

From the Institute of Information Systems
of the University of Lübeck
Director: Prof. Dr. Ralf Möller

Time Series Data Mining for Context-Aware Event Analysis

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Sciences

Submitted by

Mona Lange
from Hamburg

Lübeck 2016

First referee: Prof. Dr. Ralf Möller

Second referee: Prof. Dr. Stefan Fischer

Date of oral examination: May 24, 2017

Approved for printing. Lübeck, June 1, 2017

Abstract

Data-communication networks contain a multitude of data sources for data mining such as network traffic, vulnerabilities detected by vulnerability scanners or events reported by security sensors such as intrusion detection systems, intrusion prevention systems, or firewalls. Thereby, data are automatically produced within monitored networks. By applying time series data mining techniques, we are able to use these data to provide context-aware event analysis.

In contrast to related work, our context-aware event analysis approach does not focus on modeling an attacker, but aims to automatically learn ongoing workflows and anticipate negative effects of threats. Negative effects of threats could entail network dependencies leading to a chain of events, difficult to anticipate for network operators.

Network traffic analysis allows us to develop a deeper understanding of an event's context within a monitored network. For this purpose we propose to automatically recognize network dependencies from network traffic. To learn network dependencies, we introduce a methodology based on the normalized form of cross correlation. Cross correlation is a well-established methodology for detecting similar signals in feature matching applications. We term the network dependency discovery approach Mission Oriented Network Analysis (MONA). Network dependencies identified by MONA are the foundation for mining workflows based on network traffic. Workflow models describe the underlying dependencies of network devices and network services within data-communication networks. Thus, linking

events to workflows observed within a network, allows us to understand an event's context.

The context-aware event analysis approach introduced by this work is systematically evaluated with real-life case studies conducted within an energy distribution network. In addition, we compare MONA's performance and sensitivity to other state of the art network dependency mining methodologies. This systematic comparison shows that MONA outperforms the state of the art.

Zusammenfassung

Daten-Kommunikationsnetzwerke enthalten eine Vielzahl von Datenquellen für Data-Mining, wie zum Beispiel: Netzwerkverkehr, Schwachstellen, welche von Schwachstellen Scannern entdeckt wurden oder Ereignisse, die von Sicherheitssensoren wie Intrusion-Detection-Systemen, Intrusion-Prevention-Systemen oder Firewalls berichtet werden. Dadurch werden Daten innerhalb überwachter Netzwerke automatisch produziert, und das Anwenden von Zeitreihen-Data-Mining-Techniken ermöglicht uns, diese Daten zu verwenden, um kontextsensitive Ereignisanalyse zur Verfügung zu stellen.

Im Gegensatz zu verwandten Arbeiten, konzentriert sich unsere kontextsensitive Ereignisanalyse nicht auf das Modellieren von Angreifern, sondern zielt darauf ab, fortlaufende Arbeitsabläufe (Workflows) automatisch zu lernen und negative Auswirkungen von Bedrohungen zu antizipieren. Negative Auswirkungen von Bedrohungen könnten auf Grund von Netzwerk-Abhängigkeiten zu einer Kette von Ereignissen führen, die von Netzbetreibern schwer zu antizipieren sind.

Die Analyse von Netzwerkverkehr ermöglicht es uns, ein tieferes Verständnis für den Kontext eines Ereignisses in einem überwachten Netzwerk zu entwickeln. Zu diesem Zweck schlagen wir vor, auf der Basis von Netzwerkverkehr Netzwerk-Abhängigkeiten automatisch zu erkennen. Um Netzwerk-Abhängigkeiten zu lernen, führen wir eine Methodik ein, welche auf der normalisierten Form von Kreuz-

korrelation basiert. Kreuzkorrelation ist ein bewährtes Verfahren zum Ermitteln ähnlicher Signale in Anwendungen, welche übereinstimmende Merkmale suchen. Wir bezeichnen diese Netzwerk-Abhängigkeiten Ermittlungsmethode als Missions-Orientierte Netzwerk Analyse (MONA). Durch MONA identifizierte Netzwerk-Abhängigkeiten sind die Grundlage für das Lernen von Arbeitsabläufen auf der Basis von Netzwerkverkehr. Arbeitsablauf-Modelle beschreiben die zugrunde liegenden Abhängigkeiten von Netzwerkgeräten und Netzwerkdiensten in Datenkommunikationsnetze. Somit erlaubt uns das Verknüpfen von Ereignissen und beobachteten Arbeitsabläufen innerhalb eines Netzwerks, den Ereignis-Kontext zu verstehen.

Der im Rahmen dieser Arbeit eingeführte kontextsensitive Ereignisanalyse-Ansatz wird systematisch mit realen Fallstudien eines Energieverteilungsnetzes getestet. Zusätzlich vergleichen wir Leistung und Sensibilität von MONA mit anderen Stand der Technik Methoden des Lernens von Netzwerk-Abhängigkeiten. Dieser systematische Vergleich zeigt, dass MONA dem Stand der Technik überlegen ist.

Contents

1	Introduction	1
1.1	Research Objectives	2
1.2	Scientific Contributions	4
1.3	Dissemination Activities	5
1.4	Outline of the Dissertation	7
2	Network Dependency Analysis	11
2.1	Introduction	12
2.2	Network Model	13
2.3	Network Service Dependency Analysis	28
	2.3.1 Network Service Dependencies	29
	2.3.2 Normalized Cross-Correlation	34
2.4	Evaluation	40
	2.4.1 Real-life Case Study	42
	2.4.2 Comparative Evaluation	48
2.5	Discussion	66
3	Workflow Mining	71
3.1	Introduction	72
3.2	Workflow Model	75
	3.2.1 Event Logging	75

3.2.2	Probability Space	80
3.2.3	Hidden States Model	85
3.2.4	Hidden Markov Model Workflow	90
3.2.5	Extensions to Factorial Hidden Markov Model Workflow	103
3.2.6	Real-life Case Study	104
3.3	Network Vulnerability Assessment	113
3.4	Discussion	119
4	Event Prioritization and Correlation	121
4.1	Introduction	122
4.2	Method Description	125
4.2.1	Event Normalization	126
4.2.2	Alert Verification and Enrichment	129
4.2.3	Event Fusion	139
4.3	Operational Impact based Event Correlation	145
4.3.1	Network Activities	146
4.3.2	Event Prioritizing	147
4.4	Evaluation	148
4.4.1	Event Verification and Correlation Evaluation	149
4.4.2	Benchmarking Syslog Message Pro- cessing Time	156
4.4.3	Real-life Case Study	158
4.5	Discussion	160
5	Related Work	163
5.1	Mission Impact Modeling	164
5.1.1	Cyber Attack and Defense Modeling	165
5.1.2	Mission-based Event Correlation and Prioritization	166
5.1.3	Mission modeling	168

5.2	General Event Correlation and Prioritization Approaches	171
5.3	Network Dependency Discovery	173
5.3.1	Active Network Dependency Discovery	173
5.3.2	Passive Network Dependency Discovery	174
5.4	Workflow Mining	176
5.4.1	Petri net-based Workflow Models . . .	177
5.4.2	HMM-based Workflow Models	179
6	Concluding Discussion	183
	Bibliography	191
	Curriculum Vitae	207

Introduction

Current conventional network security approaches focus on perimeter protection instead of identifying the most business critical assets and protect those. Although Stuxnet¹ and Flame [MR12] have taught us that in order to protect critical infrastructures against advanced persistent threats, perimeter protection simply is not enough. To underpin this statement, we refer to the director Sean McGurk of the National Cybersecurity and Communications Integration Center (NCCIC) at the Department of Homeland Security²:

“In our experience in conducting hundreds of vulnerability assessments in the private sector, in no case have we ever found the operations network, the Supervisory Control and Data Acquisition (SCADA) system or energy management system separated from the enterprise network. On average, we see 11 direct connections between those networks. In some extreme cases, we have identified up to 250 connections between the actual producing network and the

¹Ralph Langner and Perry Pederson. “Bound to Fail: Why Cyber Security Risk Cannot Simply Be “Managed” Away.” In: *Cyber Security Series*. Foreign Policy at Brookings, Feb. 2013.

²Subcommittee on National Security, Homeland Defense, and Foreign Operations. *Cybersecurity: Assessing the Immediate Threat to the United States*. May 2011. (Visited on 09/30/2015).

enterprise network.”

If data communication networks in cyber-physical systems, such as operations networks, SCADA systems, or energy management systems, are interconnected with enterprise networks, they need to be included in an infrastructure’s operational risk assessment. Taking the overall infrastructure into account provides contextual information for further event analysis. Contextual information allows network operators to determine what ongoing network activities could potentially be threatened by security events or be affected by software vulnerabilities.

In the context of this chapter, initially the research objective is introduced in Section 1.1. Afterwards, scientific contributions are presented in Section 1.2. Then, all dissemination activities are discussed in Section 1.3 and an overall outline is given in Section 1.4.

1.1 Research Objectives

Malicious actors exploiting cyberspace have been identified by the United States intelligence community as the top national security threat³. Similarly, Dell’s annual threat report states a 100% increase in SCADA attacks⁴. This report is based on an analysis of data collected by Dell’s global response intelligence defense network that consists of millions of security sensors in more than 200 countries. Due to the rise of cyber attacks on data-communication networks, cyber security has become a high priority to organization with data-communication networks.

³James R. Clapper. *Statement for the Record, Worldwide Threat Assessment of the US Intelligence Community*. Feb. 2014. (Visited on 05/03/2015).

⁴Dell Inc. *Dell Security Annual Threat Report*. Feb. 2015. (Visited on 10/19/2014).

To protect against this new threat, more and more security sensors such as Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), and Firewalls (FWs) are deployed in order to monitor enterprise networks. Security sensors create a continuous stream of security events that have to be monitored by network operators. In addition, often vulnerability scanners are deployed to continuously monitor enterprise networks for new present vulnerabilities. If a known vulnerability within a monitored network is detected, network operators are notified. Generally, the resources for preventing an exploitation of all reported vulnerabilities are not available. Therefore, network operators are challenged by a time series consisting of large volumes of events occurring at a high pace. In order to face this new challenge, we propose a methodology for context-aware event analysis.

In order to be able to analyze events that occur within data-communication networks, we rely on contextual information to better understand the event. Within data-communication networks, cyber assets (e.g., network devices or network services) interact due to an underlying higher purpose. This common higher purpose that causes cyber assets to interact within a network is also referred to as workflow. Workflows within a data-communication network can depend on multiple cyber assets. Knowing what cyber assets are affected by an event allows us to analyze events based on contextual information. Let us assume a security event suggests that a company's cyber asset might be compromised. If a cyber asset is required to complete an essential task for a company, then a security event targeting this cyber asset should be prioritized over a security event which is targeting a cyber asset contributing to a non essential task for the company. Thereby, we conclude that the context of a security event provides a basis for event analysis.

An event's context consists of the ongoing workflows within the monitored network. Unfortunately, workflows are often not documented, and they are difficult to discover by relying on human expert knowledge (see Chapter 3). However, in monitored networks huge amounts of data are available, and by applying data mining techniques, we are able to extract information of ongoing workflows. From a data mining perspective, we are interested to test the potential of applying data mining techniques to real-life applications.

1.2 Scientific Contributions

As previously mentioned, data-communication networks produce huge amounts of data which are available for data mining. Monitored data-communication networks produce network traffic, streams of events reported by security sensors or vulnerability scanners. Vulnerability scanners report on known vulnerabilities, which are detected within a monitored network. The main goal of the research presented in this thesis is to use data mining techniques to analyze this information. This information is automatically produced within a monitored network, and allows a context-aware event analysis.

The context of an event consists in the workflow which is targeted by an event. This context can automatically be learned from network traffic. Based on this contextual information, events are analyzed based on the targeted workflow. Events consist of reports from security sensors or from vulnerability scanners. The major contributions of this thesis are as follows:

- Inspired by time series data mining techniques, we propose a novel non-intrusive network service dependency

discovery approach referred to as Mission Oriented Network Analysis (MONA). We discuss alternative approaches to MONA and systematically evaluate merits and drawbacks.

- We also introduce network dependency analysis as a foundation for network traffic based workflow discovery. Within a case study based on an energy distribution network we show that the introduced workflow discovery approach can be applied to real-life data-communication networks.
- We show how automatically discovered workflows can be exploited in order to provide a foundation for analyzing events reported by security sensors (IDS, IPS, or FWs).
- We provide correlation techniques for reducing the overall rate of reported events. For these event correlation techniques, we show within a case study based on an energy distribution network that they are able to reduce the overall rate of events reported by security sensors.

1.3 Dissemination Activities

Various parts of this thesis have been published previously in order to disseminate research results. The following list provides an overview of dissemination activities in chronological order.

Published:

Alexander Kott, Mona Lange, & Jackson Ludwig. “Assessing Mission Impact of Cyber Attacks: Towards

a Model-Driven Paradigm.” In: *IEEE Security and Privacy*. IEEE, 2016.

Mona Lange & Ralf Möller. “Time Series Data Mining for Network Service Dependency Analysis.” In: *9th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2016)*. San Sebastian, Spain: Springer-Verlag Berlin Heidelberg, Oct. 2016.

Mona Lange, Felix Kuhr, & Ralf Möller. “Using a Deep Understanding of Network Activities for Workflow Mining.” In: *39th Annual German Conference on Artificial Intelligence (KI 2016)*. ISBN: 978-3-319-46072-7. Klagenfurth, Austria: Springer-Verlag Berlin Heidelberg, Sept. 2016, pp. 177–184.

Mona Lange, Felix Kuhr, & Ralf Möller. “Using a Deep Understanding of Network Activities for Network Vulnerability Assessment.” In: *22nd European Conference on Artificial Intelligence (ECAI 2016)*. ISBN: 978-1-61499-671-2. The Hague, Netherlands: IOS Press, Aug. 2016, pp. 1583–1585.

Mona Lange, Felix Kuhr, & Ralf Möller. “Using a Deeper Understanding of Network Activities for Security Event Management.” In: *International Journal of Network Security & Its Applications (IJNSA)*. June 2016.

Mona Lange & Marina Krotofil. "Mission Impact Assessment in Power Grids." In: *NATO IST-128 Workshop on Cyber Attack Detection, Forensics and Attribution for Assessment of Mission Impact*. Istanbul, Turkey: Information Systems Technology Panel, June 2015.

Mona Lange et al. "Event Prioritization and Correlation based on Pattern Mining Techniques." In: *14th International Conference on Machine Learning and Applications and Workshops (ICMLA 2015)*. Miami, Florida: IEEE, Dec. 2015.

Dissemination activities also included leading an exploratory research team by hosting two workshops (Lübeck, Brussels) and being a guest editor for a special issue called "model-driven paradigms for cyber defense" in the Journal of Defense Modeling and Simulation: Applications, Methodology, Technology:

Mona Lange. "Model-Driven Paradigms for Integrated Approaches to Cyber Defense." In: *NATO IST-ET-094*. , Team Leader , Information Systems Technology Panel, 2015-2016.

In addition a talk on Mission Impact Modeling for Industrial Control Systems was given:

Mona Lange & Marina Krotofil. "Mission Impact Modelling for Industrial Control Systems." In: *1st SCADA Security Conference Latin America*. Rio de Janeiro, Brazil, 2014.

1.4 Outline of the Dissertation

The purpose of the research presented in the context of this thesis is to introduce time series data mining for context-

aware event analysis. Our contribution is divided into different chapters and we provide a brief introductory outline of their content in the following.

Network Dependency Analysis

Chapter 2 shows how a deeper understanding of network activities can be derived by analyzing network traffic. To automatically learn network dependencies from network traffic, we propose a methodology based on the normalized form of cross correlation, which is a well-established methodology for detecting similar signals in feature matching applications. We term the network service dependency discovery approach Mission Oriented Network Analysis (MONA).

To test MONA's ability to identify existing network service dependencies in data-communication networks, we conduct a real-life case study within an energy distribution network. In addition, a comparative evaluation systematically compares MONA to three other network service dependency approaches: Sherlock [Bah+07], NSDMiner [Nat+12] and Orion [Che+08]. The comparative evaluation relies on synthetically generated networks based on the network simulator ns-3⁵. The empirical validation quantitatively evaluates MONA with respect to three other network service dependency discovery methodologies. As described in Figure 1.1, network dependency analysis operates on the level of granularity of network services, applications and network devices.

⁵Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. *ndnSIM: NDN simulator for NS-3*. Tech. rep. NDN-0005. NDN Project, July 2012.

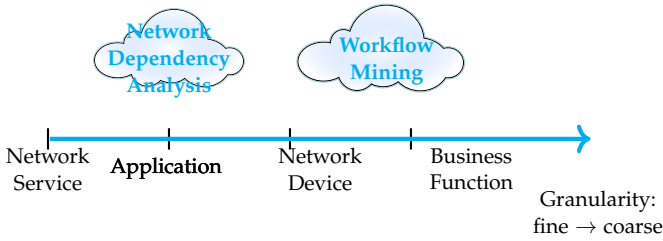


Figure 1.1: Network Dependency Analysis and Workflow Mining are adjacent knowledge domains.

Workflow Mining

Chapter 3 describes workflow mining based on network service dependency discovery. Discovering workflows relies on heuristics to mine structural descriptions. The problem statement of workflow discovery is similar to network dependency analysis, which aims to identify sequences of employed network services. This becomes even more apparent when we are looking into passive network dependency approaches for business-driven IT management [KGE06]. Workflow mining within data-communication networks aims to identify ongoing tasks executed within a company. Thereby, workflow mining operates at the level of granularity of network devices and business functions. Often workflow mining abstracts ongoing tasks beyond the granularity level of business functions. For example, business functions are used to identify a company's structure.

As illustrated in Figure 1.1, we propose that network service dependency analysis and workflow mining are adjacent knowledge domains. Moreover, we introduce time series data mining as a methodology for deriving workflows within data-communication networks.

Event Prioritization and Correlation

To protect data-communication networks, cyber security sensors are deployed to monitor networks for potentially malicious activities. Cyber security sensors report malicious activities in the form of events. Moreover, distinct cyber security sensors often report events in heterogeneous data formats. Unfortunately, often the number of reported events is too high for network operators to monitor them. Therefore, in Chapter 4 we propose event prioritization and correlation to address this issue. In order to analyze events reported by distinct cyber security sensors, we introduce a methodology for normalizing events with heterogeneous data formats. Normalized events are clustered, and we use learned workflows to prioritize events based on contextual information. The introduced event prioritization and correlation approach is tested in a real-life case study conducted within an energy distribution network.

Related Work

Research on context-aware cyber security has led to an ample variety of scientific contributions over the last decades. Hence, in Chapter 5 we give an overview over different approaches to context-aware cyber security and related previous work to the approaches in this thesis as to demonstrate a substantial advance in the state of the art.

Concluding Discussion

Chapter 6 discusses our contribution with respect to related work and with respect to new application areas. Moreover, we present promising directions for future work.

Network Dependency Analysis

To solve the problem of reliably uncovering network dependencies, in the context of this work, we propose a methodology called Mission Oriented Network Analysis (MONA). MONA is non-intrusive, does not require a modification of existing software, and relies on passively collected network traffic. The following chapter is organized as follows: First, a general introduction to network dependency analysis is given in Section 2.1. Second, the underlying network model is introduced in Section 2.2. Third, the proposed network service dependency methodology is discussed in Section 2.3. Lastly, a systematic evaluation is provided in Section 2.4.

For this evaluation, MONA is deployed within the disaster recovery site of an electrical distribution network and evaluated on a ground truth provided by network operators. In addition, we create synthetic networks and insert network dependencies randomly to evaluate MONA in comparison to three state of the art methodologies: Orion [Che+08], Sherlock [Bah+07] and NSDMiner [Nat+12]. A comparative evaluation with NSDMiner, Sherlock and Orion shows that MONA surpasses Orion, Sherlock and NSDMiner signifi-

cantly.

2.1 Introduction

For analyzing how susceptible a network is to software vulnerabilities or attacks, it is essential to understand how ongoing network activities can potentially be affected. A network is built with a higher purpose or mission in mind and this leads to interactions of network devices and applications.

We refer to these interactions with a common purpose as network activities. Network activities can involve multiple applications spanning distinct network devices. This leads to network dependencies due to the following observation: Applications within a common network activity rely on each other to fulfill a common task. If an application is subject to an attack, this can have an impact on other applications within the network activity as they might rely on this application to provide a service.

Currently, network service dependencies can be derived through human labor. However, most enterprise networks are subject to frequent modifications. Modifications in enterprise network for example consist of adding new hardware, deploying new software or removing outdated hardware. In addition, enterprise networks also often rely on applications provided by third parties. Third party software can lead to network operators not being aware of all existing network service dependencies. We, therefore, conclude that knowledge of all existing network service dependencies is often not available.

A deeper network understanding implies knowledge of all network activities within an enterprise network, including understanding how network activities link to applications

and, thereby, network devices. Such network activities result in network service dependencies and we are challenged to derive network service dependencies automatically through data mining-based network service dependency discovery. In order to automatically derive network service dependencies by analyzing network traffic, we first introduce our underlying network model in the following section.

2.2 Network Model

Modeling an IT network requires a basic understanding [EB15] of the Open Systems Interconnection (OSI) model. For understanding network connectivity, the following layers of the OSI model are of particular interest: data link layer, network layer, transport layer and application layer. We define a network device as a physical device on the network, and Media Access Control (MAC) addresses are used to identify network devices. The data link layer physically links network devices using MAC addresses. However, the data link layer only provides point-to-point connectivity. For enabling network connectivity beyond a point-to-point communication, a network layer protocol such as the Internet Protocol (IP) is required. IP addresses are used to identify source and destination of an end-to-end connection. In other words, MAC addresses allow a point-to-point connection, while IP addresses provide an end-to-end connection. Therefore, switches and routers are used to forward packets, i.e., they act as intermediate hosts. Data-communication networks are built with a common higher purpose. This leads to reoccurring interactions between distinct network devices and services, which we call network activity patterns. An example for such a network activity pattern is given in the

following example.

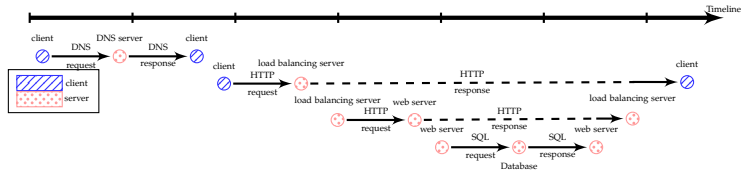


Figure 2.1: Example for network activities.

Example 2.1 (A simple network activity). An example for a network activity pattern in an IT network is given in Figure 2.1. Every node in Figure 2.1 represents a network device. Client network devices are represented in blue and server network devices are represented as red nodes. A majority of communication protocols consist of request and response pairs. Let us assume that a client wants to access a specific web server. Achieving this might require a DNS lookup. Assuming that IP addresses are returned, a load balancing server receives a HTTP request. The load balancing server passes on the HTTP request a web server. The requested information is not available locally, but it is stored in an external database. So the web server sends an SQL request to the database. The database sends the information back to the web server. The web server in its turn sends an HTTP response to the load balancing server, which forwards the HTTP response to the client that initiated the sequence of tasks.

Via network interfaces available on a network device, the network device can be connected to another subnetwork. Hence, a sequence of tasks can require multiple network

devices and subnetworks to be operational. The purpose of network service dependency discovery is to capture these interactions, and to describe how they link to network devices and applications. A network model provides us with the underlying ontology for network service dependency discovery.

Network Device

MAC addresses are necessary to enable a majority of local or metropolitan area networks, defined within IEEE 802 standards, such as Ethernet or WiFi based networking. MAC addresses are used within local or metropolitan area networks to reach every device on the same network (cabled or wireless). To allow routing across distinct networks, the Internet protocol (IP) is used. So when a network device wants to send a packet to another network device, it is first checked whether the target is on the same subnetwork as the network device itself. If it is on the same subnetwork, the destination network device can be reached directly through its *MAC* address for example via Ethernet or WiFi based networking. Otherwise, if the destination network device is not on the same subnetwork, the Internet protocol is used and the network packet is sent to the configured router. Therefore, a network device needs to be assigned to one *MAC* address in order to be able to communicate within a subnetwork. If a network device is supposed to be able to communicate across one or more subnetworks, a network device is assigned one or more IP addresses. In the following, we will start with introducing our network device model.

Definition 2.1 (Network device). A network device is defined as a non empty set of media access control (*MAC*) and Internet protocol (*IP*) addresses *MAC* and *IP*, respectively

$(MAC \cap IP = \emptyset)$, where

$$D \subseteq \mathcal{P}(MAC) \setminus \{\emptyset\} \times \mathcal{P}(IP)$$

is the set of network devices. Additionally, for a given IP address, we are able to derive the corresponding network device by

$$DEV : IP \rightarrow D.$$

◇

Following Definition 2.1 a device can of course be assigned one *IP* address and one *MAC* address or a *MAC* address can be linked to no *IP* address. Assigning multiple *IP* addresses to one *MAC* address and vice versa, allows a device to be assigned multiple *MAC* addresses and *IP* addresses. Being able to assign multiple *MAC* addresses to a network device is needed as routers and switches supply multiple point-to-point endpoints. However, switches do not necessarily need to have *IP* addresses as they work on the data link layer. From this it follows that they are not visible on the network layer. Definition 2.1 allows modeling more complex network device types, for example it allows modeling network devices where one *IP* address is linked to multiple *MAC* addresses as shown in Example 2.2.

Example 2.2 (Network device). A network device d_i as introduced in Definition 2.1, which links a single *MAC* address to a single *IP* address could be

$$d_i = (\{\text{MM:MM:MM:SS:SS:SS}\}, \{\text{XX.XX.XX.XX}\})$$

with an assigned 48 bit *MAC* address MM:MM:MM:SS:SS:SS and a 32 bit IPv4 address XX.XX.XX.XX. The first 24 bits MM:MM:MM represent the manufacturer of a network device with an organizationally unique identifier. The second

24 bits SS:SS:SS represent the network interface controller and are assigned to the adapter by the manufacturer. A network device as introduced in Definition 2.1 can represent multiple *MAC* addresses that are linked to one IP address. An example for this is Google Public DNS. To keep response time low, multiple servers all over the world exist. The closest operational server is determined via Anycast¹ routing. So, two IP addresses both are able to send requests the IP address 8.8.8.8, which is assigned to Google Public DNS, but depending on their closest operational Google Public DNS server, the server answering these requests might have different *MAC* addresses. Similarly, Definition 2.1 allows for modeling a router that links multiple subnetworks.

The network model introduced in this chapter, is used to analyze network traffic. Within network traffic, network devices cannot directly be observed. However, given a network packet is sent by a network device d_j with IP address $XX.XX.XX.XX$, the corresponding network device d_j can be determined by applying function $DEV(XX.XX.XX.XX) = d_j$.

Network Service

A network encompasses devices that are communication endpoints or additional intermediate devices, over which endpoints communicate. The underlying reason for network packets being exchanged are applications hosted on network devices communicating with other applications. Unfortunately, based on observed network packets, it is not possible to directly see which application wants to communicate. Applications communicate through network services, which are linked to transport protocols such as Transmission Control

¹J. Abley, K Lindqvist, and J. Abley. "RFC 4786: Operation of Anycast Services." In: *Internet Engineering Task Force*. 2006.

Protocol (TCP) or User Datagram Protocol (UDP). Within their segment header, TCP and UDP additionally specify a port number $p \in \mathbb{P}$. Network devices that are communication endpoints host applications, which communicate through network services. In the following definition our network service model is introduced.

Definition 2.2 (Network service). Let S be a set of network services (and let transport protocols $\Psi = \{\text{UDP}, \text{TCP}\}$)

$$S \subseteq D \times \Psi \times \mathbb{P},$$

for a network device $d \in D$, a transport protocol $\psi \in \Psi$ and a port number $p \in \mathbb{P}$. Within network traffic, observed network packets can be linked to a source and destination network device, a transport protocol Ψ and a port number from \mathbb{P} . This allows us to define a relation $SERV$, which links a network device, a transport protocol and a port number to a network service:

$$SERV : D \times \Psi \times \mathbb{P} \rightarrow S.$$

To derive all network services hosted by a device d_j , we define the relationship $HOSTS(d_j)$, which returns all network services hosted by d_j .

$$HOSTS : D \rightarrow \mathbb{P}(S)$$

In order to derive the device, on which a network service s is hosted, we define

$$HOSTS^{-1} : S \rightarrow D,$$

and write $HOSTS^{-1}(s_i)$ (slightly misusing standard notation as this not a bijective function). In general, a network service s_i refers to network service number $i \in \mathbb{N}_0$. However, within

examples a network service number can also refer to a port number.

To facilitate linking network services to network devices, we associate service s_i with device d_j by writing s_i^j . Given a service $s_i^j \in S$, the corresponding device d_j can be derived by

$$d_j = HOSTS^{-1}(s_i^j).$$

◇

Example 2.3 (Network service). For example a network service network time protocol (NTP) $s_i^j \in S$

$$s_i^j = (d_i, TCP, 123)$$

is hosted by a network device $d_j \in D$ on port number 123. Let us assume we observe network traffic and see IP address XX.XX.XX.XX receiving a network packet on TCP port 123. Relations *SERV* and *DEV* allows us to link this information to network service s_i^j , by applying $SERV(DEV(XX.XX.XX.XX), TCP, 123)$.

This allows us to derive all involved network services for a given IP-address and port pair by $HSTS(DEV(sIP)) \rightarrow \mathcal{P}(S)$. Based on network traffic analysis we will detect network services and determine how they communicate in an end-to-end manner with each other. Aside from intermediate devices (e.g., routers and switches), network devices can be categorized into client and server network devices. In the following we will refer to client network devices as clients and server network devices as servers. Clients and servers are able to send requests. Servers additionally provide network services that answer these requests. In most

communication networks, the number of clients by far surpasses the number of servers. Generally, requests are sent through ports dynamically assigned by an operating system, while the network service answering these requests are linked to statically assigned ports.

Statically and dynamically assigned Ports

Requests are often sent through a dynamically assigned port. Thus, these ports often change and retaining information about the port number does not provide important additional information. Dynamically assigned ports are chosen from specifically assigned port ranges². Ephemeral port ranges are available for private, customized or temporary purposes. Although IANA recommends ephemeral port ranges to range from $2^{15} + 2^{14}$ to 2^{16} , the range is highly dependent on the operating system. Microsoft assigns ephemeral ports starting as low as 1025 for some windows versions and a lot of Linux kernels have the ephemeral port range start at 32768. We follow the IANA recommended ephemeral port range for clustering purposes.

Definition 2.3 (Cluster Network Service). Let S be a set of services that are hosted by device d_j . All network services communicating through a dynamically assigned port are grouped by

$$s_*^j \in S,$$

as a representative, whereas $*$ represents a dynamically assigned port and j represents the device a network service is hosted on. \diamond

²Joe Touch et al. "Service Name and Transport Protocol Port Number Registry." In: *The Internet Assigned Numbers Authority (IANA)*. 2013.

Known network services have to be linked to ports statically, such that other network services can routinely communicate requests with them. IANA-assigned ports are an example for the effort of standardizing network services with respect to their names and port numbers for network services that run over transport protocols such as TCP, UDP, DCCP, and SCTP. It should also be noted that multiple statically assigned ports could be assigned to the same application.

Network Packet

In the following this network model is used to derive network service dependencies based on network traffic. Network traffic consists of network packets exchanged between network devices through network services. The basic building block of our approach are network packets exchanged between directly dependent network services.

Definition 2.4 (Network Packet). We define the set of network packets P as network services S communicating over time $T \subseteq \mathbb{N}_0$

$$P \subseteq IP \times \mathbb{P} \times IP \times \mathbb{P} \times \Psi \times T.$$

◇

Example 2.4 (Network Packet). A network packet is exchanged by a source and destination IP address $srcIP$ and $dstIP$. Network packets contain a link to transport protocols such as Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). Within their segment header, TCP and UDP additionally specify a port number $\mathcal{P} = \{x : x \in \mathbb{N}, 1 \leq x \leq 65535\}$, which is a 16-bit unsigned integer, thus ranging from 1 to 65535.

For example a network packet $p \in P$ as introduced in Definition 2.4 can correspond to

$$p = (sIP, sPort, dIP, dPort, \psi, t),$$

for source IP addresses sIP , a source ports $sPort \in \mathcal{P}$, destination IP addresses dIP , destination ports $dPort \in \mathcal{P}$, a transport protocol $\psi \in \Psi$ and a timestamp $t \in T$.

As is apparent from the network packet model introduced in Definition 2.4, MAC addresses for source and destination network device cannot be extracted based on exchanged network packets. Often, only an IP address is observed, but network packets still need to be linked to a network device in order to be considered for network service dependency analysis. Definition 2.1 allows linking IP addresses to network devices.

Network Flow

Based on network packets as described in Definition 2.4, we conduct a network dependency analysis based on packet headers (e.g., UDP and TCP) and timing data in network traffic. Network traffic contains network packets that serve different purposes. Some network packets are exchanged for establishing a connection between two network services, other network packets transfer information between two network services. For analyzing network traffic in order to detect present network service dependencies, we are primarily interested in network packets transferring information between network services. Hence, our approach operates on network flows. To identify network flow boundaries, we look into the definition of TCP and UDP flows. TCP flows start with a 3-way handshake (SYN, SYN-ACK, ACK) between a client and a server and terminate with a 4-way handshake

(FIN, ACK, FIN, ACK) or RST packet exchange. If network services communicate frequently, they may forgo the cost of repetitive TCP handshakes by using KEEPALIVE messages to maintain a connection in idle periods. In comparison, the notion of UDP flows is vague, since UDP is a stateless protocol. This is due to the protocol not having well-defined boundaries for the start and end of a conversation between server and client. In the context of this work, we consider a stream of consecutive UDP packets between server and client as a UDP flow, if the time difference between consecutive packets is below a predefined threshold. In our analysis we exclude all network packets that are necessary for establishing a communication between server and client.

Definition 2.5 (Direct Dependency). So given that additional data is exchanged between network service $s_i^j, s_k^l \in S$, which are hosted on network device d_j and d_l , respectively, we term such an end-to-end interaction between two network services as direct dependency. Direct dependencies $SDEP$ are described by

$$SDEP \subseteq (S \times S)$$

for network services $s_i^j, s_k^l \in S$ hosted on network devices d_j and d_l . \diamond

Example 2.5 (Direct Dependency). A specific direct dependency $sDEP \in SDEP$ is denoted as

$$sDEP = (s_i^j, s_k^l)$$

for network services $s_i^j, s_k^l \in S$ hosted on network devices d_j and d_l .

