The Process Matters: Cyber Security in Industrial Control Systems

Dina Hadžiosmanović

The Process Matters:
Cyber Security in Industrial Control Systems

Dina Hadžiosmanović

This research is supported by the Ministry of Security and Justice of the Kingdom of the Netherlands through the projects Hermes, Castor and Midas.

CTIT Ph.D. Thesis Series No. 13-282
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE
Enschede, The Netherlands

IPA: 2014-02
The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithms).

ISBN: 978-90-365-3604-2
ISSN: 1381-3617
DOI: 10.3990/1.9789036536042
`http://dx.doi.org/10.3990/1.9789036536042`

Typeset with LaTeX. Printed by Wöhrmann Print Service.
Cover design: Arnold Bakker. Photo by: ABB University.

THE PROCESS MATTERS:
CYBER SECURITY IN INDUSTRIAL CONTROL SYSTEMS

DISSERTATION

to obtain
the degree of doctor at the University of Twente
on the authority of the rector magnificus,
prof. dr. H. Brinksma,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday, 9th of January 2014 at 16.45

by

Dina Hadžiosmanović

born on 12th of July 1985,
in Zenica, Bosnia and Herzegovina

The dissertation is approved by:

Prof. dr.   P.H. Hartel (promotor)

mami i tati...

Abstract

An industrial control system (ICS) is a computer system that controls industrial processes such as power plants, water and gas distribution, food production, etc. Since cyber-attacks on an ICS may have devastating consequences on human lives and safety in general, the security of ICS is important. In this context, the most valuable asset is the *process* that is under the control of the ICS. As a result of attacks on the process, the behaviour of the process (i.e., the program output in a computer program) changes due to modifications in: *(i)* the automation logic (i.e., program instruction set) or *(ii)* the process input parameters (i.e., the program input). The detection of process manipulations through attacks is challenging as it requires the understanding of complex process dependencies in sensitive and often proprietary environments. Due to these conditions, the problem of process manipulations has not been thoroughly studied by security researchers.

This thesis tackles this challenge by performing pioneering work in exploring suitable techniques for detecting process attacks in ICS. The main focus of the thesis is the problem of malicious manipulations in process input. To decompose the problem, we distinguish three attack vectors used for accomplishing an input manipulation: *(i)* user application (e.g., issue legitimate but malicious user commands to the plant automation), *(ii)* network (e.g., issue network messages to divert the process by exploiting access vulnerabilities of the network infrastructure) or *(iii)* field devices (e.g., trigger inappropriate automation reaction by sending false data from the field).

In this thesis we analyse the first two types of input manipulations (i.e., threats carried through user application and network infrastructure) as they describe common cyber attacks (i.e., an exploitation of vulnerabilities in software through remote access). The third attack vector remains out of our scope as it typically includes hardware device tampering (e.g., on a measurement sensor). For the selected attack vectors we *(i)* investigate the problem and *(ii)* present and validate an approach for addressing the problem. Based on this, the core contributions of the thesis are structured into four chapters.

First, to investigate the problem of manipulations via the user application, we adapt a common methodology for hazard analysis to systematically identify and characterise potential threats on a real world plant.

Second, based on the obtained knowledge during the problem investigation, we present an approach for addressing process manipulations though the user application. The approach includes mining of event logs to detect undesirable user activities. A real world validation shows that the approach effectively decreases the workload of operators and highlights relevant events for the inspection.

Third, to investigate the problem of network manipulations, we perform an assessment of the state-of-the art detection techniques for network content analysis. The performed analysis presents insights into capabilities and shortcoming of the detectors and discusses promising approaches for addressing process manipulations.

Fourth, we present an approach for detecting process manipulations via network traffic analysis. During the problem investigation, we identified a common weakness of all analysed detectors: the lack of capabilities for the analysis and interpretation of the current process condition. To tackle this, our approach captures low-level process indicators (such as process updates to the memory of a control device) from network traces to derive patterns of normal behaviour and detect deviations. The obtained results show that the approach manages to extract and consistently monitor 98% of process features in a real world plant.

Summarizing, this thesis presents a thorough analysis of input process manipulations in an ICS and presents approaches for addressing two common attack vectors of the analysed threats. Our work shows that relevant information describing process operation can be extracted and analysed from common system traces (i.e., network traffic and system logs) to improve the awareness of the detector about the process that is under the control of the ICS. By doing this, we lay the ground for detecting critical process attacks that cannot be addressed by the existing solutions.

Samenvatting

Een industrial control system (ICS), is een computersysteem dat industriele processen zoals energiecentrales, water- en gasdistributie, voedselproductie etc. controleert. Omdat cyberaanvallen op ICS grote gevolgen kunnen hebben op mensenlevens en algehele veiligheid, is de beveiliging van ICS erg belangrijk. In deze context is het *proces* dat onder de controle van ICS valt het belangrijkste asset. Als gevolg van cyberaanvallen op het proces verandert het gedrag van het proces (d.w.z. de programma output in een computerprogramma) dankzij aanpassingen in *(i)* de automatiseringslogica (d.w.z. het programma instructieset) of *(ii)* de procesinvoer parameters (d.w.z. de programma input). De opsporing van procesmanipulaties door cyberaanvallen is uitdagend omdat het begrip vereist van complexe procesafhankelijkheden in een gevoelige en vaak private omgeving. Hierdoor is het probleem van procesmanipulaties nog niet grondig bestudeerd door veiligheidsonderzoekers.

Deze thesis pakt bovengenoemde uitdaging aan door het uitvoeren van verkennend werk in het bestuderen van passende technieken voor het opsporen van procesaanvallen in ICS. Het hoofdthema van deze thesis is het probleem van kwaadaardige manipulaties in de procesinput. Om het vraagstuk te ontleden, onderscheiden we drie aanvalsvectoren die gebruikt worden om een inputmanipulatie te veroorzaken. Deze drie zijn *(i)* gebruikerstoepassing (d.w.z. legitieme publicatie maar slecht gezinde gebruikersopdrachten aan de fabrieksautomatisering), *(ii)* netwerk (d.w.z. publicatie van netwerkberichten om het proces af te leiden door het exploiteren van toegangskwetsbaarheden in de netwerkinfrastructuur) of *(iii)* veldapparatuur (d.w.z. veroorzaken van een ongewenste automatiseringsreactie door het versturen van verkeerde data uit het veld).

In deze thesis analyseren we de eerste twee types inputmanipulatie (d.w.z. bedreigingen vanuit gebruikerstoepassing en netwerkinfrastructuur) omdat het hierbij gaat om alledaagse cyberaanvallen (d.w.z. een exploitatie van kwetsbaarheden in software door een geringe toegang). De derde aanvalsvector blijft buiten ons bereik omdat het daarbij gaat om manipulatie van de hardware (bij een meetsensor).

Voor de geselecteerde aanvalsvectoren gaan we *(i)* het probleem onderzoeken en *(ii)* een benadering presenteren en valideren om het probleem te adresseren. Hierop gebaseerd zijn de bijdragen van deze thesis gestructureerd in vier hoofdstukken.

Ten eerste willen we het probleem onderzoeken en passen we een veel gebruikte methode voor risicoanalyse toe voor het systematisch identificeren en karakteriseren van potentiele bedreigingen bij een bestaande fabrieksinstallatie.

Ten tweede presenteren we, gebaseerd op de verkregen kennis tijdens het onderzoek, een benadering voor het adresseren van procesmanipulaties door de gebruikerstoepassing. De benadering bevat het inwinnen van gebeurtenislogaritmes voor het opsporen van ongewenste gebruikersactiviteiten. Een real world ratificatie laat zien dat de benadering de werklast van operatoren effectief verlaagd en benadrukt de relevante gebeurtenissen voor de inspectie.

Ten derde onderzoeken we het probleem van netwerkmanipulaties door het uitvoeren van een beoordeling van de allernieuwste opsporingstechnieken voor de network content analysis. De uitgevoerde analyse geeft inzichten in de mogelijkheden en tekortkomingen van de detectoren en bespreekt veelbelovende benaderingen voor het adresseren van procesmanipulaties.

Ten vierde presenteren we een benadering voor het opsporen van procesmanipulaties door een analyse van het netwerkverkeer. Tijdens het onderzoeken van het probleem hebben we een gemeenschappelijk tekortkoming van alle geanalyseerde detectoren ontdekt: het gebrek aan mogelijkheden voor de analyse en interpretatie van de huidige procestoestand. Om dit op te lossen bevat onze benadering low-level procesindicatoren (zoals procesupdates in het geheugen van een controleapparaat) van netwerksporen tot het afleiden van patronen van normaal gedrag en het detecteren van afwijkingen. De verkregen resultaten laten zien dat de benadering 98% van de procesfuncties in een bestaande installatie extraheert en consequent bewaakt.

Samenvattend presenteert deze thesis een grondige analyse van input procesmanipulaties in een ICS en laat het twee benaderingen zien die de twee veelvoorkomende aanvalsvectoren van de geanalyseerde bedreigingen adresseert. Ons werk laat zien dat relevantie informatie die procesoperaties beschrijft afgeleid en geanalyseerd kunnen worden door veelgebruikte systeemsporen (d.w.z. netwerkverkeer en systeemlogaritmes) om het bewustzijn van de detector van het proces dat onder controle is van ICS te verbeteren. Hierdoor leggen we een basis voor het opsporen van kritieke procesaanvallen die niet kunnen worden geadresseerd door de bestaande oplossingen.

Acknowledgements

Sounds surreal, but it is true: I have reached the finish line. Indeed, the road was bumpy and muddy at times, but the journey was absolutely unforgettable, vivid and fun. Now is the time to remember and thank at least some of the people who took part in my great adventure of pursuing a PhD.

Dear Damiano, as my daily supervisor (and as an Italian:), you played an indispensable role in my PhD. I am completely sure that no other supervision would have been so lively and interesting as the one with having you by my side. I always felt that I was on the top of your priorities, thank you for that. And more: thanks for challenging me, fighting for me, having my back, and finally, thanks for letting me fly on my own when I felt like doing so...I hope that our roads will cross again in the future.

Dear Pieter, you taught me how to do research. Thanks for being so kind and supportive with me (even while giving really bad news). Under your wings I learnt how to present, argue and question my ideas. I am truly happy that you are also present in the next chapter of my research life.

Dear Sandro and Emmanuele, you were my extended support team. Thank you for making me feel like a part of *the maf...*, pardon, *familia*. Sandro, thanks for many words of wisdom thorough the last years. Emma, I loved working with you on the last paper! Thanks for using your magical talent to channel the communication between Damiano and me in cloudy days:)

During my PhD I spent several fruitful and exciting months in Berkeley. Robin, thanks for hosting me there, for your for patience and interest during long lasting discussions and confcalls in late afternoons (or early mornings:). I am so happy that our work has a follow up story now.

I thank all the members of the committee for taking their time to read my thesis and provide me valuable comments.

My daily life at UT would not be so memorable without social distractions

of so many interesting people. Thank you all: the old crew for initial words of support (Ayse, Trajce, Luan, Saeed), my dear PhD siblings for sharing the suffer and success (Christoph, Stefan and Arjan), new kiddies for showing that I am getting old (Jan-Willem, Marco, Michael and Elmer), the #3032 officemates for keeping the DIES girl union alive (Begul and Eleftheria), the "grown ups" for giving insights into life after PhD (Lorena, Andreas, Jonathan, Wolter, Eelco and Maarten) and the secretaries (Bertine, Nienke, Ida and Suse) for helping with all kinds of silly requests I came up with in the past years.

My dear housemates Ramazan, Samuele, Sinem, Ertug, Unai and Omkar, thank you for making Enschede feel like home in so many cold rainy days. Bella Elisa, you gave me hope that I could find a soulmate so far from home. Christian K., dear and inspiring food-dude! Prof. Hanjalić, thanks for the initial (and crucial) encouragement. Ajla, Dea, Ena and Omar, you know I carry you with me wherever I go!

Finally, I am very fortunate to have a perfectly functional, lightly-possessive, warm and wonderfully colourful family. I owe my greatest gratitude to them!

Dear Sezerim, you are my biggest inspiration and my biggest support. Your easy (and lazy:)) character perfectly complements my (sliiiiiightly) hyperactive nature. My love, thank you for holding me in hard times of PhD and loving me just the way I am. The airports and train stations of Europe know how hard we fought for our love all these years. I am so happy we made it!

Sevgili Karaoglu ailesi, Sezer'in ve benim her zaman arkamizda olup, verdigimiz kararlari suphesiz desteklediginizden oturu sizlere tesekkur ederim.

Lieve Loes and Arnold, heel erg bedankt voor jullie begrip, cover en de samenvatting!

Dragi moji Hadžiosmanovići, Karahodžići i Timotijevići, stari i mladi, hvala što ste *svi* bili uz mene i na svoj način učestvovali u mom dugogodišnjem putu prema doktoratu (strikan V. dok me je vodio na časove engleskog; daidža D. i Aida kao univerzalni *"ni-na-ni-na"* službenici; tetka J. i tetak B. kao doktori za "uho-grlo-nos" i zamjenski *mama i tata*; majka, Dedo i Dada sa zrncima mudrosti...). Tanja i Srle, hvala što ste me obogatili još jednim bratom i sestrom.

Pajdo moj najdraži, hvala za sve godine suptilne i nesebične podrške. Neco, hvala za logistiku i toplo gostoprimstvo svaki put. Djevojčice moje, vi ste razlog zašto se tako često vraćam u Sarajevo!

Najdraži moji majko i tajko, nikada vam ne mogu zahvaliti na ljubavi i podršci koju ste mi poklanjali svih ovih godina. Ne mogu ni vratiti vrijeme koje nismo proveli zajedno zbog mog doktorata. Mogu vam samo zahvaliti na strpljenju i pokloniti vam rezultat svog mog truda, posvetiti *vama* ovu tezu. Puno vas volim!

Amsterdam, December 2013.

Contents

Chapter 1

Introduction

Industrial control systems (ICS) monitor and control physical processes, often inside Critical Infrastructures like power plants and power grids, water, oil and gas distribution systems, building monitoring (e.g., airports, railway stations), production systems for food, cars, ships and other products.

