attack against federated learning. *IEEE Journal on Selected Areas in Communications*, 38(10):2430–2444, 2020.

[45] Z. Su, Y. Wang, T. H. Luan, N. Zhang, F. Li, T. Chen, and H. Cao. Secure and efficient federated learning for smart grid with edge-cloud collaboration. *IEEE Transactions on Industrial Informatics*, 18(2):1333–1344, 2021.

[46] Submariner, 2023.

[47] S. Sultan, I. Ahmad, and T. Dimitriou. Container security: Issues, challenges, and the road ahead. *IEEE access*, 7:52976–52996, 2019.

[48] N. Sun, J. Zhang, P. Rimba, S. Gao, L. Y. Zhang, and Y. Xiang. Data-driven cybersecurity incident prediction: A survey. *IEEE communications surveys & tutorials*, 21(2):1744–1772, 2018.

[49] Towards5Gs, 2023.

[50] B. Tschaen, Y. Zhang, T. Benson, S. Banerjee, J. Lee, and J.-M. Kang. Sfc-checker: Checking the correct forwarding behavior of service function chaining. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 134–140. IEEE, 2016.

[51] TShark, 2023.

[52] Y. Wei, S. Zhou, S. Leng, S. Maharjan, and Y. Zhang. Federated learning empowered end-edge-cloud cooperation for 5g hetnet security. *IEEE Network*, 35(2):88–94, 2021.

[53] M. Wen, Q. Li, K. J. Kim, D. López-Pérez, O. A. Dobre, H. V. Poor, P. Popovski, and T. A. Tsiftsis. Private 5g networks: Concepts, architectures, and research landscape. *IEEE Journal of Selected Topics in Signal Processing*, 16(1):7–25, 2021.

[54] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *SCIPY*, 2010.

[55] S. Xie, M. Mohammady, H. Wang, L. Wang, J. Vaidya, and Y. Hong. A generalized framework for preserving both privacy and utility in data outsourcing. *IEEE Transactions on Knowledge and Data Engineering*, 35(1):1–15, 2021.

[56] J. Xu, J. Fan, M. Ammar, and S. B. Moon. On the design and performance of prefix-preserving ip traffic trace anonymization. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 263–266, 2001.

[57] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon. Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 280–289. IEEE, 2002.

[58] T.-F. Yen, X. Huang, F. Monrose, and M. K. Reiter. Browser fingerprinting from coarse traffic summaries: Techniques and implications. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 6th International Conference, DIMVA 2009, Como, Italy, July 9-10, 2009. Proceedings 6*, pages 157–175. Springer, 2009.

[59] M. Zoure, T. Ahmed, and L. Réveillère. Network services anomalies in nfv: Survey, taxonomy, and verification methods. *IEEE Transactions on Network and Service Management*, 19(2):1567–1584, 2022.

[60] M.-A. Zöller, M. Baum, and M. Huber. Framework for mining event correlations and time lags in large event sequences. 07 2017.

Appendix A

List of Abbreviations

3GPP	Third-Generation Partnership Project
5G	Fifth Generation of broadband cellular network
MEC	Mobile Edge Computing
GDPR	General Data Protection Regulation
API	Application Programming Interface
NF	Network Function
PSI	Private Set Intersection
FL	Federated Learning
DAG	Directed Acyclic Graph
ETSI	European Telecommunications Standards Institute
NIST	National Institute of Standards and Technology
FPE	Format-Preserving Encryption
AMF	Access and Mobility Management Function
SMF	Session Management Function
AUSF	Authentication Server Function
UPF	User plane function
NRF	NF Repository function
NSSF	Network Slice Selection Function
PCF	Policy Control Function
AF	Application Function
UDM	Unified Data Management
UDR	Unified Data Repository
PFCP	Packet Forwarding Control Protocol
SDN	Software Defined Networks
UE	User Equipment
VM	Virtual Machine