Table 2.1 schematically illustrates direct dependencies that can be observed within Figure 2.1. For example Table 2.1

shows a network device referred to as Client, which hosts a network service that sends a UDP request. Requests are sent from dynamically assigned ports clustered according to Definition 2.3. This UDP request is sent to a network service linked to port 53, which is hosted by another network device called DNS server. This direct dependency is observed at a certain time t . Continuously analyzing network traffic results in a time series of observed direct dependencies.

Source		Destination		ψ	t
Name	Port	Name	Port		
Client	*	DNS Server	53	UDP	2016-08-21T21:30:00+00:00
DNS Server	53	Client	*	UDP	2016-08-21T21:30:00+01:00
Client	*	Load balancing server	80	TCP	2016-08-21T21:30:00+05:00
Load balancing server	*	Web server	80	TCP	2016-08-21T21:30:00+06:00
Web server	*	Database	118	UDP	2016-08-21T21:30:00+07:00
Database	118	web server	*	UDP	2016-08-21T21:30:00+07:50
Web server	*	Load balancing server	*	TCP	2016-08-21T21:30:00+08:10
Load balancing server	*	Client	*	TCP	2016-08-21T21:30:00+09:00

Table 2.1: Direct dependencies within Figure 2.1.

Definition 2.6 (Direct Dependencies). A direct dependency $sDEP \in SDEP$ between network services s_i^j and s_k^l , such that s_i^j is hosted on network device d_j and s_k^l is hosted on d_l , is denoted as

$$SDEP = SDEP^{rq} \cup SDEP^{rsp}.$$

We distinguish requests and responses exchanged between network services based on Definition 2.3. If a network service uses an ephemeral port to send a network packet to a network service on a static port range, we assume it is a request. Thus, an exchanged request $SDEP^{rq}$ is denoted by

$$SDEP^{rq} = \{(s_*^j, s_k^l) \mid s_*^j \text{ sends a request to } s_k^l \text{ in the period under consideration,}\}$$

where k is in the statically assigned port range. Conversely, this means that a network service using its static port range to answer a network service on an ephemeral port is defined as a response. An exchanged response $SDEP^{rsp}$ is written as

$$SDEP^{rsp} = \{(s_k^l, s_*^j) \mid s_k^l \text{ sends a response to } s_*^j \text{ in the period under consideration.}\}$$

◇

Definition 2.7 (First Element of Direct Dependencies). To retrieve the first element of a direct dependency $SDEP$, we define a relation *FIRST*.

$$FIRST : SDEP \rightarrow S$$

◇

Definition 2.8 (Second Element of Direct Dependencies). Similarly, to retrieve the second element of a direct dependency $SDEP$, we define a relation *SECOND*.

$$SECOND : SDEP \rightarrow S$$

◇

Example 2.6 (Retrieve Elements from Direct Dependencies).

A direct dependency $sDEP = (s_i^j, s_k^l)$ the first network service s_i^j is retrieved by applying $FIRST(sDEP) = s_i^j$. Along the same lines, for a direct dependency $sDEP = (s_i^j, s_k^l)$ the second network service s_k^l is retrieved by applying $SECOND(sDEP) = s_k^l$. ◇

Retrieving the first and second element of a direct dependency is required later for deriving indirect dependencies.

In order to facilitate a subsequent analysis of communication patterns in order to automatically derive network service dependencies, a representation of the sequence of exchanged network packets is required. This representation of a sequence of exchanged network packets is referred to as communication histograms.

Definition 2.9 (Communication Histogram). Let us suppose that we analyze network packets $P_i \subseteq P$, as introduced in Definition 2.4, exchanged within a time window $[t_{min}, \dots, t_{max}]$, with start and end time point $t_{min}, t_{max} \in T$ within an IT network. Network services exchanging these network packets P_i form a set $S_i \subseteq S$. To build a communication histogram H , we define a bin size Δ_t and count the numbers of network packets exchanged between two network services in respective time intervals. The number of bins in a communication histogram H is given by $bins = \lfloor \frac{(t_{max} - t_{min})}{\Delta_t} \rfloor$.

All communication histograms are defined by

$$H : S \times S \rightarrow (\{1, \dots, bins\} \rightarrow \mathbb{N}_0).$$

◇

In the prior definition, communication histogram bins $\{1, \dots, bins\}$ are mapped to \mathbb{N}_0 . In other words, every time frame bin contains the number of exchanged network packets.

A communication histogram H for network services $s \in S$ and $s' \in S$ provides an array, containing the numbers of network packets between s and s' exchanged in every time stamp $t \in [t_{min}, t_{max}]$. For every exchanged network packet, assuming it was received during the considered time period $[t_{min}, \dots, t_{max}]$, the corresponding bin tb in the communication histogram $H(s, s')$ is incremented $H(s, s')[tb] ++$. This

is described in Algorithm 1. The corresponding bin tb in the

Algorithm 1 Building communication histograms

```

1: Input:
2: Observed Packets  $\subseteq P$ 
3: start time  $t_{\min} \in T$ 
4: Output: The matrix of a set of communication histograms  $\mathbf{H}$ 
5: bins =  $(t_{\max} - t_{\min}) \text{ div } \Delta_t$ 
6:
7:      $\triangleright$  compute number of bins for communication histogram  $\mathbf{H}$ 
8:      $\triangleright$  all histogram vectors are initialized and filled with zeros
9:      $\triangleright$  fill histogram bins for every observed network packet
10:
11: for all  $(sIP, sPort, dIP, dPort, \psi, t) \in \text{Observed Packets}$ 
12:   do        $\triangleright$  a continuous stream of network packets is observed and analyzed
13:
14:      $tb = (t - t_{\min}) \text{ mod } \text{bins}$ 
15:      $\mathbf{H}(\text{SERV}(\text{DEV}(sIP), \psi, sPort),$ 
16:        $\text{SERV}(\text{DEV}(dIP), \psi, dPort))[tb]++$ 
17: return  $\mathbf{H}$ 

```

communication histogram is determined by $tb = (t - t_{\min}) \text{ mod } \text{bins}$, assuming a timestamp $t \in T$.

Example 2.7 (Communication histograms). To illustrate communication histograms, consider the network activities in Figure 2.1. The timeline denotes the chronological sequence of exchanged network packets with the corresponding communication histogram bins. The communication histograms

are illustrated in an exemplary manner in Figure 2.2.

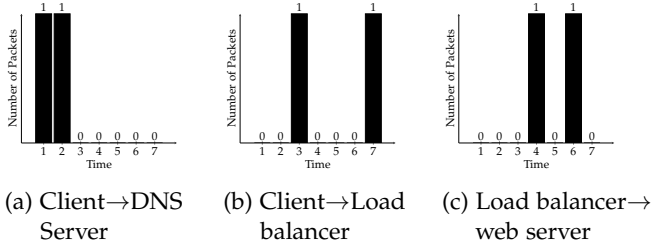


Figure 2.2: Example for communication histograms.

Given that we have now abstracted network traffic into integer vectors referred to as communication histograms, we are now able to look into network service dependency discovery. The network model is the underlying ontology describing our understanding of network devices and how they interact. This allows a deeper analysis of network activities based on network traffic through network service dependency analysis, which is introduced in the following section.

2.3 Network Service Dependency Analysis

The previously introduced network model describes our understanding of network devices, network services and how they directly interact. As stated previously, network devices host applications, which interact through network services. Thus, network services have the primary purpose of communicating for applications. A wide variety of distinct applications exist and often these applications do not

operate independently. On the contrary, applications depend on each other to provide and support network services and, thereby, applications. Hence, applications interact with each other through network services to fulfill a common mission. Thereby, a common mission causes interacting network services to be interdependent with respect to their respective applications. As applications are opaque within network traffic, and network packets can only be assigned to a network service, in the follow we focus on introducing network service dependencies.

2.3.1 Network Service Dependencies

Network services operate on distributed sets of clients and servers and rely on supporting network services, such as Kerberos, Domain Name System (DNS), and Active Directory. To fulfill a network's mission, network services need to interact. Since engineers use a bottom-up influenced divide-and-conquer approach to implement a new task, they are able to reuse network services and do not need to re-implement complex customized ones. This leads to multiple network services interacting for a common high-level task. Let us assume that multiple distinct network services interact to fulfill a common task. If one of these network services becomes unavailable, other network services will be affected over time, since they are dependent on one-another. Potentially, the common task could not be fulfilled (in time). Based on the previously defined network model, which introduces direct dependencies between network services, we expand this notion to network service dependencies. We refer to network services that depend on each other to fulfill a common task as network service dependencies.

Example 2.8 (A simple network activity (revisited)). A net-

work activity is described in Figure 2.1 and shows multiple network service dependencies. For example a client hosts an application, which is linked to a network service launching a DNS request to a DNS server. The DNS server answers the client by sending a DNS response. This is a direct dependency as introduced in Definition 2.6. Based on the answer provided by the DNS server, the client now sends an HTTP request to a load balancing server. The HTTP request cannot be sent by the client to the load balancing server, if the DNS response is not provided by the DNS server. So from the perspective of the client, the task of contacting the load balancing server is dependent on successfully resolving the load balancing server's IP address. Hence, we say the load balancing server is indirectly dependent on the DNS server. The network activity in Figure 2.1 contains more indirect dependencies. The direct dependency between client and load balancing server leads to the load balancing server forwarding the HTTP request to a web server. Therefore, we state that the client is indirectly dependent on the web server. Similarly, the web server retrieves some data through SQL queries from a database. So, the information, which the client requested from the web server can only be provided if the web server is able to retrieve information from a database. Hence, the client is also indirectly dependent on the database.

Unfortunately, indirect dependencies are often not documented and are difficult to discover by relying on human expert knowledge. Therefore, the purpose of non-intrusive network service dependency analysis is to automatically identify indirectly dependent network services based on analyzing network traffic. To automatically learn indirectly dependent network services, we propose a methodology called Mission Oriented Network Analysis (MONA) based

on the normalized form of cross correlation, which is a well-established methodology for detecting similar signals in pattern matching applications.

For the purpose of detecting indirect dependencies *ISDEP*, we analyze the communication histograms of directly dependent network services in order to derive re-occurring communication patterns. Detecting re-occurring communication patterns requires clustering direct dependencies into indirect dependencies. Before aiming to identify indirectly dependent network services, we provide a deeper insight into indirect dependencies. Similarly to previous work, we distinguish two different types of remote-remote dependencies and local-remote dependencies [Che+08].

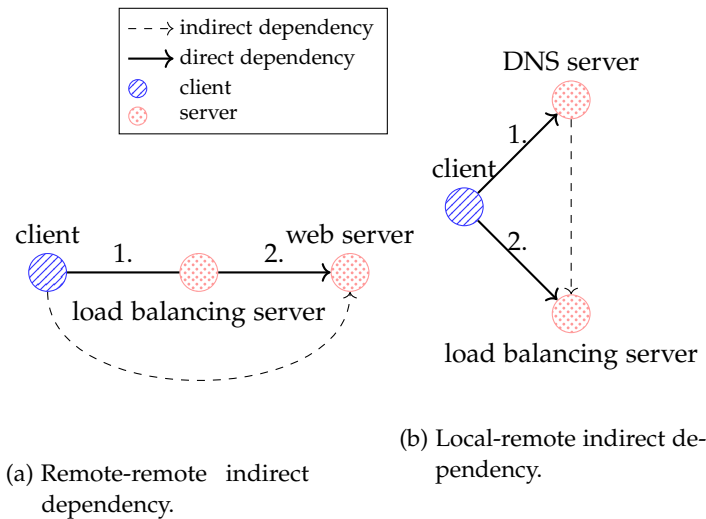


Figure 2.3: Example for indirect dependencies.

A local-remote (LR) dependency is an indirect dependency

joining two direct dependencies, if they fulfill the following criteria: a system must issue a request to a remote system in order to complete an outstanding request issued to a local service. Another indirect dependency type is referred to as remote-remote (RR) dependency and describes the following communication pattern: a system must first contact one host before issuing a request to the desired host.

Definition 2.10 (Local-remote dependency). An LR dependency joins two direct dependencies $SDEP$ (see Definition 2.6), which both have network services hosted by the same network device as a starting point. An LR dependency implies that an application causes requests to be sent to distinct network services, which might or might not be hosted by distinct network devices. LR dependencies $ISDEP_{LR}$ between two direct dependencies $sDEP_i, sDEP_j \in SDEP$, as introduced in Definition 2.6, are described by

$$ISDEP_{LR} = SDEP \bowtie_{HOSTS^{-1}.FIRST=HOSTS^{-1}.FIRST} SDEP.$$

◇

There might be different causes for LR dependencies. For example, Figure 2.3b shows an LR dependency consisting of a system issuing a request to a remote system in order to complete an outstanding request issued to a local service. An LR dependency could also be caused by a monitoring application at regular time intervals requesting current measurement values from remote substations.

Definition 2.11 (Remote-remote dependency). RR dependencies $ISDEP_{RR}$ between two direct dependencies is described by

$$ISDEP_{RR} = SDEP \bowtie_{HOSTS^{-1}.SECOND=HOSTS^{-1}.FIRST} SDEP.$$



Example 2.9 (Remote-remote dependency). An example for an RR dependency is shown in Figure 2.3b consisting of a client requesting information from a web server, which is prefaced with a load balancing server.

Definition 2.12 (Set of indirect dependencies). The set of all indirect dependencies consists of all RR and LR dependencies and is defined as


$$ISDEP = ISDEP_{LR} \cup ISDEP_{RR}.$$



After introducing our model of LR and RR dependencies in Definition 2.10 and Definition 2.11, we will focus on how these indirect dependencies can be derived. Indirect dependencies join two direct dependencies. Consequently, indirect dependencies cannot directly be derived by looking at exchanged network packets. However, based on exchanged network packets, direct dependencies, which are possible candidates for a local-remote or remote-remote indirect dependencies, have to be derived.

Definition 2.13 (Candidates for indirect dependencies). Candidates for indirect dependencies are derived based on direct dependencies and given $sDEP_i = (s_i^j, s_k^l)$, all network services hosted by

$$HOSTS^{-1}(s_k^l) = d_l \text{ or } HOSTS^{-1}(s_i^j) = d_j$$

are candidates for a LR dependency as introduced in Definition 2.10 and RR dependency in Definition 2.11. 

Let us assume that we found a candidate for an LR dependency $((s_i^j, s_k^l), (s_m^j, s_o^n))$. For analyzing whether this candidate for an LR dependency really constitutes an LR dependency, their respective histograms are compared. Both (s_i^j, s_k^l) and (s_m^j, s_o^n) are described by communication histograms as introduced in Definition 2.9. The communication histograms contain the communication pattern of all involved directly dependent network services. Normalized cross correlation finds the best possible alignment between two communication histograms and assess their correlation. Information processing by applications can lead to communication patterns being shifted by t_{delay} . To overcome the lack of a perfect alignment between two communication networks, we extend the Pearson distance to normalized cross-correlation (inspired by [BH01]).

2.3.2 Normalized Cross-Correlation

After introducing the concept of candidates for LR and RR dependencies previously in Definition 2.13, focus of the following subsection is identifying LR and RR dependencies. Communication histograms are signals providing a description of direct dependencies that are joined as candidates for LR or RR dependencies. Given that communication histograms are similar, the direct dependencies are likely to form an indirect dependency. Thus, communication histograms are compared in order to analyze whether two direct dependencies constitute an indirect dependency.

Within signal processing, the Pearson product-moment correlation [ODL07] (PPMC) coefficient has been successfully used to measure how similar two signals are. The PPMC coefficient is a measure for the linear correlation between two variables X and Y . As it measures direction and strength

of a linear relationship between two variables, the PPMC coefficient is also referred to as linear correlation coefficient.

Definition 2.14 (Pearson product-moment correlation). The PPMC coefficient ranges from $[-1, +1]$ and the plus sign denotes a positive linear correlation, whereas a minus refers to a negative linear correlation. The linear correlation $\rho_{X,Y}$ between two variables X and Y by

$$\rho_{X,Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}, \quad (2.1)$$

such that E is the expected value, μ_Z and σ_Z denote the mean and standard deviation of Z . The PPMC coefficient is a measure of the linear correlation between two variables X and Y , giving $-1 \leq \rho_{X,Y} \leq +1$, such that

- $]0, +1]$ is positive correlation: meaning that when X increases, Y has a tendency to also increase.
- $[-1, 0[$ is a negative correlation: meaning that when X increases, Y has a tendency to decrease.
- 0 is no correlation: meaning that when X increases, Y does not tend to increase or decrease.

◇

Two perfectly correlated variables X and Y will have data points, which all lie on a straight line. The corresponding PPMC coefficient for two perfectly correlated variables is $\rho_{X,Y} = \pm 1$. The PPMC coefficient measures whether there is a relationship between two variables X and Y . A mathematical property of the PPMC coefficient to keep in mind is that, the PPMC coefficient does not differentiate between dependent and independent variables, but rather states that there

is a relationship between them. So, the PPMC coefficient is a symmetric measure and will return the same result when comparing $\rho_{X,Y}$ and, vice versa, $\rho_{Y,X}$. Similarly, it should be noted that $\rho_{X,Y} = 0$ indicates there is no relationship between X and Y . Given that two variables X and Y are independent of each other, there consequently is no relationship between the two variables and $\rho_{X,Y} = 0$ would be returned. However, this result does not allow one to deduce that X and Y are independent of each other. The PPMC coefficient can be used to derive a correlation distance called Pearson distance between two variables.

Definition 2.15 (Pearson distance). The Pearson distance is defined as the distance between two variables X and Y

$$d_q(X, Y) = 1 - \rho_{X,Y}, \quad (2.2)$$

for two variables X and Y . ◇

The purpose of the Pearson distance, which is also referred to as Pearson correlation distance, is to measure the similarity between two variables. The Pearson correlation distance allows comparing the distance between two variables X and Y . The Pearson correlation coefficient $\rho_{X,Y}$ ranges between $-1 \leq \rho_{X,Y} \leq +1$, therefore, the Pearson distance lies between $0 \leq d_q(X, Y) \leq 2$. However, due to the fact that negatively correlated variables are not further pursued, the value range for compared variables X and Y is $0 \leq d_q(X, Y) \leq +1$.

In the context of this work, we are interested in using the Pearson distance to compare communication patterns. Communication patterns are stored in communication histograms as described in Definition 2.9. Communication histograms are integer vectors and, thereby, can be compared by applying the Pearson distance introduced in Definition 2.15.

A drawback to the Pearson distance is that it only detects the similarity between vectors that are aligned. Given that two vectors X and Y contain the same shifted pattern, the Pearson distance $d_q(X, Y)$ will classify the vectors X and Y as not being in a linear relationship. An obvious cause for shifted patterns within communication histogram is network latency, as information transfer between distinct network devices takes time. In addition, any information processing by an application can lead to communication patterns between indirect dependencies being shifted.

Example 2.10 (Information processing within an LR dependency). Let us consider a LR dependency $ISDEP_{LR} = ((s_i^j, s_k^l), (s_m^j, s_o^n))$. This LR dependency implies that the data is exchanged between two network services (s_i^j, s_k^l) is processed by an application on network device d_j . Processing this information leads to another network service s_m^j exchanging information with network service s_o^n . Due to the processing of data on device d_j , the request is sent to s_o^n t_{delay} time steps later. Network latency can also lead to the communication patterns describing direct dependency (s_i^j, s_k^l) and (s_m^j, s_o^n) being shifted due to the transfer time of a network packet.

Unfortunately, the Pearson distance as introduced in Definition 2.15 does not account for linearly dependent vectors containing shifted patterns. This is a common problem in template matching applications. Template matching requires developing techniques for finding areas in an image that are similar to a template image. This is a similar problem with respect to comparing two communication histograms that contain shifted patterns. In Template matching, normalized cross correlation [BH01; Lew95] has been successfully used

to solve this problem. Thus, to overcome the lack of a perfect alignment between two communication networks, we extend the Pearson distance, introduced in Equation 2.2, to normalized cross-correlation.

Normalized Cross-Correlation

Whether a shift between communication patterns is caused by network delay or processing time, both root causes entail that communication patterns within an indirect dependency are not perfectly aligned. We, therefore, conduct a comparative analysis of two communication histograms via normalized cross correlation. The communication patterns of both direct dependencies would be similar, although shifted by t_{delay} time steps. In pattern recognition, normalized cross correlation has been proposed to take a shift, such as t_{delay} , into account.

Definition 2.16 (Normalized Cross Correlation). Let us consider two communication histograms r and $s \in [0, \dots, \tau_{max} - \tau_{min}]$ with an overall number of *bins*. We extend Definition 2.2 into normalized cross correlation in order to take a potential shift between r and s into account. The Pearson distance is expanded into the normalized cross correlation $q_{r,s}(\tau)$ to measure the similarity between r and s by:

$$q_{r,s}(\tau) = \frac{\frac{1}{bins} \sum_{t=0}^{bins} (r_t - \mu_r)(s_{t+\tau} - \mu_s)}{\sigma_r \sigma_s}, \quad (2.3)$$

for communication histograms r and s mean value μ_r and μ_s and standard deviation σ_r and σ_s , respectively. Applying Equation 2.3 returns a vector of with *bins* similarity values. To identify the point in time t_{delay} where both signals are

best aligned is found by computing

$$t_{delay} = \operatorname{argmax}_{\tau \in \{0, \dots, (t_{max} - t_{min})\} \subseteq \mathbb{N}} Q_{r,s}(\tau). \quad (2.4)$$

The value range for a time delay $t_{delay} \in \{0, \dots, t_{max} - t_{min}\}$ is dependent on the starting time point t_{min} and end point t_{max} of monitoring network traffic. \diamond

Normalized cross correlation returns a measure of similarity between two communication histograms. Thresholding helps identifying whether two direct dependencies, which are described by two communication histograms r and s , constitute an indirect dependency. A threshold θ is used to automatically identify indirect dependencies by

$$Q_{r,s}(t_{delay}) \geq \theta \quad (2.5)$$

If two communication histograms r and s surpass this threshold, we consider both communication histograms r and s to be correlated and therefore indirectly dependent and shifted by t_{delay} . Normalized cross-correlation is applied to all indirect dependency candidates and returns a set $ISDEP$ consisting of all LR dependencies $ISDEP_{LR}$ and RR dependencies $ISDEP_{RR}$.

Example 2.11 (Indirect dependencies). Based on the example of network activities given in Figure 2.1, multiple indirect dependencies could be identified. These indirect dependencies are shown in Figure 2.4. An example for an RR dependency is the client, who communicates with the load balancing server, who then communicates with the web server. Also, the load balancing server exchanging network packets with the web server, who then goes on to communicate with a data base constitutes a RR dependency. The network activity in Figure 2.1 also contains a LR dependency, which consists

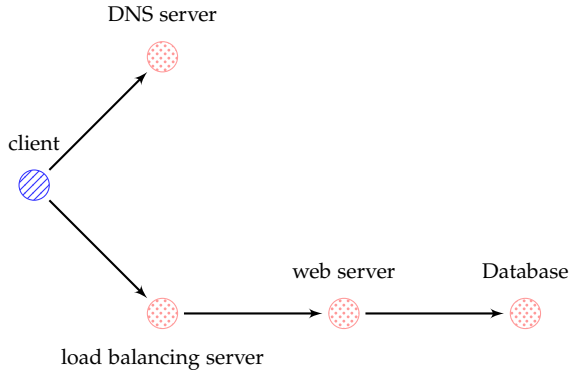


Figure 2.4: Example for indirect dependencies within the network activity shown in Figure 2.1.

of the client resolving an IP address with the help of a DNS server and then using the IP address to contact a load balancing server. Figure 2.4 illustrates an example for RR and LR dependencies.

Based on the network activities shown in Figure 2.1, a schematic illustration of LR and RR dependencies is shown in Figure 2.3. We call this newly introduced network service dependency discovery methodology Mission Oriented Network Analysis (MONA) and evaluate its performance in the next section.

2.4 Evaluation

The disaster recovery site of an energy distribution network, provided by an Italian water and energy distribution company, was available for an experimental evaluation. To allow

network service dependency analysis in the monitored network, network traffic is mirrored using SPAN and RSPAN³ on Cisco switches. SPAN and RSPAN allows a copy of network traffic to be sent to a monitoring network device. As the network service dependency method developed in the context of this work is stream based, we attach our module to the port to which network traffic is mirrored. This allows us to collect and analyze real-life network traffic based on a disaster recovery site of an energy distribution network. Within a critical infrastructure, there are legal restrictions for accessing a production environment that is safety critical. Therefore, after a thorough analysis of the production environment, in addition to the disaster recovery site, additional network devices are emulated. Based on this network, we are able to collect and analyze real-life network traffic.

There are two parts to our experimental evaluation: First, in Subsection 2.4.1 we show the results of a case study within an energy distribution network. Within this case study, MONA was deployed within the data-communication network of an energy distribution network. Real-life network traffic consists of network services frequently to rarely interacting and we are able to determine typical response times. Some network services have a common purpose and show similar communication patterns. As this network is a real-life network, absolute knowledge of all network dependencies is not available. During first experiments on data sets from the disaster recovery site, we often found new network dependencies that had been previously forgotten by the network operators. Therefore, the second part of our evaluation, which is described in Subsection 2.4.2, is

³Richard Froom, Balaji Sivasubramanian, and Erum Frahim. *Implementing Cisco IP Switched Networks (SWITCH) Foundation Learning Guide: Foundation Learning for SWITCH 642-813*. Cisco Press, 2010.

based on synthetically created data sets. We generate synthetic networks based on response times observed in the operational, real-life network and conduct a comparative evaluation with Orion [Che+08], Sherlock [Bah+07] and NS-DMiner [Nat+12]. In addition to allowing response times to be varied, synthetic networks allow experimenting with network size, number of direct and indirect dependencies, and the number of exchanged network packets.

2.4.1 Real-life Case Study

For our case study the disaster recovery site of an energy distribution network was available and provided a test environment for network traffic analysis. The test environment provided a continuous stream of real-time network traffic, and MONA was deployed within this test environment. All direct dependencies observed by MONA within the test environment are presented in Figure 2.5 by edges. Nodes represent network services in this representation and we replace IP addresses by host names to allow for easier readability. Separated by a colon, ports are appended to the host names. To illustrate the different subnetworks within the monitored network, nodes are colored depending on what subnetwork they are located in. The legend lists all subnetworks present within the experimental environment.

Direct Network Service Dependencies

Figure 2.5 shows a network device named `mferp2`, which is a communication server. This communication server `mferp2` is connected to multiple substations, which are identified as `TTY-T[125-158]`. The communication server `mferp2` hosts a network service (introduced in Definition 2.2). This network service belongs to an application, which sends requests (see

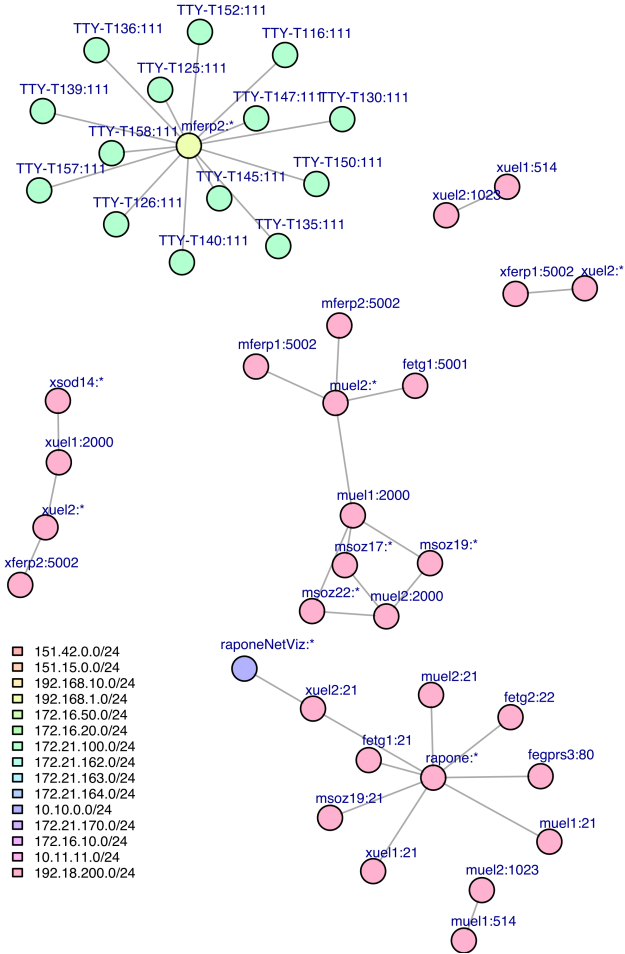


Figure 2.5: Direct network service dependencies in an energy distribution network.

Definition 2.6) to all substations in order to be updated with current measurement information.

In Figure 2.5 the node `mferp2:*` represents the network service hosted by the network device `mferp2`, which sends requests to port 111 of substations `TTY-T[116-158]`. Network device `mferp2` has two distinct IP addresses that are located in two different subnetworks `192.168.1.0/24` and `192.18.200.0/24`. Aside from sending requests to multiple substations, `mferp2` receives requests from SCADA server `muel2` on port 5002. SCADA server `muel2` also sends requests to `mferp1` on port 5002 and communication gateway `fetg1` on port 5001.

Human Machine Interfaces (HMI) for medium voltage substations `msoz17`, `msoz19` and `msoz22` are able to send requests to Supervisory Control and Data Acquisition (SCADA) servers `muel1` and `muel2` on port 2000. Another HMI `xsod14` is in charge of supervising high voltage substations and sends requests to scada server `xuel1` on port 2000.

The experimental environment also consists of emulated network devices, for monitoring and control purposes two network devices `rapone` and `raponeNetViz` were added. As the purpose of this evaluation is to focus on network service dependencies found within real-life energy distribution networks, for the purpose of further evaluating detected indirect service dependencies, these two network devices were excluded from further evaluation. Although this monitored network is a SCADA network, the mission of the analyzed network is irrelevant and the same methodology can be applied to all TCP/IP and UDP/IP based data-communication networks.

Indirect Network Service Dependencies

Direct network service dependencies, as shown in Figure 2.5, are the basis for detecting indirect network service dependencies. Direct network service dependencies imply that network packets are exchanged between network services. Therefore, complete knowledge of all currently existing and non-existing direct dependencies is given, assuming that all network traffic within a monitored network is mirrored. Complete knowledge of all existing and non-existing indirect network service dependencies is more difficult to attain. Real-life data-communication networks are dependent on third party software and operators do not have complete knowledge. Especially in critical infrastructures, often entire subnetworks are built and maintained by third parties.

In the context of this experimental evaluation, the ground truth of all existing indirect dependencies within this experimental environment is derived with the help of network operators. Monitored network devices are listed by network operators and all other network devices - including network services hosted on these network devices - are excluded from further analysis. For monitored network devices, network operators list all indirect dependencies known to them. Within the time period 28% of the analyzed network traffic involved non-monitored network devices and, thereby, this communication was excluded from further analysis. Non-monitored network devices within the experimental environment are unknown to the network operators involved within our research project. This precaution was taken in order to ensure a known ground truth for existing and non-existing indirect dependencies is given.

The ground truth consisting of a list of all monitored network devices and existing indirect dependencies between

them, was derived before deploying MONA in the test environment. Knowledge of indirect dependencies detected by MONA could bias network operators in the sense that they include indirect dependencies detected by MONA into their list of existing indirect dependencies within the test environment. Thus, the ground truth was derived before discussing MONA's detected indirect dependencies with network operators to avoid network operators being biased. Figure 2.6 shows all indirect network service dependencies that MONA in the experimental environment.

All identified indirect network service dependencies were classified as true positives. However, absolute knowledge of all existing and non existing network dependencies can only be assumed. This is why we will additionally evaluate MONA based on synthetic networks in Subsection 2.4.2.

Our evaluation illustrates how important Definition 2.1 is. Network device mferp2 is one physical device, but is assigned two IP addresses from two different subnetworks. Therefore, we linked the nodes representing both network services involving mferp2 by adding an edge, although they are hosted within two different subnetworks. Multiple LR dependencies join communication server mferp2 to multiple substations TTY-T[116-158]. Additionally, mferp2 communicates via muel2 with Human Machine Interface (HMIs) msoz19 and msoz22. Another HMI msoz17 wants to access information about the substations TTY-T[116-158]. For this, first muel1 is contacted, who passes the request on to muel2. As an energy distribution network is a critical infrastructure, it needs to be ensured that all communication pathways are always available. Hence, regularly, back up servers and alternate communication pathways are tested, even if no information needs to be transmitted. All indirect network service dependencies detected by MONA (and shown in

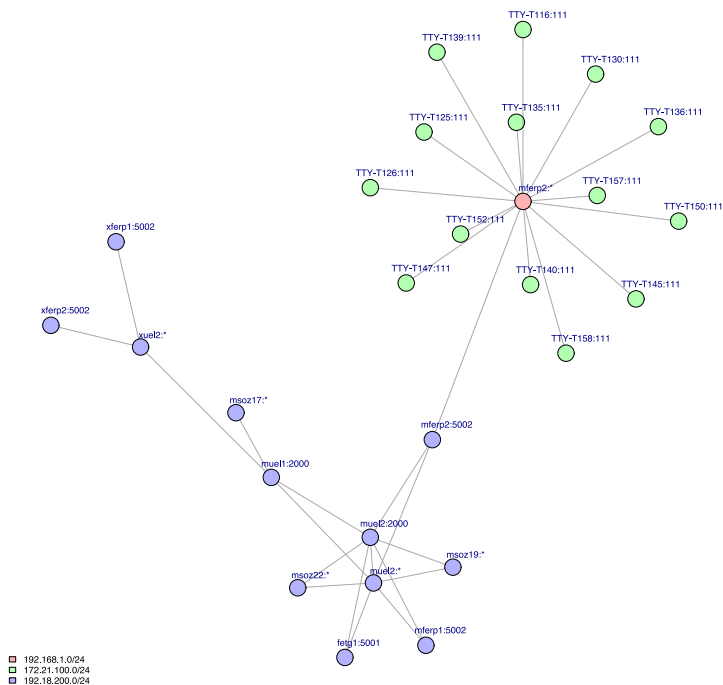


Figure 2.6: Network service dependency detected by MONA within an energy distribution network.

Figure 2.3) were verified by operators as true positives. To allow a more in-depth investigation of MONA’s performance and sensitivity, we expand our experimental evaluation to synthetic networks in the following subsection.

2.4.2 Comparative Evaluation

As every network relies on third party software, which might have their own network dependencies unknown to network operators, collecting a complete ground truth of a network is very difficult. Also, one network is not enough to investigate performance and limitations of a network service dependency detection methodology. A synthetic network generator allows for varying network size, varying number of direct and indirect dependencies, and communication patterns can be varied as well.