Although failures in the security or safety of critical infrastructures could impact people and produce damage to industrial facilities, recent reports state that current critical infrastructures are not sufficiently protected against cyber threats. For example, according to the report by the U.S. Department of Justice [89], around 2700 organisations dealing with critical infrastructures in the U.S. detected 13 million cybercrime incidents, suffered $288 million of monetary loss and experienced around 150 000 hours of system downtime in 2005.

Security of ICS raises an additional concern since ICS failures often cause cascading effects in other systems (due to inter-dependencies amongst systems). For example, known failures in energy and telecommunication services had immediate consequences on various services such as financial (e.g., ATM transaction halt), transportation (e.g.,stopping of city metro service), government (failure of the 112 emergency number) [36].

The increasing number of security incidents in ICS facilities is mainly due to a combination of technological and organizational weaknesses[130]. In the past, ICS facilities were separated from public networks, used proprietary software architectures and communication protocols. Built on the “security by obscurity” paradigm, the systems were less vulnerable to attacks leveraging ICT. Although keeping a segment of communication proprietary, ICS vendors nowadays increasingly use IP-based communication protocols and commercial off-the-shelf software. Also, it is standard to deploy remote connection mechanisms to ease the management during off-duty hours, and achieve nearly-unmanned operation.

Unfortunately, the stakeholders seldom enforce strong security policies. User credentials are often shared among users to ease day-to-day operations, seldom updated (and not always revoked), resulting in a lack of accountability [11]. An example of such practice is the incident in Australia when a disgruntled (former) employee used valid credentials to cause a havoc [100].

Due to these reasons, ICS facilities have become increasingly vulnerable to internal and external cyber attacks. Although companies reluctantly disclose incidents, there are several published cases where safety and security of ICS were seriously endangered [90].

1.1 Motivation

To begin we will present some definitions relevant for understanding the remainder of the chapter (a more comprehensive summary of definitions can be found in Chapter 2). In general terms, a threat is any intention that uses unauthorised access or activity to negatively impact system operation. A vulnerability is a weakness in the system (e.g., design or implementation) that could be exploited by a threat source. An attack is a threat that has been realised. An attack vector is the path that is used by the threat source to obtain the goal (e.g., an attacker uses malicious code to exploit software vulnerability in the process controller and discrupt the process). Like a "regular" computer system, an ICS is susceptible to threats exploiting software vulnerabilities (e.g., protocol implementation, OS, ICS application). However, an ICS environment is also prone to *process* threats which exploit weak application logic that controls the process. By *process*, we here refer to an industrial process: a systematic series of mechanical or chemical operations that produce or manufacture something [121].

In this context, a malicious, yet legitimate use of valid system commands can disrupt the physical process. Process threats also include situations when system users make an operational mistake, e.g., define the capacity of the tank to be 5 times higher than in reality. The most prominent real-life process attack was performed by Stuxnet [77]. This is the first malware that, besides performing a sophisticated exploitation of various system vulnerabilities, diverted the targeted process to cause harm to hardware, and finally result in process failure.

As process threats are specific to the ICS environment, we focus on this type of threat. On one hand, to the best of our knowledge, there are no available security solutions (both in the academic and commercial community) that offer a tailored, comprehensive protection against process threats (at best, the current network approaches monitor the performed functionalities, rather than the actual process indicators.) On the other hand, the importance of addressing these threats is widely acknowledged. For example, a guideline document by the National Institute of Standards and Technology presents a list of relevant threat scenarios in ICS environments. In this list, 7 out of 9 threat scenarios cause direct consequences on the process [105]. Also, Langner [63] states that the biggest concern of plant operators is in the area of threats which leverage legitimate process commands to change critical process parameters (e.g., a setpoint of the pump speed) and thus result in process disruption. In our opinion, the reason for this unbalance lies in the fact that the development of suitable cybersecurity techniques requires an extensive analysis of process characteristics and behaviours which are unavailable to IT cybersecurity experts.

We now discuss possible ways in which an industrial process can deviate from normal behaviour.

1.1.1 How can a process deviate?

As in any deterministic computer program, the output of a process changes due to two reasons: *(i)* the (automation) code and *(ii)* the (process) input parameters. First, a change in the automation code modifies the character of the process. For example, an update of the controller code can result in a modification of the process speed. This action effectively changes the process behaviour (until the next code update). Second, a change in input parameters can trigger a process change (e.g., insert a combination of parameters that stops the pumping procedure). This action causes a temporary modification in the process behaviour (until the next parameter input or transition to the next process state). Once misused, these actions become a threat (e.g., insert code that reverses the process or insert a combination of parameters that stop key process controllers). A threat leveraging a code update typically uses an administrative command (e.g., Modbus function code 24 - write file record). By contrast, input parameter manipulation uses the same set of commands that are used by process operators (e.g., Modbus function codes 3, 16 - write registers). Since there is no evident difference between normal user commands and input parameter manipulation, the detection of the threats that result from input parameter manipulation is more difficult than the detection of genuine admin commands.

This thesis therefore focuses on detecting input parameter manipulation. We distinguish three general attack vectors for accomplishing input parameter manipulation:

- user application (e.g., issue legitimate but malicious user commands via user application to cause process to halt),

- network (e.g., send malicious network messages to the input interpreter to divert the process),
- field devices (e.g., trigger inappropriate automation reaction by sending false measurement data from the field).

Each attack vector implies specific requirements on the attacks. In particular, the attacker needs to perform the following actions:

- via user application – *(i)* get access to the ICS software (typically through legitimate/stolen credentials) and *(ii)* obtain knowledge which user commands can endanger the process,
- via the network – *(i)* bypass network access control (e.g., by obtaining control over a trusted workstation that is located in the process control network) and *(ii)* obtain knowledge for generating messages that will be valid for the input interpreter (e.g., develop a client for the protocol user by the targeted controller).
- via field devices – *(i)* perform hardware device tampering (e.g., on a measurement sensor) or *(ii)* physical damage.

This thesis primarily focuses on the analysis of threats that use software as a part of their attack vector (thus vectors: user application and network). Therefore, the analysis of threats using field devices is out of the scope of this work.

We now analyse challenges for defending against process attacks.

1.1.2 Cyber security for process manipulations

There are several commonly accepted strategies for securing IT systems against cyber attacks. For example, "defence in depth" is a multilayer application of security controls (authentication, network segmentation, firewalls, physical security, etc.) to protect an IT environment against diverse cyber threats [9]. The practical aim of a multilayer strategy is to introduce complementary defence mechanisms and thus to decrease the probability of an attacker penetrating the system. As the level of attack sophistication increases (think of a targeted process attack compared to password guessing), the mitigation strategy requires a higher level of threat understanding to tackle it. In the context of traditional IT, process manipulations resemble the class of internal penetrations. According to Anderson [14], internal penetrations are threats which involve the misuse of access and data rights in the system. Since the attacker is authorised to use the system, the detection of these attacks is hard. In addition to this, cyber security for process manipulations is specific due to two main reasons. First, process threats represent a type of threat that does not exist in the IT domain (i.e., there is no physical process that can be influenced by cyber threats). This means that, to address them, the threats need to be analysed and decomposed to understand how they are manifest in the system. Second, ICS environments differ from traditional IT (e.g., in architecture, mode of operation, network protocols). This means that traditional IT cyber security strategies (e.g., network intrusion detection) often need to be adjusted to work in the new environment (e.g., build suitable protocol analysers for the environment). We identified three general problems that, in our opinion, represent challenges for addressing process threats in ICS: *(i)* threat characterisation, *(ii)* applicability of common IT countermeasures and *(iii)* inclusion of process semantics. We now describe each challenge in more detail.

Threat characterisation A key precondition for a reliable threat detection is the identification of descriptive threat artefacts (i.e., clues that uniquely distinguish the threat from benign behaviour). The analysis and description of process threats is not trivial and differs from threats in traditional IT. We explain this challenge by highlighting the differences amongst the two environments (ICS and traditional IT). In particular, we see two important differences. First, process threats directly influence a physical environment. For example, an attack on a gas distribution facility may have effects on human lives while an IT threat typically targets information availability or integrity. The identification of potentially undesirable threats in a continuous physical process requires the understanding of various process dependencies, and is thus different from the identification of for example information theft. Second, realisations of process threats differ for each plant setup (e.g., an attack targeting a specific water plant may not work for other plants). In practice, this means that the characteristics of a threat are different for each specific environment. This inevitably leads to difficulties in identifying descriptive and common characteristics of the analysed threats.

Applicability of common countermeasures A straightforward strategy for mitigating cybersecurity threats in the ICS context is the application of common IT strategies (e.g., firewalls, encryption, intrusion detection systems, password policies). Although best practices apply (e.g., enforcing access control, network segmentation), many techniques face practical difficulties. For example, some ICS sectors have real-time requirements (such as in energy, transportation), so the latency and ”analysis throughput” issues may introduce unacceptable delays and degrade or prevent acceptable system performance [9]. Also, common network-based solutions have limited applicability in the ICS context. For example, the most common intrusion detection systems (IDS) are misuse-based. They are convenient as there is a large range of signatures for many network and host architectures using modern protocols in common IT environments. However, due to a low number of published attacks in the ICS domain, the range of available signatures in the ICS domain is small and inadequate [9]. In addition, ICS environments often use specific protocols (e.g., MMS, Profinet) whose analysis capabilities have not been included in current IDS (with exceptions of Modbus and DNP3 preprocessors in SNORT [128]). On the other hand, due to mostly automated behaviour, some techniques suit the ICS environment better than the traditional IT (e.g., whitelisting common functionalities [81]). We believe that standard IT mitigation strategies have not been studied sufficiently in the ICS context to identify promising fields of application and acknowledge the limitations.

Inclusion of process semantics The existing literature suggests that the success of attack detection directly depends on the level of context knowledge used during the analysis [101]. Basically, the more we understand about the system environment and the way an attack occurs, the better chance we have to detect malicious behaviour. In the field of business analysis, *process mining* aims to discover, monitor and improve business processes by extracting knowledge from event logs [110]. Similarly, for monitoring *industrial processes*, we need to discover and analyse the semantics of the industrial process (i.e., the knowledge describing normal process behaviour and implications of a process change). The acquisition of the process semantics differs depending on the type of data source. We discuss two sources of process information: ICS log and network data.

ICS event logs represent interpreted process information (e.g., "tank level is high"). Since this type of data already carries interpreted process semantics, there is no need for an additional extraction of process semantics. However, we identify two important challenges in using this type of data, namely: *(i)* log integrity *(ii)* compatibility issues during log extraction and *(iii)* incomplete process interpretation.

First, logs can be corrupted by an attacker (e.g., by tampering measurements that will trigger different log events). In general, there is no mechanism that can detect and isolate such manipulated log entries.

Second, depending on the vendor and software version, event logs are held in different log formats and may require different extraction methods.

Third, the logs are preconfigured to interpret user-defined process events only (e.g., raise an alert if the tank level is high and pressure is high). In theory, such logging should be sufficient to capture relevant process activities. However, in practice, this type of data capturing is not comprehensive, and thus might miss some process activity (e.g., an attack might cause an inconsistency between measurements that will not be logged as that situation was not predefined for log generation).

We now discuss the second information source: the network data. Network data carries process information "as is" (e.g., network traces carry raw process measurements, instead of interpreted process activities). This is good because the network data can provide a comprehensive view on the process (i.e., capture all communication passed on the network). However, the extraction of the process semantics is challenging. To illustrate the problem, we discuss the capabilities of common network analysis techniques for the detection of process attacks. The content of a process attack is carried in packet payload (since the payload holds application/process data). Thus, a promising technique for detecting process attacks must include network payload analysis. Generally, payload analysis is used for detecting attacks that target applications (e.g., shell-code attacks). We distinguish two general approaches for payload analysis: *(i)* functional analysis and *(ii)* statistical content analysis.

First, network parsers are used to decode raw network data and interpret the information carried within the packet (i.e., identify protocol functionalities used in the packet). For example, by decoding the received packets, information about the frequency of specific operations (read parameter, write to file, update parameter) can be extracted and used to characterise the daily process operation [45]. To fully understand the process, the decoder has to parse the protocol up until the application level of the OSI model. In practice, many decoders do not have this ability. For example, decoders for Modbus/TCP protocol available in two widely utilised environments (Bro [83] and Wireshark [138]) do not fully parse the application level. Because of this, the analysis towards the interpretation of process parameters is not possible. Provided that the full decoder is available, the main challenge remains: understand and evaluate the semantics of the observed data within the context of the current process state.

Second, statistical payload analysis is used as an alternative to protocol decoding (i.e., when the decoder is not available). Statistical analysis works under the assumption that the statistics of benign and malicious data packets differ significantly (e.g., shell code vs. HTTP network trace). In the context of process manipulations, this assumption generally does not hold as the malicious behaviour here can be represented by only one bit of difference compared to benign packets (e.g., turn the controller off by flipping one bit in network packet). Therefore, the detection of process manipulations using this approach can be unreliable.

1.2 Research question

Based on the analysis of process threats in ICS, this work focuses on answering the following research question:
"How to design techniques for the detection of process attacks in ICS?"

To achieve this we perform a set of studies on real ICS plants and real data from the plants. This work focuses on two attack vectors targeting process disruptions via input parameter manipulation: user application and network (described in Section §1.1.1).

To address the first attack vector we pose two detailed research questions. First, to characterise the threats we try to answer the following research question:

RQ 1 *What are the process threats occurring via the user application?*

Second, we aim at deriving an effective technique for detecting process threats via the user application. We identify event logs as a promising source of information. Based on this we pose our next research question:

RQ 2 *How can we automate the detection of undesirable user actions in ICS logs?*

To tackle the second attack vector, network threats, we first investigate how current network-based techniques cope with advanced network attacks. To do this we formulate the following research question:

RQ 3 *How can network-based state-of-the art techniques detect process attacks on ICS?*

Finally, after analysing the difficulties in detecting network threats, we focus on improving the capabilities of monitoring approaches for the ICS context. For this we address the following research question:

RQ 4 *How can we enrich network monitoring with process context data?*

We now provide more details on the goals of each research question.