To ensure the synthetic network generates realistic communication patterns, we extracted communication patterns of known network service dependencies from the experimental environment in Subsection 2.4.1. Communication patterns vary depending on the number of network flows exchanged per communication between indirectly dependent network services. We used this information to develop a random network generator based on the network simulator ns-3⁴. We added some random variations to the communication delays to mimic noise, which is always present in real-world applications. The developed random network generator can create synthetic data sets with a known ground truth for evaluating our network dependency analysis.

Network simulation is widely used to design and evaluate new protocols and applications. The chosen network simulator in the context of this work is ns-3. and it has been used in numerous domains from simulating peer-to-peer, wireless and ad-hoc networks, business processes or peer-to-peer network. Figure 2.7 illustrates the network simulation as an ns-3 module. The ns-3 module is comprised of

⁴Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. *ndnSIM: NDN simulator for NS-3*. Tech. rep. NDN-0005. NDN Project, July 2012.

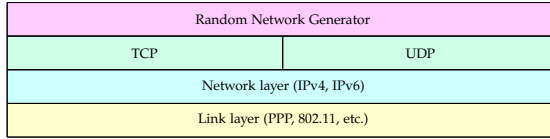


Figure 2.7: Network layer model of the ns-3 based random network generator.

a random network generator, which is built to run on top of any available link-layer protocol model/on top of network-layer/transport-layer protocols. For our experimental evaluation, link, network, and transport layers are provided by ns-3. Based on the random network generator, we randomly generate networks with varying communication patterns and a predefined number of network nodes and indirect dependencies. The focus of the experimental evaluation based on synthetic networks is to conduct a detailed analysis of MONA's performance and sensitivity compared to other state of the art network service dependency methods.

Comparable Network Dependency Analyzers

We chose two state of the art network service dependency analyzers named Orion [Che+08], NSDMiner [Nat+12] and Sherlock [Bah+07] for a comparative evaluation with MONA.

Orion

Orion [Che+08] introduces the terminology of local-remote and remote-remote dependencies also relied on in the context of this work and leverages the delay distribution between network services to infer network dependencies. Orion infers network dependencies between two network services

when the delay between consecutive accesses follows a constant pattern. As an example for such a constant pattern, consider an application, which needs consecutive access to two distinct network services. Then, the delay between to dependent network services will follow a non-random distribution. It has already been pointed out previously [Mar13] that Orion contains several fixed constants that are selected without prior data calibration, which might lead to shortcomings in the robustness of the proposed solution. Also, Orion requires a minimum flow count to analyze the delay distribution of network services. Within the context of this work, Orion was reimplemented according to the algorithm described in the context of [Che+08]. Therefore, we will evaluate Orion's and MONA's robustness with respect to thresholds in the following.

NSDMiner

NSDMiner [Nat+12] is another methodology for network service dependency discovery. NSDMiner locates nested connections, wherein one complete request-response pair starts and completes between the request and response of another connection. So the located nested connection has to match an expected recursive connection pattern. Network flows are monitored for their chronological order, and NSDMiner detects a network service dependency when the probability that the life span of connections to one network service is included in the life span of a connection to another network service is higher than a predefined threshold. NSDMiner focuses on detecting local-remote dependencies.

Sherlock

Sherlock [Bah+07] infers network service dependency based on co-occurrence within network traffic. Similar to NSD-Miner, Sherlock builds on the notion of nested connections. However, unlike NSDMiner, Sherlock learns LR and RR dependencies. As a result, the strength of a network service dependency is computed as the probability of a network service being accessed within a time interval in which another network service is accessed. It focuses on detecting remote-remote dependencies and leverages packet capture running at each end-host to infer dependencies. After inferring network service dependencies, a directed dependency graph is built, modeling network device states as up, troubled, or down. Within the context of this work, Sherlock was reimplemented according to the algorithm described in the context of [Bah+07].

After introducing three state of the art network service dependency methods that can be compared to MONA, an effective testing method for comparing these four methods is needed. As a score for testing the methodology, we rely on the F-measure, the weighted harmonic mean of Recall and Precision, hence we shortly discuss the F-measure score in the following.

Testing Methods

To allow a comparative evaluation of all network service dependency methods, we randomly generate network traffic with the previously described ns-3 based module. For an evaluation, it is important to have a known ground truth. Ground truth means that each data-communication network contains network traffic between directly and indirectly dependent network services, and we have knowledge of all

existing network service dependencies. We focus the experimental analysis on determining how correct the learned network service dependency model is in comparison to other network dependency analyzers. As we are able to control the direct and indirect dependencies in the network, we are able to analyze precision and recall of the learned model.

Precision and Recall

True Positive (TP) refers to a correctly learned indirect dependency, False Positive (FP) refers to a learned indirect dependency that is false and False Negative (FN) is an existing indirect dependency that was not found. Based on these values we can compute Precision and Recall with the following definitions.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

We are equally interested in maximizing precision and recall.

F-measure

To evaluate whether an evaluated network service dependency method equally maximizes precision and recall, we rely on the F-measure. The F-measure combines precision and recall into a common measure

$$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

which represents the weighted harmonic mean of recall and precision. The F-measure allows differential weighting of recall and precision, however as we place equal importance

on precision and recall, we balance precision and recall equally.

Test Environment

All of the following experiments with synthetic data sets were executed on the same architecture. The environment used for the evaluation is the following:

- Virtual machine on Macbook Pro, 2,8 GHz Intel Core i7, OSX 10.10.4 (14E46)
- OS: Ubuntu Release 12.04 (precise) 64-bit, Kernel Linux-3.13.0-32-generic
- CPU: Intel Core i7-4558U @ 2.80GHz (1 processor)
- RAM: 8 GB, 1600 MHz DDR3

Performance Evaluation

Based on the previously described random network generator, we are able to conduct experiments to test performance and sensitivity of the network service dependency discovery methodologies.

To provide a proof that our proposed methodology is sound, we first evaluate the performance with respect to all approaches. Afterwards, a sensitivity analysis will evaluate the robustness of MONA's threshold compared to Orion's in order to address potentially limiting factors.

In order to analyze the quality of all network service dependency discovery methods, we generate a data-communication network containing 100 network devices with 30 indirect dependencies. We adjusted the threshold for all methodologies to reflect the best possible result.

The results of this evaluation are shown in Figure 2.8 and illustrate how MONA outperforms Orion, Sherlock and NS-

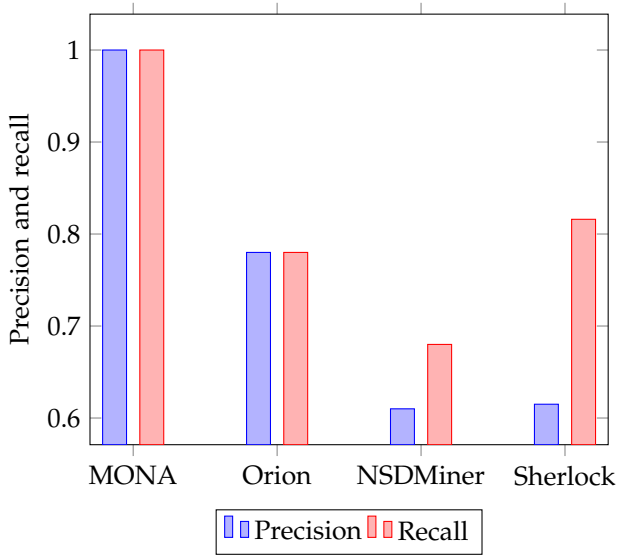


Figure 2.8: Comparison of Orion’s, NSDMiner’s and Sherlock’s precision and recall compared to MONA’s.

DMiner in terms of precision and recall rate. More precisely we evaluated whether Orion, NSDMiner and Sherlock are also able to detect correctly identified indirect dependencies uncovered by MONA. To make the comparison more comprehensible, we chose a synthetic network setup, which is most similar to what we finding within the previously mentioned real-life power distribution network. Within this synthetic network setup shown in Figure 2.8, MONA’s precision and recall value is 1. To facilitate comparison, we deploy Orion, NSDMiner and Sherlock on the same data set, which reveals that Orion outperforms NSDMiner and Sherlock.

Orion's precision and recall values are equivalent, while NSDMiner and Sherlock have a higher recall than precision value.

NSDMiner's performance is partially due to the methodology only detecting local-remote dependencies. Therefore, it misses remote-remote dependencies and we will exclude it from all further evaluation as all other methodologies detect LR and RR dependencies.

Sherlock detects every pair of frequently occurring network services as depending on each other. Thus, it creates a large number of false positives, and typically these false positives include frequently communicating network services. Looking into the precision and recall values of Sherlock, it seems clear that Sherlock aims to maximize its recall value. Similar to Rippler's experiment [Zan+14], the results of our experiments verified this property.

Orion maximizes precision and recall equally, however, MONA outperforms Orion's, Sherlock's and NSDMiner's results. As we are interested in equally maximizing precision and recall, we rely on the F-measure in the following as a test methodology.

Figure 2.9 shows the F-measures based evaluation for MONA, Sherlock and Orion in increasingly large networks with 10 direct dependencies 20 indirect dependencies. The number of flows per communication between indirectly dependent network services is varied between 5-10, 5-50 and 5-90.

As all four compared methodologies rely on analyzing network traffic patterns, a correlation between resulting F-measure curves becomes apparent. Generally, Orion surpasses Sherlock except for smaller networks with less than 275 network devices and 5-50 flows per communication between indirectly dependent network services. MONA almost

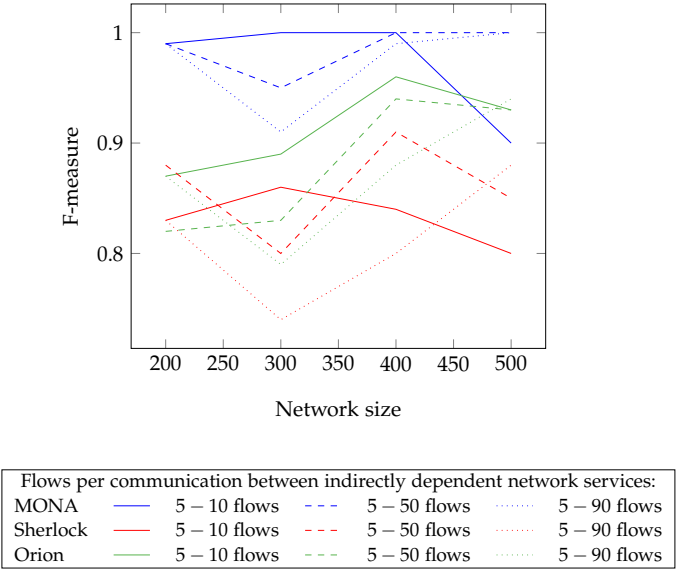


Figure 2.9: F-measures for MONA, Sherlock and Orion with network traffic containing 20 indirect dependencies.

always surpasses Sherlock and Orion, except for networks with 475 to 500 network devices and 5-10 flows per communication between indirectly dependent network services. In this case Orion surpasses MONA with a margin of less than 0.15 within Figure 2.9 in x-coordinate 0.91 and y-coordinate 460.

Figure 2.10 shows the evaluation for MONA, Sherlock and Orion based on the F-measure in increasingly large networks with 70 direct dependencies and 70 indirect dependencies. The number of flows per communication between indirectly dependent network services is varied between 5-10, 5-50 and

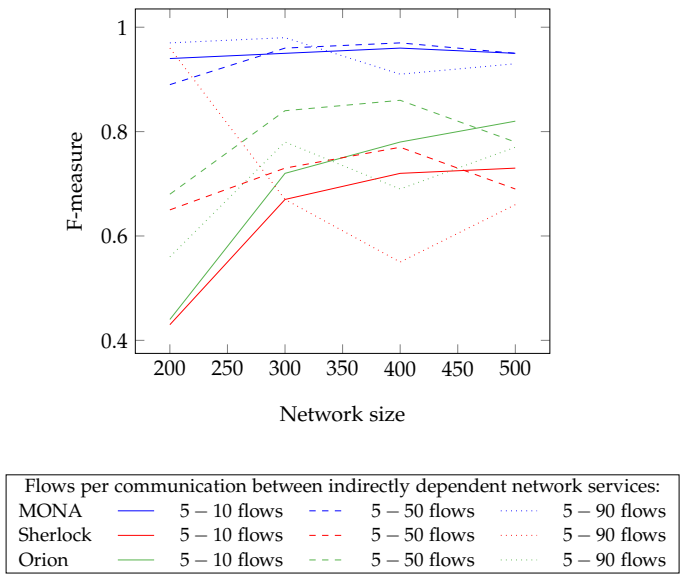


Figure 2.10: F-measures for MONA, Sherlock and Orion with network traffic containing 70 indirect dependencies.

5-90.

MONA’s F-measure results clearly surpass Sherlock’s and Orion’s in this experimental set up with more direct and indirect dependencies. General, Orion surpasses Sherlock’s F-measure results except for networks with less than 275 network devices and 5-90 flows per communication between indirectly dependent network services. For networks with 200 network devices (x-coordinate 200) Sherlock and MONA’s y-coordinates diverge by less then 0.01 for 5-90 flows per communication between indirectly dependent network services.

Computation Time Analysis

While MONA does not need to be real-time capable, as network service dependencies generally do not change as rapidly, it is still necessary to analyze how long computing network service dependencies take. This allows us to understand what resources are necessary to process arbitrary network traffic.

In MONA, computing direct network service dependencies is conducted online, while computing indirect network service dependencies can be conducted offline. Within the operational environment, indirect network service dependencies are computed online. Nevertheless, we need to make sure that network dependency information can be gathered based on practically relevant data for network traffic in realistic scenarios in sensible times.

For this purpose, we use synthetic network traffic data. Within each of this experiments, the computation time t_{time} represents the total amount of time for reading capture file, potential indirect dependency generation and calculation of indirect dependencies.

Proposition 2.1 (Computation time). *MONA's computation time increases superlinearly with the number of distinct TCP connections between network devices.*

In the following, we will conduct multiple experiments, to investigate whether the computation time is dependent on the number of TCP connections.

To test Proposition 2.1, Figure 2.11 shows the results of evaluating the performance of network dependency detection in a network containing 10 devices, 40 indirect dependencies and an increasing number of TCP connection. The number of TCP connections was increased from 10 to 410. This experiment showed that more communication between

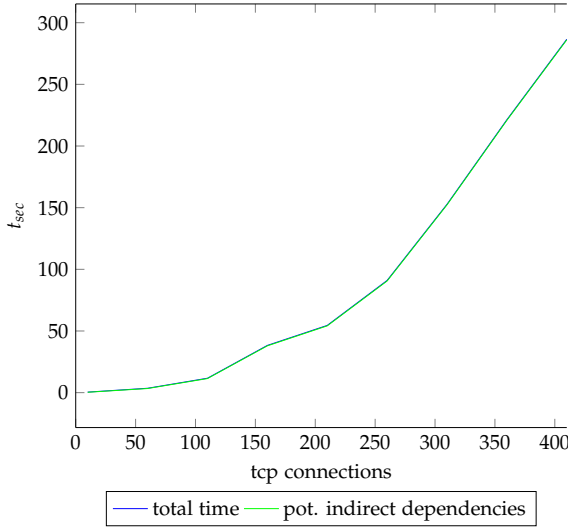


Figure 2.11: MONA's computation time for a single network size with network traffic containing an increasing number of TCP connections.

network devices, i.e., an increasing number of TCP connections, leads to an increase of computation time within the module generating potential indirect dependencies. Overall, the computation time t_{time} increases with the number of TCP connections increasing.

To further investigate whether the number of TCP connections increases MONA's computation time, we simulated networks of different size containing the same number of TCP connections. Similar to the previous experiment illustrated in Figure 2.11, we then increase the number of TCP connections for different sized networks and record MONA's computation time. In Figure 2.12 networks with 120, 220,

320, 420, 520, 620 and 920 network devices were simulated with 30 indirect dependencies.

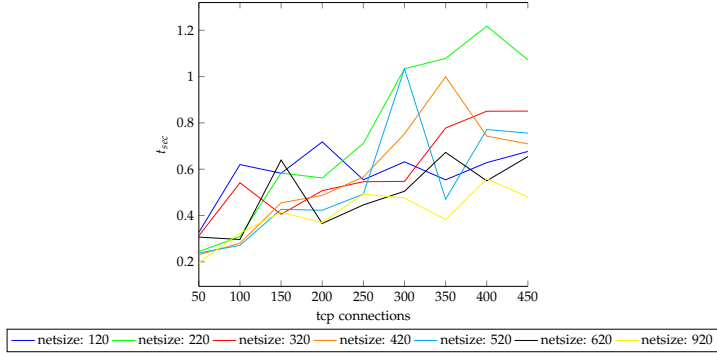


Figure 2.12: MONA's computation time for multiple network sizes with network traffic containing an increasing number of TCP connections.

We rely on standard error, mean average deviation $MAD_{\tilde{x}}$, and median deviation to further evaluate how far on average the computation times for different sized networks containing the same number of TCP connection diverge. For a univariate data set x_0, x_1, \dots, x_{n-1} , the $MAD_{\tilde{x}}$ is defined as the median of the absolute deviations from the data's median \tilde{x} :

$$MAD_{\tilde{x}} = \frac{\sum_{i=1}^n |x_i - \tilde{x}|}{n} \quad (2.6)$$

Similarly, the mean \bar{x} over the same univariate data set with n elements is defined as

$$\bar{x} = \frac{\sum_{i=0}^{n-1} x_i}{n} \quad (2.7)$$

For the same data set, the average mean standard error SE_x

is computed

$$SE_x = \frac{\sum_{i=0}^{n-1} |x_i - \bar{x}|}{n} \quad (2.8)$$

based on the mean standard error $SE(x_i)$ for an element x_i with a standard deviation σ_{x_i} . To compare error measures over all different sized network which are tested within the experiment, we average the error measures over all network sizes.

Error measure	Result
Mean standard error	0.059410
Average of the median deviation	0.083820
Average of the middle deviation	0.121169

Table 2.2: Error measures results for the experiment shown in Figure 2.12.

Error measure results based on Equation 2.6, 2.7 and 2.8 for the experiment shown in Figure 2.12 are shown in Table 2.2. The error measure results suggest that regardless of network size, the number of TCP communications is the decisive factor for MONA's computation time. We come to this conclusion due to the following observation (see Figure 2.12): increasing the number of TCP connections within a synthetic network of unvarying size leads to an increase of computation time.

Sensitivity Evaluation

Orion and MONA both rely on a threshold in order to quantify indirectly dependent network services.

Proposition 2.2 (Threshold Sensitivity). *MONA's precision and recall results are less sensitive to the chosen threshold than Orion's precision and recall results.*

In order to be able to compare both thresholds, we used our random network generator to create data-communication networks containing 100 network devices, 60 directly communicating network services and 20 indirect dependencies. Figure 2.13 show the results of this analysis. Random networks are generated to mimic communication patterns of the real-life energy distribution network, which is a part of our case study, however we are able to increase specific characteristics. For example, we can simulate networks with an increasing number of network devices or add more indirect dependencies. This allows for testing the capabilities and limitations of Orion and MONA. To overcome a possible bias of the results due to the added random variations, we generated every network multiple times and averaged the results. As we are interested in equally optimizing precision and recall, based on the results of our sensitivity experiment shown in Figure 2.13, we come to the conclusion that MONA's threshold is more robust than Orion's threshold for medium-sized networks. We come to this conclusion due to the experiment shown in Figure 2.13 revealing Orion's precision and recall values being optimal for threshold $x = 0.17$. A lower and higher threshold x strongly decreases Orion's precision or recall value. MONA's precision and recall values within the experiment shown in Figure 2.13 are less sensitive to threshold variations than Orion's.

To further compare MONA's and Orion's threshold, we additionally simulate a larger network containing 450 network devices with 100 communicating network services and between 120 and 140 indirect dependencies. For a larger

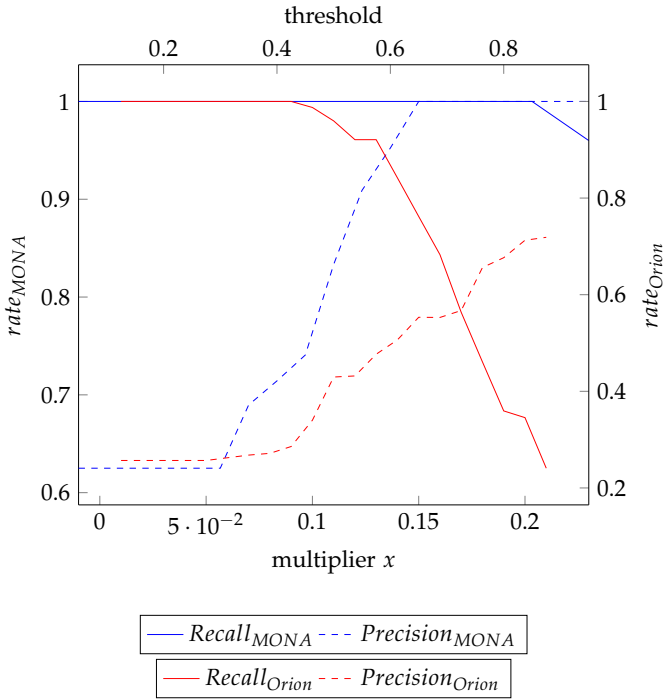


Figure 2.13: Comparison between MONA and Orion in a medium size network.

network it is even more apparent that MONA's threshold is more robust than Orion's. The results for this experiment are shown in Figure 2.14.

Similar to the experiment shown in Figure 2.13, this experiment points out that MONA's threshold is more robust than Orion's as we aim to achieve a high FP rate, while reducing the overall FN rate. The results of this sensitivity experiment based on synthetically created data sets are shown in

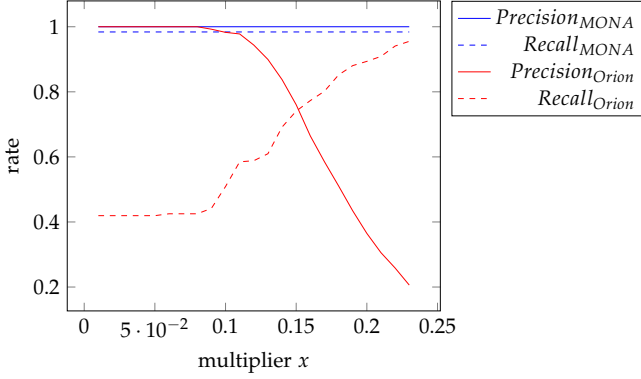


Figure 2.14: Comparison between MONA and Orion in a large network.

Figure 2.14 and reflect our previous assessment shown in Figure 2.13.

By analyzing Orion’s underlying methodology, it becomes apparent why Orion’s threshold is less robust within synthetic networks which recreate communication patterns of an energy distribution network. In Orion’s own experimental evaluation, Orion has a high rate of TP and a low rate of FN indirect dependencies in data-communication networks with small delays or huge amount of network communication between indirect dependent network services. However, not all data-communication networks fulfill such criteria. Given that communicating network devices are physically distributed as for example remote terminal units within an energy distribution network, delays automatically increase compared to physically neighboring network devices.

Proposition 2.3 (MONA’s Threshold Sensitivity (see θ in Equation 2.5)). *Within networks containing large numbers of*

direct dependencies and no indirect dependencies, MONA becomes more sensitive to the chosen threshold.

To assess the sensitivity of our introduced approach MONA, we generated a network containing 500 network devices and 500 direct dependencies. In other words, this network contains a lot of communication between network services, but it does not contain any indirect dependencies that could be identified. The results of this analysis are shown in Figure 2.15.

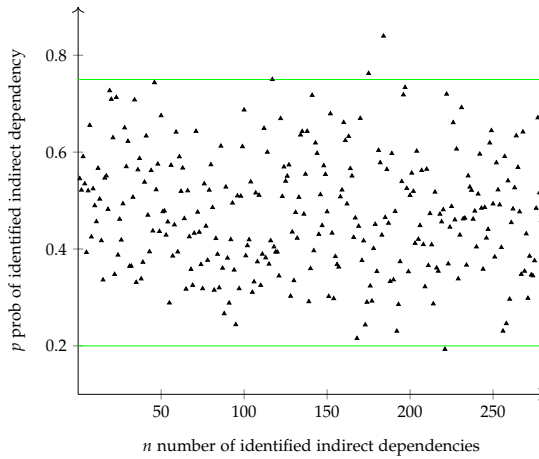


Figure 2.15: Results for MONA for network traffic containing no indirect dependencies.

In Figure 2.15 it becomes apparent that normalized cross correlation identifies similar communication patterns. Thereby, although all communication patterns are not caused by present indirect dependencies, MONA assigns indirect dependency candidates higher probability values. By choosing a higher threshold, such false positive indirect dependency

candidates can be excluded from indirect dependencies identified by MONA. Based on this experiment we come to the conclusion that Proposition 2.3 holds as networks with a large number of direct dependencies make MONA more sensitive to its threshold.

Generating synthetic networks with 500 network devices and 500 direct dependencies illustrates the drawbacks to any correlation-based methodology by showing that a lot of communication leads to false indirect dependencies being assigned a probability higher than zero. Thus, a domain-appropriate threshold has to be chosen carefully, to avoid false positive indirect dependencies being identified.

2.5 Discussion

Automatically identifying network service dependencies in large, distributed networks provides tangible benefits to network operators. Large networks consist of multiple network devices which host applications that interact through network services to fulfill a common goal. This interaction between distinct network devices results in network service dependencies. Knowledge of network service dependencies is helpful for network management as this enables network operators to minimize downtime and costs, while preparing for and responding to system failures. Large, distributed networks often contain a high number of network service dependencies, hence we introduce an automated approach to non-intrusive network service dependency mining.

Non-intrusive network service dependency mining approaches analyze network traffic in order to deduce existing network service dependencies. Network traffic consists of continuously exchanged network packets. To analyze net-

work traffic for existing network dependencies, we formalize exchanged network packets into direct dependencies. As network traffic consists of network packets that are continuously exchanged over time, a time series of direct dependencies is available for further data mining.

Based on the time series of direct dependencies observed within monitored networks, we abstract direct dependencies into communication histograms. By abstracting observed network traffic into communication histograms, we are able to use normalized cross correlation for deriving network service dependencies. The stream-based discovery of network service dependencies based on normalized cross correlation results in a general framework called Mission Oriented Network Analysis (MONA).

To investigate MONA's ability to correctly identify existing network service dependencies, we conduct a real-life case study based on an energy distribution network. In the context of the case study, network operators provide a ground truth by listing all known indirect network service dependencies beforehand. Based on this ground truth, we come to the conclusion that MONA detects all existing indirect dependencies within the test environment. In addition, no false indirect dependencies are derived by MONA. Therefore, all indirect network service dependencies detected by MONA are classified as true positives.

Within this case study we also observed that network operators do not always know all existing network service dependencies within their infrastructure. Multiple times MONA was able to uncover network service dependencies which were previously unknown to operators. Network service dependencies within our case studies were unknown to operators due to three reasons:

- third party software,

- network support provided by third parties or
- human error.

Hence network operators could not be trusted to provide reliable ground truth.

To allow a more in depth evaluation, we rely on synthetically generated networks and compare MONA to Sherlock, NSDMiner and Orion. Sherlock is another approach to non-intrusive network dependency discovery which learns an inference graph of network service dependencies based on co-occurrences within network traffic. NSDMiner addresses the same problem of network service dependency discovery for network stability and automatic manageability. NSDMiner is available as open source software.

Orion is another popular non-intrusive approach. Orion relies on spike detection based on the delay distribution of direct dependencies (also referred to as flow pairs) to infer network dependencies.

After showing precision and recall for all four network service dependencies, we exclude NSDMiner from further evaluation as it only detects LR dependencies. In addition, we point out that Sherlock's results suggest that it focuses primarily on optimizing its recall. As we are equally interested to optimize precision and recall, we rely on F-measures as a test methodology. A thorough analysis with increasingly large networks and varying numbers of network flows for communicating indirectly dependent network services and varying numbers of direct and indirect dependencies shows that MONA's F-measure results surpass Orion's and Sherlock's. Our observation regarding Orion failing to detect high-confidence dependencies is mirrored by the experimental evaluation of the active network dependency analyzer Rippler [Zan+14]. Our results from extensive experiments

show MONA significantly improving the state of the art.

In this chapter, we showed how network traffic provides a solid foundation for a non-intrusive detection of existing network service dependencies. We refer to the introduced time series data mining technique as MONA and in the following chapter we will discuss how to use MONA to automatically derive workflows.

Workflow Mining

In Chapter 2 Mission Oriented Network Analysis (MONA) was introduced as a means of automatically detecting network service dependencies based on network traffic. A network service dependency joins two direct communications and as a result joins four network services into a common data structure called indirect dependency. To further expand on this notion, we introduce network service dependencies as a basis for mining workflows. First, in Section 3.1, we provide a general introduction to the topic of workflow mining. Second, in Section 3.2, we introduce how we rely on network service dependency analysis to mine workflow events based on network traffic. We model detected network service dependencies with a probability space. Based on this probability space, we model workflows with Hidden Markov models (HMMs see [RJ86] for more information on HMMs). To test the ability of the introduced workflow mining approach to detect workflows in real-life applications, a case study is conducted within the disaster recovery site of an energy distribution network. Additionally, in Section 3.3, network vulnerability assessment is introduced as an example for using workflows.

3.1 Introduction

Workflow and business process models describe the underlying dependencies of network devices and network services within data-communication networks. Thus, they can be used as a foundation for context-aware information systems as this enables a security information and event management system to take the overall workflow into account while processing security information. The process of taking the overall workflow into account is also referred to as operational impact assessment. Operational impact assessment can be used to assess the impact of a software vulnerability on a monitored data communication network. For more information on operational impact assessment of software vulnerabilities, see Section 3.3.

The task of managing distinct information, such as software vulnerabilities and security events, is fulfilled by security information and event management (SIEM) systems or unified security management. Within the cyber security community, SIEM systems and unified security management are in huge demand. Thus, the cyber security community has developed an increasing interest in workflows. Workflows represent orchestrated, repeated patterns of business activity¹. Business activities allow for the systematic organization of resources into processes. Processes rely on multiple resources in order to provide a service or process information. Within enterprise networks, multiple workflows exist in order to fulfill different challenges for a company. Workflows typically involve multiple network devices and applications provided by multiple third party vendors.

¹Franchise Tax Board, State of California. *Business Process Management Center of Excellence Glossary*. 2009. URL: https://www.ftb.ca.gov/aboutFTB/Projects/ITSP/BPM_Glossary.pdf.

Unfortunately, workflows are often not documented since it is a very time consuming process to design handmade workflow models and, additionally, requires a lot of knowledge about a monitored infrastructure. Thereby, manual workflow modeling is expensive. Additionally, handmade workflows are often idealized descriptions of the process under consideration and often describe more what should be done, rather than the actual process. See [Van+03] for a detailed description of recurring issues with manual workflow modeling.

Verifying whether a workflow really describes the actual process is an additional, time consuming process. For handmade workflows it is difficult to detect if a workflow model is outdated due to concept drifts that have occurred. Thus, workflow models need to be verified regularly and updated when necessary. To counter all these issues, the problem of automatically mining workflows has been introduced already in 1998 [AGL98].

Workflow mining aims to automatically derive structured descriptions of executed tasks within an infrastructure. Commonly workflow mining relies on event logs, which directly list what task was executed at a specific point in time. Often also the user executing the task is listed. Event logs are for example provided by Enterprise Resource Planning (ERP) software. An example for an event log is given in Table 3.1.

Relying on event logs, workflow mining methods automatically deduce workflows as sequences of activities executed by users. Given that no logs of activities executed by users are available, to the best of our knowledge no workflow mining algorithms exist. We argue that other sources for workflow mining exist. For example, network traffic contains information about ongoing workflows within a network.

Case 18:			
Description	Event	User	Time
Register order Prepare shipment Ship goods Send bill Receive payment	Start	Alice	2016/02/05 15:00
		Alice	2016/02/05 15:00
		Alice	2016/02/05 15:00
		Alice	2016/02/05 15:01
		Alice	2016/02/05 15:01
		Alice	2016/02/06 17:00
	End	Alice	2016/02/06 17:00

Table 3.1: Event log produced by ERP software.

Workflows within data-communication networks consist of executed network activities that link multiple cyber assets. For example, a workflow could describe the following network activity: a user aims to retrieve information from a web server. This workflow is illustrated in Figure 2.1 and describes multiple resources (client, DNS server, load balancing server, web server and database), which interact to provide information to a user. In order to continuously provide information to users, multiple cyber assets interact. This results in a repeatable pattern of business activity, which is observable as a recurrent network activity.

We argue that network service dependencies are basic building blocks for deriving repeatable patterns of business activity. Business activities are observable as interaction between network services with a common underlying purpose such as providing services or processing data. Hence, network service dependency analysis provides the basic building blocks for learning workflows. To automatically derive workflows based on network traffic in the following Section 3.2, we propose a methodology relying on network service dependency mining techniques introduced before.

3.2 Workflow Model

As described previously, we assume that organizations have workflows translating into network activities which lead to interactions between cyber assets. Interactions between cyber assets can be observed within network traffic. Hence, we argue that network traffic can serve as a foundation for workflow mining.

We come to this conclusion due to the following observation: Network traffic consists of network packets, which are exchanged between applications hosted by network devices in order to share information to fulfill a common task. This common task corresponds to a so-called workflow event. Over time, we are able to record sequences of workflow events. Workflow events consist of distributed applications hosted on network devices interacting through network services to fulfill a common task. Moreover, mining sequences of workflow events provides the foundation for deriving Hidden Markov Model based workflows. The derivation of workflow events is based on MONA, which we introduced in Chapter 2. In the following, we introduce how we model workflow events based on automatically detected network service dependencies.

3.2.1 Event Logging

The purpose of workflow mining in the context of this work is to construct a structural representation of recurrent activities with an infrastructure. Workflow mining (see Section 5.4) relies on event logs in order to deduce a workflow model. Generally, event logs within most workflow mining approaches are supposed to list activity descriptions and contain references to resources such as, for example, cyber

assets. As in the context of this work we aim to derive workflows by relying on network traffic as a source of information, we rely on direct dependencies to derive so-called event logs. Event logs serve as input for the workflow mining approach introduced in the context of this work.

Definition 3.1 (Activity). The purpose of activities is to link network services S , which are introduced in Definition 2.2, to activity descriptions. Activity description are given as a non-empty alphabet Σ and based on this information we define an activity labeling function *aname* as

$$aname : \mathcal{PS} \setminus \{\emptyset\} \mapsto \Sigma^*$$

such that sets of network services from \mathcal{S} are mapped to activity descriptions. An activity set A_S is defined by:

$$A_S := \{name \in \Sigma^* \mid \exists S' \subseteq S : aname(S') = name\}$$

for activity descriptions $name \in \Sigma^*$ and a set of network services S . To identify which activities network-services are linked to, we define relation *DESCRIBES*. Relation *DESCRIBES* is defined as:

$$DESCRIBES \subseteq S \times A$$

We define *DESCRIBES* as

$$DESCRIBES = \{(s, name) \mid \exists S' \subseteq S \\ : \exists s \in S' \wedge aname(s) = name\}$$

Slightly misusing formal notation we write $DESCRIBES(s_j) = a_i$ for $s_j \in S$ and $a_i \in A_S$.

◇

Example 3.1 (Activity). From this definition it follows that we write $A_S = \{a_0, \dots, a_i, \dots, a_{n-1}\}$ for n activities. Table 3.2 illustrates activities for the workflow described in Figure 2.1. For example the Domain Name Server (DNS) in Figure 2.1 translates domain names into IP addresses with a network services mapped to port number 53 with the UDP protocol. Surfing the world wide web relies on a network service either linked to port 80 or port 8080 and the TCP protocol. In order to provide database information, a network service is linked to port 118 and the UDP protocol.

We use wildcards to cluster ports within the ephemeral port range and represent clusters by the character “*”. Clustering ephemeral ports was previously introduced in Definition 2.3. Within the ephemeral port range, ports are dynamically assigned by an operating system. Thus, the same request sent at different points in time is very likely to be sent from different ports within the ephemeral port range. Therefore, separately analyzing port numbers within the ephemeral port range provides no additional information. The ephemeral port range is generally used to request information. Thereby, network services for requesting information are linked to port “*” and they can rely on the UDP or TCP protocol.

Activity ID	Network Service	Port	Protocol	Activity description
a_0	s_{17}, s_2	53	UDP	translate domain names into IP addresses
a_1	s_1, s_2, s_3	80, 8080	TCP	surf the world wide web
a_2	s_{15}	118	UDP	provide database information
a_3	s_1, s_2, s_3, s_{17}	*	UDP, TCP	request information

Table 3.2: Activities based on Figure 2.1.

Based on Definition 3.1 the activities shown in Table 3.2 are defined as follows:

- $aname(\{s_{17}, s_2\}) = \text{"translate domain names into IP addresses"}$
- $aname(\{s_1, s_2, s_3\}) = \text{"surf the world wide web"}$
- $aname(\{s_{15}\}) = \text{"provide database information"}$
- $aname(\{s_1, s_2, s_3, s_{17}\}) = \text{"request information"}$

By analyzing network traffic, we are able to identify network services by port number and protocol, and with Definition 3.1 we are able to link network services to names for workflow activities. Based on the workflow illustrated in Figure 2.1, Example 3.1 describes how network services are linked to activities.

Generally, two different types of ports are distinguished: dynamic and static ports.

The range of ephemeral ports varies, although efforts of standardizing ephemeral port ranges have been made by the IANA. The range of ephemeral ports is a preset parameter. There are two negative occurrences linked to automatically preset the ephemeral port range: the ephemeral port range could be chosen too large or too small. An ephemeral port range chosen too large leads to statically assigned ports being erroneously assumed to be dynamic ports. An ephemeral port range chosen too small leads to dynamically assigned ports being wrongly assumed to be static ports. Thereby, should the ephemeral port range be assumed too small, the number of activities increases. Given that the number of erroneously assigned ports increases, network service dependency discovery may lead to biased results. We use activities to derive event logs in the following.

Definition 3.2 (Workflow event log entry). Based on activities (see Definition 3.1), network devices (see Definition 2.1),

and communication histograms (see Definition 2.9), we derive workflow events WFE as

$$WFE \subseteq D \times A \times D \times A \times H$$

for source and destination network device $d_i, d_j \in D$, activities for source and destination network device $a_j, a_l \in A$ and a communication histogram $h_m \in H$. \diamond

Source		Destination		Communication Histogram
Hostname	Activity	Hostname	Activity	
Client	a_3	DNS Server	a_0	h_0
Client	a_3	Load balancing server	a_1	h_1
Load balancing server	a_3	Web Server	a_3	h_2
Web server	a_3	Database	a_2	h_3

Table 3.3: Workflow event log for the workflow shown in Figure 2.1.

Example 3.2 (Workflow event log). Based on network flows, a simple workflow as shown in Figure 2.1 could lead to the workflow event log shown in Table 3.3. Event logs contain references to the source and destination point of a communication. Within Table 3.3, source and destination point of communication are network devices and represented by hostnames. Every network device is linked to an activity, and the communication between the two network devices is described by a communication histogram.

Workflow event logs are the basis for deriving workflow events based on indirect dependencies based on MONA as introduced in Chapter 2.

3.2.2 Probability Space

Previously, we introduced MONA as a novel methodology for network service dependency discovery. In the context of this work, we introduce network dependency analysis as a basis for workflow mining. Detecting network service dependencies in an enterprise network is limited to a set of possible network service dependencies. The underlying characteristics of an enterprise network lead to recurring network service dependencies. Modeling network service dependency discovery as a probability space [Kol50] implies that the occurrence network service dependency is a random experiment with a fixed set of possible outcomes.

Network service dependencies are separated into two different categories. As discussed above, similarly to previous work, we distinguish remote-remote (RR) dependencies and local-remote (LR) dependencies [Che+08]. Examples for both dependency types are shown in Figure 2.3. Colloquially introduced, a LR dependency implies that a local host requires information from a remote host before issuing another request to second remote host. An example for a LR dependency $ISDEP_{LR}$ is shown in Figure 2.3b.

An RR dependency refers to the following communication pattern: A host contacts another remote host, who then goes on to contact another remote host. Figure 2.3a shows an RR dependency $ISDEP_{RR}$. For more information on network service dependencies, which are also referred to as indirect dependencies, we refer to Section 2.2. In the following example, we will introduce network service dependencies shown in Figure 2.1 in order to lay the foundation for an ongoing example.

Example 3.3 (Network service dependencies). Based on the workflow shown in Figure 2.1, Figure 3.1 shows three in-

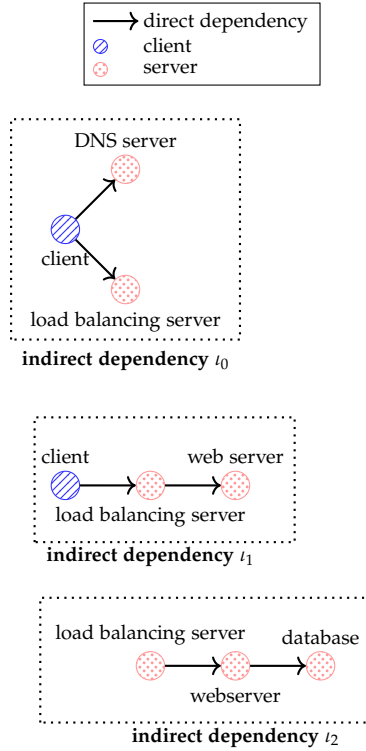


Figure 3.1: Indirect dependencies within the workflow shown in Figure 2.1.

direct dependencies. An LR dependency ι_0 links a direct dependency (introduced in Definition 2.6) between client and DNS server to the direct dependency between client and load balancing server. An RR dependency ι_1 links a direct dependency between client and load balancing server to the direct dependency between load balancing server and web

server. Another RR dependency ι_2 joins the direct dependency between load balancing server and web server to the direct dependency between web server and a database.

LR dependencies and RR dependencies $\iota_0, \iota_1, \iota_2 \in ISDEP$ are indirect dependencies consisting of direct dependencies from $SDEP$. Each indirect dependency joins two distinct direct dependencies, respectively.

We use MONA to detect how multiple network services, and thereby network devices, interact for a common higher purpose. This information is not obvious when looking into workflow event logs. Thereby, we consider network service dependencies as a foundation for deriving hidden states within our HMM workflow model.

Network service dependency discovery relies on normalized cross correlation, as described in Chapter 2, and provides an heuristic approach for deriving indirect network service dependencies.

Network service dependency discovery is an experiment as it can be repeated a number of times and will provide an outcome for each repetition. Also, we assume that past outcomes of the experiment provide no information about future outcomes. In other words, we assume that, due to an infrastructure and its workflows constantly evolving, network service dependencies can change. Therefore, we chose a probability space [Kol50] to represent network service dependency discovery.

Definition 3.3 (Probability space of network service dependencies). Network service dependency discovery results

in indirect dependencies, which we understand as atomic events ι . The set of all indirect dependencies is referred to as Ω . In addition, network service dependency discovery results in a set of observed indirect dependencies F and the set of all possible indirect dependency events \mathcal{F} . The results of network service dependency discovery are described by a probability space (Ω, \mathcal{F}, P) with

- (1.) a set of all possible indirect dependencies Ω ,
- (2.) a set of all possible indirect dependency set \mathcal{F} , and
- (3.) a probability function $P : \mathcal{F} \rightarrow [0, 1]$.

$\mathcal{F} \subseteq \mathcal{P}(\Omega)$ such that \mathcal{F} is a σ -algebra due to the following criteria being fulfilled:

- All possible indirect dependencies Ω are a part of the set of all possible indirect dependency sets $\Omega \in \mathcal{F}$,
- an arbitrary independent event set $F \in \mathcal{F}$, such that $\Omega \setminus F \in \mathcal{F}$, and
- is closed under countable unions, such that $F_0, F_1 \in \mathcal{F} \implies F_0 \cup F_1 \in \mathcal{F}$.

A probability function $P : \mathcal{F} \rightarrow [0, 1]$ mapping \mathcal{F} to $[0, 1]$ with $P(\Omega) = 1$, such that for an event set $\mathcal{F}' \subseteq \mathcal{F}$

$$P(\mathcal{F}') = \sum_{F \in \mathcal{F}'} P(F).$$

To derive a probability function P , we take a closer look into indirect network service dependencies. An indirect dependency $\iota \in ISDEP$ joins two direct dependencies. Due to every direct dependency describing the communication between two network services, the elementary building blocks

for every indirect dependency are network services. Thus, an indirect dependency ι is a tuple of four network services (s_0, s_1, s_3, s_4) .

Given an event set F with n distinct indirect dependencies ι , the number of occurrences $C_F(s)$ of a network service $s \in S$ is derived by

$$C_F(s) = \sum_{\iota \in F} [s \in \iota].$$

Based on the number of occurrences $C_F(s)$ of a network service $s \in S$, the overall number of occurrences $C_F(\iota)$ of every network service within an indirect dependency ι can be derived by

$$C_F(\iota) = \sum_{s \in \iota} C_F(s).$$

Based on this information, the probability $P(\iota)$ of an indirect dependency $\iota \in F$ is defined as follows:

$$P(\iota) = \frac{C_F(\iota)}{\sum_{\iota_i \in F} C_F(\iota_i)}.$$

Thus, overall, the probability for all indirect dependencies Ω is

$$P(\Omega) = \sum_{\iota \in \Omega} P(\iota) = 1.$$

◇

Network service dependencies capture how multiple distinct activities serve a common higher purpose. Network service dependency discovery estimates indirect dependencies based on communication patterns, which essentially are noisy observations. The upper level of our HMM workflow is a Markov process and the states are unobservable as activities serving a higher purpose cannot directly be observed

by monitoring network traffic. Thus, we rely on network service dependencies to derive hidden states within our HMM workflow in the following subsection.

3.2.3 Hidden States Model

We use the previously introduced probability space (Ω, \mathcal{F}, P) , which is derived based on network service dependency discovery, to derive an HMM describing an enterprise network's workflow. Network service dependency discovery is used to analyze network traffic in order to observe sets of indirect dependencies $F \subseteq \mathcal{F}$. The event set $F = \{\iota_1, \iota_2 \dots, \iota_n\}$ consists of n observed indirect dependencies. Observed indirect dependency sets F are the basis for modeling hidden states within an HMM workflow.

Definition 3.4 (Hidden States). Let an event set $F = \{\iota_0, \iota_1 \dots, \iota_n\}$ consist of n observed indirect dependencies. Indirect dependencies *ISDEP* join two distinct direct dependencies *SDEP*. However, two indirect dependencies $\iota_0, \iota_1 \in F$ can contain the same direct dependency δ_0 . We refer to two indirect dependencies ι_0 and ι_1 as overlapping in δ_0 . Based on indirect dependencies within an observed event set F , a set X with no overlapping direct dependencies is determined by

$$\begin{aligned} \text{NoOverlap}(X) = \neg \exists \iota_0 \in X : \exists \iota_1 \in X \setminus \{\iota_0\} \\ : \exists \delta_0 : \delta_0 \in \iota_0 \wedge \delta_0 \in \iota_1. \end{aligned}$$

The set of hidden states X is derived based on

$$X := \text{chooseOne} \left(\underset{\substack{X \in \mathcal{P}(F) \\ \text{NoOverlap}(X)}}{\text{argmax}} \quad |x| \right).$$

Should there be two or more possible hidden state sets with the same maximum number of hidden states, then a random one is chosen by *chooseOne*. Randomly choosing a hidden state set could affect the accuracy of the overall sequence of hidden states within the HMM workflow model. \diamond

Example 3.4 (Hidden States). Figure 3.1 illustrates three indirect dependencies $\iota_0, \iota_1, \iota_2 \in ISDEP$ contained in the workflow shown in Figure 2.1. Indirect dependencies represent an activity serving a higher purpose. Indirect dependencies are derived by analyzing communication pattern, which essentially constitute noisy observation. The set $X = \{\{\iota_0, \iota_2\}, \{\iota_1\}\}$ corresponds to all sets consisting of indirect dependencies with no overlapping direct dependencies. Thus, within our HMM workflow, the states are hidden as activities serving a higher purpose cannot directly be observed by monitoring network traffic. Thus, we rely on network service dependency discovery to derive hidden states within our HMM workflow. The indirect dependency ι_1 overlaps with ι_0 on the direct dependency client \rightarrow load balancing server. In addition, ι_1 overlaps with ι_2 on the direct dependency load balancing server \rightarrow web server. Thus, there are two possible sets that have no overlapping direct dependencies: The set containing a single indirect dependency ι_1 and its complement containing ι_0, ι_2 . Based on Definition 3.4, the set of hidden states is the set with the maximum number of elements. Thus, given all sets with no overlapping direct dependencies $X = \{\{\iota_0, \iota_2\}, \{\iota_1\}\}$, the set of hidden states is $X = \{\iota_0, \iota_2\}$. The set of hidden states is illustrated in Figure 3.2.

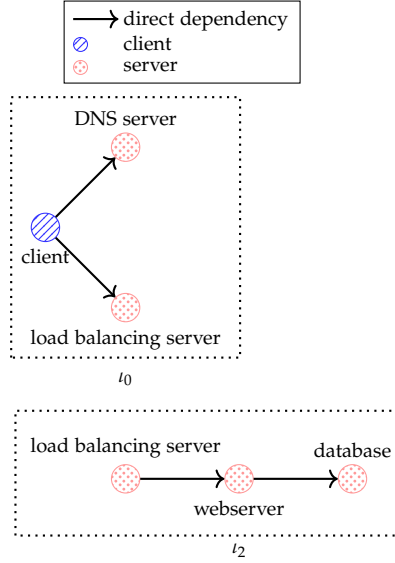


Figure 3.2: Hidden states within the workflow shown in Figure 2.1.

Definition 3.5 (Candidates for state transitions). Based on an event set $F = \{\iota_1, \iota_2, \dots, \iota_n\}$ consisting of n observed indirect dependencies and set of hidden states X , the remaining observed indirect dependencies \tilde{C} that do not constitute hidden states X is derived by

$$\tilde{C} = F \setminus X.$$

The set of state transition candidates C captures indirect dependencies that overlap with two hidden states. This is

the underlying foundation for deriving state transition probabilities within an HMM workflow. The set of candidates for state transitions C is

$$C = \left\{ \iota_0 \in \tilde{C} \mid \exists (\iota_1, \iota_2 \in X) [\delta_0 \in \iota_0 \wedge \right. \\ \delta_0 \in \iota_1 \wedge \\ \delta_1 \in \iota_0 \wedge \\ \left. \delta_1 \in \iota_2] \right\}.$$

◇

Example 3.5 (Candidate for state transitions). Figure 3.1 illustrates three indirect dependencies $\iota_0, \iota_1, \iota_2 \in ISDEP$ contained in the workflow shown in Figure 2.1. Based on observed indirect dependencies, the hidden state set of indirect dependencies with no overlapping direct dependencies $X = \{\iota_0, \iota_2\}$ is derived.

The remaining observed indirect dependency $\tilde{C} = \{\iota_0, \iota_1, \iota_2\} \setminus \{\iota_0, \iota_2\} = \{\iota_1\}$ is a candidate for a state transition due to its overlapping direct dependencies. The indirect dependency ι_1 overlaps with ι_0 on the direct dependency client \rightarrow load balancing server. In addition, ι_1 overlaps with ι_2 on the direct dependency load balancing server \rightarrow web server. The overlapping is understood as a state transition between the two hidden states ι_0 and ι_2 . Thereby, we come to the conclusion that the set of candidates for state transitions is $C = \{\iota_1\}$, which is shown in Figure 3.3.

In this subsection, we presented our model of hidden states in Definition 3.4 and described the concept of candidates for state transitions in Definition 3.5. This provides the

underlying foundation for introducing our HMM workflow in the following Subsection 3.2.4.

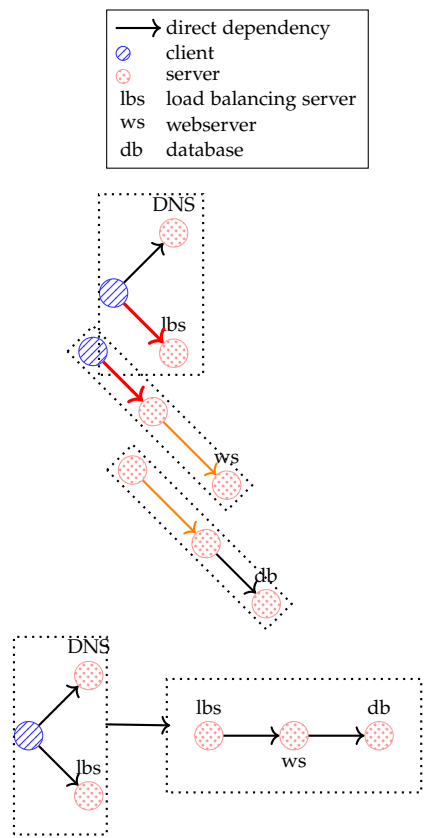


Figure 3.3: Candidate for state transitions within the workflow shown in Figure 2.1.

3.2.4 Hidden Markov Model Workflow

Based on activities that were introduced in Definition 3.1, an unsupervised temporal clustering has to be achieved to deduce sequences of activities, which are referred to as workflows, are represented as Hidden Markov Models (HMMs). We chose to represent workflows as HMMs as this allows a probabilistic analysis of transitions between activities and discovering a probabilistic workflow model from activities. Observations are identified based on network packets exchanged between network devices through network services utilized by each activity. We use HMM workflows as a means for representing network service dependencies as hidden states and consider direct dependencies as observables generated by hidden states. Given a sequence of observations, and an HMM, the probability of the observation sequence given the model can be derived. Based on network service dependency discovery, we are able to introduce a novel approach for unsupervised workflow discovery, assuming temporal consistency and cyclically repeated communication patterns.

Normalized cross correlation provides a heuristic approach for detecting network service dependencies and the result is described by a probability space (Ω, \mathcal{F}, P) described in Subsection 3.2.3. Observed indirect dependencies $F \in \mathcal{F}$ are the basis for modeling a set of hidden states X and a set of candidates for state transitions C . A monitored network may have a fully meshed topology, i.e., all network devices could be interconnected. In addition, every network device could use its entire static port range to host communicating network services. We rely on normalized cross correlation and an algorithm inspired by the Baum-Welch algorithm (see [Wel03] for more information) as offline learning methodologies to provide an efficient heuristic for deriv-

ing an HMM workflow. In the following we will introduce the underlying HMM workflow.

Definition 3.6 (HMM). We model a workflow as a first-order hidden Markov model (HMM). An example for an HMM is given in Figure 3.4.

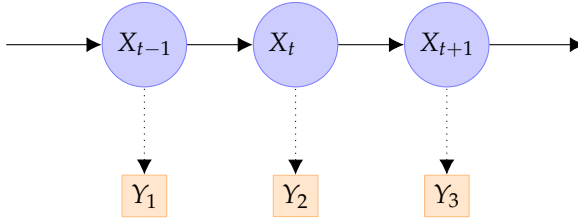


Figure 3.4: A Hidden Markov Model (HMM).

Our first-order HMM is based on the introduction to HMMs provided by Jurafsky et al. [JM09]. A HMM workflow $\lambda = (X, Y, A, B, \Pi)$ is defined as follows:

- a set of hidden states $X = \{\iota_1, \dots, \iota_n\}$ and $\iota_i \in \Omega$ with $\text{dom}(X^t) = \Omega$,
- a set of output symbols $Y = \{\delta_1, \dots, \delta_m\} \subseteq \text{SDEP}$ with $\text{dom}(Y^t) = \mathcal{P}(\text{SDEP})$,
- a state transition matrix $A : \Omega \times \Omega \rightarrow [0, 1]$,
- an observation matrix $B : \Omega \times \text{SDEP} \rightarrow [0, 1]$, and
- an initial state distribution vector Π over X .

There are two assumptions to fulfill for first-order HMMs. The first one being the Markov assumption, which states that the probability of a particular state only depends on the

previous state. In the following we write ι^t to denote $\iota \in X^t$. The Markov assumption for an indirect dependency ι , which constitutes a hidden state, observed over time $\{\iota^1, \iota^2, \dots, \iota^n\}$ is

$$P(\iota^t \mid \iota^{t-1}, \iota^{t-2}, \dots, \iota^1) = P(\iota_i^t \mid \iota_i^{t-1}).$$

The second assumption is the assumption of output independence. Similarly, we write δ^t to denote $\delta \in Y^t$. Output independence refers to an output observation δ only depending on the state that produced the observation ι . Given a direct dependency δ , which constitutes an observation, observed over time $Y = \{\delta^1, \dots, \delta^m\}$, output independence is defined by

$$P(\delta^t \mid \iota^t, \dots, \iota^{t-1}, \dots, \iota^1, \delta^t, \delta^{t-1}, \dots, \delta^1) = P(\delta^t \mid \iota^t).$$

By analyzing network traffic over time, windowing is achieved by a function *LINK*, which associates time windows W with hidden states X , output symbols Y and a tuple counting how the number of occurrences \mathbb{N}_0 for direct dependencies *SDEP*. Windowing is achieved by the following function:

$$Link : W \rightarrow dom(X) \times dom(Y) \times \mathcal{P}(SDEP \times \mathbb{N}_0).$$

In the following, we will denote consecutive time windows as w_t and w_{t+1} . \diamond

Example 3.6 (MONA over multiple time windows). Table 3.4 gives an example for three time windows w_1, w_2 , and w_3 , considering the workflow described in Figure 2.3, hidden states shown in Figure 3.2 and a state transition illustrated in Figure 3.3. Slightly missusing notation, the direct dependency occurrence counter set M is also used as a function

Time window	Hidden state set X , candidate for state transition set C and direct dependency occurrence counter set M
w_1	$(\{\iota_2(\delta_3(s_*^c, s_{53}^{DNS}), \delta_1(s_*^c, s_{80}^{lbs}))\};$ $\{\};$ $\{(\delta_1, 5), (\delta_3, 3)\})$
w_2	$(\{\iota_1(\delta_1(s_*^c, s_{80}^{lbs}), \delta_2(s_*^{lbs}, s_{80}^{ws}))\};$ $\{\};$ $\{(\delta_1, 5), (\delta_2, 4)\})$
w_3	$(\{\iota_2(\delta_3(s_*^c, s_{53}^{DNS}), \delta_1(s_*^c, s_{80}^{lbs})); \iota_3(\delta_2(s_*^{lbs}, s_{80}^{ws}), \delta_4(s_*^{ws}, s_{118}^{db}))\};$ $\{\iota_1(\delta_1(s_*^c, s_{80}^{lbs}), \delta_2(s_*^{lbs}, s_{80}^{ws}))\};$ $\{(\delta_1, 10), (\delta_2, 7), (\delta_3, 8), (\delta_4, 6)\})$

Table 3.4: Example for multiple time windows.

$M : \delta \rightarrow \mathbb{N}$ in order to derive the number of occurrences of members of its set.

Time window w_1 consists of a hidden state set $X = \{\iota_2\}$, an empty state transition set $C = \{\}$ and a set of tuples counting how often direct dependencies δ_1 and δ_3 occur. For example, network services hosted on a client c sent 5 network packets to a load balancing server lbs within time window w_1 . Similarly, 3 network packets were exchanged between client c and DNS server DNS .

Time window w_2 consists of a hidden state set $X = \{\iota_1\}$, an empty state transition set $C = \{\}$ and a set of tuples counting how often direct dependencies δ_1 and δ_2 occur. For example, network services hosted on a client c sent 5 network packets to a load balancing server lbs within time window w_2 . Similarly, 4 network packets were exchanged between load balancing server lbs and web server ws .

Within a time window w_3 , two indirect dependency ι_2 and ι_3 , which do not contain identical - “overlapping”- direct dependencies are observed. Thus, ι_2 and ι_3 are joined with a hidden state set X . Since ι_1 contains direct dependencies that are also contained within ι_2 and ι_3 , we consider ι_1 overlapping with ι_2 and ι_3 . Thereby, ι_1 indicates a state transition from ι_2 to ι_3 . A client c communicates with a DNS server and a load balancing server (see ι_2). As ι_1 indicates a state transition, we are able to see that ι_2 leads to the load balancing server communicating with a data base (see ι_3).

Definition 3.7 (State transition matrix). Given that multiple time windows $w \in W$ are analyzed, the set of hidden states in a time window w is derived by $w(X)$, such that the set of all hidden states

$$X' = \bigcup_{w \in W} w(X).$$

Given that $\iota_i, \iota_j \in X'$ the frequency of occurrence for state transitions from ι_i to ι_j is derived by

$$C_{a_{ij}} = \sum_{w \in W} [\exists (\iota_i, \iota_j \in X' \wedge \delta_0 \in \iota_i, \delta_1 \in \iota_j \wedge \delta_0 \neq \delta_1) : \iota_m(\delta_0, \delta_1) \in w_n(C)].$$

The frequency of occurrence for state transitions $C_{a_{ij}}$ from hidden state ι_i to ι_j is normalized over the overall frequency of occurrence of all state transitions starting in hidden state ι_i , such that

$$a_{ij} = \frac{C_{a_{ij}}}{\sum_{q=\{1, \dots, |X'|\}} C_{a_{iq}}}.$$

Based on the previously described equation, we are able to learn parameter a_{ij} , allowing us to build a state transition matrix

$$A = \begin{matrix} & \begin{matrix} l_1 & l_2 & \dots & l_n \end{matrix} \\ \begin{matrix} l_1 \\ l_2 \\ \vdots \\ l_n \end{matrix} & \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \end{matrix}$$

within an HMM $\lambda = (X, Y, A, B, \Pi)$ over multiple tumbling windows $\{w_1, \dots, w_n\} \in W$, given that n time windows were observed. Within communication networks, generally such a state transition matrix is generally sparse. \diamond

Example 3.7 (State transition matrix). Considering the workflow described in Figure 2.3, hidden states shown in Figure 3.2 and a state transition illustrated in Figure 3.3, an example for a state transition matrix A is given in the following. Given hidden states X and a candidate for state transition set C , based on Equation 3.7 the following state transition matrix A is derived. The hidden states X and candidate for state transition set C in the context of this example is:

- Hidden states $X = \{\iota_1(\delta_{11}(s_*^c, s_{53}^{DNS}); \delta_{12}(s_*^c, s_{80}^{lbs})), \iota_2(\delta_{21}(s_*^{lbs}, s_{80}^{ws}); \delta_{22}(s_*^{ws}, s_{118}^{db}))\}$,
- Candidate for state transition set $C = \{\iota_3\delta_{12}(s_*^c, s_{80}^{lbs}); \delta_{21}(s_*^{lbs}, s_{80}^{ws})\}$.

$$A = \begin{matrix} & \begin{matrix} \iota_1 & \iota_2 \end{matrix} \\ \begin{matrix} \iota_1 \\ \iota_2 \end{matrix} & \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{matrix}$$

This state transition matrix A has $a_{12} = 1$ and represents a state transition from ι_1 to ι_2 . As we consider state transitions to be bidirectional, there is also a state transition from ι_2 to ι_1 leading to $a_{21} = 1$.

Definition 3.8 (Observation matrix). Similarly, an emission probability $b_i(\delta_k)$ for direct dependency δ_k being emitted in hidden state ι_i is derived as

$$b_i(\delta_k) = \frac{\sum_{w \in W} w(M(\delta_k))}{\sum_{w \in W} \sum_{\delta \in M} w(M(\delta))},$$

where the frequency of a direct dependency δ_k occurring in a time window $w \in W$ is computed by $w(M(\delta_k))$. The previous equation is the foundation for deriving an observation matrix B , which represents the probability of state transitions within an HMM $\lambda = (X, Y, A, B, \Pi)$, such that

$$B = \begin{matrix} & \begin{matrix} \delta_1 & \delta_2 & \dots & \delta_m \end{matrix} \\ \begin{matrix} \iota_1 \\ \iota_2 \\ \vdots \\ \iota_n \end{matrix} & \begin{pmatrix} b_1(\delta_1) & b_1(\delta_2) & \dots & b_1(\delta_m) \\ b_2(\delta_1) & b_2(\delta_2) & \dots & b_2(\delta_m) \\ \vdots & \vdots & \ddots & \vdots \\ b_n(\delta_1) & b_n(\delta_2) & \dots & b_n(\delta_m) \end{pmatrix} \end{matrix}.$$

◇

Example 3.8 (Observation matrix). Considering the workflow described in Figure 2.3, hidden states shown in Figure 3.2 and a state transition illustrated in Figure 3.3, an example for an observation matrix B is given in the following. The hidden states X and candidate for state transition set C in the context of this example is:

- Hidden states $X = \{\iota_1(\delta_{11}(s_*^c, s_{53}^{DNS}); \delta_{12}(s_*^c, s_{80}^{lbs})), \iota_2(\delta_{21}(s_*^{lbs}, s_{80}^{ws}); \delta_{22}(s_*^{ws}, s_{118}^{db}))\}$, and
- Frequency of occurrences $\{(\delta_{11}, 6), (\delta_{12}, 4), (\delta_{21}, 2), (\delta_{22}, 2)\}$.

This leads to the following (2×4) observation matrix B :

$$B = \begin{matrix} & \delta_{11} & \delta_{12} & \delta_{21} & \delta_{22} \\ \begin{matrix} \iota_1 \\ \iota_2 \end{matrix} & \begin{pmatrix} 0.6 & 0.4 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \end{pmatrix} \end{matrix}$$

Definition 3.9 (Initial state distribution vector). In addition, an initial state distribution vector Π consists of counting the occurrences of a hidden state ι_i over n time windows and normalized by the overall number of hidden states in all time windows. This is computed by

$$\pi(\iota_i) = \frac{\sum_{w \in W} [\iota_i \in w(X)]}{\sum_{\iota \in X'} \sum_{w \in W} \iota \in w} \quad (3.1)$$

for multiple time windows $w \in W$ and a set all hidden states X' observed over all monitored time windows. Thereby, the more time windows a hidden state is to be observed in, the higher the hidden state's initial probability. \diamond

Example 3.9 (Initial state distribution vector). Considering the workflow described in Figure 2.3, hidden states shown in Figure 3.2 and a state transition illustrated in Figure 3.3, an example for an initial state distribution vector Π is given in the following. The hidden states X observed within three time windows are:

- $w_1(\{\iota_2(\delta_3(s_*^c, s_{53}^{DNS}), \delta_1(s_*^c, s_{80}^{lbs}))\});$
- $w_2(\{\iota_1(\delta_1(s_*^c, s_{80}^{lbs}), \delta_2(s_*^{lbs}, s_{80}^{ws}))\});$
- $w_3(\{\iota_2(\delta_3(s_*^c, s_{53}^{DNS}), \delta_1(s_*^c, s_{80}^{lbs}));$
 $\iota_3(\delta_2(s_*^{lbs}, s_{80}^{ws}), \delta_4(s_*^{ws}, s_{118}^{db}))\});$

This leads to the following initial state distribution vector Π :

$$\Pi = \begin{matrix} & \iota_1 & \iota_2 & \iota_3 \\ \begin{pmatrix} 0.25 & 0.5 & 0.25 \end{pmatrix} \end{matrix}$$

Definition 3.7, Definition 3.8 and Definition 3.9 enable an HMM to be learned by observing network traffic within an monitored network traffic based on network service dependency analysis.

Definition 3.10 (HMM Workflow-based Impact Assessment). HMM workflow-based impact assessment aims to link events, such as for example identified software vulnerabilities, IDS, IPS or FW events, to affected workflows. Given an HMM workflow $\lambda = (X, Y, A, B, \Pi)$, events are modeled as external events, which are associated with network services $S_{affected}$. Based on these network services $S_{affected}$, a

set $X_{affected}^{i=0}$ of hidden states X , which are affected at time horizon $i = 0$ is derived by

$$X_{affected}^i = CC \left(\left(S_{affected}, \text{map}(asSet, X) \right) \right)$$

where CC denotes connected components of the hypergraph $(S_{affected}, Nodeset)$. \diamond

Viterbi Algorithm

Given

- a HMM workflow model $\lambda = (X, Y, A, B, \Pi)$, and
- observed direct dependencies Y , which are emitted by hidden states X

HMMs can be used to infer how output strings Y were generated. There are many possible state sequences given an HMM λ and an observed sequence $y = y_1 y_2 \dots y_T \in Y$, there are many possible state sequences $x = x_1 x_2 \dots x_T \in X$ that produce y . State sequences are also referred to as paths.

Many applications are interested in deriving the path with the highest probability of occurring ζ^* . For HMMs that describe ongoing workflows within a data-communication networks, the most probable path corresponds to the sequence of events with the highest likelihood of taking place.

$$\zeta^* = \underset{\zeta}{\operatorname{argmax}} P(y|\zeta) \quad (3.2)$$

The path with the highest likelihood of taking place is also referred to as Viterbi path as the Viterbi algorithm is a dynamic programming method for deriving this path. The Viterbi algorithm (see [For73] for more details) is computed by:

$$\text{Initialization:} \quad V(1, k) = P(y_1|k) \cdot \zeta_k \quad (3.3)$$

Recursion allows deriving the most probable state sequence $P(x_1 \dots x_T | y_1 \dots y_T)$ which produces the first t observations that have k as a final state. Recursion is derived by:

$$\text{Recursion: } V(t, k) = \max_{x \in X} (P(y_t | k) \cdot a_{x,k} \cdot V_{t-1,x}) \quad (3.4)$$

Every chosen $V(t, k)$ must be stored in order to retrieve the overall path ζ^* . The joint path and emission probability is

$$P(y | \zeta^*) = V(T, y) \quad (3.5)$$

For an HMM with $|S|$ states, the overall the Viterbi algorithm has the complexity $O(|S|^2 T)$.