The objective of **RQ 1** is to investigate the problem of process threats occurring through user application. The problem investigation is important as it represents the first step towards the design of viable solutions for problem treatment. In particular, an essential requirement for designing a mitigation strategy is the understanding of different types of threats that can be mounted against the system, and how these threats may manifest themselves in the data.

The objective of **RQ 2** is to derive a technique that can perform an automated analysis of plant operation logs and find potentially undesirable activities. This problem is relevant since even a small plant installation generates thousands of events per day thus the manual inspection of logs is practically infeasible.

The main goal of **RQ 3** is to investigate the problem of network threats and to understand how process threats are manifest in the network traces. While there are several works that benchmark detection capabilities of different approaches, the literature generally lacks works that investigate the core problems of a particular performance (e.g., why a detector has a high false positive rate for a specific type of threat). To address this question we perform an in-depth assessment of the state-of-the art detection techniques for network content analysis.

The common weakness of all analysed detectors is the lack of capabilities for the analysis and interpretation of the current process condition. The main goal of **RQ 4** is to tackle this problem and derive an approach which will be able to evaluate the process condition. In this context, the main challenge is the extraction and interpretation of process indicators from the network data.

1.3 Thesis overview

We start by presenting relevant background information and notation that are necessary to understand the remainder of the thesis (Chapter 2). In Chapter 3 we present a methodology for identifying threats that aim at manipulation of ICS process via the user application. In Chapter 4 we present *MELISSA*, our tool for semi-automated detection of undesirable user actions in ICS. Chapter 5 investigates the problem of network process manipulations and presents a comparative analysis of the state-of the art techniques for network payload analysis. In Chapter 6 we present our approach, *SONICS*, for performing semantic network monitoring in ICS networks. Finally, Chapter 7 presents conclusions and future directions. Figure 1.1 depicts the overview of thesis contributions. We now elaborate further the contribution of each chapter.

On the Misuse of ICS Applications via User Activity (Chapter 3) We present a two-step approach for characterising process threats occurring via the user activity: *(i)* systematic identification of user actions within the ICS application and *(ii)* an analysis of user actions. We demonstrate our approach as a case study on a real-life ICS plant controlling a water treatment plant. We start by performing a structured analysis of internal ICS documentation to identify user actions. We analyse the identified activities by adapting a well known methodology for hazard analysis (HAZOP). By using a series of focus groups sessions with process experts, we identify and characterise process threats. Our analy-

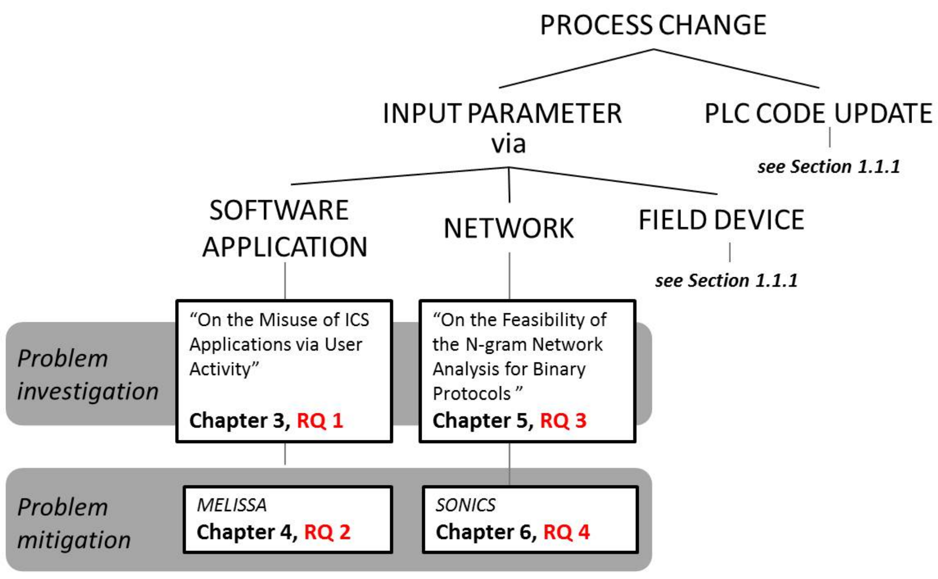


Figure 1.1: Thesis outline

sis identified 36 potentially undesriable actions on the engineering workstation in the analysed plant. Finally, we discus promising approaches for detecting the identified threats. This work has appeared as a journal article [1] and a refereed workshop paper [6].

A Log Mining Approach for Monitoring User Activity in ICS (Chapter 4) We present an approach for addressing process threats that occur through ICS application. The approach includes semi-automated analysis of event logs to detect undesirable user activities. In essence, our tool, *MELISSA* leverages an algorithm for pattern mining to detect more and less frequent actions. A real world validation shows that the approach effectively decreases the workload of operators and highlights relevant events for the inspection. This work has appeared in a refereed conference papers [4] (as full paper) and [3] (as an extended abstract).

N−gram Against the Machine: On the Feasibility of the N−gram Network Analysis for Binary Protocols (Chapter 5) We perform a comparative assessment of four state-of-the art network- content-based detectors: PAYL [115], Anagram [113], POSEIDON [25] and McPAD [84]. The assessment includes two common network protocols from two environments: LAN (SMB/CIFS) and ICS (Modbus/TCP). During the analysis we discuss the reasons why certain attack instances are (not) detected by the chosen approaches. Also, we discuss the feasibility of deploying such approaches in real-life environments, in particular w.r.t the false positive rate, an issue that is seldom discussed in IT research community. This work has appeared in a refereed conference paper [5].

Through the Eye of the PLC: A Network Monitoring Approach for ICS (Chapter 6) We present an approach that reconstructs and models the process behaviour. In particular, our tool, *SONICS* captures and extracts low-level process indicators (such as process updates to the memory of a control device) from network traces to derive patterns of normal behaviour and detect deviations. The obtained results during the validation show that the approach manages to consistently monitor 98% of process features in a real world plant. Our results confirm the feasibility of process monitoring via network analysis and represent a step ahead towards a viable process-aware intrusion detection system. This work has appeared as a technical report [7] and will be submitted to a refereed conference.

Chapter 2

Preliminary topics

In this chapter provide background information necessary to understand the remainder of the thesis. We start the chapter by introducing definitions and concepts that will be used in the following chapters. Next, we explain a typical ICS environment, focusing on the architecture, daily operation and the differences between traditional IT systems and ICS. Finally, we give a brief overview on the techniques for detecting intrusions in traditional IT systems.

2.1 Glossary and basic definitions

We now introduce the concepts and terms used in this thesis. We adopt (and reproduce) the definitions presented in the Guide on Information Security by the National Institute of Standards and Technology [42].

> An **industrial control system** is an information system used to control industrial processes such as manufacturing, product handling, production, and distribution.
>
> A **critical infrastructure** is a system and assets, whether physical or virtual, so vital to a nation that the incapacity or destruction of such systems and assets would have a debilitating impact on security, national economic security, national public health or safety, or any combination of those matters.
>
> A **threat** is any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, individuals, other organizations, or a nation through an information system via unauthorized

access, destruction, disclosure, or modification of information, and/or denial of service.

A **threat scenario** is a set of discrete threat events, associated with a specific threat source or multiple threat sources, partially ordered in time.

A **vulnerability** is a weakness in a system, system security procedures, internal controls, or implementation that could be exploited by a threat source.

An **attack** is an attempt to gain unauthorized access to system services, resources, or information, or an attempt to compromise system integrity, availability, or confidentiality.

An **incident** is an occurrence that actually or potentially jeopardizes the confidentiality, integrity, or availability of an information system.

Cyberspace is a global domain within the information environment consisting of the interdependent network of information systems infrastructures including the Internet, telecommunications networks, computer systems, and embedded processors and controllers.

A **cyber attack** is an attack via cyberspace, targeting an enterprise's use of cyberspace for the purpose of disrupting, disabling, destroying, or maliciously controlling a computing environment/infrastructure; or destroying the integrity of the data or stealing controlled information.

Cyber security is the ability to protect or defend the use of cyberspace from cyber attacks.

An **attack vector** is a path or a means by which an attack can be made on critical infrastructure [58].

A **countermeasure** is a management, operational or technical control prescribed for an information system to protect the confidentiality, integrity, and availability of the system and its information.

Intrusion detection is the process of monitoring the events occurring in a computer system or network and analysing them for signs of possible incidents.

We now describe industrial control system in more detail.

2.2 Industrial control systems

An industrial control system is a general term that comprises several types of systems like: SCADA (Supervisory Control and Data Acquisition), DCS (Distributed Control system), IA (Industrial Automation), IACS (Industrial Automation and Control Systems), PCS (Process Control System). Although the literature often disagrees in the correct usage of the terms in specific situations, we can highlight the general differences amongst the most popular terms: SCADA, DCS and PCS.

SCADA systems are highly distributed systems used to control geographically dispersed assets, where centralized data acquisition and control are critical to system operation [105]. A typical application is in water distribution and wastewater collection systems, oil and natural gas pipelines, electrical power grids, and railway transportation systems. By contrast, a DCS system is usually located in one plant area. DCS are used to control industrial processes such as electric power generation, oil refineries, water and wastewater treatment, and automotive production. As a more specific term, PCS refers to the automation logic that operates the actual process (e.g., exact controllers composing the water plant). In the remainder of the thesis we do not differentiate between the specific terms but instead use ICS as the general, superset term.

In general, an ICS consists of two main domains: the process field and a control room (Figure 2.1). Large systems may have more than one control room. The network infrastructure binds the two domains together. The control room provides an interface between the field and ICS operators (with a real-time overview of the process field statuses). The process field consists of control elements that operate the field devices (e.g., pumps, tanks, pipes, valves).

Depending on the underlying process, the systems differ from each other. For example, a power-related installation contains power switches and transformers while a water-related installation contains water pumps and valves. Based on the interviews with ICS experts from different domains (described in Chapter 3), the computer systems controlling these processes still behave in a similar way.

2.2.1 Architecture

Despite the fact that there are different vendors producing ICS equipment, the system architectures in various ICS facilities are similar and the terminology is interchangeable. Figure 2.2 shows an adapted architecture from a well known ICS vendor. Layer 1 consists of physical field devices, PLCs (Programmable Logic Controllers) and RTUs (Remote Terminal Units). The PLCs and RTUs are responsible for controlling the industrial process, receiving signals from the field devices

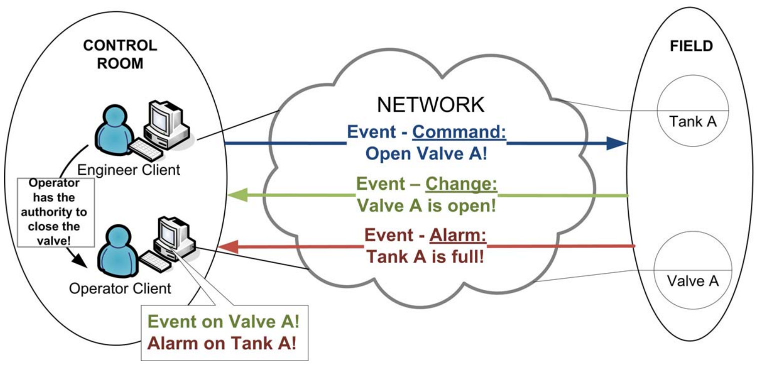

CONTROL ROOM
Engineer Client
Operator has the authority to close the valve!
Operator Client
Event on Valve A!
Alarm on Tank A!
NETWORK
Event - Command: Open Valve A!
Event – Change: Valve A is open!
Event - Alarm: Tank A is full!
FIELD
Tank A
Valve A

Figure 2.1: ICS overview: control room and process field

and sending notifications to upper layers. In practical terms RTU are commonly used to support automation over large geographical areas (e.g., via telemetry and wireless networks) while PLCs are typically used in setups using local networks. Layer 2 consists of ICS servers responsible for processing data from Layer 1 and presenting process changes to Layer 3. Connectivity Servers aggregate events received by PLCs and RTUs and forward them to ICS users in the control room. The Domain Controller in Layer 2 holds local DNS and authentication data for user access. The Aspect Server is responsible for implementing the logic required to automate the industrial process. For example, an Aspect Directory in the Aspect Server holds information about working ranges of the field devices, the device topology, user access rights, etc. Besides, the Aspect Server collects and stores data from the Connectivity Servers into audit and event logs. The various clients in Layer 3 represent ICS users.

We now present more details on the most important ICS component for process automation, the automation controller. To describe the concepts of control automation, we use PLC as an example (since it typically operates on local computer networks).

2.2.2 Programmable logic controller (PLC)

A PLC is an embedded device that holds the logic to automate and control the industrial equipment. A PLC is interesting since the operation on it directly influences the process. In typical setups, a set of PLCs might control a process safely without human intervention for extended periods of time, sometimes for days. A PLC consists of CPU, memory, I/O modules, and communication interfaces. The CPU executes logical operations while program and memory hold program code

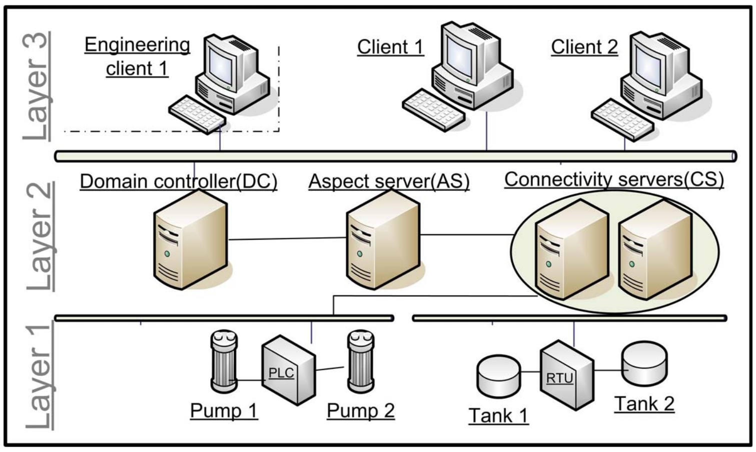


Figure 2.2: A simplified ICS architecture

and data values. I/O modules interface to the controlled field devices as well as other PLCs part of the same process.

The control strategy of a PLC executes a program repeatedly over time as an "infinite" cycle of *(i)* reading inputs, *(ii)* executing logic, and *(iii)* writing outputs. The read operations collect the status from connected field devices (e.g., pump speed, tank level), the execution logic then computes updates to the process (e.g., a new pump speed based on the current tank level). Finally, the write operations put the changes to the process flow into effect (e.g., decrease pump speed setpoint).

Process Variables. Inside the PLC, two components determine the process control: *(i)* the code, and *(ii)* the transient state in the form of process variables. The code consists of logic that regulates the field devices, and drives interaction with the external infrastructure. For example, the code defines the procedure for filling in a tank, along with necessary preconditions that need to be satisfied (e.g., the water level and pressure). PLCs are typically programmed in derivatives of languages such as Pascal and Basic.

Process variables characterize the current operation state in a PLC. Examples of typical variables include the setpoint for a physical process, the current value of a valve sensor, and the current position in a cycle of program steps. Process variables serve as input to the PLC code. For example, a variable value representing a high pressure level might trigger the start of a draining stage. Likewise, the PLC carries out operator commands by writing into corresponding variables. For example, a command to open a valve would update a variable that the program code is regularly checking; once it notices the update, it outputs the corresponding analog signal to the physical device.

2.2.3 Communication

We now describe communication in ICS. Conceptually, we commonly find two semantic groups of network communications between ICS components: *(i)* process awareness, and *(ii)* process control.

The *awareness* communication propagates status information about the controlled process across devices. In particular, the ICS servers requests regular updates from the PLCs to the HMI to report the current plant status to ICS users. In addition to escalating critical updates for timely reaction, awareness also collects trending data for long-term process analysis. PLCs also propagate awareness information across themselves to ensure that each device learns sufficient information about critical variables before entering the next process stage (e.g., PLC 1 might require information about the state of a field device connected to PLC 2 before starting a subsequent process stage).

The *control* communication is generally exercised in one of two ways: *(i)* by PLCs (according to the embedded logic); and *(ii)* by user commands that override the PLC internal logic. Note that in either case it is the PLC that carries out the action, and hence will reflect the process change as updates to its internal state.

Protocols In an ICS architecture, the communication between different ICS servers is typically performed via OPC [95], a communication standard for industrial automation. Depending on the vendor, the communication towards PLCs uses legacy or open protocols. While some protocols are used in general deployment (e.g., Modbus [119], Profinet [85], IEC 60870 [118]), others remain industry-specific (e.g., BACnet [13] for building automation; DNP3 [47] for power networks).

In general, all protocols used in ICS are binary protocols. In contrast to text-based network protocols (such as HTTP, POP and SMTP), binary protocols are not readable by humans. Such protocols are largely used in network services, such as distributed file systems, databases, etc. In practical terms, the network payload of a binary protocol is more compact when compared to text protocols, often unreadable by a human and may resemble attack payloads (since malware packets often consists of binary fragments too).

Network representation Within a PLC device, process variables map directly to PLC memory cells. At the network, ICS protocols define corresponding *net-*

work representations to refer to variables as part of commands, e.g., to specify the target variable for a read operation. In this thesis we focus on analyzing one of the most used ICS protocols, Modbus. Modbus represents process variables in the form of a PLC-specific *memory map*, consisting of 16-bit *registers* and 1-bit *coils*. Some vendors also deploy variations of the default specification, such as combining 2 or 4 registers to hold 32-bit or 64-bit values, respectively. The layout of a Modbus memory map remains specific for each device instance, and is generally determined by a combination of device vendor, programmer, and plant policies (see, e.g., [97] for a generic memory map, which typically act as a starting point).

2.2.4 Users

An ICS is operated by two types of users: operators and engineers. An engineer is responsible for managing access rights, setting working ranges for devices, writing automation scripts, etc. An operator monitors the system status and reacts to events, such as alarms, so that the process runs correctly. Typical operator actions, depending on the underlying industrial process, include commands such as: change switch status, increase temperature, open outlet, start pump. Although industrial processes in various domains differ in the details (or some user roles may be assigned to external parties such as vendors), the user interaction with an ICS is broadly similar. The process experts acknowledged that an engineer is a more powerful system user than an operator (e.g., an engineer writes scripts that define process automation while operators usually only run the script). Also, operators perform actions that are predefined by engineers (e.g., an engineer defines pump speed range, while an operator works within the range only). This means that operator actions are security and safety constrained depending on the way the engineer implemented controls. In contrast, there is no mechanism that will ensure that engineer actions are safe for the process (e.g., an engineer can, by mistake, assign a capacity 10 times bigger than in reality to a tank, and thus shut tank level alarms off). Although individual operator actions are legitimate and should be safety constrained, the stakeholders acknowledge that a sequence of operator actions can still produce damage to the process. In our work we focus on the activities of process engineers. They have more privileges than operators and their activities are therefore the most dangerous.

2.2.5 Operation

In Section §2.2.3 we explain two communication flows in ICS: control and awareness. By using the control flow, a user can make modifications on the process. To do this, they leverage ICS supervisory applications.

ICS supervisory application Each class of users is supported by specific software. First, operators use an HMI interface to perform daily monitoring of the process operation. This application is connected to *(i)* field sensors (to provide current field measurements) and *(ii)* ICS servers (to provide information on the current process configuration).

Second, process engineers use engineering workstations to define process configurations in ICS servers. The operational environment of process engineers varies across different vendors, but is commonly organised in a hierarchical of structure, sometimes referred to as the *Aspect directory*. The directory holds configuration settings of the whole plant (e.g., setup parameters of field devices, user access settings, alarming parameters). In Chapter 3 we further analyse the specifics of an ICS engineering application.

ICS event log System logs capture information about process activity. Depending on the size of the facility, an ICS records thousands of events per day. Such events describe system status updates, configuration changes, condition changes, user actions, etc.

ICS users actively use logs during operation. In particular, operators gather alarms triggered in real time. An alarm represents the event that is predefined, by experts, to be suspicious. For example, an alarm trigger is designed to go off when a specific field value reaches the threshold (e.g., tank level less than 100L). In a nutshell, alarms represent filtered and interpreted log information (since they are generated based on events that occur at the same time as log entries).

User activity in event logs Generally, a user action leaves a trace in the log in two ways: (1) as a direct action (e.g., the exact user action of performing a reconfiguration), (2) as a consequence (e.g., an consequence of a performed action or a sequence of actions-process script) or (3) no trace at all. The first type of trace implies a log entry that captures the time, the location and the user name of the person who performed the action.

The second type of trace implies an indirect action consequence or system response. Although caused by a user action, this trace typically does not consist of user name who performed the initial action nor the location of the failure source.

This is because the captured trace does not represent the source but the victim of the specific action that propagated [51, 79].

We now compare ICS environment to traditional IT systems.

2.2.6 Comparing ICS and IT systems

We compare ICS and IT with respect to two aspects: technological and operational. First, technologically, ICS nowadays generally resemble standard IT systems. This is because an ICS uses off the shelf operating systems and components (e.g., Windows OS). Historically, this was not the case as ICS facilities used to leverage proprietary software and specialised hardware. The change occurred with the wide adoption of low-cost Internet Protocols which increased the connectivity and access capabilities (e.g., corporate connectivity helps run business). ICS components have a lifetime of 15-20 years, which is significantly longer than the lifetime of the traditional IT components of 3 to 5 years. The practical differences are manifest in the set of communication protocols (e.g., ICS environments still use a set of domain-specific protocols like Profibus, Bacnet, Modbus) and the constraint of resources of ICS components (e.g., computational resources of PLC devices limit the application of security solutions).

Second, there are significant differences in various aspects of operation, namely: performance requirement, time-critical response, change management [105]. In a traditional IT system, the most important security risks refer to data confidentiality and integrity (e.g., prevent leaking or tampering the data). In an ICS, the main concern is data availability (e.g., a continuous ICS process cannot allow unexpected outages). An ICS is generally a time-critical system where process information needs to be treated without delays (e.g., late valve closure can cause equipment damage). In a traditional IT system, time is generally not an essential requirement (e.g., a delay in the information flow will not normally cause system failure).

Best security practices advise timely application of security patches and software updates. Such change management is generally not a problem in traditional IT. However, the updates in ICS need to be thoroughly tested by *(i)* the vendor of the ICS application (to ensure that the control application will not be hampered) and *(ii)* the end user (to ensure that the specific process will not be hampered). As a practical consequence, the application of updates takes more time than in traditional IT. Table 2.1 summarises the differences between ICS and traditional IT.

2.2.7 Cyber security in ICS

Historically, ICS long remained isolated from communication with other infrastructures, and Internet (i.e., operating as a communication island). Due to this isolation (and the natural difficulty of applying changes in ICS), practitioners in

Table 2.1: A summary of differences between IT and ICS

Aspect	**Category**	**Traditional IT**	**ICS**
Technology	Component lifetime	Lifetime on the order of 3-5 years.	Lifetime on the order of 15-20 years.
	System operation	Few operating systems.	Common and proprietary operating systems and software.
	Communication	Common communication protocols, few proprietary protocols.	Open and proprietary communication protocols over different types of media (e.g., wire, wireless and satellite).
	Resource constraint	Systems typically have enough resource power to support additional security solutions.	Systems often have constrained resources to support additional functionalities.
Operation	Security focus	The biggest focus is on the central server and the information stored there.	The biggest focus is on the process, thus edge devices (e.g., controllers) that are operating the process.
	Performance requirements	Most important requirements are information integrity and confidentiality. A high throughput is demanded.	Most important requirements are information/system availability. A modest throughput is acceptable.
	Time critical interaction	There is a low/medium requirement for timely interaction.	There is a high requirement for timely interaction.
	Change management	Software changes and updates are performed on a regular basis.	Software changes must be thoroughly tested before deployment. Because of this, slow system changes and updates occur.

this field often disregarded best practices of cyber security [105]. In addition, the design of ICS components and communication protocols is, even today, often legacy property. These conditions led to the common convention that ICS environments are operated in the "security by obscurity" manner.

The situation has changed in the last decade. On one hand, ICS environments have adopted corporate business connectivity and remote access capabilities to modernize their operation. For example, plants nowadays implement business enterprise networks which hold and communicate different types of information to external stakeholders (i.e., regulatory information to government authorities, trending data to management). On the other hand, the increased adoption of standard IT has revealed a number of cyber security issues. For example, various security assessments revealed vulnerabilities in software deployed in PLCs and smart meters [29, 86]. Also, there are several real life incidents that demonstrated the weaknesses of ICS cyber components [77, 100, 134, 136].

We distinguish two general strategies for improving the cyber security in ICS. The first strategy aims at a adapting best IT security practices in the ICS domain. For example, authors adjust common approaches for detecting intrusions to support ICS communication protocols [66, 67, 128], implement "'defence in depth" [9], incorporate encryption into network protocols [73], apply defensive deception behaviour in ICS [92].

The second strategy leverages the specifics of the ICS field to perform a more tailored monitoring of cyber activities. For example, fingerprint the details of ICS field controllers [81, 99], analyse field measurements to perform state estimation [23, 69], monitor system functionality from network protocol [45].

A field of computer security focusing on building techniques capable of detecting malicious activity in computer systems is called *intrusion detection*. We now present a brief overview of common approaches in intrusion detection that apply to both ICS and traditional IT environment.

2.3 Intrusion detection

The main task of intrusion detection is to monitor events occurring in the system and analyse them for possible incidents (i.e., violations of security and user policies).

Based on the type of the analysis, there are two general types of detection systems: misuse- and anomaly-based. A misuse-based system uses predefined patterns of behaviour to identify benign or malicious activities (e.g., a pattern of bytes in a specific network attack). An anomaly-based system first "learns" what is normal behaviour (e.g., by extracting the statistics of network communication).

Then, during the monitoring, the system looks for anomalies by comparing the models of normal behaviour to the current activities.

Based on the information resource, the detection systems are categorised into host- and network-based. A host-based system monitors the activities of a single computer. For example, a host-based monitoring includes the analysis of system logs, processes, application activities, file accesses, application configuration changes. A network-based system monitors network traffic (e.g., analysis of network flows, packets headers, packet content).

This thesis focuses on network-based approach. We now briefly present techniques for network analysis.

Network-based detection Based on the data source, there are two types of network-based detection techniques: *(i)* flow- and *(ii)* packet-based techniques. Flow-based techniques use aggregated connection information to detect attacks whose realisation causes effects on communication patterns. Packet-based techniques analyse each packet on the network to detect packet segments consisting of suspicious content. While a flow-based analysis can detect global shifts in the patterns of communication (e.g., DDoS attack), a packet-based analysis focuses on attacks whose malicious content can be hidden in only one packet, and thus invisible at the flow level (e.g., a buffer overflow attack). There are two types of packet-based detection techniques: *(i)* header-based and payload-based. Header-based approach analyses TCP/IP header information in the packet to detect the misuse of header parameters (e.g., in [71]). Payload-based techniques analyse data payload to capture segments carrying malicious content. This thesis explores the area of content-based network detectors. We now present the most common technique for content analysis in anomaly-based systems, $n-$gram.

2.3.1 $N-$gram analysis

$N-$gram analysis is a common technique for capturing features of data content. This technique is used in various areas, such as monitoring system calls [39], text analysis [34], packet payload analysis [114]. An $n-$gram is a contiguous sequence of n items (e.g., words, bytes) from a given sequence of system calls, text, network payload. In the context of network payload analysis, the current approaches use the concept of $n-$grams in different ways. In particular, we distinguish two aspects:

1. The way an $n-$gram builds feature space - The extracted $n-$grams can be used for building different feature spaces [37]: (a) count embedding (count the number of different $n-$grams to describe the payload), (b) frequency

embedding (use relative frequency of byte values of an $n-$gram to describe the payload, e.g. [15, 25, 115]) and (c) binary embedding (use the presence/absence of specific $n-$grams to describe the payload, e.g., [114]).