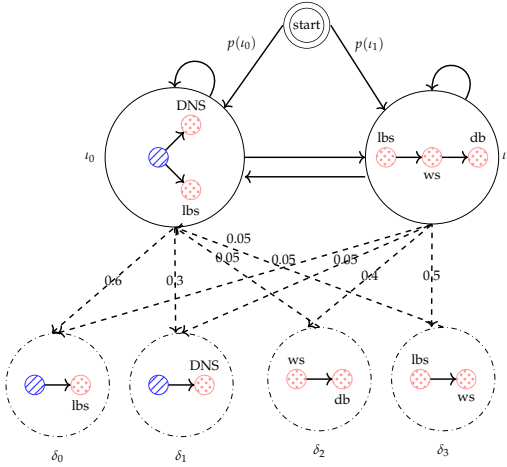


Figure 3.5: HMM workflow model example.

Example 3.10 (Viterbi Algorithm). To illustrate how the Viterbi Algorithm is applied on an HMM workflow model, Figure 3.5 introduces a HMM workflow model example. The

HMM workflow model illustrates state transition probability as well as the observation matrix. In addition, the initial starting probability is denoted by $p(i_0)$ and $p(i_1)$, which are shown within Figure 3.5. The Viterbi algorithm assumes a sequence of observables is known, so we suppose that the sequence δ_0, δ_0 is observed.

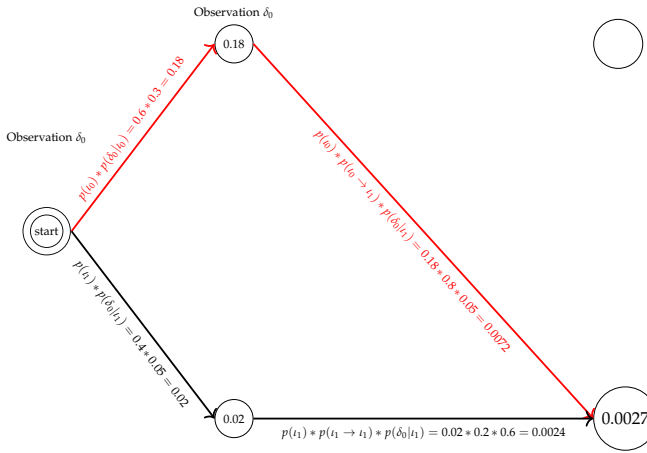


Figure 3.6: Viterbi algorithm on an HMM workflow model example.

The initialization step of the Viterbi algorithm takes the initial starting probability $p(i_0)$ or $p(i_1)$ of a hidden state i_0 or i_1 and combines this information with the probability $p(\delta_0|i_0)$ or $p(\delta_1|i_1)$ of the observed symbol δ_0 being emitted within the hidden state. The result from the initialization step is now taken as the probability $p(i_0)$ and $p(i_1)$ of being in the hidden state. The hidden state most likely to emit the observed symbol δ_0 becomes a part of the Viterbi path. The Viterbi path is drawn in red within this example.

The recursion step of the Viterbi algorithm takes the probability of being in a hidden state, the probability of emitting an observed symbol given that one is within the hidden state and then, additionally, takes the probability of a state transition into account. Figure 3.6 shows the initialization step of the Viterbi algorithm as well as the first recursion step with respect to the hidden state ι_1 .

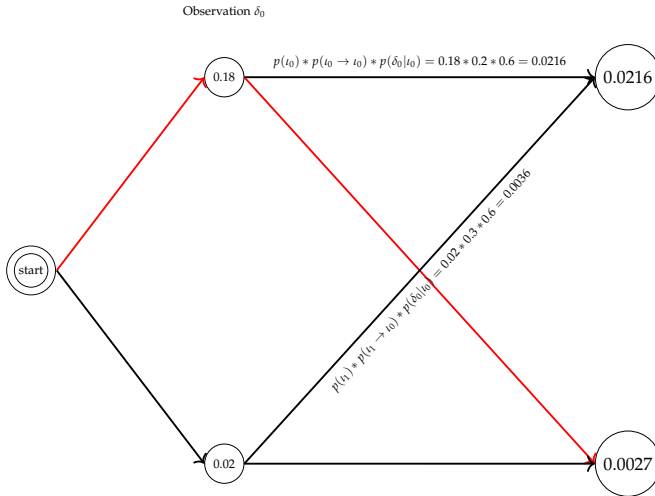


Figure 3.7: Viterbi algorithm on an HMM workflow model example.

Figure 3.7 illustrates the first recursion step with respect to the hidden state ι_0 . The overall Viterbi path is drawn in red and consists of the most likely sequence of states.

3.2.5 Extensions to Factorial Hidden Markov Model Workflow

The HMM-based workflow and identified Viterbi paths can be used to derive a Factorial Hidden Markov Model (FHMM) workflow. FHMM are a special case of an HMM, which were introduced in [GJ97]. FHMM extend HMM by modeling severally loosely coupled stochastic random processes. Given m HMM chains with n states for each variable an HMM with $m \times n$ variables is needed. Thus, applying the forward or backward algorithm for exact inference quickly becomes intractable. An example for a two layer FHMM is shown in Figure 3.8.

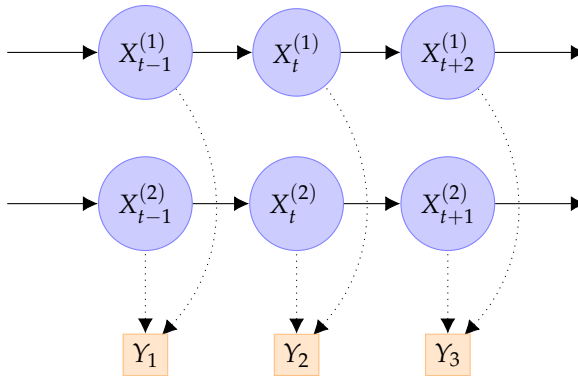


Figure 3.8: A Factorial Hidden Markov Model (FHMM) with two layers.

Every layer within an FHMM is an independent HMM, but the observation vector depends upon the state of all layers such that observed direct dependencies depend on workflows, represented by Viterbi paths, and hidden states.

To represent the dependency of the observation vector on hidden states and workflows equally, we derive a multi-state variable as a combination of all states such that

$$X_t = X_t^{(1)}, X_t^{(2)}, \dots, X_t^{(n)}$$

for a multi-state X_t at a time t and $1, 2, \dots, n$ layers. This multi-state definition is based on the FHMM introduced in [Che+09]. Within Figure 3.8 the multi-state variable is $X_t = X_t^{(1)}, X_t^{(2)}$. Additionally, every multi-state variable is independent from other state variables such that

$$P(X_t^{(n)} | X_{t-1}^{(n)})$$

the previous equation holds. A FHMM allows representing the dependency of observed direct dependencies on existing workflows, which are detected by the Viterbi algorithm, and hidden states, which are derived based on network service dependency detection.

3.2.6 Real-life Case Study

The disaster recovery site of an energy distribution network, provided by an Italian water and energy distribution company, was available for non-invasive experimentation. We integrated our framework into this test network to test the ability of our newly introduced workflow mining approach to rediscover workflows based on network traffic. The implementation of our introduced methodology is stream-based and provides a continuous analysis of network traffic in order to detect ongoing workflows. Within the operational environment, network traffic is mirrored by routers and switches in the test environment. Based on network traffic within this test environment, we are able to deploy MONA in order to detect network service dependencies.

Figure 3.9 shows all network service dependencies detected by MONA by representing network service dependencies as edges and network services as nodes. Based on

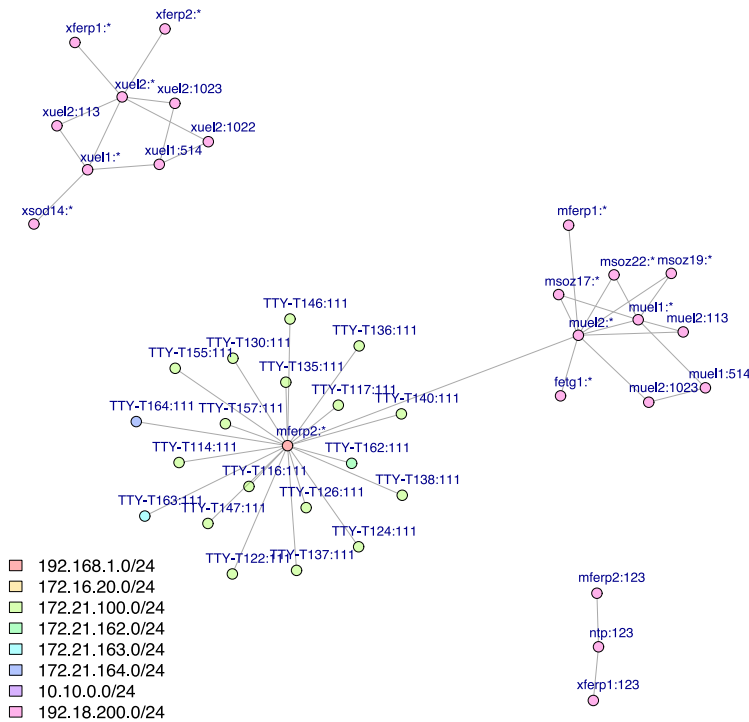


Figure 3.9: Network service dependencies detected by MONA within an energy distribution network.

SCADA protocols, remote terminal units { TTY-T114, TTY-T117, TTY-T122, TTY-T124, TTY-T126, TTY-T130, TTY-T135, TTY-T136, TTY-T137, TTY-T138, TTY-T140, TTY-T146, TTY-T147,

TTY-T155, TTY-T157 } in substations of medium voltage acquire data from electrical devices (e.g., programmable logic controllers, sensors, etc.) and send them via front end servers mferp1, mferp2 to the supervisory SCADA servers muell1 and muell2 of the power grids main office.

Within the emulation environment two front end servers xferp1, xferp2 and two supervisory SCADA servers xuell1 and xuell2 are emulated for monitoring high voltage substations, however no substation for high voltage exists. Therefore, SCADA software hosted on xuell1 and xuell2 interacts via xferp1 in search of possible remote terminal units. In addition, a human machine interface xsod14 is emulated to represent a network operator in charge of high voltage substations.

Within this real-life case study all network devices within the data-communication network of the electrical distribution company are included, regardless of whether they are monitored by network operators or not. No ground truth of all existing indirect dependencies was acquired beforehand, as MONA's ability to identify existing indirect network service dependencies within this test environment has already been evaluated in Section 2.4.1. A ground truth for workflows within this monitored network would require knowledge of all existing applications and their interactions. As the company owning this data-communication relies on third parties to provide services, as most networks do, network operators are not aware of all workflows within their monitored network. Thus, we discussed indirect dependencies identified by MONA afterwards with network operators and all identified workflows were classified as existing indirect network service dependencies by network operators.

In order to identifying workflows based on analyzing network traffic, capturing network service dependency dis-

covery as a probability spaces is the first step and, therefore, discussed in the following.

Probability Space

The purpose of Figure 3.10 is to illustrate the additional information contained in the probability space adds to network service dependencies. We chose to focus on representing the number of occurrences of network services within all observed indirect dependencies to point out characteristics of a probability space derived based on the network service dependencies shown in Figure 3.9.

Representing network services as nodes and the number of occurrences by the node size is obviously a more self-explanatory representation compared to representing network service dependencies as nodes. Network service dependencies link multiple network services and can only be understood with that knowledge. Of course, representing network service dependencies as nodes, while preserving this information is a very challenging task. Especially given that the number of network service dependencies exceeds the number of existing network services.

Figure 3.10 shows the number of occurrences network services within indirect dependencies contained within the network represented in Figure 3.9. The size of nodes within Figure 3.10 represent the number of occurrences of all network services. The more frequently a network service occurs, the larger it is. The representation was inspired by BPMN 2.0² as we model subnetworks as swimlanes to point out dependencies spanning multiple subnetworks.

To represent the size difference between nodes, we plot

²Bruce Silver and Bruce Richard. *BPMN method and style*. Vol. 2. Cody-Cassidy Press Aptos, 2009.

the nodes representing the minimum, average and maximum number of occurrences of a network service. Given $F = \{\iota_1, \dots, \iota_n\}$ with n distinct indirect dependencies, the minimum number of occurrences $\min(f_0)$ of a network service within $f_0 \in F$ is computed by

$$\min(f_0) = \min_{\iota_k \in f_0} C(\iota_k), \quad (3.6)$$

the maximum number of occurrences $\max(F)$ of a network service within F is computed by

$$\max(f_0) = \max_{\iota_k \in f_0} C(\iota_k) \quad (3.7)$$

and the average number of occurrences $avg(F)$ is derived by

$$avg(f_0) = \frac{\sum_{\iota_k \in f_0} C(\iota_k)}{n}. \quad (3.8)$$

Based on Equation 3.6, Equation 3.7 and Equation 3.8, the minimum, maximum and average number of network services occurrences are derived.

To point out the different activities that network services are linked to, we color coded nodes within Figure 3.10. In Figure 3.10 also an explanatory legend listing an activity's color coding and description is added. Additionally, nodes are labeled according to the hostname of the network device hosting the network service.

The representation of network services within the test environment shown in Figure 3.10 points out how important the SCADA communication server mferp2 is for the activity (see Definition 3.1) with the description "Management" on all remote terminal units $\{\text{TTY-T114}, \dots, \text{TTY-T157}\}$.

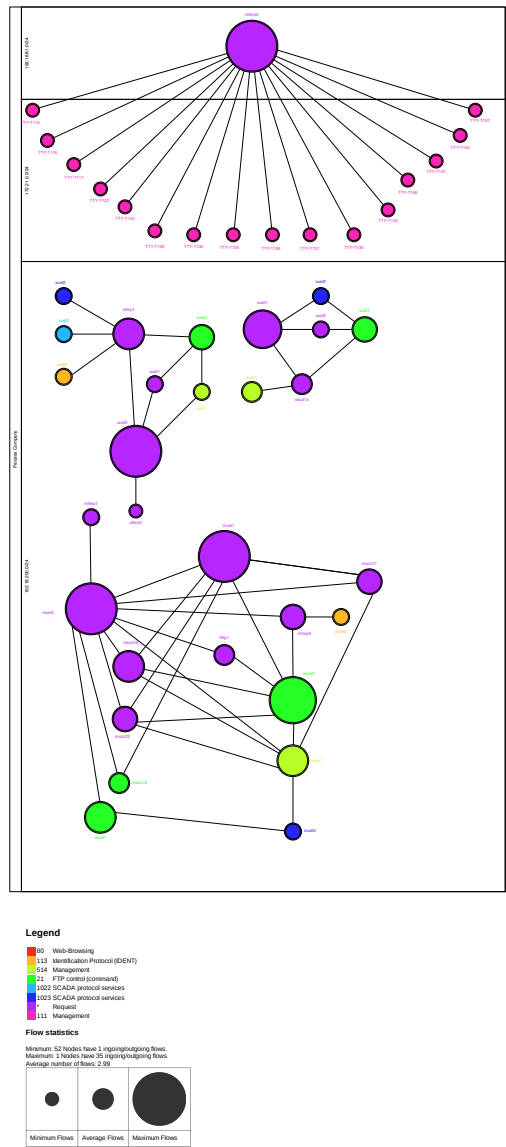


Figure 3.10: Activities derived from network traffic in an energy distribution network.

Network services hosted on the gateway servers for medium voltage substations muel1 and muel2 or the gateway servers for high voltage substations xuel1 and xuel2 are also of above-average importance. Network services hosted on human machine interfaces (HMI) msoz17, msoz19 and msoz22 interact with network services hosted by gateway server muel1 and muel2.

In order to monitor high voltage substations, HMI xsod14 interacts with network services hosted by gateway server xuel1 and xuel2. According to network operators monitoring power distribution networks, these network services hosted by gateway server xferp1, mferp2 and front end servers muel1, xuel1, muel2 and xuel2 are indeed of above average importance for keeping a power distribution network safe.

It is also interesting to see that the SCADA communication server mferp2 has two IP addresses in two different subnetworks. This can be seen as mferp2 is represented by two different nodes within two different subnetworks. The network device model introduced in Definition 2.1 allows a network device to have two distinct IP addresses. Based on this definition, we can attribute, as described in Definition 2.2, network services in different subnetworks to the same network device.

Compared to the network service dependencies shown in Figure 3.9, Figure 3.10 shows that a probability space derived based on detected network service dependencies contains a lot of additional information.

Discussing Figure 3.10 with security analysts confirmed that large network service nodes are also of higher importance for a power distribution network. The HMM state model is derived based on this probability space representation in Subsection 3.2.3.

Workflows

To allow human operators to analyze workflows, workflows are often represented by BPMN 2.0³ or Yet Another Workflow Language (YAWL)⁴. Thus, based on the probability space, we define the problem of mining such a workflow structure as the problem of finding the most likely sequence of hidden states. This is a problem often associated with HMMs. We are interested in the most likely sequence of workflows in a given communication data network. Given a predefined HMM, the most likely complete sequence of hidden states can be calculated using the dynamic programming Viterbi algorithm [For73]. Finding a sequences of hidden states with a probability that represents how likely a specific hidden state sequence is to be completed, allows us to determine whether a specific workflow is in place or not. Such a sequence of hidden states can be represented by BPMN 2.0⁵ and shown to network operators for qualitative validation of the workflows.

The hidden state set X , consisting of indirect dependencies, within the previously described workflows are:

- $\iota_1(\delta_{11}(msoz19 : *, muel2 : *), \delta_{12}(muel2 : *, muel : 2000))$
- $\iota_2((\delta_{21}(muel2 : 2000, mferp2 : 5002), \delta_{22}(mferp2 : 5002, mferp2 : *)))$
- $\iota_3(\delta_{31}(msoz22 : *, muel2 : 2000), \delta_{32}(msoz22 : *, muel2 : *))$

³Bruce Silver and Bruce Richard. *BPMN method and style*. Vol. 2. Cody-Cassidy Press Aptos, 2009.

⁴Wil M.P. Van Der Aalst and Ter A. H. M. Hofstede. *YAWL: Yet Another Workflow Language*. Tech. rep. 2003.

⁵Bruce Silver and Bruce Richard. *BPMN method and style*. Vol. 2. Cody-Cassidy Press Aptos, 2009.

- $\iota_4(\delta_{41}(msoz17 : *, muel1 : 2000), \delta_{42}(muel1 : 2000, muel2 : *))$
- $\iota_5(\delta_{51}(mferp2 : *, TTY - T116 : 111), \delta_{52}(mferp2 : *, TTY - T130 : 111))$
- $\iota_6(\delta_{61}(mferp2 : *, TTY - T136 : 111), \delta_{62}(mferp2 : *, TTY - T157 : 111))$
- $\iota_7(\delta_{71}(mferp2 : *, TTY - T150 : 111), \delta_{72}(mferp2 : *, TTY - T145 : 111))$
- $\iota_8(\delta_{81}(mferp2 : *, TTY - T158 : 111), \delta_{82}(mferp2 : *, TTY - T140 : 111))$
- $\iota_9(\delta_{91}(mferp2 : *, TTY - T147 : 111), \delta_{92}(mferp2 : *, TTY - T152 : 111))$
- $\iota_{10}(\delta_{101}(mferp2 : *, TTY - T126 : 111), \delta_{102}(mferp2 : *, TTY - T125 : 111))$
- $\iota_{11}(\delta_{111}(mferp2 : *, TTY - T139 : 111), \delta_{112}(mferp2 : *, TTY - T135 : 111))$
- $\iota_{12}(\delta_{121}(xsod14 : *, xuel1 : *), \delta_{122}(xuel1 : *, xuel2 : *))$
- $\iota_{13}(\delta_{131}(xuel2 : *, xferp2 : 5002), \delta_{132}(xuel2 : *, xferp1 : 5002))$

Workflows are sequences of indirect dependencies and two distinct workflows were identified within the operational environment. The first workflow is shown in Figure 3.11 and spans human machine interfaces msoz17, msoz19 and msoz22, which are communicating with medium voltage substation TTY-T116, TTY-T130, TTY-T136, TTY-T157, TTY-T150, TTY-T145, TTY-T158, TTY-T140, TTY-T147, TTY-T152, TTY-T126, TTY-T125, TTY-T139, TTY-T135 through front end server mferp2 and communication gateways muel2 and muel1.

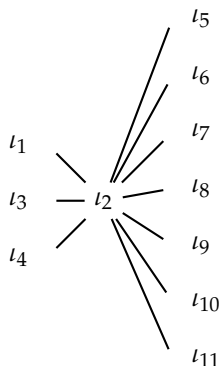


Figure 3.11: Workflow for communicating with medium voltage substations as identified within the real-life case study.

The second workflow spans human machine interface xsod14 which relies on communication gateways xuel1 and xuel2 to communicate with high voltage substations. Due to no high voltage substations being integrated into the test environment, no communication beyond xuel1 or xuel2 can be seen.

Both identified workflows have been verified by network operators as the main workflows within the test environment.

3.3 Network Vulnerability Assessment

In data-communication networks, network reliability is of great concern to both network operators and customers. Therefore, network operators want to determine what tasks

could be affected by software vulnerabilities being exploited that are present within their data-communication network. To determine what tasks could be affected by a software vulnerability being exploited, it is fundamentally important to know the ongoing workflows in a network. A particular task may depend on multiple network services, spanning many network devices.

Current network vulnerability approaches [Mur13] focus on identifying critical nodes in a network without focusing on the impact of software vulnerabilities. Even though, software vulnerabilities can be remotely exploitable and sometimes even exploits are readily available online, network vulnerability assessment currently does not take currently present known vulnerabilities into account.

Developing a deeper understanding of network activities allows network vulnerability assessment to analyze what network services would be potentially be affected by a software vulnerability that was detected in a monitored network. Knowing what network activities would be affected by a software vulnerability being exploited, supports network operators in developing a deeper understanding on how their network is affected by software vulnerabilities.

Vulnerability Assessment

Network vulnerability assessment consists of two parts: detecting present software vulnerabilities in a monitored network and analyzing a network's sensitivity to particular software vulnerabilities. In a monitored network, vulnerability scanners detect present software vulnerabilities. According to the ISO 27005 standard, a vulnerability is a "weakness of an asset or group of assets that can be exploited by one

or more threats”⁶. Whereas an asset is defined by ISO13355 ISO/IEC TR13355-1⁷ as being “anything that can have value to the organization, its business operations and their continuity, including information resources that support the organization’s mission”. Since 1999 the non-profit organization MITRE defines common Vulnerabilities and Exposures (CVE) identifiers for software vulnerabilities⁸. The purpose of vulnerability scanning is to identify all software vulnerabilities, which can be linked to a monitored data-communication network.

Vulnerability scanning

Vulnerability scanning⁹ a data-communication network is the process of assessing whether software vulnerabilities can be linked to monitored network devices. Software vulnerabilities can be linked to operating systems, software or firmware¹⁰. Network vulnerability analysis helps network operators to verify whether a software vulnerability linked to an application within the monitored network might endanger ongoing workflows. In the following we rely on the network model introduced in Chapter 2. Vulnerabil-

⁶ISO ISO and IEC Std. “ISO 27005: 2011.” In: *Information technology—Security techniques—Information security risk management*. ISO. 2011.

⁷ISO ISO and IEC Std. *ISO/IEC 13335-1: Management of information and communications technology security—Part 1: Concepts and models for information and communications technology security management*. 2004.

⁸MITRE. *Common Vulnerabilities and Exposures*. <https://cve.mitre.org/>. 2000.

⁹The Government of the Hong Kong Special Administrative Region. *An Overview of Vulnerability Scanners*. <http://www.infosec.gov.hk/english/technical/files/vulnerability.pdf>. 2008.

¹⁰Bhadreshsinh G. Gohil, Rishi K. Pathak, and Axaykumar A. Patel. “Federated Network Security Administration Framework.” In: 2013.

ity scanning provides us with a mapping function $SVULN$, which links CVE identifiers $cveId_j$ to network services in a monitored data-communication network.

Definition 3.11 (Link vulnerabilities to network services). Such that we are able to associate a network service S

$$SVULN : S \rightarrow CVEID$$

with a $CVEID \subseteq \Sigma^*$. Given that network device d is affected by a vulnerability $CVEID$, then $SVULN(d) = SVULN(HOSTS(d))$ lists all hosted network services are linked to a vulnerability. \diamond

Hence, we assume that an affected operating system will lead to an application hosted by that network device being compromised. A software vulnerability with confidentiality impact signifies the threat of information disclosure, in comparison a vulnerability with an integrity impact signifies the threat of data modification and a vulnerability with availability impact could lead to performance degradation. As network activities often span multiple network services for a higher mission, not only network services directly linked to a vulnerability could be affected by an attacker exploiting this vulnerability. All network services relying on requests or responses from a network service linked to a vulnerability with a confidentiality, integrity or availability impact could also be affected.

Consider a vulnerability with a confidentiality impact. Given that a network service is linked to this vulnerability, all information provided by other network services could be leaking, too. Hence, these network services would also be affected by data theft due to a cyber attacker exploiting this vulnerability. A network service, which is linked to a vulnerability with an integrity threat, implies that requests

sent from this network service could potentially be modified. Similarly, a network service relying on information from another network service, which is linked to a vulnerability with an availability impact, would also be affected by performance degradation of this vulnerability.

Definition 3.12 (Workflow-based Vulnerability Assessment). Based on a workflow HMM $\lambda = (X, Y, A, B, \Pi)$ the set of affected network services AS , which are directly affected by detected software vulnerabilities is defined as

$$AS = CC((SVULN(s_i), map(asSet, X))), \quad (3.9)$$

where CC denotes the connected components of the hypergraph given as parameter ($asSet$ maps a tuple into a set of components). \diamond

Definition 3.12 allows the context aware analysis of software vulnerabilities by taking affected workflows into account. This is possible by linking vulnerabilities detected by vulnerability scanners to network services as introduced in Definition 3.11.

Motivating Example

The disaster recovery site of an energy distribution network, provided by ACEA SPA¹¹, which is an Italian water and energy distribution company, was available for non-invasive experimentation. Based on this network, we are able to collect and analyze real-life network traffic and also scan the network for present software vulnerabilities. Figure 3.9 shows all network service dependencies detected by MONA. These network service dependencies were considered complete and correctly identified by network operators.

¹¹ACEA SpA. <http://www.acea.it/Home2.aspx?lang=en>. 2017.

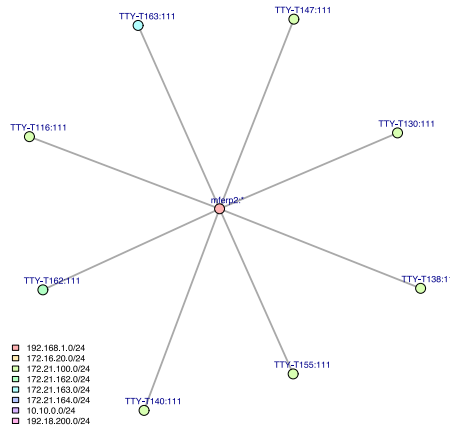


Figure 3.12: Workflow based vulnerability assessment for vulnerabilities CVE-2007-5423 and CVE-2010-2075, which were detected on mferp2.

Figure 3.12 shows the result of network dependency based vulnerability assessment for software vulnerabilities CVE-2007-5423 and CVE-2010-2075 that were detected via network scanning on network device mferp2. Both software vulnerabilities can be exploited with exploits readily available online. CVE-2007-5423 is a vulnerability that allows remote attackers to execute arbitrary code in TikiWiki 1.9.8 and CVE-2010-2075 is an unauthorized-access vulnerability due to a backdoor in UnrealIRCd 3.2.8.1. TTY-T[116-163] are remote terminal units of substations, which are dependent on requests from the front end server mferp2. Given the HMM workflow described in Subsection 3.2.6, workflow-based vulnerability assessment concludes that TTY-T[116-163] are affected by CVE-2007-5423 and CVE-2010-2075, which were detected

on mferp2. Hence, we understand that both vulnerabilities affect the workflow communicating with medium voltage substations, which was illustrated in Figure 3.11. This provides network operators in charge of patching software vulnerabilities with contextual information and allows them to better prioritize the chronological sequence of patching.

3.4 Discussion

In the context of this chapter, we introduced an approach for continuous workflow mining based on Mission Oriented Network Analysis (MONA) (see Chapter 2). MONA is a passive network service dependency discovery method, which analyzes network traffic in order to correlate communication patterns of interacting network services. Based on correlated communication patterns, MONA is able to derive network service dependencies within a monitored infrastructure. Automatic network service dependency discovery is the foundation for deriving an HMM based workflow model. To the best of our knowledge this is the first workflow mining approach, which is able to deduce an HMM based workflow model by analyzing network traffic.

To evaluate the ability of this workflow mining approach to discover ongoing workflows within an enterprise network, we tested this methodology within the data-communication network of an energy distribution company. In the context of our experimental evaluation, we came to the conclusion that network operators have a high level understanding of workflows in their monitored network. However, they lack a detailed understanding on what applications and network services are involved. This was generally due to the energy distribution network, which we were able to use as a test

environment, relying heavily on third party software. Third party software is often also updated and maintained by the third party, thus network operators are often not aware of updates and modifications. Thereby, we concluded that deriving manual workflow models is costly and requires specialist know how and good communication within a company.

Luckily, network traffic based workflow mining can support network operators in understanding workflows in their monitored network on application layer level. We discovered thus discrepancies by automatically deriving workflows and discussing them with network operators. Automatically derived workflows were found more detailed and accurate than manually derived information. Although, it should be noted that applications cannot be observed directly within network traffic, but are deduced indirectly based on network services that applications chose to communicate through.

Based on HMM workflows, we have introduced a novel workflow-based vulnerability analysis approach. Workflows are derived based on network service dependency analysis, which allows automatically capturing ongoing network activities with a workflow model. Based on automatically mined workflows, we are able to link exploitable software vulnerabilities to ongoing network activities. The proposed framework is fully automated and is able to integrate vulnerability specification from the bug-reporting community and helps network operators develop a deeper understanding on how networks are affected by software vulnerabilities.

After investigating network-based vulnerability assessment, in the following we will investigate security information and event management with the aim of reducing the overall number of events and using workflows to add a contextual understanding of events.

Event Prioritization and Correlation

Systems providing real-time analysis with the aim of reducing the overall number of reported low level events are referred to as Security Information and Event Management (SIEM) systems. In addition, some SIEM systems focus on adding contextual information to low level events in order to support network operators with interpreting events. Workflow mining, as introduced in Chapter 3, allows us to understand how network services interact within a joint network activity for a common mission. Thereby, we are able to link events to ongoing workflows and prioritize them according to this information. Thus, in the following we will investigate security information and event management with the aim of reducing the overall number of events and using workflows to add a contextual understanding of events.

The structure of this chapter is outlined in the following. First a general introduction to the topic of event prioritization and correlation is given in Section 4.1. Section 4.2 describes how events provided by heterogeneous IDS sensors, IPS sensors and FWs are normalized, verified and merged. In the content of this work we have conducted a network de-

pendency analysis (see Chapter 2) based on network traffic to identify network activity patterns. Section 4.3 illustrates how, based on identified network activity patterns, cyber incidents can be linked to ongoing network activities within a monitored network. Events are reported by IDS sensors, IPS sensors or FWs with monitored data-communication networks. Section 4.4 outlines a systematic evaluation of the introduced event correlation.

4.1 Introduction

As the United States intelligence community has identified malicious actors exploiting cyberspace as a top national security threat [Cla14], protecting enterprise networks from cyber attackers has become increasingly important. A first step to cyber defense is perimeter protection. Perimeter protection in data-communication networks is primarily achieved through Firewalls (FWs), which aim to keep cyber attackers out. To achieve in-depth cyber defense, it has to be assumed that perimeters can be penetrated. To illustrate this point, IBM's 2015 cyber security intelligence index¹ reveals that approximately half of all cyber attacks originate from within a company's own network. Hence, a second step to cyber defense is to be aware of malicious events within a monitored network. For this purpose, Intrusion Detection Systems (IDSs) have been developed to monitor an enterprise network for malicious events. If actions can be taken based on malicious activities identified by security sensors, this security system is referred to as Intrusion Prevention System (IPS). An example for such an action is dropping a packet that was determined to be malicious and blocking

¹IBM Corporation. 2015 *Cyber Security Intelligence Index*. July 2015.

all further traffic from this IP address. Legitimate network traffic should be forwarded to its intended destination with no apparent disruption or delay of service.

With the growing deployment of IPS sensors, IDS sensors and FWs in increasingly large and complex communication networks, managing information provided by these systems becomes critically important. An IDS or IPS can either be a software or hardware-based system and is classified as host-based or network-based IDS. Network-based IDS or IPS sensors are generally placed in positions where high volumes of network traffic occur, as they can only report on malicious activity that they are able to observe. Similar to the workflow mining methodology introduced in the previous chapter, network-based security sensors analyze network traffic. With, however, a different goal: network-based security sensors, just as all security sensors, aim to identify malicious behavior within a monitored network.

Host-based IDS sensors are installed on hosts and locally monitor activities. Host and network-based IDS or IPS sensors rely on signatures or algorithms to monitor hosts or networks for malicious activities and resulting reports of potentially malicious activities are referred to as events. If an event is caused by an IDS that has observed evidence of malicious abuse, the event is referred to as alert. To summarize: In the context of this work, on the one hand, any network activity, whether it is benign or malicious, is referred to as an event, on the other hand, alerts refer to evidence of malicious activities reported by security sensors.

IDS, IPS and FW monitor networks in order to detect malicious activities. These systems are generally based on a low-level attacker model and therefore report low level events. Thus, knowledge of both areas, networking technology and infiltration techniques, is required to correctly

interpret large amounts of highly paced low-level events. Low-level event correlation has been investigated in electric power systems [WKK07], chemical processes [Zhu+14] and patient-care monitoring systems [KOB12].

In the context of this work we focus on the interpretation and correlation of events. We understand events as an external manifestation of exceptional conditions occurring during the everyday operation of software or hardware within a monitored system. Events are reported cyber incidents, and events can only be prioritized if the implications of a network service or network device failing are understood. For example, a cyber incident that is linked to a critical network service needs to be prioritized over an incident that is linked to a network service of little importance. Therefore, we propose low-level event correlation in order to reduce the overall number of reported events and context-aware event analysis based on automatically learned workflows. The methodology proposed in the following was implemented as software component referred to Low-Level Correlator (LLC) in the following sections.