2. The accuracy of payload representation - $N-$grams can represent the payload in the following ways: (a) as an exact payload description ($n-$grams represent continuous sequences of bytes, e.g. in [25, 114, 115]) and (b) as an approximated payload description ($n-$grams represent a compression or a reduction of the exact payload, e.g., [48, 84]).

Also, various systems employ different architectures and combinations of approaches to analyze $n-$grams (e.g., Markov models in [15], Self-Organizing Maps in [25], hashing in [48]).

Detection evaluation There are four terms that are commonly used to evaluate the accuracy of detection approaches, namely: *true positive, true negative, false positive and false negative*. A *true positive* refers to the correct detection of a malicious activity. A *true negative* indicates a situation in which the detector correctly identified a benign situation as benign. The *false positive* indicates a situation that the system identifies a benign activity as a malicious activity. A *false negative* indicates to a situation when the system fails to identify a malicious activity [96]. Ideally, the number of *true positives* and *true negatives* will be equal to the total number of malicious and benign activities, respectively (indicating that the detector correctly classified all instances of malicious and benign activity). Also, the number of *false positives* and *false negatives* will be zero (indicating that no benign activities will be identified as malicious, and the detector will not miss to report any of the malicious activities). In reality this is not the case. The terms *false positive* and *false negative* are related to the specific type of the detection system. More specifically, misuse-based systems more often suffer from a higher number of *false negatives* (dues to the lack of signatures), while anomaly-based systems more often suffer from a high number of *false positives* (due to the inaccurate modelling of normal behaviour).

We have now explained all the main concepts needed to understand the remainder of the thesis. In the next Chapter we address the first research question, **RQ1**.

Chapter 3

On the Misuse of ICS Applications via User Activity

In this chapter we answer **RQ1** by analysing the threats that occur as a result of legitimate user activity on ICS application software. As described in Chapter 1, this is the first attack vector we consider for analysing potential process threats. This type of threat considers two scenarios: *(i)* a malicious use (e.g., an attacker uses the engineering application to change configuration settings) or *(ii)* accidental misuse, i.e., an operational mistake of users that have legitimate credentials in an ICS application. An example of an incident is the havoc caused by a disgruntled (former) employee in Australia who used valid application credentials to issue commands which overflew a park and residential area with sewage water [124]. Since this attack did not create behaviour of the ICS that significantly deviated from the normal behaviour (e.g., the attacker leveraged the same set of commands as legitimate system users), *the attack* remained undetected for two months.

A common approach for mitigating the misuse of user commands is a thorough analysis of user actions. A trace of user actions can be captured through different types of logs (e.g., security software, operating system, application logs [60]). To highlight interesting events from the log, and thus to detect an undesirable event, experts analyse logs using different techniques (e.g., pattern mining, correlation analysis [49]). The main challenge is the extraction of relevant events from a large number of log entries. The literature suggests that the success of the log mining approach depends on the context in which it is applied [80]. For example, episode mining (mining of distinctive sequences of log entries within a time window) of events in an IDS showed poor performance. In particular, due to the dynamic nature of timing patterns in the analysed events, the mining algorithm raised 99% of events as alarms which turned to be *false positives* [56]. This shows that applying a suitable techniques for the context of mining is important.

Problem Descriptive characteristics of user behaviour in ICS are, to the best of our knowledge, not well studied. This knowledge is necessary for building reliable models of user behaviour and identifying potential security threats. The lack of the knowledge is mainly due to the fact that the context relates to industrial processes, environments whose operational details are sensitive and specific to each different ICS instance.

In this chapter we present a combined methodology for identification and analysis of threats occurring as a result of user activities in ICS applications. The proposed approach adapts a common safety analysis methodology (Hazard and Operability Study–HAZOP) and uses two types of resources: *(i)* internal ICS documents and *(ii)* focus group sessions.

We use the internals of an ICS application to systematically identify possible user actions within the analysed system. More specifically, we leverage an internal structure of application software (often referred as the *Aspect Directory*) that holds information about objects and operations that can be performed by ICS users. The extracted user activities are further evaluated within the HAZOP study which requires feedback from domain experts. We do this through focus group sessions. A focus group consists of ICS process experts who are familiar with the analysed environment, process operation and potentially undesirable process consequences. During focus group sessions we identify and characterise classes of potential process-related threats.

Contribution The main contributions of this work are:

- we present a methodology to identify and analyse process threats occurring as a result of user activities in an ICS application,
- we perform a case study on a real-life environment to apply the proposed methodology and present the results.

We structure the remainder of the chapter as follows. Section §3.1 presents our methodology. In Section §3.2 we present the case study describing the applied methodology and the results obtained from a real life ICS environment. Section §3.3 presents related work and Section §3.4 presents conclusions.

3.1 Approach

We leverage knowledge of process experts to find potential threats in user actions within an ICS application. More specifically, we perform a systematic assessment of user actions through focus group sessions. A focus group consists of experts (process engineers and operators) who are familiar with the specific process configuration and operation. The task of the focus group is to analyse possible user actions within an ICS application and evaluate the severity of them. The sessions are organised as structured interviews where experts systematically discuss possible consequences and constraints of the identified user actions. The result is the assignment of a severity label with each user action. We now discuss the scope and the structure of the analysis.

Scope of the analysis Although we acknowledge that the analysis of sequences of user actions is important (e.g., an engineer issues a sequence of commands where each command is legitimate, but the overall effect is undesirable), our focus is on the analysis of individual user actions. This is because the understanding of sequence consequences requires a substantial understanding of the consequences of individual steps. Therefore, as a starting point, we constrain our analysis to single user actions. We argue that this constraint still captures actions that may represent undesirable behaviour. For example, a process engineer can change the process character (e.g., by changing the scale of the pump speed) in one single command.

Structure of the analysis We structure the analysis in two phases: *(i)* a systematic identification of possible user actions within the ICS application and *(ii)* an analysis of user actions. In the first step we analyse ICS application to systematically identify the set of activities that can be performed by a signed on user (e.g., a user can change process configuration settings or stop process operation). For this we use internal ICS documentation. In the second step we analyse the identified actions. We perform focus group sessions where experts discuss causes, consequences and mitigation strategies for different user actions.

We implement our approach as a case study on data coming from a real-world water treatment plant. The plant serves a metropolitan area and operates on two common vendor applications, creating us a representative case for the analysis. We now discuss both steps in more detail.

3.1.1 Identification of possible user actions

As the first phase of our approach we aim at systematically identifying possible user actions within an ICS application. We do this by following the idea of HAZOP (Hazard and Operability) study [41]. HAZOP methodology is a common approach for addressing process safety problems. The main goal of the study is an investigation and evaluation of possible process safety deviations. Here, each deviation is described by a *keyword* and a *guideword.* A HAZOP *keyword* describes the analysis context (e.g., for a water company: temperature, pressure, speed). A HAZOP *guideword* describes a change that may occur in the context (e.g., for pump speed: increase, decrease). Following this, a process deviation is defined as a combination of one process *keyword* and one process *guideword* (e.g., "speed increase").

For our approach, we translate HAZOP deviations to possible user actions. For us, a HAZOP *guideword* represents a *type of action* that a user can perform within an application. We use three generic types of user action as HAZOP guidewords: *add, modify and delete*. As HAZOP *keywords*, we use *objects* of the performed user action (e.g., tank, configuration panel). To systematically identify the *keywords* in the context (i.e., all process components that can be manipulated by a user), our approach uses internals of ICS application. More specifically, we leverage the *Aspect directory* to extract possible objects for user activity. The Aspect directory has a hierarchical structure holding clusters (subdirectories) of different process objects that can be manipulated by the users. For example, process objects under the path *plant_1/production/cleaning/..* represent devices from the cleaning area of the *plant_1*. To systematically enumerate all objects that can be manipulated by the user within an application, we transform the directory structure into a tree where higher branches represent higher subdirectories in the hierarchy (e.g., cleaning, ozon area) while lower branches represent process objects (e.g., field devices). We then enumerate the leaves of the tree to identify HAZOP keywords (e.g.,*plant1/functional mode/street1/../pumps/*pump3). To this end, by combining one HAZOP *keyword* and one HAZOP *guideword* we compile a set of possible user actions on the analysed ICS application.

3.1.2 Analysis of user actions

As the second step of our approach we analyse identified user actions through focus group sessions. We perform the analysis in two steps: *(i)* threat identification and *(ii)* threat mitigation.

First, to identify user actions that may represent a potential threat, we perform two sessions with focus groups. In the first round experts discard actions that are

(a) safety constrained (e.g., a deletion of a PLC in online mode), (b) not related to the process directly (e.g., modification of HMI presentation) or (c) make little sense for the environment (e.g. an item on the access list can only be added or deleted, but not modified). In the second round, for each of the remaining user actions, experts perform an in depth analysis of four aspects: severity, cause, effect, and mitigation recommendation. These four aspects represent common discussion topics across different safety methodologies such as Hierarchical Task Analysis (HTA), Predictive Human Error Analysis (PHEA), HAZOP [41]. For each user action, experts also discuss if the potential misuse of that action can be mitigated with the existing ICS application. We consider a specific action as a threat if *(i)* the action is evaluated as highly severe and *(ii)* there are no existing mitigation measures to address the misuse of this action. The result of this is an enumeration of potential process threats in the analysed ICS environment.

As the second step, we analyse possible threat mitigation strategies. Here we aim at understanding the constraints of the current application and deriving a promising strategy for the threat mitigation. We structure the discussion to address three main questions: *(i)* Why the current version of the ICS application cannot address the identified threats? *(ii)* What can be done to improve the resilience of the ICS application against the threats? and *(iii)* How can the improvements be deployed?

By answering the posed questions together with process experts, we gather the information required to understanding the characteristics of a promising threat mitigation strategy.

We now present a case study implementing the proposed approach.

3.2 Case study: Application misuse in a water treatment plant

To demonstrate our approach, we perform an assessment of a real-life ICS engineering application controlling a water treatment facility. We first present details on the setup of the case study and then discuss the results of both steps of the approach.

Environment and setup The plant serves a metropolitan area of around 700 000 residents and uses control application of a well known vendor. We perform the analysis during two types of sessions with process experts. First, we perform a round of interviews with operators, IT and process engineers to obtain the information about possible user actions within the application (i.e., extraction of the *Aspect directory*) and common plant operations. Secondly, we perform two rounds of focus group sessions to analyse identified user actions. The focus groups consist of two process engineers and one operator.

We now present the results of the two step approach applied on the case study: Identification of user actions (Section §3.2.1) and Analysis of user actions (Section §3.2.2).

3.2.1 Identification of possible user actions within the application

As explained earlier, we use HAZOP study as the basis of our analysis. Within this context, we use three distinctive type of user actions (add, modify, delete) as HAZOP *guidewords*. To compile possible user actions, we extract HAZOP *keywords* (i.e., *process objects*) from the plant *Aspect directory*. Figure §3.1 repre-

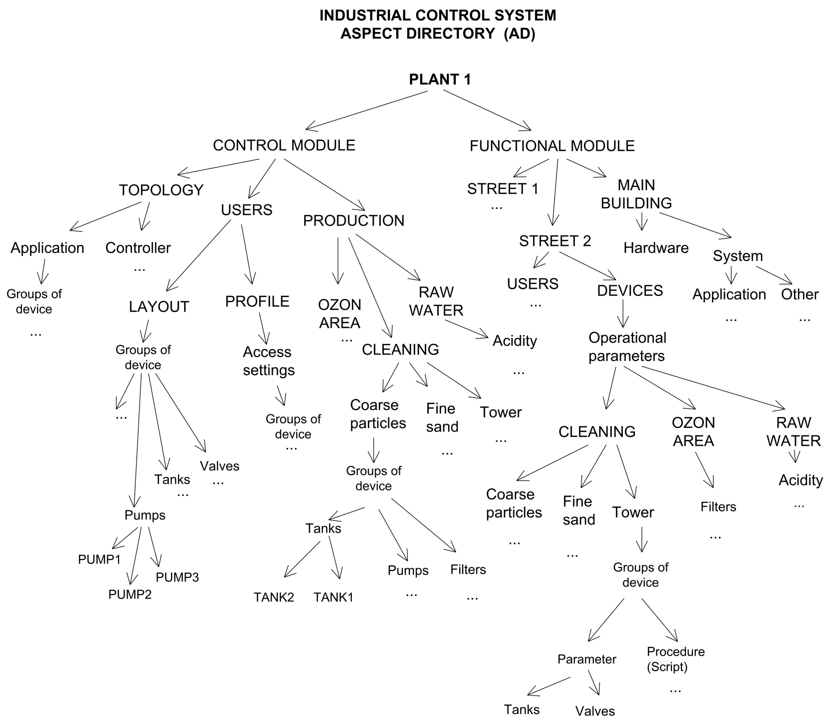


Figure 3.1: Hierarchical task tree of the target ICS application in a water facility

sents a part of the anonymised and simplified tree made from the custom *Aspect directory*.

The complete tree produces 170 different HAZOP *keywords* (i.e., the leaves of the tree). To simplify the analysis, we perform aggregation of the keywords. Here, instead of considering each specific keyword individually (e.g., *PUMP1, PUMP2, PUMP3*), we analyse a group of same keywords (*pumps*) on one branch. We perform the aggregation by taking the substring of the tree path with up to and excluding the leaves of the tree (e.g., plant1/functional mode/street1/../pumps/*pump3*). This way we semantically group together devices which are on the same location (or configuration paths) and represent similar devices (e.g., pumps). By doing this we decrease the total number of keywords from 170 down to 36. Following the HAZOP methodology, we generate user actions (i.e., deviations in HAZOP terminology) as all possible combinations of given *keywords* and *guidewords*. Table 3.1 shows examples of deviation generation. The first column of the table consists of all chosen keywords (paths of the HTA tree from Figure 3.1). The second column consists of three guidewords. Each keyword from the first column is combined with all three guidewords from the second column to build a deviation in the third column. The character of the specific deviation depends on the position of the keyword in the tree. Typically, one keyword can be found on several branches of the *Aspect directory* tree (e.g., *tanks* in Figure 3.1). A deviation built on the keyword *tanks* in the *Plant1/Functional mode/Street2/Devices/Operational parameters/../Groups of devices/tanks* branch of the tree implies actions on operational parameters of the device such as capacity, desirable tank level, etc. Therefore, the devised deviations are: a user modifies the capacity of tank, a user modifies the value of the desirable tank level. Deviations of the same keyword (*tanks*) from a different branch of the tree imply different types of actions. For example, the path to *tank* in the *Plant1/Control module/Users/Profile/Access settings/Groups of devices/Tanks* branch of the tree implies access settings that relate to tanks (e.g., modify user access rights).

We now analyse identified user actions.

3.2.2 Analysis of user actions within the application

In the first step of the approach we identified 108 possible user actions. We then presented these user actions to the focus groups and discussed them in two sessions to identify potential threats.

Threat identification During the first session, the focus groups analyse identified user actions and discarded actions that are considered as uninteresting. For

Table 3.1: Examples of deviation generation

Keyword	Guideword	Deviation / Potential threat
Valve (topology)[1]	add	a user **adds** a valve to the group
	modify	a user **modifies** the name of a valve
	delete	a user **deletes** a valve from the group
Tanks (operational parameters)[2]	add	no
	modify	a user **modifies** the capacity of tank (e.g., the presumed capacity of tank is increased by double)
	delete	no
Pumps (access settings)[3]	add	a user **adds** action type to the allowed actions (e.g., a user adds ”inserting setpoint” and/or ”changing pump status” to the list of allowed operator actions on a pump)
	modify	no
	delete	a user **deletes** action type from allowed actions (e.g., a user deletes ”inserting setpoint” and/or ”changing pump status” from operator actions on a pump)

[1] *Plant1/Control module/Topology/Groups of devices/Valves*

[2] *Plant1/Functional mode/Street2/Devices/Operational parameters/Groups of devices/Tanks*

[3] *Plant1/Control module/Production/Cleaning/Access settings/Groups of devices/Pumps*

example, depending on the context (i.e., the specific software control implementation), some combinations of keywords and guidewords do not apply (e.g., acidity parameters cannot be deleted) or are not considered as severe (e.g., add tank). To this end, the focus group selected 35 user actions that represent a potential threat. In Table §3.2 we present these actions. We note the reader that the table entries are, to avoid repetitions, aggregated over guidewords (add, modify, delete) and process components (e.g., we use higher tree levels instead of leaves to present action objects from Figure §3.1).

During the second session, focus group analysed the remaining user actions to identify severity, causes, effects, and mitigation strategies. Table 3.3 presents an example results of the detailed analysis of three user actions. The examples exhibit various process consequences (e.g., equipment damage, product quality, unreliable alarming).

After performing an in depth analysis of all selected user actions, the result is a list of actions that are *(i)* evaluated as severe by the focus group and *(ii)* cannot be addressed by the current ICS application. We refer to these actions as process-related threats.

We distinguish two main types of process threats: scripting errors and misconfiguration. Both types of threats typically originate from the activity of engineers. The threats exploiting a scripting error imply writing (and loading) faulty process automation scripts or leveraging scripts already developed by system engineers. A misconfiguration implies forcing the settings of unsafe configurations. Misconfiguration threats can have functional or control consequences. A functional misconfiguration implies the usage of settings that modify the character of process

Table 3.2: An aggregated overview of potential process threats

Process component	**Action**
Control_module/production/ozon_area/ozon_parameter/	modify
Control_module/production/raw_water/acidity/chemical_parameter/	modify
Control_module/production/cleaning/Coarse_particles/groups_of_devices/	modify
Control_module/production/cleaning/Fine_sand/groups_of_devices/	modify
Control_module/production/cleaning/Tower/groups_of_devices/	modify
Functional_module/Street1/devices/Op_param/cleaning/Tower/script/	modify, delete
Functional_module/Street1/devices/Op_param/cleaning/Fine_sand/script/	modify, delete
Functional_module/Street1/devices/Op_param/cleaning/ Coarse_particles/script/	modify, delete
Functional_module/Street2/devices/Op_param/cleaning/Tower/script/	modify, delete
Functional_module/Street2/devices/Op_param/cleaning/Fine_sand/script/	modify, delete
Functional_module/Street2/devices/Op_param/cleaning/ Coarse_particles/script/	modify, delete
Functional_module/Building/hardware/operation/PLC/	modify, delete
Functional_module/Building/hardware/operation/OPCserver/	modify, delete
Control_module/users/profile/access_settings/groups_of_devices/	add, modify, delete
Control_module/topology/application/groups_of_devices/	add, modify, delete
Functional_module/main_building/system/application/	modify

Table 3.3: Understanding mitigation strategies

Guideword	**MODIFY** - A user modifies the quantity value of chemicals (e.g., input 2).
Cause	Inside/outside malicious attack Human error
Effects	Errors in calculations Equipment damage Influence the product quality
Recommendations	Additional input checks Track user behaviour
Guideword	**DELETE** - A user deletes a device from the device topology (e.g., pump 13)
Cause	Inside/outside malicious attack Human error
Effects	Device becomes inaccessible Equipment damages due to overload Inconsistent alarms
Recommendations	Increase the number of access levels Track user behaviour Safety checks on engineer actions before execution
Guideword	**MODIFY** - A user modifies a tank capacity (e.g., tank 1).
Cause	Inside/outside malicious attack Human error
Effects	Tank damage due to overload No alarm when real maximum reached Damage of interdependent equipment
Recommendations	Include safety checks during manual working mode Additional configuration checks Track user behaviour

operation (e.g., change capacity of a tank to prevent the alarming system from going off and triggering the start of the next process stage). A control misconfiguration implies the usage of settings that corrupt the common plant operation in such a way that ICS users can no longer supervise and recover the process properly (e.g., modify access rights so a pump can no longer be stopped by an operator). In Table 3.4 we present examples of process threats in the system context.

Table 3.4: Examples of identified process-related threats

Type of threat	Threat	Example
Scripting error	A user loads a script that causes errors in the system automation	Insert a script that calculates wrong ratio of chemical components
Misconfiguration /functional	A user modifies a device parameter	Change capacity of a tank to prevent the alarming system from going off
Misconfiguration /functional	A user modifies auditing policy	Turn off all auditing
Misconfiguration /control	A user modifies the range of allowed actions for a specific device	Modify access rights so a pump can no longer be stopped by an operator
Misconfiguration /control	An user modifies the system topology	Delete devices from the toplogy so some devices become invisible, and thus inaccessible to operators

Threat mitigation After the performed analyses, we discuss possible methods for mitigating the identified threats. We structure the discussion with focus groups to answer three related questions:

- Can the current version of the ICS application address identified threats?
- What can be done to improve the resilience of the ICS application against the threats?
- How can the improvements be deployed?

First, we discuss can the current ICS application address identified threats. More specifically, we analyse why the targeted application cannot apply threat recommendations (Recommendations in Table 3.3). During the focus group session we identify two weaknesses of the ICS application: *(i)* lack of logic controls (e.g., no process safety checks are in place during the manual system mode, an input is not validated before executing an engineer command) and *(ii)* user auditing (e.g., no detail user behaviour analysis). According to the engineers, it is hard to have a comprehensive set of logic controls on all process inputs due to the complex character of the ICS environment. For example, the implementation of logic controls requires intensive manual work on defining the rules for each specific input parameter (which might change over time).

Also, daily user auditing is not performed due to heavy manual load of the auditing process. For example, the analysed plant generates around 8000 log entries every day. Without an advanced mechanisms for log transformation and manipulation, the manual inspection is infeasible in the current setup.

Second, we analyse suitable approaches for addressing the identified threats. A common approach implies *shadow operation monitoring* where different components of plant operation are monitored and verified to comply to a specified behaviour (i.e., analogous to network- and host- based intrusion detection/prevention systems). For this, the experts confirm that ICS environments indeed represent a promising ground for exploration. This is mainly because system (and user) behaviour in ICS is repetitive and mostly automated. For example, an ICS operates on a limited number of algorithms that repeat over time (e.g., each process phase is one algorithm). Also, user involvement is typically seldom and scheduled (i.e., changes in system configuration are often periodical). This implies that the extracted patterns of common system behaviour may reveal unexpected user actions. The challenge lies in building techniques that will include and interpret process semantics related to user actions.

Finally, we analyse possible approaches for implementing the proposed approaches for system monitoring. In general, there are two ways in addressing the identified system weaknesses: (1) by upgrading the proprietary ICS software to support additional functionalities (such as an additional input check in manual mode) and (2) by employing an independent tool to analyse data resources from ICS and detect suspicious behaviour from user workstations. In theory, an upgrade of the ICS software is more suitable for avoiding various compatibility issues. However, in practice, due to the proprietary character of ICS environments, the experts confirm that the employment of an independent tool is often a more straightforward strategy for implementing a security addition.

Following the results of the discussion with plant experts we pursue research in a related direction. In Chapter 4 we present an approach and an independent tool for mining ICS event logs.

3.3 Related work

There are different approaches for identifying and analysing safety threats in industrial control systems. Most common methodologies include FMEA (Failure Mode and Effects Analysis), FTA (Fault Tree Analysis), HAZOP (Hazard and Operability Study) [41]. The main goal of these methodologies is to identify potential scenarios that may produce negative effects on the process. For example, FTA and FMEA decompose possible process failures to identify their root causes.

PHEA (Predictive Human Error Analysis) analyses the effects of human errors on the underlying process. More specifically, PHEA uses a set of standard errors in human operation (e.g., action too late, too soon, too long, too short) to discuss the consequences of these actions on the process behaviour. In this work we take a more generic view on possible user actions by considering three general operations (i.e., add, modify, delete). Although we do not perform an in-depth analysis of parameter relations in the process (e.g., by considering timing conditions as in PHEA), the usage of generic operations allows us easier analysis of general effects on the system. Keeney et al. [59] study the characteristics of insider threats that occur across various sectors in critical infrastructures. Here, the authors primarily focus on identifying motivations and pre-attack behaviour, rather than identifying different types of activities that can be performed.

Traditional methodologies for addressing safety problems in process control systems focus on hazard analysis (i.e., a combination of factors in normal plant work that may lead to undesirable consequences) and do not consider security threats (i.e., threats that occur as a result of malicious activity on the system). By introducing a special set of guidewords, Winther et al. [117] show how HAZOP can be extended to identify security threats. Srivantakul et al. [104] combine HAZOP study with UML use case diagrams to identify potential misuse scenarios in computer systems. We take a similar approach and adapt HAZOP study to explore ICS application and analyse user (engineer) behaviour in an ICS environment.

3.4 Conclusion

In this chapter we present an approach for the analysis of potential threats as a result of user activity in an ICS software application. The approach leverages internal ICS application structure to identify user actions. We use the HAZOP methodology to analyse user actions and, together with the plant experts, identify actions that represent potential threats. By extending the existing method (HAZOP) our approach fits in best practice and is made accessible to practitioners.

There are two main assumptions for applying the presented approach: *(i)* an active involvement of process experts and *(ii)* the availability of ICS application structure. First, as in similar threat studies, this work requires the active involvement of process experts in the evaluation process (i.e., via two rounds of focus group sessions, each performing an in depth analysis of causes and action effects).

Second, the core ingredient for the performed analysis is the extraction of possible actions. This is done by transforming the ICS application structure into a hierarchical tree. The usage of the internal application structure is suitable as it provides a comprehensive overview of possible user actions on an ICS. We acknowledge, however, that the internal structure sometimes might not be available (e.g., due to proprietary vendor design). Also, users could be using two software applications at the same time, thus the extraction of all possible user actions, and their combinations, could be complex. In both cases, possible user actions would have to be enumerated manually, which would possibly be incomplete.

Due to constraints in the availability of process experts, we validate our approach on only one use case. The performed case study confirmed the applicability of the proposed approach on a real life environment and revealed a set of potential threats. We acknowledge that the analysis on multiple case studies would result in more complete findings. For example, we foresee that the results from other case studies would extend the list of possible misuses on a particular system. However, we argue that the three classes of threats identified in this work (functional and control misconfiguration, scripting error) are applicable to other environments as well.

Chapter 4

A Log Mining Approach for Monitoring User Activity in ICS

In Chapter 3 we investigated the problem of process threats as a result of user activities in ICS application software. We performed an analysis of possible user actions within an ICS engineering software and, together with process engineers, identified potentially undesirable user activities (e.g., commands resulting in process misconfigurations).

In this chapter we address **RQ2** by presenting an approach to automate the detection of the identified threats as a result of user activity in ICS application software. In Chapter 3 we describe the challenges of analysing single user commands and command sequences. This chapter focuses on the analysis of threats that are performed through single user commands. We consider two threat scenarios in which the attacks are performed by: *(i)* a signed-on user or *(ii)* on a known user workstation.

To detect undesirable user activities, we use ICS event logs as the main data source. ICS event logs represent a data trace that provides a complete view on the industrial process that is continuous over time and captures information about user activities, system changes in the field as well as system status updates [98]. By contrast to network traces and field sensor data (e.g., current temperature), event logs represent high-level data traces carrying interpreted process information (e.g., an event entry typically carries information *when/where/who/what* performed an action, while this information primarily needs to be retrieved from network traces and sensor updates). This is hard because it implies that low-level data has to be interpreted in high-level semantics. Our starting assumption is that ICS will, with their mostly automated character and a limited number of process procedures, display an overall regularity in log activity. We aim at using this regularity to extract uncommon process operation.

Problem Even an ICS system used in a small installation generates thousands of potentially alarming log entries per day. Thus the size (and high dimensionality) of logs make manual inspection practically infeasible.