Often, multiple heterogeneous security sensors are deployed to monitor enterprise networks. To allow the integration of distributed heterogeneous IPS sensors, IDS sensors or FWs, cyber incidents can be reported as Syslog² messages. Syslog messages contain information provided by IPS sensors, IDS sensors or FWs. Depending on the sensor and the respective configuration, provided information can differ. If a sensor reports abnormal behavior, the alert could point towards an on-going cyber attack. If a FW or IPS sensor detects the cyber incident, these security sensors are also able to report on actions taken against an observed cyber

²R. Gerhards. *The Syslog Protocol*. Mar. 2009. URL: <http://www.ietf.org/rfc/rfc5424.txt>.

incident. Unfortunately, network administrators cannot manage the number of Syslog messages occurring per second in a real data-communication network. Hence, an emerging track of security research [PFV02; NCR02; CLF03; HS14] has focused on Syslog message correlation to identify potentially relevant cyber incidents and analyzing cyber incidents.

In the following in Section 4.2, we look into real-time event analysis with the purpose of reducing the overall number of reported events. Reducing the overall number of reported events provides the foundation for operational-impact based event correlation.

4.2 Method Description

Cyber security sensors, such as IDS sensors, IPS sensors, or FWs, report on cyber incidents within a monitored network. These cyber incident reports are referred to as events. Heterogeneous cyber security sensors rely on different data formats to pass on reports on cyber incidents. The goal of LLC is to provide for event normalization, verification, and merging. In addition, LLC introduces operational impact based event correlation based on automatically mined workflows.

Event normalization transforms events with heterogeneous data formats into a common data format and is introduced in Subsection 4.2.1. Sometimes events mention potential vulnerabilities being exploited with the monitored network. Of course, vulnerabilities within a monitored network can only be exploited if they exist within the network without any countermeasures addressing them, e.g., added patches or FWs blocking malicious actors from accessing the vulnerability. Thus, vulnerability scanners have the aim of identifying vulnerabilities within a monitored network. The

purpose of event verification is to link events to vulnerabilities within a monitored network and determine whether a vulnerability has been detected by vulnerability scanners to be present on the targeted network device or not. Event verification is described in Subsection 4.2.2. Often a single cyber incident can be lead to multiple events being reported. Event merging aims to fuse events that were caused by the same cyber incident. This is presented last in Subsection 4.2.3.

4.2.1 Event Normalization

In the overall architecture, the above-mentioned security sensor report events describing security incidents. These events are shared via Syslog³ and fed into a Syslog engine to allow further processing. Before correlating events, event normalization is necessary, due to the heterogeneous nature of data formats used by cyber security sensors. This allows for coherent processing of all events in subsequent processing steps.

Following the recommendation of [Cup01; CM02], we focus on retaining the essential attributes of an event: sensorID, eventID, source IP address, destination IP address, source port, destination port, create time and event type. An example for an event with these essential attributes is given in Table 4.1.

sensorID	eventID	source IP address	destination IP address	source port	destination port	create time	event type
10	270	85.1.1.8	132.8.1.5	49154 UDP	2404 UDP	2015-01-24 11:02:31	1

Table 4.1: Selected fields of an event.

³Balabit. *Syslog-ng*. <https://www.balabit.com/network-security/syslog-ng>. 2015.

In an environment with multiple heterogeneous cyber security systems such as IDS sensors, IPS sensors, and FWs, cyber incidents are reported in inherently heterogeneous data formats. Some cyber security systems report cyber incidents with a data format containing only basic information, such as source, target, name and time of the reported cyber incident. Other cyber security systems provide more details, such as ports or the network services, process information, and more. The Intrusion Detection Message Exchange Format (IDMEF) defines a data model that is able to accommodate those different needs. In the context of this work, we rely on IDMEF to model events and introduce our data model in the following definition.

Definition 4.1 (Event). A reported cyber incident is defined as an element of *EVENT* which is introduced as:

$$\begin{aligned} EVENT = & Analyzer \times \\ & T \times T \times T \times Source \times Target \times \\ & Classification \times Assessment \end{aligned}$$

with *Analyzer* = *analyzerId* \times *Node*, Time stamp *T*, *Source* = *Node* \times *process* \times *user* \times *service*, *Target* = *Node* \times *process* \times *user* \times *service*, *Node* = *location* \times *name* \times *address*, *Classification* = *text* \times *Reference* with *Reference* = *name* \times *url* and *Assessment* = *impact* \times *confidence* \times *action*.

Names starting with lowercase letters denote variables and names starting with uppercase letters denote aggregation classes for other classes and variables. A description of the classes and variables in our data model is listed in the following.

- **Analyzer:** Identifying information about the sensor sending the event.

- **Node:** Information about the network device.
 - * **Address:** IP address
 - * **Name:** Hostname
 - * **Location:** Equipment location hosting the network device
- **CreateTime** T: Timestamp denoting when the event was created.
- **DetectTime** T: Timestamp of the cyber incidents causing the event to be created. If there are multiple incidents, the timestamp of the first incident is used.
- **AnalyzerTime** T: Timestamp denoting when the analyzer sent the event.
- **Source:** Possible origin of the event. Information about the apparent cause of the event is given by the strings Node, Process, User and Service.
- **Target:** Possible target of the event. Information about the apparent cause of the event is given by the strings Node, Process, User and Service.
- **Classification:** Description of the potential cyber incident allowing the operator to determine what is going on.
 - **text:** Description of the cyber incident
 - **Reference:** Descriptive information about a cyber incident.
 - * **name:** Name of the alert or vulnerability identifier
 - * **url:** A url for additional information.
- **Assessment:** An event's impact, confidence within this report and action taken against the event.

- Impact: Severity of the event
- Confidence: Score describing the level of confidence an analyzer has in its own evaluation of a reported event
- Action: Action(s) taken by an analyzer against an event.



Alert normalization is realized by a specific Syslog configuration and means transforming heterogeneous events, such as described in Table 4.1, into the data format described in Definition 4.1. Descriptive attributes and structure of normalized events are inspired by attributes and structures defined by the Intrusion Detection Message Exchange Format (IDMEF). For details on IDMEF see [DCF07]. After normalizing events, LLC proceeds with event verification. In the following we describe details on event verification using examples with normalized event.

4.2.2 Alert Verification and Enrichment

Not all events reported by security sensors are alerts. Some events report on potential malicious activities detected by security sensor. We refer to these events as alerts. Alert verification and enrichment is subdivided into three parts:

- Alert vulnerability verification,
- Alert severity verification, and
- Alert vulnerability enrichment.

Alert vulnerability verification is described first, with the central idea of correlating incoming alerts with pre-acquired

knowledge of existing vulnerabilities in a monitored network. This knowledge is acquired by relying on vulnerability scanners. Vulnerability scanners provide a listing of all monitored network devices and present vulnerabilities. To detect vulnerabilities in a monitored network, a network is periodically scanned with security or vulnerability scanners. An example for an open source security scanner NMAP⁴ or vulnerability scanner OpenVAS⁵.

The LLC process relies on information provided by security or vulnerability scanners to link normalized alerts to the monitored network. If an alert does not provide vulnerability information, LLC adds the vulnerabilities known for the device the alert refers to and generates an LLC alert. This is referred to as alert vulnerability enrichment. Normalized alerts can have many fields, and in the following special cases are described in order to demonstrate central ideas using examples.

Alert Vulnerability Verification

Alert vulnerability verification aims to identify whether an event is trying to exploit a vulnerability known to be present on the network. Thereby, alerts that report a cyber attacker trying to exploit a vulnerability not present within the monitored network are less relevant to a network operator than alerts reporting a cyber attacker trying to exploit a vulnerability present within the network. We refer to this process as alert vulnerability verification. LLC can be configured by network operators to eliminate alerts with not verified vulnerabilities.

⁴Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

⁵Greenbone Networks GmbH. *The Open Vulnerability Assessment System (OpenVAS)*. 2016. URL: <http://www.openvas.org/>.

Analyzer ID	Alert ID	Time	Source IP address	Destination IP address	CVE ID
CEDET01IDS	1	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	CVE-2016-0034

Table 4.2: An alert issued by an IDS probe with the analyzer ID CEDET01IDS reports a local exploit.

Let us assume that an event (see Definition 4.1) is reported by an IDS with the analyzer ID CEDET01IDS. The reported event contains a reference to a local exploit taking advantage of a vulnerability. For the example in Table 4.2 we assume that an IDS (CEDET01IDS) has generated an alert referring to the CVE of a vulnerability. An exploit can only be successful if the respective vulnerability or bug is indeed present on the targeted network device. Thus, alert vulnerability verification needs to check whether the vulnerability has been detected on the targeted device by extracting respective information from the network inventory. All vulnerabilities detected in the monitored network are listed in the network inventory.

Definition 4.2 (Network inventory). A network inventory lists all vulnerabilities that were detected by vulnerability scanners as present within a monitored network. Network scanners scan all network devices and link vulnerability identifiers $v_0^{id}, v_1^{id}, \dots, v_n^{id} \in \Sigma_v^*$ to the IP addresses IP of the network device they were detected on. We model a vulnerability N as a link between a vulnerability identifier v^{id} and an IP address IP as

$$N : IP \rightarrow \mathcal{P}(\Sigma_v^*). \quad (4.1)$$

This links an IP address IP belonging to a network device, as introduced in Definition 2.1, to vulnerability identifier v^{id} .

◇

Example 4.1 (Network inventory). For the example in Table 4.3 we assume that a vulnerability scanner linked IP address 132.8.1.5 to vulnerability identifier CVE-2016-0034.

IP	Vulnerability identifier
132.8.1.5	CVE-2016-0034

Table 4.3: Vulnerability information extracted from the network inventory.

The process of alert vulnerability verification relies on the network inventory as a knowledge base to link FW/IDS/IPS alerts to the detected vulnerabilities in monitored network. The network inventory is a list of vulnerabilities detected by vulnerability scanners within a monitored network. Vulnerability scanners rely on heuristics methods to detect the presence of vulnerabilities, hence there are false positive and false negative vulnerabilities. Our understanding of false positive and false negative vulnerabilities is described in the following paragraph. The following understanding only refers to known vulnerabilities.

False Positive Vulnerability

Vulnerability scanners have the purpose of detecting vulnerabilities that are present within a monitored system. Similar to IDS designers, algorithms of vulnerability scanners are forced to make assumptions on whether a vulnerability is present or not. So if a vulnerability scanner classifies a vulnerability as present, however the monitored host is not vulnerable to this vulnerability being exploited, then this is referred to as a false positive vulnerability. A false positive

vulnerability can for example be caused by a vulnerability already being patched. Another possibility that can lead to a false positive vulnerability is that a vulnerability scanner's assumption of a vulnerability being present is imprecise or wrong.

False Negative Vulnerability

Vulnerability scanners rely on heuristic algorithms and therefore do not inevitably detect all vulnerabilities in a monitored system. Especially unknown vulnerabilities, often cannot be detected based on the underlying algorithms of state-of-the art vulnerability scanners. Alerts can be caused by an attacker trying to exploit a vulnerability, however the issued alert does not contain a link to the vulnerability. Thus, we state that such an alert is linked to a false negative vulnerability. The process of enriching alerts with corresponding vulnerability information, essentially aims to reduce potentially false negative vulnerabilities.

Cyber security sensors report on vulnerabilities potentially being exploited with the monitored network. Similar to vulnerabilities scanners, cyber security systems rely on heuristic methods and, thereby, also produce false positive and false negative alerts. Our understanding of false positive and false negative alerts is explained in the following paragraphs.

False Positive Alert

Should an IDS alert report evidence of malicious abuse, however the reported incident is actually benign, then the alert reporting the incident is commonly referred to as false positive. From the perspective of an IDS, this is not an error. The underlying algorithm is just making an assumption that

does not only reflect malicious behavior but also reflects benign network activities. Models of malicious behavior are referred to as attack models. IDS designers are forced to make assumptions in their attack models on how to detect malicious network activities, hence false positive alerts commonly occur with all state-of-the art IDS.

False Negative Alert

Given that a cyber attack occurred, but an IDS was not able to detect this attack, then this is referred to as a false negative alert. IDS designers make assumptions on how to detect malicious network activities. An undetected cyber attack implies that no evidence of malicious activities could be observed according to the attack model of the IDS. Thus it can be concluded that the assumptions made by the IDS designer within the attacker model were insufficient for picking up evidence for this cyber attack. Especially more skilled cyber attacks often lead to false negative alerts due to a skilled attacker's malicious actions not being reported by cyber security sensors. Generally, reducing the number of false negative alerts often increases the number of false positive alerts. This is due to the following observation: Adding more assumptions to an attacker model often reduces the number of false negative alerts, however more benign network activities will also be classified as evidence for malicious network activities. This is especially true, if an attacker model tries to capture a more skilled cyber attacker. If an attacker model is supposed to represent a skilled attacker, any activity with even a remote possibility of indicating an attack will trigger an alert.

For the purpose of alert vulnerability verification, the network inventory shown in Table 4.3 is searched for vul-

nerabilities present on the targeted network device of the reported cyber incident. Considering for example an alert, as shown in Table 4.2 an alert vulnerability verification requires checking the network inventory for vulnerabilities present on 132.8.1.5.

Definition 4.3 (Alert vulnerability verification). To derive the set of all vulnerabilities that are listed in the network inventory for an IP address IP , we define

$$VERIFYVULN : IP \rightarrow \mathcal{P}(N^{id}) \quad (4.2)$$

for a set of vulnerability identifiers N^{id} . To continue the example, we assume that data shown in Table 4.3 are available and $VERIFYVULN(132.8.1.5)$ indicates that CVE-2016-00034 is indeed associated with 132.8.1.5. \diamond

Example 4.2 (Alert vulnerability verification). The network inventory allows the alert vulnerability verification process to link a reported local exploit to vulnerabilities detected on source and destination host, as shown in Table 4.3. Table 4.4 describes how an LLC alert is issued based on this alert verification process.

Analyzer ID	Alert ID	Time	Source IP Address	Destination IP Address	CVE ID	Tag
CEDET01IDS	1	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	CVE-2016-00034	VULNVERIFIED

Table 4.4: An LLC alert with a verified vulnerability with an added tag VULNVERIFIED.

The purpose of alert vulnerability verification is to verify whether a vulnerability, which is reported within an alert, is a known vulnerability that is present within the targeted

network. However, as previously mentioned, vulnerabilities can be unknown and false positive vulnerabilities can be reported by security or vulnerability scanners. Thus, we introduce alert severity verification in the following in order to prevent alerts with a high severity being eliminated.

Alert Severity Verification

Alert severity verification has the goal of ensuring that alerts with a high severity are not eliminated.

Definition 4.4 (Alert severity verification). To support alert severity verification, severity values \mathbb{N} are predefined. The predefined severity values depict severity levels that describe a negative impact. Given an event from *EVENT* could not be linked to any vulnerability, then an event's severity value $VERIFYSEVERITY(e)$ is correlated to predefined severity values. If the severity value $VERIFYSEVERITY(e)$ is larger than a predefined threshold then true is return, otherwise false is returned. This comparison is described by the following definition:

$$VERIFYSEVERITY : EVENT \rightarrow \mathbb{N}_0 \quad (4.3)$$

If the severity event verification $VERIFYSEVERITY(e)$ return true, the even is considered to be an alert and, therefore, is passed as an LLC alert. \diamond

Given that cyber security sensors pick up a potentially malicious network activity, an alert is issued. Given a sensor is certain of abnormal activity with a negative impact occurring, a data field addressing the severity of the alert is added. Not all alerts stem from a vulnerability being exploited and, therefore, some alerts cannot be linked to vulnerabilities.

Example 4.3 (Alert severity verification). To continue the example, we assume that severity level values critical = 2, high = 1 and medium = 0 were predefined as severity levels, and an event $e \in EVENT$ is reported by an IDS sensor with the analyzer ID CEDET01IDS as shown in Table 4.5 occurred.

Analyzer ID	Event ID	Severity	Source IP Address	Destination IP Address	Time
CEDET01IDS	2	0	85.1.1.8	132.8.1.5	2016-01-24 11:02:31.20

Table 4.5: Example for an event reported by IDS sensor CEDET01IDS with medium severity.

The predefined severity threshold is 0 and, therefore, all events with a severity value equal to or larger than 0 are reported as severity verified events. Hence, the event's severity $VERIFYSEVERITY(e)$ might not be vulnerability verified, however it is severity verified. Depending on LLC's configuration, this alert is either eliminated or this alert is passed on for further consideration with a tag NOTVERIFIED as shown in Table 4.6.

Alert ID	Analyzer ID	Severity	Source IP Address	Destination IP Address	Time	Tag
2	CEDET01IDS	0	85.1.1.8	132.8.1.5	2016-01-24 11:02:31.20	NOTVERIFIED

Table 4.6: Example for a non verified severity alert.

Alert Vulnerability Enrichment

Often IPS, IDS or FW sensors do not contain references to vulnerabilities within reported events, although reported events

could be caused by a vulnerability present within the monitored network being exploited. Hence, LLC enriches alerts with vulnerability information from the network inventory. The process of enriching alerts containing no vulnerability information relies on adding vulnerability identifiers to the issued LLC alerts. This is achieved by checking the network inventory for vulnerabilities on the network device linked to an alert's targeted IP address.

For the example in Table 4.7 we assume that an IDS (CEDET01IDS) has generated an alert referring to no CVE vulnerability. By applying Definition 4.3, the targeted

Analyzer ID	Alert ID	Time	Source IP Address	Destination IP Address	CVE ID
CEDET01IDS	1	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	

Table 4.7: An alert issued by an IDS probe with the analyzer ID CEDET01IDS reports a local exploit.

IP address 132.8.1.5 is checked for present vulnerabilities *VERIFYVULN*(132.8.1.5) within the network inventory as shown in Table 4.3. The results indicate that CVE-2016-00034 is indeed associated with 132.8.1.5. Thus, as shown in Table 4.8, an LLC alert with the CVE identifier CVE-2016-00034 and the tag NOTVERIFIED is created.

Analyzer ID	Alert ID	Time	Source IP Address	Destination IP Address	CVE ID	Tag
CEDET01IDS	1	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	CVE-2016-00034	NOTVERIFIED

Table 4.8: An enriched LLC alert with an added vulnerability identifier with an added tag NOTVERIFIED.

After normalizing incoming Syslog messages, vulnerabil-

ities contained within incoming Syslog messages are verified. In addition, alerts are enriched with vulnerability information. This information supports network operators in analyzing security events and identifying potential attacks within an enterprise network. To overcome limitations that stem from perusing a vulnerability centric approach, the low level correlation process offers additional analysis, such as alert vulnerability verification, alert severity verification, and alert vulnerability enrichment. The purpose of this additional analysis is to add contextual information to alerts and support network operators in understanding potentially malicious incidents within their monitored networks. The additional analysis is optional and can be activated and deactivated as necessary.

4.2.3 Event Fusion

FWs, IDS sensors or IPS sensors report alerts, which are indications for abnormal activities in a monitored network. As argued above, some alerts are clearly linked to an exploited vulnerability or are assigned a high severity by the cyber security sensor reporting the abnormal activity. However, the vast majority of events are simply reports of observable benign security policy violation occurring within a monitored network. Also, in 2015 Check Point reported that zero day attacks are rising⁶. Check Point discovered that organizations are targeted by 106 unknown malware attacks per hour. Thus, reporting a rate 48 times higher than the rate reported in 2013. As these vulnerabilities are unknown, they cannot be detected by vulnerability scanners or be known to

⁶Check Point. *Check Point Security Report*.
<https://www.checkpoint.com/resources/2015securityreport/CheckPoint-2015-SecurityReport.pdf>. 2015.

IDS sensors, IPS sensors or FWs.

Insider attacks might not be reported as malicious activities in an information system, but as an observable occurrences within the monitored network. Targeted attacks on an information system are often prefaced by reconnaissance attacks. An example for such reconnaissance attacks are port scans. Port scans do not automatically imply an ongoing cyber attack, as monitoring tools also conduct port scans to search for open ports or vulnerabilities in a network. However, targeted attacks require mapping the targeted network by investigating information systems for example through port scans in order to find out how they can be attacked. Depending on the configuration of the monitoring cyber security sensors, port scans are often reported as events rather than alerts.

We argue that misconfiguration, insider attacks, or reconnaissance attacks, e.g., port scans, can lead to multiple events being issued. Separately, every alert cannot necessarily directly be attributed to malicious behavior. Hence, network operators are likely to overlook these events. By automatically merging events with the same source and/or same destination address, events possibly caused by the same cyber incident are fused and passed on as LLC alert to the network operator. We attempt to perform event fusion to report multiple events that occurred within the same time period as indicators for an ongoing cyber incident.

The low level correlation process keeps a tumbling time-based window of events. In the context of this work, the size of the tumbling time-based window is predefined as a 1 second window. On the one hand, this allows for a constant computation time of the low level correlation process, regardless of how long the component is running, on the other hand, events can only be fused, if they are in the same

window. Hence, it introduces false negatives into the low level correlation process as events sorted into different time windows cannot be merged into a common LLC alert. Time windows, in the context of this work, are tumbling windows with a 1 second tumble.

Duplicate Event Fusion

The goal of duplicate event fusion is to collect events reporting the same ongoing cyber incidents into a single LLC alert. For accomplishing this, alerts referring to nodes (see Definition 4.1) with the same source and destination IP address and port are merged into a common LLC alert. It should be noted that a single cyber security sensor is able to produce duplicate events when a cyber incident matches multiple rules. This phenomenon is also referred to as event splitting. Given that there are multiple duplicate events describing the same cyber incident, displaying all available cyber security sensors as condensed as possible reduces the overall number of reported events without losing potential sources of information.

Definition 4.5 (Merging Events with same Source and Destination Address). Consider two events e_i and $e_j \in Event$. Given that they have the same source IP address and port, we merge both events e_i and e_j into one LLC alert. This process is described in the following term:

$$\prod_{\substack{\text{Source.Node,} \\ \text{Target.Node}}} e_i \bowtie_{\substack{e_i(\text{Source.Node})=e_j(\text{Source.Node}) \\ e_i(\text{Target.Node})=e_j(\text{Target.Node})}} e_j$$



Example 4.4 (Merging Events with same Source and Destination Address). Table 4.9 describes how an IDS sensor with an

analyzer ID CEDET01IDS splits incident reports caused by the same cyber attack into multiple alerts. The low level correlation process performs duplicate low-level event fusion by merging events into an LLC alert. The LLC alert combines all time stamp into a common data field. The description of both alerts within Table 4.9 and Table 4.10 is taken from events caused by exploiting a vulnerability with the identifier CVE-2006-3439. However within both tables, the alert descriptions are shortened and originally were MSRPC-TCP_CPS-Microsoft-Windows-Server-Service-Buffer-Overrun and CPS-Windows-MSRPC-SRVSV-Unicode-Buffer-Overflow. Thus, similarly to all previous examples, the following example is taken from a real-life occurrence. Taking the same example

Analyzer ID	Event ID	Source IP Address	Destination IP Address	Time	Description	Classification ID
CEDET01IDS	1	85.1.1.8	132.8.1.5	2016-01-24 11:02:31.10	Windows-Server-Service-Buffer-Overrun	
CEDET01IDS	2	85.1.1.8	132.8.1.5	2016-01-24 11:02:31.20	Windows-Unicode-Buffer-Overrun	
CEDET01IDS	{1,2}	85.1.1.8	132.8.1.5	{2016-01-24 11:02:31.10, 2016-01-24 11:02:31.20}		DUBLIMERGE

Table 4.9: Example for merging duplicate events issued by the same sensor with analyzer ID CEDET01IDS.

of an exploit of a known vulnerability being executed, it is possible that the cyber attack is detected by distinct IDS sensors, IPS sensors or FWs. Given source and destination with events are identical, the low level correlation processes performs event fusion. Table 4.10 describes how two IDS sensors with an Analyzer ID CEDET01IDS and CEDET02IDS both report events caused by the vulnerability being exploited. The low level correlation process performs duplicate event fusion by merging them into a single LLC alert. The LLC alert lists the time stamps of the two alerts in a single data

field.

Analyzer ID	Event ID	Source IP Address	Destination IP Address	Time	Description	Classification ID
CEDET01IDS	1	85.1.1.8	132.8.1.5	2016-01-24 11:02:31.10	Windows-Server-Service-Buffer-Overrun	
CEDET02IDS	2	85.1.1.8	132.8.1.5	2016-01-24 11:02:31.20	Windows-Unicode-Buffer-Overrun	
{CEDET01IDS, CEDET02IDS}	{1,2}	85.1.1.8 85.1.1.8	132.8.1.5	{2016-01-24 11:02:31.10, 2016-01-24 11:02:31.20}		DUBLIMERGE

Table 4.10: Example for merging duplicate events issued by the same sensor with Analyzer IDs CEDET01IDS, CEDET02IDS.

Event fusion is not only applied to merge duplicate events. In the following section we will introduce how event fusion is additionally used to merge events with the same source or destination address.

Merging Events with same Source or Destination Address

Definition 4.6 (Merging Events with same Source Address). Consider two events e_i and $e_j \in Event$. Given that they have the same source IP address and port, we merge both events e_i and e_j into on LLC alert. This process is described in the following term:

$$\prod_{Source.Node} e_i \bowtie_{e_i(Source.Node)=e_j(Source.Node)} e_j \quad (4.4)$$

◇

An example for merging events with the same source address is given in Table 4.11. A port scan with source address 85.1.1.8 scans destination addresses 132.8.1.4 and 132.8.1.5. This leads to two events being detected by IDS sensor with the analyzer ID CEDET01IDS, which are merged into a single IDS alert.

Analyzer ID	Event ID	Source	Destination	Time	Description	Tag
CEDET01IDS	4	85.1.1.8	132.8.1.4	2016-01-24 11:02:31.10	Port scan	
CEDET01IDS	5	85.1.1.8	132.8.1.5	2016-01-24 11:02:31.20	Port scan	
CEDET01IDS	{4,5}	85.1.1.8	{132.8.1.4, 132.8.1.5}	{2016-01-24 11:02:31.10, 2016-01-24 11:02:31.20}		SRCMERGE

Table 4.11: Example for merging events with the same source IP-address. It is irrelevant, whether both events are issued by the same IDS sensor, IPS sensor or FW.

Definition 4.7 (Merging Events with same Destination Address). Consider two events e_i and $e_j \in Event = (Analyzer, CreateTime, DetectTime, AnalyzerTime, Source, Target, Classification, Assessment)$. Given that they have the same destination IP address and port, we merge both events e_i and e_j into on LLC alert. This process in the following equation.

$$\prod_{Target.Node} e_i \bowtie_{e_i(Target.Node)=e_j(Target.Node)} e_j \quad (4.5)$$

◇

Example 4.5 (Merging Events with same Destination Address). An example for merging event with the same destination address is given in Table 4.12. As a part of a distributed

denial of service attack, two distinct IP addresses 85.1.1.8 and 85.1.1.9 send HTTP requests to destination address 132.8.1.4. Thus, two events are detected by an IDS sensor with the analyzer ID CEDET01IDS, which are merged into a single IDS alert.

Analyzer ID	Event ID	Source	Destination	Time	Description	Tag
CEDET01IDS	6	85.1.1.8	132.8.1.4	2016-01-24 11:02:31.10	GET /search?p=double HTTP/1.1	
CEDET01IDS	7	85.1.1.9	132.8.1.4	2016-01-24 11:02:31.20	GET /search?p=trouble HTTP/1.1	
CEDET01IDS	{6,7}	[85.1.1.8, 85.1.1.9]	132.8.1.4	[2016-01-24 11:02:31.10, 2016-01-24 11:02:31.20]	[GET /search?p=double HTTP/1.1, GET /search?p=trouble HTTP/1.1]	DSTMERGE

Table 4.12: Example for merging events with the same destination IP-address. It is irrelevant, whether both alerts are issued by the same IDS sensor, IPS sensor or FW.

Event normalization, alert verification and enrichment, and event fusion are preprocessing steps for the operational impact based event correlation introduced in the following section.

4.3 Operational Impact based Event Correlation

In the context of this section, we introduce workflows as a basis for operational impact based event correlation. As investigated above, workflows are automatically derived based on network activities, which are detected by conducting a stream-based network service dependency analysis on network traffic. Workflows span multiple network services

that depend on each other to fulfill a common goal. These network activities constitute hidden states within the automatically derived Hidden Markov Model (HMM) workflow introduced in Chapter 3. In order to automatically derive network activities, we need to identify direct and indirect dependencies between network services as described in Section 2.3.

4.3.1 Network Activities

Indirect dependencies are elementary building blocks for deriving HMM workflows within communication networks. An existing indirect dependency implies that, according to our communication approach, two direct dependencies have a similar communication pattern. Hence, we conclude that all network services within the involved direct dependencies are dependent on each other. Given that two indirect dependencies are joined into a common workflow, a failure or delay of one indirect dependency could have an operational impact on the other one. We consider a single indirect dependency as the smallest possible workflow. By linking events to events to affected workflows, we allow a context-aware event analysis.

Definition 4.8 (Affected network activities). Based on a workflow HMM $\lambda = (X, Y, A, B, \Pi)$ the set of affected network activities NA , which are directly affected by an event is defined as

$$NA = CC((S, \text{map}(\text{asSet}, X))),$$

where CC denotes the connected components (CC) of the hypergraph given as parameter (asSet maps a tuple into a set of components). Figure 4.1 illustrates a possible set of

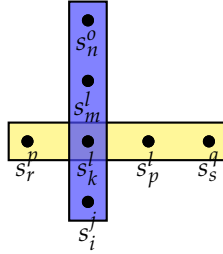


Figure 4.1: An example of a set of network activities NA .

network activities NA . To find the devices TD associated to a network activity, we use $dev : \mathcal{P}(S) \rightarrow \mathcal{P}(D)$. \diamond

4.3.2 Event Prioritizing

Considering events $EVENT$, every event can be linked to a set of network services according to

$$ES : EVENT \rightarrow \mathcal{P}(S),$$

with a mapping function ES , an event $EVENT$ and a network service set S . Based on the identified set of network services, a set of involved devices can be derived with the function dev .

Events that affect the same devices are correlated into a single IDMEF event. Whether two events $e_x, e_y \in EVENT$ are correlated is derived by,

$$\begin{aligned} \forall e_x, e_y \in EVENT : ES(e_x) \cap ES(e_y) \neq \emptyset \\ \implies \{e_x, e_y\} \in IDMEFEV \end{aligned}$$

Correlated network services $CORREV$ potentially operationally impacted by reported events are defined as the

CCs of the graph (S , $IDMEFEV$). In order to support an operator in understanding how a correlated event affects the monitored network, we map events to a set of devices.

$$\begin{aligned} affectedDevices &: EVENT \rightarrow \mathcal{P}(D) \\ affectedDevices(e) &= \bigcup_{\substack{t \in NA, \\ \tilde{e}_{xy} \in CORREV}} \{dev(t) \mid ES(\tilde{e}_{xy}) \cap t \neq \emptyset\} \end{aligned}$$

An operator is then able to see all events and all devices, which may potentially be affected by these events. All network devices are assigned criticality values according to the following equation.

$$CRIT : D \rightarrow \{\text{low, medium, high}\}$$

Based on the criticality map for devices, we are able to prioritize correlated events by defining the following order relation on $CORREV$:

$$\begin{aligned} \leq &= \{(e_x, e_y) \in CORREV \times CORREV \mid \\ &\quad reduce(\max, map(CRIT, affectedDevices(e_x)), low) \\ &\quad \leq reduce(\max, map(CRIT, affectedDevices(e_y)), low)\} \end{aligned}$$

A correlated event is ranked as the highest criticality category of all affected network devices. So, a correlated event is assigned a criticality value and it can be ranked into one of the three criticality categories $\{\text{low, medium, high}\}$. This allows a network operator to prioritize events that might potentially affect more critical network devices.

4.4 Evaluation

The PANOPTESEC testbed is an authentic replication of an Italian water and energy distribution company's corporate

enterprise systems and SCADA system. To test our hypothesis that the previously introduced LLC is sufficient for detecting cyber incidents among all reported Syslog messages, we need Syslog messages with a known ground truth. To derive ground truth data, we set up attack scenarios within the PANOPTESec testbed. Kali Linux and metasploit is used to emulate a cyber attack on the test bed.

4.4.1 Event Verification and Correlation Evaluation

To test the vulnerability-centric parts of LLC as well as the source, destination and multiple event merging parts, we focus on the average event reduction in normal operational mode of the simulation environment.

Performance Metrics

The performance metrics introduced in the following paragraphs are used to evaluate LLC's performance within cyber attacks conducted within the PANOPTESec testbed. To evaluate the performance of LLC, metrics with respect to outgoing LLC alerts $alert_{LLC}$, incoming Syslog messages msg_{Syslog} , Syslog messages that are incorporated by LLC alerts msg_{Syslog}^{LLC} , and Syslog messages that can be attributed to a cyber incident msg_{Syslog}^{attack} are defined. The respective number of occurrences is written as $|alert_{LLC}|$, $|msg_{Syslog}|$, $|msg_{Syslog}^{LLC}|$, and $|msg_{Syslog}^{atk}|$.

Based on these parameters we define the Syslog message reduction rate as:

$$\frac{(msg_{Syslog} - msg_{Syslog}^{LLC})}{msg_{Syslog}} \times 100. \quad (4.6)$$

To prove that the Syslog message reduction only eliminate useless Syslog messages, we investigate LLC's ability to identify true positive incidents $msg_{Syslog}^{atk} \subseteq msg_{Syslog}$ within all incoming Syslog messages msg_{Syslog} . The true positive incident identification rate is described as:

$$\left(1 - \frac{(msg_{Syslog}^{atk} - msg_{Syslog}^{LLC})}{msg_{Syslog}^{atk}}\right) \times 100. \quad (4.7)$$

An additional benefit of LLC is its ability to aggregate Syslog messages. To measure how well LLC aggregates true positive incidents msg_{Syslog}^{atk} into LLC alerts $alert_{LLC}$, we define the following metric to describe the Syslog message aggregation rate:

$$\frac{(msg_{Syslog}^{atk} - alert_{LLC})}{msg_{Syslog}^{atk}} \times 100. \quad (4.8)$$

Given that an attack atk was observed, all incoming Syslog messages that can be attributed to an attack atk , but, however, have not been linked to vulnerability by the sensor reporting the Syslog message are denoted as $msg_{Syslog}^{atkNoCVE}$. LLC alerts that report $msg_{Syslog}^{atkNoCVE}$ are denoted as $msg_{LLC}^{atkNoCVE}$. The set of LLC alerts $msg_{LLC}^{atkNoCVE} \subseteq msg_{Syslog}^{LLC}$. Hence, we consider a particular vulnerability has been enriched by LLC, when the incoming Syslog message leading to the LLC alert was not linked to the vulnerability. The vulnerability enrichment rate is described by the following metric:

$$\left(1 - \frac{(msg_{Syslog}^{atkNoCVE} - msg_{LLC}^{atkNoCVE})}{msg_{Syslog}^{atkNoCVE}}\right) \times 100. \quad (4.9)$$

Attack simulation with little noise present within the test environment

The attack scenario was executed within the emulation environment. For a deeper analysis in addition to the online tests, all incoming Syslog messages during each experiment were recorded. A description of the cyber attack was given and provides a known ground truth which allows a detailed evaluation of the produced LLC Alerts. In the following experiments we describe the number of incoming Syslog alerts and the resulting LLC alerts. The attack scenario assumes as a first step that an attacker with the IP address 172.16.10.20 exploits a vulnerability CVE-2006-3439 to gain control of 172.16.10.10 (dorete). In a second step, after gaining control of dorete, he is able to use the other IP address of dorete 192.18.200.230 to exploit CVE-2004-2687 on 192.18.200.230 (ARCHIVESRV). In a third step, the SCADA server 192.18.200.230 is used to exploit CVE-2004-2687 on 192.18.200.146 (xferp2).