This is a relevant and challenging problem to tackle. It is relevant because process threats affect the security and safety of critical infrastructures, which in turn could endanger human life. It is challenging because in the past the analysis of system logs has been applied to other security domains (e.g., in system logs, IDS logs) but struggled to deliver convincing detection results [56].

A common approach for detecting user activity in a system is by filtering log entries. In Chapter 2 we explain that user actions may leave a trace in the system in two ways: as a direct user action and as a consequence. In principle, a direct user activity could be detected by implementing the following filters:

- extract entries which include a signed on engineer,
- extract entries that are performed on critical workstations (e.g., main system server),
- extract entries that are performing an action on a critical path of the Aspect Directory (e.g., reconfiguration in access settings)

However, the extraction of action consequences is difficult. This is because the prediction of potential consequences of a performed action and the propagation of such consequences is not straightforward as it implies an in-depth analysis of process dependencies. This means that detecting direct user actions solely might not reveal the undesirable character of the user action. For example, a user might write an erroneous (or malicious) script that produces postponed faults in the system. The act of writing a script is not unusual (and is typically scheduled), thus the log trace of the action is legitimate. Therefore, a rule-based approach would either: (1) raise an alert and be cleared by the operators as legitimate or (2) not raise an alert at all. However, the script might produce indirect consequences (or faults) that are undesirable for the process work. Due to the fact that the enumeration of all possible faults that a script might trigger is typically infeasible, such fault would not be detected. Without understanding that the erroneous script has triggered the undesirable set of actions, operators cannot identify the problem.

We propose a semi-automated approach to aggregate and simplify user monitoring in ICS. Our approach uses a known algorithm to analyse ICS logs. We validate our findings with process experts. To the best of our knowledge, there are no publicly available datasets of logs capturing some sort of process attacks. Likewise, the data traces that we analysed did not consist of any known attacks. Because of this, this work represents an exploratory study to evaluate the feasibility and usability of the proposed analysis.

Contributions The main contributions of this chapter are the following:

- we propose an approach to detect process threats occurring through the ICS engineering software and build a tool to automate the analysis of ICS logs, which can be used to monitor the industrial process,
- we perform experiments to validate our approach using data from a real facility.

The rest of the chapter is organised as follows. Section §4.1 presents the approach. In Section §4.2 we present the architecture of the tool and benchmark the tool in Section §4.3. Section §4.4 presents related work. Finally, Section §4.5 presents an extended discussion on approach usability and future work. Section §4.6 concludes the Chapter.

4.1 Approach

We propose a new approach to detect process threats based on an automated way of processing ICS logs. Our goal is to identify the most interesting events from the logs, and thus allow operators to focus on a set of potentially suspicious events than can be inspected manually. To this end we built a tool called MELISSA (Mining Event Logs for Intrusion in ICS Systems).

The tool uses an approach that builds the model of common process behaviour. The approach is heuristic because, to evaluate the usefulness and promising directions of the analysis, we use the experience of process engineers.

We argue that the *content* of ICS logs seldom changes over time. This is because usually new devices are not frequently added (or removed), operators and engineers repeat a finite set of actions, the system is semi-automated, etc. Some events are highly frequent (e.g., one event repeated 1,115 times in 8 hours of plant work in the log). Due to these reasons, we believe that the pattern-based analysis of system behaviour is suitable for the ICS context.

The basic idea of our approach is that a frequent behaviour, over an extended period of time, is likely to be normal because event messages that reflect normal system activity are usually frequent [20, 74, 108]. Similar to Burns et al. [28], we found a large fraction of events that always appear with the same number of daily occurrences (e.g., timer-triggered event). Thus, a rare event, in a semi-automated and stabile environment as ICS, is more likely to be anomalous. For example, an engineer operating from a machine that is usually inactive outside the working hours is considered suspicious.

Scope of the analysis The ultimate goal of this research is to have an online analysis system. This would imply having the knowledge which describes the common behaviour in an ICS and capable of detecting uncommon activities. In this Chapter we describe the process of building this knowledge. Therefore, as an initial step, we use offline analysis to gather cumulative features describing daily ICS operation.

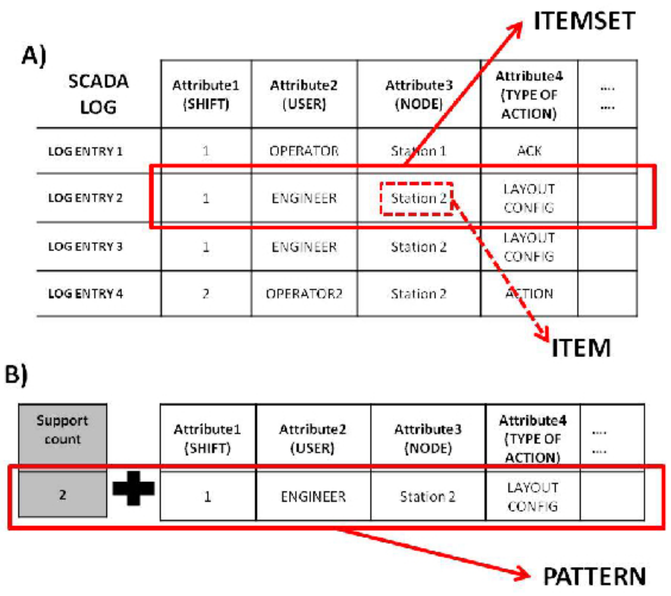
By including the context information (e.e., severity of process attributes), we limit the scope of our analysis to the ICS environment. This means that our approach cannot seamlessly be applied in a different environment. However, the tailored context information contributes to higher chances or providing meaningful results.

Structure of the analysis Our approach consists of two main phases: *(i)* mining of logs and *(ii)* manual inspection of uncommon events.

As the first step, we process logs to find uncommon activities. More specifically, we perform the following:

- Attribute subset selection
- Log analysis

Attribute subset selection identifies the subset of log attributes that carry relevant process information (Section §4.1.2). Log analysis performs transformation and mining of logs to extract uncommon activities. To do this we use a state-of the art algorithm for mining frequent patterns (described in Section §4.2.2.1) to identify the most and the least frequent (expected to be anomalous) patterns of system behaviours. In essence, the patterns are used to describe the cumulative characteristics of log entries. Figure 4.1 depicts the relation between a log entry, an itemset, an item and a pattern. Each unique log entry, with several attributes, represents a single itemset (Figure §4.1.A). A unique value of an attribute in the log entry represents one item. A support count is the number of log entries that contain the given itemset. Formally, if the support count of an itemset I exceeds a predefined minimum support count threshold, then I is a pattern [49]. In Figure §4.1.A, log entries 2 and 3 are the same, thus the corresponding pattern has the support count 2 (Figure §4.1.B).



SCADA LOG	Attribute1 (SHIFT)	Attribute2 (USER)	Attribute3 (NODE)	Attribute4 (TYPE OF ACTION)	....
LOG ENTRY 1	1	OPERATOR	Station 1	ACK	
LOG ENTRY 2	1	ENGINEER	Station 2	LAYOUT CONFIG	
LOG ENTRY 3	1	ENGINEER	Station 2	LAYOUT CONFIG	
LOG ENTRY 4	2	OPERATOR2	Station 2	ACTION	

Support count	Attribute1 (SHIFT)	Attribute2 (USER)	Attribute3 (NODE)	Attribute4 (TYPE OF ACTION)	....
2	1	ENGINEER	Station 2	LAYOUT CONFIG	

Figure 4.1: Log translation: A) mapping log entries into itemsets and items, B) mapping itemsets into patterns

As the second phase, we perform a manual inspection of extracted patterns. We do this by leveraging the knowledge of process engineers to manually evaluate the character of the extracted patterns. Similar to focus group sessions described in Chapter 3, process engineers represent experts who are aware of the semantic implications of specific actions, but typically cannot provide information on automatic extraction of log entries (since they are not experts in data analysis).

As an additional phase of our heuristic approach, we explore the usefulness of patterns once the process knowledge is included in the mining process (i.e., inclusion of attribute severity in Section §4.3.3).

We now present more details on the analysed data, the ICS logs.

4.1.1 Input data for analysis

The initial, raw, dataset consists of 11 attributes. The given attributes can be grouped in four semantic groups:

- time (*Timestamp*),
- type of action (*Type of action, Aspect of action*),

- action details (*Message description, Start value, End value*),
- user (*Username, User full name*),
- location (*Object path, Source, ICS node*)

Often the raw dataset consists of features that are redundant, irrelevant or can even misguide mining results. This is why we need to perform data preprocessing, analyse the current feature set and select a suitable subset of attributes.

4.1.2 Attribute subset selection

The main purpose of the attribute selection is to identify the most relevant attributes in a multidimensional data which will contribute to the mining process (and prevent from having distorted and biased conclusions). Common approaches for attribute selection exploit class labels to estimate information gain of specific attribute (e.g., induction of decision tree [49]). Unfortunately, our input data does not consist of class labels (i.e., labels for normal and undesirable behaviour), thus we cannot perform the "traditional" attribute evaluation. However, some approaches may evaluate attributes independently. For example, principal component analysis (PCA) [49] searches for *k* n-dimensional orthogonal vectors that can be used to represent the data. The original data is thus projected into a much smaller space and represented through principal components. The principal components are sorted in the order of decreasing "significance". Finally, the dimensionality reduction is performed by discarding weaker components, thus those with low variance. By performing the PCA on our data, we discard two low variance attributes (*Start Value, End Value*) since they only had few distinct values in the whole dataset. Also, we identify two redundant attributes (*Username* and *User full name*). Thus we discard one of them.

As expected, the attribute *Timestamp* showed the highest variance. We aggregate this attribute in three working shifts. We describe the details of this aggregation in Section 4.2.3.

Now we try to understand the behaviour of the remaining attributes.

Due to the fact that the highly variable attributes can produce overfitting [65, 93], we try to lower the number of distinct values in the most variable attributes (in our context, the ones over 150 distinct values- *Object path, Source, Message description*). The attribute *Object path* represents structured text. In Section 4.2.3 we describe the details of generalising the values of this attribute.

The attribute *Source* represents an ID of the field or network device and consists of around 350 different values. This attribute is highly variable, but does not contribute to the data mining process due to fact that it uniquely identifies a device.

For example, a creditcard number almost uniquely identifies a customer and thus does not represent a useful attribute to generalize customers behaviour thus to be used in the data mining. To tackle this, Lim et al. [65] and Oliner et al.[80] perform de-parameterisation of data by replacing IP addresses, memory locations and digits by tokens. We decide to omit *Source* attribute from the analysis since we observe that attribute *Object path* holds aggregated clusters of entries from *Source* attribute.

The attribute *Message description* represents unstructured text and consists of 280 values. We perform an in-depth analysis of values to determine means of aggregation. We conclude that a portion of values represents redundant data to other attributes (e.g., information in *Message description: "Action A on source B is acknowledged by C"* is repeated already in the same entry by the attributes *Type of action: A, Source: B, User: C*). The rest of messages are presented in an inconsistent way and provide information which, at this moment, we cannot parse and aggregate in a meaningful way. An alternative approach would be unsupervised clustering of messages, such as in [120]. We point out that such clustering does not guarantee semantic similarity of messages. Due to the redundancy in data we argue that the remaining attributes can compensate the information loss of discarding the attribute "Message description". In addition, we are sure that such highly variable attribute in the current format does not contribute to the data generalisation. Thus we do not consider this attribute during the analysis.

Our final set consists of 6 nominal attributes: *Working shift, Aspect of action, Type of action, Object path, User account* and *ICS node*. Some attributes are not applicable for all entries. As a result, every entry uses between 3 and 6 attributes. A *ICS node* represents a computer that sends event details to the log. In our case, there are 8 different nodes. All nodes in the network have a dedicated and predefined role that typically does not change (e.g., there are 2 engineering workstations, 4 operator workstations and 2 connectivity servers). The attribute *Type of action* takes one out of 12 nominal values. This attribute describes the general type of action, such as: operator action, configuration change, process simple event, network message, etc. For types of action which are performed by users, the attribute *Aspect of action* is applicable. This attribute takes one out of 6 nominal values in the log and details the character of the user action, such as: change of workplace layout, change in workplace profile, etc. The attribute *Object path* provides information about the location of the device which is the object of the performed action (e.g., plant1/control module/users/profile/access settings/groups of devices/tanks). The attribute *User account* represents the username of the signed-on user. Table 4.1 represents a sample of the analysed log.

Some events in the log are more severe than others. The severity of a ICS event depends on the combination of attribute values. Thus, a correct evaluation of

Table 4.1: Example of a simplified ICS log

Work shift	Aspect of action	Type of action	Object path	User account	Node
2	-	Process Simple Event	-	-	CS01
1	-	Configuration change	Plant1 / ../ tanks	Operator2	OP03
2	Layout change	Configuration change	Plant1 /../ layout	Engineer1	EN01
2	Function change	Operator action	Plant1/../ Op. param.	Operator 1	OP04

specific attribute values can help to detect events that are undesirable for the normal process flow. For example, the value *AuditEventAcknowledge* of the attribute *Type of action* is semantically less important than the value *AspectDirectory*. This is because the first value implies an action where an operator acknowledged an alarm while the latter value implies that a new action was performed on the main configuration directory. Leveraging the expert knowledge about the process and the semantics of nominal attribute values can help to distinguish critical and non critical events in the complete log. In Section 4.3.3 we describe how we use this knowledge to improve our detection results.

4.1.3 On the dataset validation

The process experts argue that, at the time of logging, there were no known security incidents. We investigate the ways of validating this claim. We argue that due to size and high dimensionality of the log, manual inspection is infeasible. Thus a (semi)automated approach is required. Typically, common log analysis tools imply the usage of predefined rulesets which filter events out of logs. For example, in [91], various rulesets for analysing logs, such as syslog and ssh log, are maintained. Unfortunately, such ruleset for analyzing ICS system logs does not exist. Thus we cannot perform a reliable log analysis to establish the ground truth.

An alternative approach for establishing the ground truth would imply the log capture in a controlled environment. In reality, this means either (1) performing the log capture in a lab setup or (2) performing the log capture in a constrained real environment (e.g., by reducing the number of process components to the ones that are validated to be correctly working). We argue that neither of the cases can compare to the actual real data.