Afterwards, the attacker uses xferp2 as a starting point to gain control of 192.18.200.181 (mferp1) by exploiting CVE-2004-2687. All exploited vulnerabilities have been detected by vulnerability scanners within the emulation environment and are listed within the Network Inventory used by LLC.

The attack scenario was executed twice, once with little additional “noise” present and once in the presence of additional noise. Noise refers to cyber activities, which lead to additional Syslog messages. Attackers often create noise in order to mask their cyber attacks. The attack scenario with little additional noise is described in Table 4.13 and the attack scenario with added noise is described in Table 4.14.

Reported incidents SOURCE → DESTINATION	Syslog messages corresponding to the potential attack step	LLC alert
Step 1: 172.16.10.20 → 172.16.10.10 attacker → dorete	13 (1 with vulnerability)	1 VULNVERIFIED 1 NOTVERIFIED 2 DUBLIMERGE (9)
Step 2: 192.18.200.230 → 192.18.200.200 dorete → ARCHIVESRV	1 (1 with vulnerability)	1 VULNVERIFIED
Step 3: 192.18.200.200 → 192.18.200.146 ARCHIVESRV → xferp2	1 (1 with vulnerability)	1 VULNVERIFIED
Step 4: 192.18.200.146 → 192.18.200.181 xferp2 → mferp1	1 (1 with vulnerability)	1 VULNVERIFIED
Others:		
192.18.200.2 → 192.18.200.1 xuel2 → xuel1	5	2 DUBLIMERGE (4)
TOTAL	4749	9

Table 4.13: Reported incidents, Syslog messages and LLC alerts for the attack scenario with little noise present.

Overall within an attack scenario with little noise present, as described in Table 4.13, 4797 Syslog messages were received and 9 LLC alerts were produced. This results in an overall Syslog message reduction rate of 99.8%. The attack scenario described in Table 4.13 starts at 15:38:25 and lasts until 15:45:49. The previously described attack scenario results in 16 Syslog messages and LLC passes on 14 Syslog messages with 7 LLC alerts. In addition, 4 Syslog messages, which cannot directly be attributed as belonging to the previously described attack scenario, are passed on by LLC with 2 LLC alerts. In the following we will carefully analyze all reported cyber incidents and the corresponding LLC alerts.

Within the first step of the executed attack scenario, 13

Syslog messages are issued overall. Three of these Syslog messages are linked to vulnerabilities by the sensor reporting the cyber incident; the other Syslog messages are not linked to vulnerabilities. LLC reports 11 of the 13 overall issued Syslog messages. One reported vulnerability can be verified as present on dorete and results in an LLC alert with the tag verified vulnerability (VULNVERIFIED). Another reported vulnerability cannot be verified and results in an LLC alert with the tag NOTVERIFIED. Within every one-second-time window of LLC, every reported incident between the same source and destination pair is reported once as a verified (VULNVERIFIED) or non-verified (NON-VERIFIED) LLC alert. Additional Syslog messages, reporting the same incident are aggregated into LLC alerts with the tag DUBLIMERGE. In the attack scenario with no additional noise, this leads to two LLC alerts with the tag DUBLIMERGE, which aggregates 7 Syslog messages in total. Executing the second step results in one Syslog message, which is linked to vulnerability CVE-2004-2687 and results in one LLC alert with tag VULNVERIFIED.

The third attack step is linked to vulnerability CVE-2004-2687 and also is reported by LLC as verified vulnerability. An additionally executed attack uses xferp2 as a starting point and targets mferp1. This leads to one Syslog message containing a link to CVE-2004-2687 and this is reported as an LLC alert with the tag VULNVERIFIED. The executed attack leads to a reported incident between xuel2 and xuel1. A closer analysis of these Syslog messages reveal tags such as "Analyzer_Log-Flood-Protection", "Analyzer_Compress-SIDs", "Firewall". These cyber incidents are all reported by sensor CEDETClusterIPS node 1. These reported incidents result in 2 aggregated LLC alerts with the tag DUBLIMERGE.

Attack simulation with additional noise present within the test environment

The previously described attack scenario was executed a second time with additional noise. As shown in Table 4.14, overall 28477 Syslog messages were received during the execution of the attack scenario with additional noise. Reducing overall 28477 Syslog messages that were received to 129 LLC alerts results in a false positive reduction rate of 99.5% according to Equation 4.6. The second attack scenario execution starts at 15:55:57 and lasts until 16:28:41. The previously described attack scenario is reported within 1,884 Syslog messages and LLC alerts report 1396 of these Syslog messages by issuing 94 LLC alerts.

Additionally, 12 LLC alerts are issued for 24 Syslog messages. In the following, we will analyze these results systematically by starting with attack scenario step 1. The first step of the attack scenario is reported within 1615 Syslog messages, whereas 816 of the overall 1615 Syslog messages are linked to vulnerabilities. LLC reports 1,126 of these Syslog messages within 73 LLC alerts.

The second step of the attack scenario is reported within 9 Syslog messages, where one reported incident is linked to a vulnerability. Due to these Syslog messages being linked to a vulnerability, LLC identifies all these Syslog messages as relevant and passes on 5 LLC alerts.

The third step is reported by 8 Syslog messages with one Syslog message being linked to a vulnerability. LLC identifies 7 of these Syslog messages as relevant and reports them within 3 LLC alerts.

The additional attack using xferp2 as a starting point for targeting mferp1 leads to 252 Syslog messages. LLC

Reported incidents SOURCE → DESTINATION	Syslog messages corresponding to the potential attack step	LLC alert
Step 1: 172.16.10.20 → 172.16.10.10 attacker → dorete	1615 816 with vulnerability	1 VULNVERIFIED 13 NOTVERIFIED 33 DUBLIMERGE (1086)
Step 2: 192.18.200.230 → 192.18.200.200 dorete → ARCHIVESRV	9 1 with vulnerability	1 VULNVERIFIED 2 NOTVERIFIED 2 DUBLIMERGE (6)
Step 3: 192.18.200.200 → 192.18.200.146 ARCHIVESRV → xferp2	8 1 with vulnerability	1 VULNVERIFIED 2 NOTVERIFIED 2 DUBLIMERGE (6)
Step 4: 192.18.200.146 → 192.18.200.181 xferp2 → mferp1	252 19 with vulnerability	1 VULNVERIFIED 18 NOTVERIFIED 12 DUBLIMERGE (233)
Others:		
192.18.200.2 → 192.18.200.1 xuel2 → xuel1	15	6 DUBLIMERGE (14)
192.18.200.200 → 192.18.200.230 ARCHIVESRV → dorete	2 1 with vulnerability	2 NOTVERIFIED
192.168.1.4 → 172.21.170.5, 172.21.170.6	28 28	4 4 SRCMERGE (8)
mferp2 → TTY-T5, TTY-T6		
TOTAL	28477	129

Table 4.14: Reported incidents, Syslog messages and LLC alerts for the attack scenario with noise present.

considers 244 Syslog messages as relevant and reports them within 13 LLC alerts.

Starting at 2016-07-25 15:59:38 until 16:28:38, Syslog message report a cyber incident with source mferp2, which is a front-end scada server and destination 172.21.170.5. One second after this cyber incident, a cyber incident with source mferp2 and destination 172.21.170.6 is reported. To our knowledge, these cyber incidents are not related to the executed attack scenario. As already explained within the previous attack scenario with little additional noise, we also

observed cyber incidents between xuel2 and xuel1. The 15 reported cyber incidents between xuel2 and xuel1 have the same message content as in previous attack scenario execution and result in 5 LLC Alerts containing 14 merged Syslog messages. Two cyber incidents report 192.18.200.200 (ARCHIVESRV) targeting 192.18.200.230 (dorete). One of the Syslog messages contains a link to a vulnerability, which LLC cannot verify as present within the emulation environment. Due to these Syslog messages being linked to a vulnerability, LLC identifies this Syslog message as relevant and passes on 1 LLC alert with the tag NOTVERIFIED.

4.4.2 Benchmarking Syslog Message Processing Time

As a number of low-level Syslog messages are being sent from sensors at a high pace, fairly sophisticated knowledge of both networking technology and infiltration techniques is required to understand them. However, human operators are far too slow for dealing directly with the numbers of alerts of a real-life system⁷. IDS alert correlation systems try to solve this problem by post-processing the event stream from one or many IDS sensors. The goal of post-processing is to aggregate low-level events and enrich them with vulnerability information as explained above. Assuming a time-based system, the processing time in seconds depends on the number of simultaneously processed alerts. If the alert processing time exceeds one second, the LLC would not be able to hold that pace, should it continuously occur over multiple time windows.

⁷FireEye. *The SIEM Who Cried Wolf: Focusing Your Cybersecurity Efforts on the Alerts that Matter*. <https://www2.fireeye.com/rs/fireeye/images/fireeye-alerts-that-matter.pdf>. 2014.

Benchmarks based on synthetic events

To test the limitations of LLC as a single component, additional tests with a stand-alone version of LLC were conducted. To carry out additional tests, Syslog messages from the emulation environment were replayed continuously at varying message per second rates. Based on this, for the LLC as a standalone component, processing up to 10,000 alerts per second on average is possible. It cannot be guaranteed that a continuous event rate of 100,000 events per second can be processed by LLC. We come to this conclusion due to the scalability experiment illustrated in Figure 4.2.

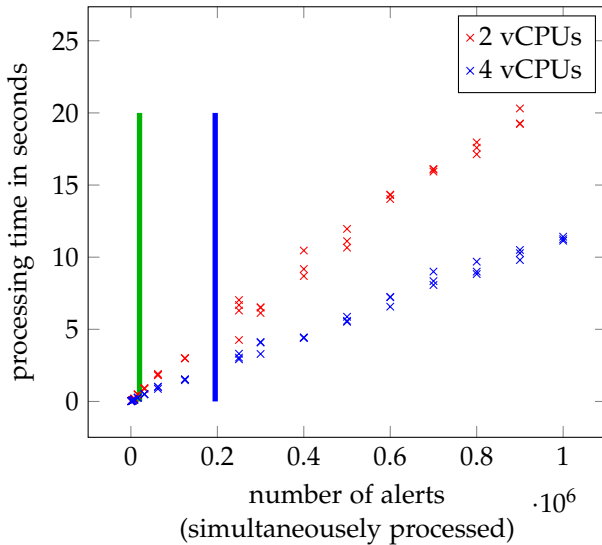


Figure 4.2: Alert processing times.

Figure 4.2 shows the processing time in seconds with respect to the number of alerts (Syslog messages) that are

simultaneously processed. To demonstrate the parallelizability of LLC, the LLC was deployed in a virtual machine with 2 virtual CPUs and 4 CPUs. The results show a linear increase of computation time for an increasing number of alerts (Syslog messages). A continuous Syslog message rate can be held by LLC, if the alert processing time is below one second.

Within the emulation environment, the LLC was tested with a continuous average Syslog message rate of 1500-2000 Syslog messages per second. LLC is able to hold that Syslog rate easily. Hence, this rate is illustrated as a green line in Figure 4.2. The hypothetical upper limit of the production environment is 200,000 Syslog messages per second. This line was added as a blue line to Figure 4.2. We conclude that LLC is parallelizable and, thereby, adding more cores will reduce LLC's alert processing time.

4.4.3 Real-life Case Study

The LLC was also test within two different operational environments. The first operational environment is based on the disaster recovery site of an energy distribution network, provided an Italian water and energy distribution company. The disaster recovery site is used to emulate the production environment of an energy distribution company. During the time of integration, 2,000 Syslog messages were observed within the emulation environment and LLC was able to easily handle this message rate. LLC has been integrated for multiple months and remains able to sustain the continuous Syslog message stream. Figure 4.3 shows a time frame of 65 seconds and displays the range of incoming Syslog messages and outgoing LLC alerts within the emulation environment.

In addition to the operational environment test within the

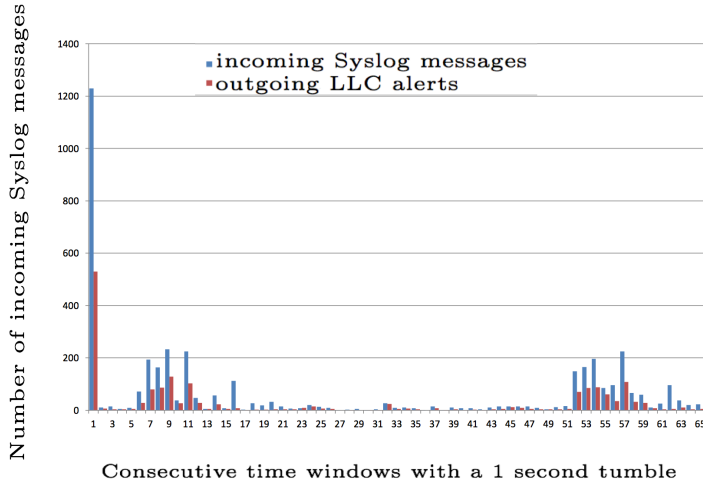


Figure 4.3: A time range showing incoming Syslog messages and outgoing LLC alerts within the emulation environment.

emulation environment based on the disaster recovery site of an Italian energy distribution company, LLC was also deployed within the production environment of the Italian energy distribution company. As this a real-life production environment, we are not allowed to reveal detailed information about the energy distribution network. The production environment contains around 1015 medium voltage substations.

The production environment test was conducted on 19th July 2016 and started at 10:22:40 and ended on 17:41:51. LLC was able successfully deployed and overall 6,769,533 Syslog messages were received. Due to the criticality of the production environment, IPS sensors and FWs are used to block

unexpected attempts of communication. This procedure is referred to as white listing. Therefore, as is to be expected no LLC alerts were produced. This real-life case study showed us that LLC is able to perform within an operational environment and is able to reduce the overall number of reported events.

4.5 Discussion

A FireEye report⁸ based on customer information estimated that most security centers receive up to 500,000 security events per day. Thus, security analysts monitoring data-communication networks experience event fatigue and become desensitized to security events. Event correlation aims to aid security analysts by reducing the overall number of security events.

In this chapter we propose an event prioritization and correlation approach that is able to use automatically discovered workflows to prioritize incoming events based on this information after conducting simple event correlation. This allows a network operator to see what ongoing network activities in the network are potentially affected by occurring events. In critical situations, decision makers need as much information situationally relevant as possible to make an informed choice on how to protect a network. The focus of the approach introduced in the context of this work is to monitor how events affect ongoing workflows in the monitored network. The reasoning behind this approach is that events affecting business-critical workflows have more

⁸FireEye. *The SIEM Who Cried Wolf: Focusing Your Cybersecurity Efforts on the Alerts that Matter*. <https://www2.fireeye.com/rs/fireeye/images/fireeye-alerts-that-matter.pdf>. 2014.

impact in the eyes of a network operator than others.

Due to little information being available about threat agents, we argue that it is very difficult to conduct a similar evaluation with attacker-model based approaches. Knowledge-based approaches [CM02; Mor+02; NCR02] rely on human input and the results directly depend on the quality of the human input.

Due to the growing complexity of computer networks, the quality of the human input is questionable. In contrast to knowledge-based approaches we focus on minimizing the required human input, and, thus, argue that a novel approach to context-aware event correlation and prioritization was introduced.

Related Work

Many approaches for protecting data-communication networks against cyber threats exist. For example mission impact modeling aims to supporting cyber security decision-making processes by analyzing the impacts of cyber attacks. Most mission impact modeling approaches [Jak11; Jak13; Mus+11; BCY12; KC13] use impact dependency graphs to assess whether a cyber attack had an impact on ongoing missions. The concept of missions is sometimes also referred to as mission-centricity in cyber security. Ongoing missions are dependent on cyber assets, which interact for a higher purpose. Network dependency analysis investigates how to improve understanding how cyber assets, such as network devices and network services interact for a higher purpose. Similarly, the research domain of mission impact modeling aims to identify and model the higher purpose of a network which is referred to as mission. Mission impact modeling relies on a detailed understanding on how ongoing missions depend on cyber assets, which is also referred to as workflow. Most of the time workflow models within mission impact modeling are acquired by relying on human input. Unfortunately, in data-communication networks, workflows are generally not documented [Van+03].

Mission impact modeling can also be used to assess the impact of cyber events occurring in a monitored data-communication network. However, as previously mentioned, a FireEye report¹ estimated based on information gathered from customers that most security centers receive up to 500,000 cyber events daily. This leads to network operators experiencing event fatigue and becoming desensitized to security events. To counter this issue, event correlation aims to reduce the overall number of security events. Therefore, in the context of this work we proposed to couple mission impact modeling with event correlation.

The purpose of this chapter is to provide an overview over the current state of the art in all of these related knowledge domains. So, in Section 5.1 related work in the domain of mission impact modeling is listed. In section 5.2 an overview over different approaches to event correlation is provided. Section 5.3 describes network dependency discovery methods, Section 5.4 introduces workflow mining approaches.

5.1 Mission Impact Modeling

Mission impact modeling addresses the fact that within data-communication networks, workflows rely on communication and information systems to fulfill a higher task. This higher task is referred to as a mission. There are different approaches to mission impact modeling. For example, some approaches, which are listed in Section 5.1.1, focus on modeling intent, motivation and capabilities of cyber attackers in order to understand a cyber attacker's intended and ac-

¹FireEye. *The SIEM Who Cried Wolf: Focusing Your Cybersecurity Efforts on the Alerts that Matter*. <https://www2.fireeye.com/rs/fireeye/images/fireeye-alerts-that-matter.pdf>. 2014.

tual mission impact. Other approaches to mission impact modeling, which are described in Section 5.1.2, focus on assessing the impact that observed cyber events have on a monitored infrastructure. Section 5.1.3 introduces mission impact modeling approaches, which concentrate on how to model an infrastructure.

5.1.1 Cyber Attack and Defense Modeling

An example for a mission impact model methodology [MC16] uses stochastic Petri nets to capture the interaction between cyber attacker and defenders within a electrical power grid. In this model, attackers are distinguished according to their intentions as surveillant and destructive attackers and three types of failure within an electrical power grid are modeled:

- control nodes not being able to accomplish their intended mission,
- a high density of compromised control nodes, and
- an attacker being able to extract critical information.

Similar to attack-trees [Sch99], mission impact modeling, with the focus of modeling cyber attack and defense, measures the effect that a cyber attack has on a monitored infrastructure. However, mission impact modeling is explicitly different from attack-trees [Sch99] as attack-trees typically are not able to handle new attacks (e.g., zero-day attacks). Given that no signature is available for an attack, there will be no attack-tree available for this attack. In addition, mission impact modeling focuses on determining the impact of an attack on ongoing workflows within a data-communication network. Thus, it comes as no surprise that attack graph-based approaches have been extended to allow a mission-

centric approach [Jaj+11] to cyber defense. Cauldron [Jaj+11] relies on an attack graph to understand the vulnerabilities that an attacker is able to exploit and uses a mission impact model to assess the impact that an attack step has on a monitored system.

For mission impact analysis, these approaches [Jaj+11; MC16] assume that the mission workflows are given. A mission workflow describes which task requires which cyber asset to complete a mission, and a value is assigned to each task in order to quantify how important it is for the overall mission. The workflow-mining method introduced in the context of this work advances the state of the by automatically deriving workflows by analyzing network traffic and, thus, it is not necessary to acquire workflow models by hand. As previously stated, acquiring workflows by hand is a time consuming and expensive process. In addition it is very difficult to assess the accuracy of an acquired mission workflow model.

5.1.2 Mission-based Event Correlation and Prioritization

Mission impact analysis has also been investigated as a tool for prioritizing events. For example, A Configurable Cyber Event Prioritization Tool (ACCEPT) [Kim+14] prioritizes an event based on the effect that an event has on the targeted host. Event correlation within ACCEPT is limited to a rule-based complex event processing engine², which allows network operators to add their own event correlation rules based on their domain knowledge. Similar to the approaches listed in Section 5.1.1, ACCEPT relies on information such as asset criticality and network connectivity information being

²Inc. Red Hat. *Drools*. <http://www.drools.org/>. 2016.

provided and imperfect knowledge is not considered. Yet, the event prioritization process depends on the availability and accuracy of this information.

Porras et al. [PFV02] introduce Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) and a Mission-based Correlation System (M-Correlator). EMERALD M-Correlator is a mission impact based approach to event correlation and prioritization of distributed heterogeneous cyber security sensors and ranks a continuous stream of event based on a given network connectivity model and given operational objectives. The network connectivity model is derived based on Nmap³, and operational objectives are derived based on a mission specification, which is provided by network operators. This mission specification lists network devices and network services, which are the most critical, and also lists event types which are considered most critical by network operators. Similarly, event correlation is provided based on a predefined clustering policy. Thus, event correlation and prioritization is based on information provided by network operators, and similarly to ACCEPT, the event correlation and prioritization process depends on the availability of this information.

In the context of this work, we introduced LLC as a novel event correlation and prioritization method that combines event correlation with operational impact based event prioritization. Operational impact assessment requires knowledge of underlying workflows within a monitored network. In contrast to previously mentioned approaches, we focus on limiting the information provided by network operators as much as possible by relying on data mining techniques to derive our mission model. To our knowledge, the event cor-

³Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

relation and prioritization method introduced in the context of this work is the first approach to operational-impact based event correlation and prioritization with an automatically derived workflow model.

5.1.3 Mission modeling

In the context of this work, we investigate the potential operational impact of reported security incidents on a business or mission. Mission impact modelling research focuses on documenting the interdependency relationship between cyber assets and mission workflows so that this information can be used operationally. We refer to these mission impact modeling approaches as focusing on mission modeling. Thus, in the following section we will investigate the state of the art of mission modeling research. Multiple distinct approaches to mission modeling approaches have been proposed [BCY12; GDK09; Jak11; Jak13; Mus+11].

Barreto et al. introduce an impact assessment methodology [BCY12] incorporating vulnerability descriptions⁴ and information acquired through a BPMN model via human input. We understand a mission as network activities with a common purpose, as illustrated in Figure 2.1. So our understanding of what constitutes a mission corresponds to Barreto's [BCY12]. Similarly to the approach introduced in the context of this work, Barreto et al. investigates the operational impact of reported security incidents on a monitored network. However, in contrast to the approach introduced in the context of this work, Barreto et al. rely on a mission model acquired through human input.

Camus (Cyber Assets to Missions and Users) [GDK09]

⁴MITRE. *Common Vulnerabilities and Exposures*. <https://cve.mitre.org/>. 2000.

introduces a mission model mapping cyber assets and users to cyber capabilities that support missions. The relationship between cyber assets, users and cyber capabilities is referred to as core ontology for integrating available data into a common model. This core ontology is integrated into a Snort⁵ web interface in order to highlight what cyber assets are targeted by reported cyber security events. Similarly to Barreto et al., Camus requires human input to derive the mission model opposed to the workflow model introduced in the context of this work, which acquired by mining network traffic. Mining mission information based on possible raw data sources such as File Transfer Protocol (FTP)⁶ logs and an Lightweight Directory Access Protocol (LDAP)⁷ is mentioned, however not introduced or evaluated. To point out possibilities for acquiring parts of an infrastructure's mission model from other data sources, three other methods of acquiring this information are described [GDK09]:

- direct translation from raw data sources,
- inferred translation through heuristics or statistical methods, which are undisclosed to the public, and
- from other mission models.

Albanese et al. [Alb+13] present a mission model for minimizing a mission's exposure to vulnerabilities. This approach minimizes a mission's exposure to vulnerabilities by taking available information about vulnerabilities and dependencies into account. To fulfill this task, an interdependency

⁵Martin Roesch. "Snort - Lightweight Intrusion Detection for Networks." In: *Stanford Telecommunications*. 1999, pp. 229–238.

⁶J. Postel and J. Reynolds. *File Transfer Protocol*. RFC 959 (Standard). Updated by RFCs 2228, 2640, 2773, 3659, 5797. Internet Engineering Task Force, Oct. 1985.

⁷Brian Arkills. *LDAP Directories Explained: An Introduction and Analysis*. ISBN: 9780201787924. Addison-Wesley Professional, 2003.

model is described which links cyber assets to mission tasks. The interdependency model is also acquired via human input. A mission $M = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_m\}$ is described as a set of tasks τ_i . Every mission is assigned a criticality value $c : \mathcal{P}(M) \rightarrow \mathbb{R}$ and is allocated to a physical host h_i , which is represented by a vector of resources (e.g., CPU, memory) and vulnerability score. A mission is considered as fulfilled, if all tasks are correctly executed. A mission's exposure to vulnerability is deduced through a cost minimization algorithm. Similarly to previous approaches, this mission model is based on information acquired via human input.

Another mission-centric approach based on human input is introduced by Jakobson [Jak11; Jak13], who presents an interdependency model representing an infrastructure's operational capacity. Within this model, cyber attacks are represented as extended versions of conceptual graphs [Sow99] such that a hardware platform hosts a cyber asset. Cyber assets have vulnerabilities, which are exploited by cyber attacks. Every cyber attack has an impact factor describing to what degree the attack is capable of compromising the attacked cyber asset. Given that a cyber attack on a cyber asset is executed, it affects the operational capacity of the cyber asset. Thus, the cyber attack model in the context of this work assumes that, given that a cyber attack is able to exploit a vulnerability, the cyber attack will definitely do that. The operational capacity of a cyber asset and the impact factor of the cyber attack are linked. An impact dependency graph then describes how cyber assets are linked to mission tasks. Unfortunately, the impact dependency graph is acquired via human input.

Musman et al. [Mus+11] introduce a framework called Computing the Impact of Cyber Attacks on Complex Missions (CMIA). CMIA evaluates mission impact by under-

standing which cyber assets are important to accomplish mission goals. All these models do not focus on how to acquire the information used to derive interdependency models or verifying the accuracy of the acquired information. By relying on time series data mining to require our workflow model, we are able to automatically acquire the information used for operational impact assessment. Additionally, a thorough evaluation verified the ability of the introduced workflow mining methodology to acquire accurate workflow models.

5.2 General Event Correlation and Prioritization Approaches

Aside from mission impact modeling approaches, which were listed in Section 5.1.2, of course there are other approaches to event correlation and prioritization. Multiple approaches correlate intrusion events and extract attack scenarios to predict the next attacker action [VS01; FAK15; GG15; CM02; Pen+08]. In the following, we will investigate approaches to event correlation and prioritization.

Several researchers focus on how to reduce the amount of events as well as decreasing the false positive rate [Mor+02; Pie04; Smi+08]. Other researchers cluster similar events [CM02; Pen+08] in order to discover high-level attack scenarios or represent and reason with operator preferences regarding the events they analyze [Tab+11]. Knowledge-based approaches [CM02; Mor+02; NCR02] rely on human input and the results directly depend on the quality of the human input.

With the complexity of computer networks growing fast, it becomes increasingly difficult for network operator's to

ensure they have a full understanding of their monitored network. Therefore, the quality of the human input is questionable. In contradiction to knowledge-based approaches we focus on minimizing the required human input.

Other researchers focus on solving the problem of aggregating events into multi-step attacks as a data mining problem [FAK15; GG15]. Unfortunately, most of the presented approaches implicitly rely on a threat agent model [VS01; FAK15; GG15; CM02; Pen+08]. Valdes et al. [VS01] use statistics-based methodologies to develop event correlation metrics which establish a lower level of correlation before discovering attack step correlations. Statistics-based event correlation methods depend on underlying attribute distributions (such as Gaussian distribution) of deviations from what is expected. Farhadi et al. [FAK15] studied how to correlate event patterns using Hidden Markov Models and focused on machine learning based attack plan recognition to correlate and predict events. The issue with all event correlation methods that rely on a threat agent model is the following: Attackers have different intentions and skill levels and, thereby, predicting their next possible action and validating the proposed adversary model is a complex task. Evaluating the quality of a threat model is difficult as little information is available about different threat agents. All previously introduced methodologies do not address the issue of evaluating the accuracy of their threat model. By comparison, the workflow model introduced in the context of this work, which provides the foundation for event correlation and prioritization, is evaluated with respect the workflows model's accuracy.

5.3 Network Dependency Discovery

Network dependency discovery approaches can be divided into two different categories: active and passive network dependency discovery. On the one hand active network dependency discovery intrusively manipulates network traffic to find out network dependencies, on the other hand passive network dependency discovery does not actively perturb system components to identify network dependencies. Passive approaches utilize heuristic algorithms to analyze communication patterns in network traffic in order to deduce network dependencies. However, passive approaches [Nat+12; Bah+07; Che+08; Bah+06] have a high false positive rate. Active approaches manipulate timing or content of network traffic to identify dependencies and, thereby, generally have a lower rate of false positives compared to passive network dependency methods.

The following sections are organized as depicted in the following: Section 5.3.1 presents active network dependency discovery approaches and Section 5.3.2 describes passive network dependency discovery approaches.

5.3.1 Active Network Dependency Discovery

Active Dependency Discovery (ADD) [BKK01] is an active approach, which models network dependencies as a directed, acyclic graph. Nodes in this dependency graph represent system components, which are colloquially introduced as representing operating systems, network services, applications, hardware and networks. ADD performs load injection to perturb different components within a system to observe whether workload on one component has an effect on another component. ADD was later expanded [BKH01] to

address correlated faults in multiple system components. To deploy this methodology, implementation details of the applications need to be known.

Rippler [Zan+14] is an active network dependency discovery method that injects temporal perturbations to determine whether or not this communication pattern propagates to other network services. Temporal perturbations are injected by manipulating communication patterns within data-communication networks.

Unfortunately, a high level of access to a cyber asset is required by active approaches [BKK01; BKH01; Zan+14]. This precondition cannot be met within all data-communication networks. Intrusive network dependency discovery approaches are referred to as active network service dependency approaches. Similarly, non-intrusive network dependency approaches are called passive network service dependency approaches. Unfortunately, in some data-communication infrastructures, such as critical infrastructures, manipulating network traffic is not possible for security reasons. Hence, we were not able to use active network dependency discovery approaches in the context of this work.

5.3.2 Passive Network Dependency Discovery

Passive approaches focus on non-intrusive network service dependency discovery and treat each host as a black box and passively analyze communication patterns between hosts. In the context of this work, we introduce passive network dependency discovery methods Sherlock, Orion, and NSD-Miner in a comparative evaluation with MONA. To ensure completeness, we will list other passive network dependency discovery methods in the following paragraphs.

Two distinct approaches for learning a network dependency structure called “Leslie Graphs”, which are described as a powerful abstraction describing the complex dependencies between network, host and application components are “Constellation” and “Analysis of Network Dependencies” (AND) [Bah+06]. Leslie Graphs are supposed to be a simple abstraction for representing complex network dependencies in order to support network administrators which plan to upgrade or reorganize existing applications. Knowledge of complex dependencies helps in the process of identifying network services and hosts, which may potentially be affected and prevent unexpected consequences. Leslie Graphs capture interdependencies at different levels of granularity ranging from the network layer to the transport layer. Constellation is a distributed system, which reactively constructs a Leslie Graph for any node on-demand. AND maintains an approximate Leslie Graph at a centralized inference engine.

Kind et al. [KGE06] introduce a NetFlow⁸-based passive approach that aims to identify network dependency as well as link cyber assets to business process to enable business-driven IT management. NetFlow is a networking protocol designed by Cisco Systems for logging and recording the flow of traffic received and sent within a network. To derive network dependencies flow pairs, a flow correlation function is built. While the introduced methodology is deployed in an enterprise environment, unfortunately, false positive or false negative rates are not evaluated or discussed.

Dechouniotis et al. [Dec+07] developed another NetFlow-based passive network dependency detection approach. Within the introduced approach, a fuzzy inference engine classifies detected network dependency as high confidence

⁸B Claise. “RFC 3954: Cisco systems NetFlow services export version 9.” In: *Internet Engineering Task Force*. 2004.

and low confidence dependencies.

All previously mentioned approaches are based on analyzing communication patterns and are non-intrusive network dependency discovery methods. As Netflow information was not available within our operational environment, we were unable to deploy the passive approaches introduced by Kind et al. [KGE06] and Dechouniotis et al. [Dec+07]. Orion [Che+08], NSDMiner [Nat+12], and Sherlock [Bah+07] are used in the systematic evaluation of MONA in Section 2.4.2.

5.4 Workflow Mining

Network dependency analysis and workflow mining are adjacent knowledge domains as illustrated in Figure 1.1. Discovering workflows relies on heuristics to mine structural descriptions of ongoing network activities with a monitored infrastructure. This problem is similar to network dependency analysis, which becomes even more apparent when looking into passive network dependency approaches for business-driven IT management [KGE06].

Workflow mining is not new [Her00; Her04; VWM04; Van04; VV04; Van+11]. Herbst [Her00; Her04] introduces a learning algorithm that is able to induce concurrent workflow models. The generated workflow model is described in the ADONIS⁹, [Jun+00] modeling language. The purpose of this workflow mining algorithm is to improve workflow management systems by reducing the acquisition time for workflow models and providing a higher quality of workflow models with less errors and support for the detection of changing requirements.

⁹BOC GmbH. "ADONIS Version 3.81 - User Manual." In: 2005.

Cook et al. [CW98] present three distinct approaches to process discovery for sequential workflow sequences: one discovery approach is based on neural network, one finite state machine based approach, and a third one is a Markovian approach using a mixture of algorithmic and statistical methods. Cook et al. consider the algorithmic and Markovian approach the most promising.

Apart from workflow modeling languages such as ADONIS, workflow models can be formally described by Petri nets or HMMs. Thus, in Section 5.4.1 Petri net-based workflow models and in Section 5.4.2 HMM-based workflow models are described. Currently, workflow mining is dependent on event logs, and research in this domain can be divided into three topics: discovery, conformance and enhancement of workflows [Van11]. Thus, we distinguish these three topics within subsequent sections.

5.4.1 Petri net-based Workflow Models

The benefits of modeling a workflow based on Petri nets are that Petri nets can easily be translated into workflow formats, which are easily readable for humans. Another benefit of Petri nets is their similarity to workflow models established in business science. As we focus on workflow discovery in the context of this work, we list Petri net-based workflow discovery techniques in the following. Many workflow mining methods [VWM04; Van04; VV04; Van+11; Mar+02a; Mar+02b; AS12; ASM13; Fen+13; Zen+13] rely on Petri nets to represent workflows.

A very popular workflow mining technique is the α algorithm [Van04; VWM04], which searches for ordering relations in event logs and models workflows based on Petri nets. For the α algorithm, it is proven that for certain subclasses it

is possible to find the right workflow model.

Beyond the α algorithm, workflow discovery has evolved into different knowledge domains such as healthcare. Maruster et al. [Mar+02a; Mar+02b] focus on higher level workflows, such as patient information flow in a medical use case to learn Petri net like models. The focus lies on learning and modeling concurrent behavior. Based on a synthetic data set with 500 event traces, which consist of events like surgery or blood test, an evaluation was conducted. Rebuge et al. [RF12] introduce another Petri net-based workflow model, which is derived through event data analysis of events recorder in healthcare.

Another application domain of workflow discovery provides a basis for security audits [AS12; ASM13; Fen+13]. Workflow discovery provides a structure, which allows an analysis to take place and look into compliance with security and privacy policies, as check for security breaches and vulnerabilities. For security audits, workflow discovery can answer multiple questions, for example workflow mining can provide

- further information on dynamic system behavior such as recovery time rate of degeneration,
- assesses about whether the actual workflow model corresponds with the intended concepts or
- checks whether observed system behavior meets requirements of the respective compliance standard.

Zeng et al. [Zen+13] introduce cross-organizational process mining based on distributed running log collected from different servers located in different organizations to learn a complex workflow that spans multiple organizations. Interest in learning workflows that span multiple organizations

stems from the following observation: The final product or service of most organizations is an outcome of activities based on a wider inter-organization value chain. Coordination workflow patterns spanning multiple organizations are represented by Petri nets. Zeng et al. assume that logs contain no noise, although real-world application scenarios obviously will contain noise.

Previously introduced Petri net-based workflow models rely on event logs for example produced by enterprise resource planning software. Unfortunately, enterprise resource planning software is not available within all data-communication networks. Thus, we introduce a methodology for deriving workflow models based on network traffic, which is produced within all data-communication networks.

5.4.2 HMM-based Workflow Models

As listed in Section 5.4.1, a lot of workflow mining methods rely on Petri nets due to their similarities to workflow models established in business science. HMMs have been successfully applied to other problem domains, such as gesture recognition [WB99] or even complex activity recognition problems [VRC05]. HMMs are used for a wide range of applications and efficient techniques exist for modeling a HMM and estimating the likelihood that observed data was produced by this HMM.

Naseri et al. [NL13] introduce a HMM-based workflow trust model, which helps to determine a workflow's rate of reliability. The Viterby algorithm [For73] is used to verify, whether the most likely sequence of underlying hidden states might have generated a sequence of observed events. Thereby, the validity of the underlying HMM-based workflow model and the validity of assessed probabilities is

verified.

Silva et al. [SZS05] describe a workflow by an and/or graph, which is a directed acyclic graph. A workflow model is represented as a specialized HMM and Silva et al. introduce a polynomial time algorithm for process discovery. Silva et al. point out that workflow models can obviously be represented by readily available probabilistic structures such as dynamic Bayesian networks or stochastic Petri nets. Especially, factorial HMMs [GJ97] seem like an obvious choice for representing chains of business processes, which are executed in parallel.

Blum et al. [Blu+08] build an HMM of surgical workflows in order to provide a human-understandable workflow model. The issue of event logs being not complete, which result in partially labeled surgical phases, is also addressed. The main motivation of this work is to intuitively visualize surgical workflow by representing the HMM workflow as a graph.

Sequence clustering [Fer+07] deduces a sequence of tasks and groups similar sequences into a common cluster. These sequences of tasks are modeled as a first-order Markov chain and, due to their probabilistic nature, this makes the technique more robust to noise. Trace clustering is another technique, which extracts features from traces and clusters them based on these extracted features [SGV08].

Unlike Petri nets, HMMs are able to model properties such as the transition probability between workflow events. However, Petri nets can be efficiently mapped to HMMs [RVV08; PM16].

Not all workflow mining techniques rely on Petri nets. For example, inductive workflow acquisition [HK98] derives a HMM by analyzing event logs and instance graphs [VV04] by aggregating multiple graphs to model a workflow.

All introduced workflow mining methods rely on event logs for example produced by enterprise resource planning software, which is not available within all data-communication networks. Thus, we introduce a methodology for deriving workflow models based on network traffic, which is produced within all data-communication networks. The methodology introduced for network traffic based workflow mining is able to counter a possible concept drift by periodically relearning the workflow model.

Concluding Discussion

Data-communication networks contain a multitude of data sources for data mining such as network traffic or events, which are reported by security sensors such as intrusion detection system, intrusion prevention systems or firewalls. This information is be used to provide context-aware event analysis using time series data mining techniques. In order to conclude the research presented in the context of this work, we discuss our contributions with respect to related methodologies and, afterwards, we conclude with an outlook on promising future research direction.

Network Service Dependency Analysis and Workflow Mining

Mission Oriented Network Analysis (MONA) is a network dependency analysis methodology, which solves the problem of discovering workflows within a monitored network. Building on the concept of network service dependency discovery that allows for deriving workflows based on network traffic, we introduce workflow mining based on MONA. To the best of our knowledge this is the first workflow mining methodology which derives workflows by analyzing network traffic.

Network service dependency discovery and workflow mining have been tested in a real-life case study within the data-communication network of an energy distribution network. Also, a systematic evaluation compares MONA to Orion, Sherlock and NSDMiner. The systematic evaluation revealed MONA's ability to identify existing indirect dependencies outperforms Orion, Sherlock and NSDMiner. The focus of workflow mining is to allow a context-aware event analysis by helping network operators understand how events affect ongoing tasks in the monitored network. The reasoning behind this approach is that events affecting business-critical tasks have more impact in the eyes of a network operator than others. Also, network operators can monitor ongoing workflows and adjust security policies accordingly. In addition, workflow mining provides the foundation for a context-aware approach to event prioritization and correlation.

Event Prioritization and Correlation

There are different approaches to event analysis. As opposed to the event correlation and prioritization method introduced in the context of this work, most event correlation approaches aim to extract attack scenarios and predict the next attacker action [VS01; CM02; Pen+08; FAK15; GG15]. The presented approaches implicitly rely on a threat agent model to extract attack scenarios. However, threat agents range from individual hackers, to organized hacker groups, organized crime, industrial espionage, disgruntled employees, terrorist groups to nation state attacks. Due to the lack of information on these different threat agents, predicting their next possible action and validating the proposed adversary model is very difficult.

Rather than focusing on how events affect ongoing tasks in the monitored network, the above mentioned approaches focus on aggregating similar events or extracting attack scenarios to predict the next attacker action. Evaluating the quality of a threat model is difficult as little information is available about different threat agents and their behavioral characteristics. Due to little information being available about threat agents, we argue that it is very difficult to conduct a similar evaluation with attacker model based approaches. In contradiction to knowledge-based approaches we focus on minimizing the required human input. The introduced context-aware event analysis approach is able to (i) automatically discover activities in the network and (ii) correlate incoming events based on targeted workflows.

Future Work

The introduced time series data mining technique MONA can be transferred to other knowledge domains such as intrusion detection. Observed communication patterns are represented by communication histograms. In order to enable intrusion detection, communication histograms which describe the normal behavior of a data-communication network need to be stored. If diverging communication patterns are observed within a monitored data-communication network, a potential intrusion has been detected.

In the context of this work we also introduced a novel workflow mining technique, which relies on network traffic in order to learn workflows. Visualizing workflows learned from network traffic observed within a monitored network could support network operators in decision making. However, future work is required for investigating visualizing techniques for workflows and evaluating their usability.

Another interesting aspect for future investigation is the longest common subsequence mining problem. Longest common subsequence mining is a string matching problem, where the longest common subsequence between two strings needs to be identified. For more details on longest common subsequence see [PD94]. Future work is required to investigate, whether the methodology introduced for network service dependency discovery in Chapter 2, is an approximation to the longest common subsequence mining problem.

List of Figures

1.1	Network Dependency Analysis and Workflow Mining are adjacent knowledge domains. . . .	9
2.1	Example for network activities.	14
2.2	Example for communication histograms. . . .	28
2.3	Example for indirect dependencies.	31
2.4	Example for indirect dependencies within the network activity shown in Figure 2.1.	40
2.5	Direct network service dependencies in an energy distribution network.	43
2.6	Network service dependency detected by MONA within an energy distribution network. 47	
2.7	Network layer model of the ns-3 based random network generator.	49
2.8	Comparison of Orion's, NSDMiner's and Sherlock's precision and recall compared to MONA's.	54
2.9	F-measures for MONA, Sherlock and Orion with network traffic containing 20 indirect dependencies.	56

2.10	F-measures for MONA, Sherlock and Orion with network traffic containing 70 indirect dependencies.	57
2.11	MONA's computation time for a single network size with network traffic containing an increasing number of TCP connections.	59
2.12	MONA's computation time for multiple network sizes with network traffic containing an increasing number of TCP connections.	60
2.13	Comparison between MONA and Orion in a medium size network.	63
2.14	Comparison between MONA and Orion in a large network.	64
2.15	Results for MONA for network traffic containing no indirect dependencies.	65
3.1	Indirect dependencies within the workflow shown in Figure 2.1.	81
3.2	Hidden states within the workflow shown in Figure 2.1.	87
3.3	Candidate for state transitions within the workflow shown in Figure 2.1.	89
3.4	A Hidden Markov Model (HMM).	91
3.5	HMM workflow model example.	100
3.6	Viterbi algorithm on an HMM workflow model example.	101
3.7	Viterbi algorithm on an HMM workflow model example.	102
3.8	A Factorial Hidden Markov Model (FHMM) with two layers.	103
3.9	Network service dependencies detected by MONA within an energy distribution network.	105

3.10	Activities derived from network traffic in an energy distribution network.	109
3.11	Workflow for communicating with medium voltage substations as identified within the real-life case study.	113
3.12	Workflow based vulnerability assessment for vulnerabilities CVE-2007-5423 and CVE-2010-2075, which were detected on mferp2.	118
4.1	An example of a set of network activities <i>NA</i>	147
4.2	Alert processing times.	157
4.3	A time range showing incoming Syslog messages and outgoing LLC alerts within the emulation environment.	159

Bibliography

- [AS12] Rafael Accorsi and Thomas Stocker. “On the Exploitation of Process Mining for Security Audits: the Conformance Checking Case.” In: *27th Annual ACM Symposium on Applied Computing*. ACM, 2012, pp. 1709–1716.
- [ASM13] Rafael Accorsi, Thomas Stocker, and Günter Müller. “On the Exploitation of Process Mining for Security Audits: the Process Discovery Case.” In: *28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1462–1468.
- [AGL98] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. “Mining Process Models from Workflow Logs.” In: *International Conference on Extending Database Technology*. Springer-Verlag Berlin Heidelberg, 1998, pp. 467–483.
- [Alb+13] Massimiliano Albanese, Sushil Jajodia, Ravi Jhawar, and Vincenzo Piuri. “Reliable Mission Deployment in vulnerable Distributed Systems.” In: *43 Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W 2013)*. IEEE/IFIP, 2013, pp. 1–8.

- [BKH01] Saurabh Bagchi, Gautam Kar, and Joe Hellerstein. "Dependency Analysis in distributed Systems using Fault Injection: Application to Problem Determination in an e-commerce Environment." In: *12th International Workshop on Distributed Systems: Operations & Management*. ISBN: 978272611190. Oct. 2001.
- [Bah+07] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A. Maltz, and Ming Zhang. "Towards highly reliable Enterprise Network Services via Inference of multi-level Dependencies." In: *ACM SIGCOMM Computer Communication Review*. ISBN: 978-1-59593-713-1. ACM, Nov. 2007, pp. 13–24.
- [Bah+06] Paramvir Bahl, Paul Barham, Richard Black, Ranveer Chandra, Moises Goldszmidt, Rebecca Isaacs, Srikanth Kandula, Lun Li, John McCormick, David A Maltz, et al. "Discovering Dependencies for Network Management." In: *ACM SIGCOMM 5th Workshop on Hot Topics in Networks (Hotnets-V)*. ACM, Nov. 2006, pp. 97–102.
- [BCY12] Alexandre de Barros Barreto, Paulo Cesar G. Costa, and Edgar T. Yano. "A Semantic Approach to Evaluate the Impact of Cyber Actions on the Physical Domain." In: *7th International Conference on Semantic Technologies for Intelligence, Defense, and Security (STIDS 2012)*. CEUR-WS.org, Oct. 2012.
- [Blu+08] Tobias Blum, Nicolas Padoy, Hubertus Feußner, and Nassir Navab. "Workflow Mining for Visualization and Analysis of Surgeries." In: *In-*

- ternational Journal of Computer Assisted Radiology and Surgery*. Springer-Verlag Berlin Heidelberg, 2008, pp. 379–386.
- [BH01] Kai Briechele and Uwe D. Hanebeck. “Template Matching using fast Normalized Cross Correlation.” In: *Aerospace/Defense Sensing, Simulation, and Controls*. International Society for Optics and Photonics, 2001, pp. 95–102.
- [BKK01] Aaron Brown, Gautam Kar, and Alexander Keller. “An active Approach to characterizing dynamic Dependencies for Problem Determination in a distributed Environment.” In: *IEEE/IFIP International Symposium on Integrated Network Management (IEEE IM 2001)*. ISBN:0-7803-6719-7. IEEE/IFIP, May 2001, pp. 377–390.
- [Che+09] Changhong Chen, Jimin Liang, Heng Zhao, Haihong Hu, and Jie Tian. “Factorial HMM and parallel HMM for gait recognition.” In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 39.1 (2009), pp. 114–123.
- [Che+08] Xu Chen, Ming Zhang, Zhuoqing Morley Mao, and Paramvir Bahl. “Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions.” In: *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2008)*. USENIX, Dec. 2008, pp. 117–130.
- [CLF03] Steven Cheung, Ulf Lindqvist, and Martin W. Fong. “Modeling multistep Cyber Attacks for Scenario Recognition.” In: *DARPA Information*

- Survivability Conference and Exposition*. Vol. 1. IEEE, 2003, pp. 284–292.
- [CW98] Jonathan E Cook and Alexander L Wolf. “Discovering models of software processes from event-based data.” In: vol. 7. 3. ACM, 1998, pp. 215–249.
- [Cup01] Frédéric Cuppens. “Managing Alerts in a multi-Intrusion Detection Environment.” In: *Annual Computer Security Applications Conference (ACSAC 2001)*. Vol. 1. IEEE, 2001, p. 22.
- [CM02] Frédéric Cuppens and Alexandre Mieke. “Alert Correlation in a cooperative Intrusion Detection Framework.” In: *IEEE Symposium on Security and Privacy*. ISBN: 0-7695-1543-6. IEEE, May 2002, pp. 202–215.
- [DCF07] Hervé Debar, David A. Curry, and Benjamin S. Feinstein. “The intrusion detection message exchange format (IDMEF).” In: *IETF*. 2007.
- [Dec+07] Dimitrios Dechouniotis, Xenofontas Dimitropoulos, Andreas Kind, and Spyros Denazis. “Dependency Detection using a fuzzy Engine.” In: *International Workshop on Distributed Systems: Operations and Management*. Springer-Verlag Berlin Heidelberg, 2007, pp. 110–121.
- [EB15] James Edwards and Richard Bramante. *Networking self-teaching Guide: OSI, TCP/IP, LANs, MANs, WANs, Implementation, Management, and Maintenance*. ISBN: 978-0-470-40238-2. John Wiley & Sons, 2015.

- [FAK15] Hamid Farhadi, Maryam AmirHaeri, and Mohammad Khansari. "Alert Correlation and Prediction using Data Mining and HMM." In: *The ISC International Journal of Information Security*. Vol. 3. 2. ISBN: 2008-3076. 2015.
- [Fen+13] Stefan Fenz, Thomas Neubauer, Rafael Accorsi, and Thomas Koslowski. "FORISK: Formalizing Information Security Risk and Compliance Management." In: *43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W 2013)*. IEEE/IFIP, 2013, pp. 1–4.
- [Fer+07] Diogo Ferreira, Marielba Zacarias, Miguel Malheiros, and Pedro Ferreira. "Approaching Process Mining with Sequence Clustering: Experiments and Findings." In: *Business Process Management*. Springer-Verlag Berlin Heidelberg, 2007, pp. 360–374.
- [For73] G. David Forney Jr. "The Viterbi Algorithm." In: *Proceedings of the IEEE*. Vol. 61. 3. IEEE, 1973, pp. 268–278.
- [GJ97] Zoubin Ghahramani and Michael I. Jordan. "Factorial Hidden Markov Models." In: vol. 29. 2-3. Springer-Verlag Berlin Heidelberg, 1997, pp. 245–273.
- [GG15] Mohammad GhasemiGol and Abbas Ghaemi-Bafghi. "E-Correlator: an Entropy-based Alert Correlation System." In: *Security and Communication Networks*. Vol. 8. 5. Wiley Online Library, July 2015, pp. 822–836.

- [GDK09] John R. Goodall, Anita D'Amico, and Jason K. Kopylec. "CAMUS: Automatically mapping Cyber Assets to Missions and Users." In: *Military Communications Conference (MILCOM 2009)*. IEEE, Oct. 2009, pp. 1–7.
- [Her00] Joachim Herbst. "A Machine Learning Approach to Workflow Management." In: *11th European Conference on Machine Learning (ECML 2000)*. Springer-Verlag Berlin Heidelberg, June 2000, pp. 183–194.
- [Her04] Joachim Herbst. *Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen*. Tenea Verlag Ltd., 2004.
- [HK98] Joachim Herbst and Dimitris Karagiannis. "Integrating Machine Learning and Workflow Management to support Acquisition and Adaptation of Workflow Models." In: *9th International Workshop on Database and Expert Systems Applications*. IEEE, 1998, pp. 745–752.
- [HS14] Neminath Hubballi and Vinoth Suryanarayanan. "False Alarm Minimization Techniques in Signature-based Intrusion Detection Systems: A Survey." In: *Computer Communications*. Vol. 49. Elsevier, 2014, pp. 1–17.
- [IS04] ISO and IEC Std. *ISO/IEC 13335-1: Management of information and communications technology security—Part 1: Concepts and models for information and communications technology security management*. 2004.

- [IS11] ISO ISO and IEC Std. "ISO 27005: 2011." In: *Information technology—Security techniques—Information security risk management*. ISO. 2011.
- [Jaj+11] Sushil Jajodia, Steven Noel, Pramod Kalapa, Massimiliano Albanese, and John Williams. "Cauldron Mission-Centric Cyber Situational Awareness with Defense in Depth." In: *Military Communications Conference (MILCOM 2011)*. IEEE, 2011, pp. 1339–1344.
- [Jak11] Gabriel Jakobson. "Mission Cyber Security Situation Assessment using Impact Dependency Graphs." In: *14th International Conference on Information Fusion (FUSION 2011)*. IEEE, July 2011, pp. 1–8.
- [Jak13] Gabriel Jakobson. "Mission-centricity in Cyber Security: Architecting Cyber Attack resilient Missions." In: *5th International Conference on Cyber Conflict (CyCon 2013)*. ISBN: 978-9949-9211-5-7. IEEE, June 2013, pp. 1–18.
- [Jun+00] Stefan Junginger, Harald Kühn, Robert Strobl, and Dimitris Karagiannis. "Ein Geschäftsprozessmanagementwerkzeug der nächsten Generation—ADONIS: Konzeption und Anwendungen." In: vol. 42. 5. Springer-Verlag Berlin Heidelberg, 2000, pp. 392–401.
- [JM09] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009.
- [Kim+14] Anya Kim, Myong H Kang, Jim Z Luo, and Alex Velasquez. *A framework for event prioritization in*

- cyber network defense*. Tech. rep. DTIC Document, 2014.
- [KGE06] Andreas Kind, Dieter Gantenbein, and Hiroaki Etoh. "Relationship Discovery with Netflow to enable Business-driven IT Management." In: *First IEEE/IFIP International Workshop on Business-driven IT Management*. IEEE/IFIP, Apr. 2006, pp. 63–70.
- [Kol50] Andrei Nikolaevich Kolmogorov. "Foundations of the Theory of Probability." In: Chelsea Publishing Co., 1950.
- [KOB12] Avinash Konkani, Barbara Oakley, and Thomas J. Bauld. "Reducing Hospital noise: a Review of medical Device Alarm Management." In: *Biomedical Instrumentation & Technology*. Vol. 46. 6. AAMI, 2012, pp. 478–487.
- [KC13] I. Kotenko and A. Chechulin. "A Cyber Attack Modeling and Impact Assessment Framework." In: *5th International Conference on Cyber Conflict (CyCon 1013)*. ISBN: 978-9949-9211-5-7. IEEE, June 2013, pp. 1–24.
- [KLL16] Alexander Kott, Mona Lange, and Jackson Ludwig. "Assessing Mission Impact of Cyber Attacks: Towards a Model-Driven Paradigm." In: *IEEE Security and Privacy*. IEEE, 2016.
- [Lan16] Mona Lange. "Model-Driven Paradigms for Integrated Approaches to Cyber Defense." In: *NATO IST-ET-094*. , Team Leader , Information Systems Technology Panel, 2015-2016.

- [LK14] Mona Lange and Marina Krotofil. "Mission Impact Modelling for Industrial Control Systems." In: *1st SCADA Security Conference Latin America*. Rio de Janeiro, Brazil, 2014.
- [LK15] Mona Lange and Marina Krotofil. "Mission Impact Assessment in Power Grids." In: *NATO IST-128 Workshop on Cyber Attack Detection, Forensics and Attribution for Assessment of Mission Impact*. Istanbul, Turkey: Information Systems Technology Panel, June 2015.
- [LKM16a] Mona Lange, Felix Kuhr, and Ralf Möller. "Using a Deep Understanding of Network Activities for Network Vulnerability Assessment." In: *22nd European Conference on Artificial Intelligence (ECAI 2016)*. ISBN: 978-1-61499-671-2. The Hague, Netherlands: IOS Press, Aug. 2016, pp. 1583–1585.
- [LKM16b] Mona Lange, Felix Kuhr, and Ralf Möller. "Using a Deep Understanding of Network Activities for Workflow Mining." In: *39th Annual German Conference on Artificial Intelligence (KI 2016)*. ISBN: 978-3-319-46072-7. Klagenfurth, Austria: Springer-Verlag Berlin Heidelberg, Sept. 2016, pp. 177–184.
- [LKM16c] Mona Lange, Felix Kuhr, and Ralf Möller. "Using a Deeper Understanding of Network Activities for Security Event Management." In: *International Journal of Network Security & Its Applications (IJNSA)*. June 2016.
- [LM16] Mona Lange and Ralf Möller. "Time Series Data Mining for Network Service Dependency Analysis." In: *9th International Conference on*

- Computational Intelligence in Security for Information Systems (CISIS 2016)*. San Sebastian, Spain: Springer-Verlag Berlin Heidelberg, Oct. 2016.
- [Lan+15] Mona Lange, Ralf Möller, Gregor Lang, and Felix Kuhr. "Event Prioritization and Correlation based on Pattern Mining Techniques." In: *14th International Conference on Machine Learning and Applications and Workshops (ICMLA 2015)*. Miami, Florida: IEEE, Dec. 2015.
- [Lew95] J.P. Lewis. "Fast Normalized Cross-Correlation." In: *Vision interface*. Vol. 10. 1. 1995, pp. 120–123.
- [Mar13] Scott Marshall. "CANDID: Classifying Assets in Networks by Determining Importance and Dependencies." MA thesis. University of California at Berkeley, 2013.
- [Mar+02a] Laura Maruster, Wil M.P. Van Der Aalst, Ton Weijters, Antal van den Bosch, and Walter Daelemans. "Automated discovery of Workflow Models from Hospital Data." In: vol. 18. IOS Press, 2002, pp. 183–190.
- [Mar+02b] Laura Maruster, AJMM Ton Weijters, Wil M.P. Van Der Aalst, and Antal van den Bosch. "Process mining: Discovering direct Successors in Process logs." In: *International Conference on Discovery Science*. Springer-Verlag Berlin Heidelberg, 2002, pp. 364–373.
- [MR12] Bill Miller and Dale Rowe. "A Survey of SCADA and Critical Infrastructure Incidents." In: *Proceedings of the 1st Annual conference on Research in information technology*. ISBN: 978-1-4503-1643-9. ACM, 2012, pp. 51–56.

- [MC16] Robert Mitchell and Ray Chen. "Modeling and analysis of attacks and counter defense mechanisms for cyber physical systems." In: vol. 65. 1. IEEE, 2016, pp. 350–358.
- [Mor+02] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. "M2D2: A formal Data Model for IDS Alert Correlation." In: *Recent Advances in Intrusion Detection*. Springer-Verlag Berlin Heidelberg, 2002, pp. 115–137.
- [Mur13] Alan T. Murray. "An Overview of Network Vulnerability Modeling Approaches." In: *GeoJournal*. Vol. 78. 2. Springer-Verlag Berlin Heidelberg, 2013, pp. 209–221.
- [Mus+11] S. Musman, M. Tanner, A. Temin, E. Elsaesser, and L. Loren. "Computing the Impact of Cyber Attacks on complex Missions." In: *IEEE Systems Conference (SysCon 2011)*. ISBN: 978-1-4244-9493-4. IEEE, Apr. 2011, pp. 46–51.
- [NL13] Mahsa Naseri and Simone A Ludwig. "Evaluating Workflow Trust using Hidden Markov Modeling and Provenance Data." In: *Data Provenance and Data Management in eScience*. Springer-Verlag Berlin Heidelberg, 2013, pp. 35–58.
- [Nat+12] Arun Natarajan, Peng Ning, Yao Liu, Sushil Jajodia, and Steve E Hutchinson. "NSDMiner: Automated Discovery of Network Service Dependencies." In: *IEEE International Conference on Computer Communications (IEEE INFOCOM 2012)*. ISBN: 978-1-4673-0775-8. IEEE, Mar. 2012, pp. 2507–2515.

- [NCR02] Peng Ning, Yun Cui, and Douglas S. Reeves. "Constructing Attack Scenarios through Correlation of Intrusion Alerts." In: *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 245–254.
- [ODL07] Anthony J. Onwuegbuzie, Larry Daniel, and Nancy L. Leech. "Pearson Product-Moment Correlation Coefficient." In: *Encyclopedia of Measurement and Statistics*. SAGE Publications, Inc., 2007, pp. 751–756.
- [PD94] Mike Paterson and Vlado Dančík. "Longest Common Subsequences." In: *International Symposium on Mathematical Foundations of Computer Science*. Springer-Verlag Berlin Heidelberg, 1994, pp. 127–142.
- [Pen+08] Xi Peng, Yugang Zhang, Shisong Xiao, Zheng Wu, JianQun Cui, Limiao Chen, and Debao Xiao. "An Alert Correlation Method based on improved Cluster Algorithm." In: *Pacific-Asia Workshop on Computational Intelligence and Industrial Application (PACIIA'08)*. Vol. 1. IEEE, 2008, pp. 342–347.
- [Pie04] Tadeusz Pietraszek. "Using adaptive Alert Classification to reduce false positives in Intrusion Detection." In: *Recent Advances in Intrusion Detection*. Springer-Verlag Berlin Heidelberg, 2004, pp. 102–124.
- [PFV02] Phillip A. Porras, Martin W. Fong, and Alfonso Valdes. "A Mission-Impact-based Approach to INFOSEC Alarm Correlation." In: *International Workshop on Recent Advances in Intrusion Detec-*

- tion. Springer-Verlag Berlin Heidelberg, 2002, pp. 95–114.
- [PM16] V. Priyadharshini and A. Malathi. “Analysis of Process Mining Model for Software Reliability Dataset using HMM.” In: *Indian Journal of Science and Technology*. Vol. 9. 4. Indian Society for Education and Environment, Jan. 2016.
- [RJ86] Lawrence R. Rabiner and Biing-Hwang Juang. “An Introduction to Hidden Markov Models.” In: *ASSP Magazine, IEEE*. Vol. 3. 1. IEEE, 1986, pp. 4–16.
- [RF12] Álvaro Rebugue and Diogo R Ferreira. “Business Process Analysis in Healthcare Environments: A Methodology based on Process Mining.” In: vol. 37. 2. Elsevier, 2012, pp. 99–116.
- [RVV08] Anne Rozinat, Manuela Veloso, and Wil M.P. Van Der Aalst. “Using Hidden Markov Models to evaluate the Quality of discovered Process Models.” In: *Extended Version. BPM Center Report BPM-08-10, BPMcenter.org*. 2008.
- [Sch99] Bruce Schneier. “Attack trees: Modeling security threats.” In: *Dr. Dobb's journal*. People's Computer Company, Dec. 1999.
- [SZS05] Ricardo Silva, Jiji Zhang, and James G. Shahan. “Probabilistic Workflow Mining.” In: *11th ACM International Conference on Knowledge Discovery in Data Mining (SIGKDD 2005)*. ACM, 2005, pp. 275–284.

- [Smi+08] Reuben Smith, Nathalie Japkowicz, Maxwell Dondo, and Peter Mason. "Using unsupervised Learning for Network Alert Correlation." In: *Advances in Artificial Intelligence*. Springer-Verlag Berlin Heidelberg, 2008, pp. 308–319.
- [SGV08] Minseok Song, Christian W. Günther, and Wil M.P. Van Der Aalst. "Trace Clustering in Process Mining." In: *Business Process Management Workshops*. Springer-Verlag Berlin Heidelberg, 2008, pp. 109–120.
- [Sow99] John F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Course Technology, Aug. 1999.
- [Tab+11] Karim Tabia, Salem Benferhat, Philippe Leray, and Ludovic Mé. "Alert Correlation in Intrusion Detection: Combining AI-based Approaches for Exploiting Security Operators' Knowledge and Preferences." In: *Security and Artificial Intelligence (SecArt)*. ACM, July 2011.
- [VS01] Alfonso Valdes and Keith Skinner. "Probabilistic Alert Correlation." In: *Recent Advances in Intrusion Detection*. Springer-Verlag Berlin Heidelberg, 2001, pp. 54–68.
- [Van04] Wil M.P. Van Der Aalst. "Business Process Management demystified: A Tutorial on Models, Systems and Standards for Workflow Management." In: *Lectures on Concurrency and Petri nets*. Springer-Verlag Berlin Heidelberg, 2004, pp. 1–65.

- [Van11] Wil M.P. Van Der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. ISBN: 978-3-642-19345-3. Springer-Verlag Berlin Heidelberg, 2011.
- [Van+03] Wil M.P. Van Der Aalst, Boudewijn F. van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and Anton J.M.M. Weijters. "Workflow Mining: a Survey of Issues and Approaches." In: *Data and Knowledge Engineering*. Vol. 47. 2. Elsevier, 2003, pp. 237–267.
- [VWM04] Wil M.P. Van Der Aalst, Ton Weijters, and Laura Maruster. "Workflow Mining: Discovering Process Models from Event Logs." In: *IEEE Transaction on Knowledge and Data Engineering*. Vol. 16. 9. IEEE, 2004, pp. 1128–1142.
- [Van+11] Wil M.P. Van Der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos Buijs, et al. "Process Mining Manifesto." In: *Business Process Management Workshops*. Springer-Verlag Berlin Heidelberg, 2011, pp. 169–194.
- [VV04] Boudewijn F Van Dongen and Wil M.P. Van Der Aalst. "Multi-Phase Process Mining: Building Instance Graphs." In: *Conceptual Modeling—ER 2004*. Springer-Verlag Berlin Heidelberg, 2004, pp. 362–376.
- [VRC05] Namrata Vaswani, Amit K. Roy-Chowdhury, and Rama Chellappa. "'Shape Activity': A continuous-state HMM for moving/deforming Shapes with Application to abnormal Activity

- Detection." In: *IEEE Transactions on Image Processing*. Vol. 14. 10. IEEE, 2005, pp. 1603–1616.
- [Wel03] Lloyd R Welch. "Hidden Markov models and the Baum-Welch algorithm." In: *IEEE Information Theory Society Newsletter* 53.4 (2003), pp. 10–13.
- [WB99] Andrew D. Wilson and Aaron F. Bobick. "Parametric Hidden Markov Models for Gesture Recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 21. 9. IEEE, Sept. 1999, pp. 884–900.
- [WKK07] Y. Wu, M. Kezunovic, and T. Kostic. "An advanced Alarm Processor using Two-level Processing Structure." In: *IEEE Power Tech.* IEEE, July 2007, pp. 125–130.
- [Zan+14] Ali Zand, Giovanni Vigna, Richard Kemmerer, and Christopher Kruegel. "Rippler: Delay Injection for Service Dependency Detection." In: *IEEE International Conference on Computer Communication (INFOCOM 2014)*. IEEE, May 2014, pp. 2157–2165.
- [Zen+13] Qingtian Zeng, Sherry X Sun, Hua Duan, Cong Liu, and Huaiqing Wang. "Cross-organizational collaborative Workflow Mining from a multi-source Log." In: *Decision Support Systems* 54.3 (2013), pp. 1280–1301.
- [Zhu+14] Jianfeng Zhu, Yidan Shu, Jinsong Zhao, and Fan Yang. "A dynamic Alarm Management Strategy for chemical Process Transitions." In: *Journal of Loss Prevention in the Process Industries*. Vol. 30. Elsevier, 2014, pp. 207–218.



Mona Lange

Personal Information

Name **Mona Lange**
Birthday **June 23, 1987**
Place of birth **Hamburg, Germany**

Education

- 2014 **Award**, *ASQF Advancement Award Winner*.
- 2013 **Diploma in Computer Science**, *Friedrich-Alexander University Erlangen-Nuremberg (FAU)*.
- 2013 **Diploma Thesis**, *FAU*, Image Encryption in Social Networks.
- 2006 **Baccalaureate**, *Richard Wagner Gymnasium, Bayreuth*.

Professional Experience

- 10/2014–10/2016 **Research Assistant at the Institute of Information Systems**, *Universität zu Lübeck*, European Research Project PANOPTESec.
- 10/2013–09/2014 **Research Assistant at the Institute for Software Systems**, *Hamburg University of Technology*, European Research Project PANOPTESec.
- 10/2012–08/2013 **Student Assistant at the Institute for IT Security Infrastructures**, *FAU*.
- 07/2011–09/2011 **Software Developer at IBM**, *Extreme Blue*, Smarter Healthcare (Student Project).
- 10/2010–05/2012 **Student Assistant at the Institute for Pattern Recognition**, *FAU*.