We point out that, lacking the notion of the ground truth, we cannot perform an extensive discussion on the percentage of events that are undesirable but have not been alerted by our approach (i.e., false negatives). We are aware of this shortcoming in our approach. Nevertheless, we note that the primary goal of our approach is to help operators uncover security-related events from real data which would be overlooked otherwise. We validate this by presenting the mining results to a group of process experts.

4.2 Architecture

MELISSA consists of two interacting components: the Data Preparator (DP) and the Pattern Engine (PE). Figure 4.2 depicts MELISSA and its internal components.

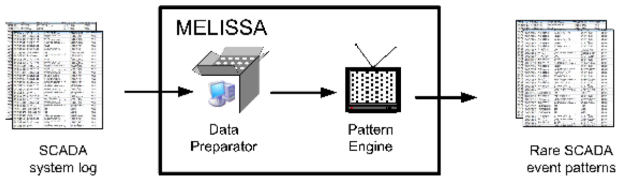


Figure 4.2: MELISSA architecture

4.2.1 Data Preparator

We perform data aggregation (e.g., variance reduction) and transformation (e.g., value coding) on the dataset to get a suitable data format for pattern mining. We describe performed operation in Section 4.2.3.

4.2.2 Pattern Engine

The PE runs the algorithm for mining frequent patterns over log and outputs an ordered list of patterns based on the frequency of the occurrence.

We now explain how we selected the specific implementation of the pattern mining algorithm. Patterns can be mined for different purposes. Various algorithms, depending on the purpose of mining, deliver itemsets with different features (e.g., complete, closed, maximal).

To select the most suitable algorithm for mining frequent patterns in our context, we identified a list of required features. The requirements are as follows:

- maximal pattern mining,
- scalability,
- selection of interesting events based on the absolute support count.

Maximal pattern mining An itemset can be frequent but not (necessarily) interesting and useful for stakeholders in a specific context. Mining large frequent itemsets often generates a large number of itemsets satisfying the minimum threshold. This is because, if an itemset is frequent, each of its subsets is frequent as well. For example, for an itemset of length 70, such as $\{a_1, a_2, ..., a_{70}\}$, there would be $\binom{70}{1}$=70 1-itemsets: a_1, a_2,..., a_{70}, $\binom{70}{2}$ 2-itemsets: (a_1, a_2), (a_1, a_3),..., (a_{69}, a_{70}), and so on. The total number of mined itemsets, for a data set consisting of 70 items is $2^{70} - 1$. This value is too big to be stored and used for manual inspection.

There are various strategies to extract a useful subset of itemsets from the complete set. For our context, our process engineers agreed that no subset of attributes carries enough semantics to distinguish between anomalous and normal events. For example, it is not sufficient to describe an event with only two attributes (e.g., itemset attributes {*Type of action, User account*}; itemset instance {Operator action, Operator 2}). Therefore, we set a requirement that the algorithm should deliver output patterns which consist of as many attributes as possible (take the superset itemset that satisfies the minimum threshold). This type of mining is in data mining terminology referred as mining maximal patterns [49]. Formally, an itemset X is a maximal frequent itemset in set S if X is frequent and there exists no super-itemset Y such that $X \subset Y$ and Y is frequent in S [49].

Scalability For the cases when the same plant setup is running for years, we might want to run the tool continuously, and receive events as they occur. Thus, the tool needs to scale well when processing logs that may consist of years of plant work. The tool can then leverage the knowledge of past behaviours to update the top patterns and detect anomalies.

Also, the speed of processing is important as operators must take immediate action in case of an alarm. There are two main types of mining algorithms: 1) algorithms that use candidate generation [12] and 2) algorithms that do not use candidate generation (FP-growth algorithms) [49].

For mining a k-size itemset, an algorithm that uses candidate generation may need up to 2^k scans of the dataset. By contrast, an algorithm that does not use candidate generation typically requires only two scans of the dataset to mine itemsets of arbitrary size. These algorithms are based on a recursive tree structure and are referred in the literature as the FP-growth methods. During the data preparation, we already scan the whole dataset several times. We expect our log size to grow up to several million entries in a year (e.g., around 2,500,000 entries correspond to the stakeholder's annual system logs). Also, benchmark results on a Frequent Itemset Mining Implementation (FIMI) workshop [44] show that the FP-growth methods scale better for most datasets. Therefore, we choose to use an FP-growth algorithm to comply with the scalability and speed requirements.

There are various implementations of the FP-growth method [27, 46]. These algorithms implement different structures to improve algorithm performances (e.g., Grahne [46] use array structures, Burdick [27] use a bitmap compression schema). We acknowledge one general problem of the FP-growth methods. These algorithms may scale bad with respect to memory consumption for small values of minimum support count (i.e., the threshold for the frequency of total occurrences). This is because a small value for the minimum support count, depending on the dataset character, may produce a large number of unique itemsets that each need a separate tree branch. This results in a complex FP-tree building and mining. However, with respect to our context (a limited number of items to mine: number of users, system nodes and a low number of different operations), we believe that there will not be a significant growth in the total number of items in our logs. Thus we expect the memory consumption to remain in ranges of our initial experiments when scaling up to millions of entries.

Selection of interesting events based on absolute support count To distinguish between interesting and uninteresting itemsets, algorithms use the concept of "cut off" parameter. For example, some algorithms use an absolute minimum support count (e.g., consider an itemset frequent if it appears at least 5 times in the dataset) while others use a relative minimum support (e.g., consider an itemset frequent if it appears in at least 10% of total dataset entries). Some algorithms use top k ranking of patterns (e.g., consider frequent if an itemset is in top 5 ranked patterns, and satisfies absolute minimum support). In our context, the output produced by the algorithm is then inspected by security operators. This implies that the number of extracted patterns directly influences usability. Thus we believe that an absolute support count with ranking suits our context better than a relative support count. We determine the final "cut off" parameter with process experts (discussed in Section 4.3.1).

An algorithm that meets most of our requirements is the FP-growth algorithm by Grahne et al. [46].

In the next section, we describe the general concept of the FP-growth algorithm.

4.2.2.1 FP-growth algorithm

The first dataset scan in an FP-growth method finds and counts all individual items in the dataset (Figure 4.3.A). The items found are inserted into the header table in decreasing order of their count (Figure 4.3.B). In the second scan, dataset entries are read and inserted in the FP-tree as branches, where items represent tree nodes. If an itemset shares some of the items with a branch previously inserted in the tree, then this part of tree will be shared between entries. Every tree node holds a count which represents the number of entries where the item occurs (with considering preceding items).

After the second dataset scan, all entries are inserted in the FP-tree. The header table holds links to tree nodes for each item. For every item in the header table a conditional pattern base and a conditional tree are built. The conditional pattern base represents a list of tree paths that a given item (e.g., item F) appears in. This represents a new dataset restricted for item F (Figure 4.3.C). The main algorithm is now repeated on the restricted dataset. As a result, a new tree of paths is built (Figure 4.3.D). The branches of the tree that satisfy minimum support count represent frequent patterns (Figure 4.3.E).

4.2.3 Implementation

We have implemented a prototype of MELISSA using Java. Data aggregation operations gather and summarize data for easier analysis. We transform the *Timestamp* attribute to represent usual working shifts in the company. In this way we aggregate a timeseries attribute into a 3-value discrete format that is more suitable for mining workload patterns. In our case, “Working shift 1” covers all events occurring between 00:00 and 08:59hrs. “Working shift 2” includes events occurring between 09:00 and 16:59hrs. “Working shift 3” includes events occurring between 17:00 and 23:59hrs.

The structured text describing the attribute *Object path* is presented as a directory path (e.g., plant1/functional mode/street1/pumps/*pump3*). As described in Chapter 3, we aggregate this attribute to 36 unique values by grouping similar objects together (i.e., paths that differ only in the lowest level of the directory hierarchy).

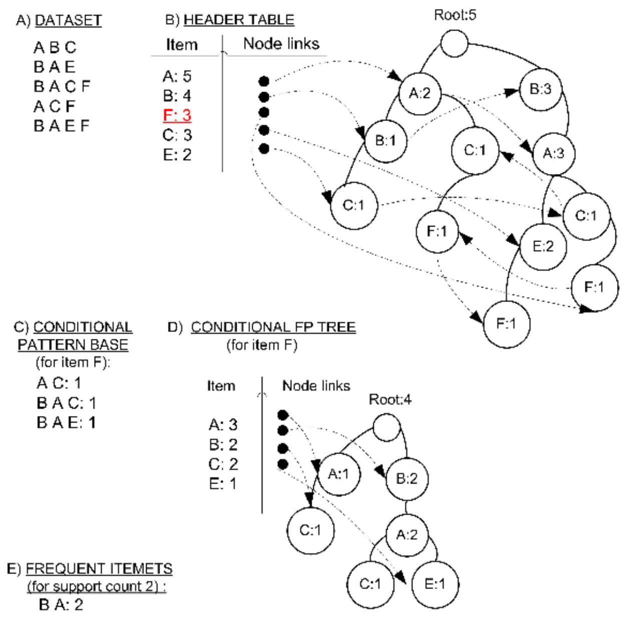


Figure 4.3: FP growth algorithm:A) full dataset, B) building FP tree from the dataset, C) extracted itemsets that preceed item F, D) building recursive tree from the conditional base, E) extracted itemsets that satisfy minimum support count

Finally, for all nominal attributes in the dataset we code distinct values as our algorithm only accepts numerical values.

In the Pattern Engine we use an algorithm for mining maximal frequent patterns proposed in [46].

4.3 Benchmarks

To evaluate the effectiveness of our approach we collected a dataset of logs generated by the ICS system of the stakeholders, which processes waste, surface and drinking water. The 101,025 log entries were collected during a 14 day period, and each log entry consists of at most 12 attributes. The logs were captured with the default audit set up of the ICS system that collects events continuously through time. In Section §4.1.2 we present attribute selection process. As a result, we use the subset of 6 log attributes that consist of 69 unique values (i.e., items). Since we aim at identifying the least frequent patterns, our minimal support count is

1. This means that each unique event which occurred at least once represents a pattern.

4.3.1 Testing MELISSA

As a proof of concept, we run our analysis in ofline mode. This means that a user runs a “day after” analysis. For example, each day the user receives up to 20 least frequent events from the day before (normally, in the stakeholder’s facility under analysis, a user gets approximately 7,000 unclassified events per day, so a reduction to 20 is significant). We decide to run the analysis offline because:

- we were provided with only two weeks of system logs;
- we cannot claim that these two weeks represent a complete set of behaviours that occur in the facility through a year;
- water-related systems are considered as slow processes (the consequences of actions are delayed - e.g., it takes several hours to overload a tank even while pumping at maximum speed), thus we can afford to run the analysis with a delay.

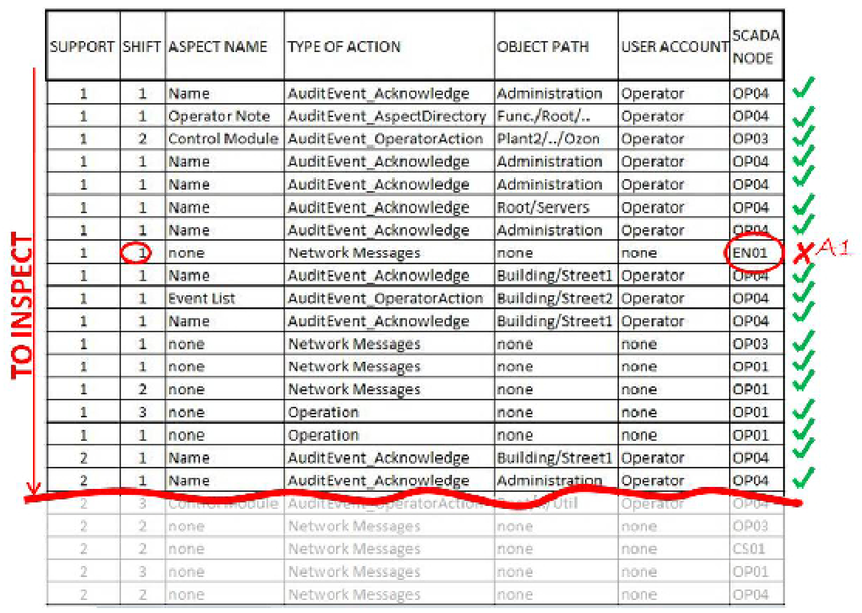
This approach can detect silent mimicry attacks as operators have a daily overview of events and can spot unusually infrequent user actions spread over several days (e.g., unplanned configuration changes [18]).

4.3.2 Preliminary results

We first summarize the results of daily inspections.

MELISSA found 486 unique patterns from the 14 days long ICS log. The number of unique patterns per day varies from 12 to 79. Also, the support count per day per pattern varies from 1 to 1,151.

According to process experts, an acceptable level of usability is that they receive up to 20 events per day for manual inspection, with the exception that all events with a support count of 1 should be reported. We use these requirements to set the threshold for extracting the most interesting events. After applying the threshold on the whole dataset, approximately 198 events (represented in 131 patterns) are labeled for inspection. During the daily inspections, the stakeholders label 20 patterns as suspicious. After having collected additional information about the context of the events, the stakeholders finally label 1 pattern as anomalous.



SUPPORT	SHIFT	ASPECT NAME	TYPE OF ACTION	OBJECT PATH	USER ACCOUNT	SCADA NODE	
1	1	Name	AuditEvent_Acknowledge	Administration	Operator	OP04	✓
1	1	Operator Note	AuditEvent_AspectDirectory	Func./Root/..	Operator	OP04	✓
1	2	Control Module	AuditEvent_OperatorAction	Plant2/../Ozon	Operator	OP03	✓
1	1	Name	AuditEvent_Acknowledge	Administration	Operator	OP04	✓
1	1	Name	AuditEvent_Acknowledge	Administration	Operator	OP04	✓
1	1	Name	AuditEvent_Acknowledge	Root/Servers	Operator	OP04	✓
1	1	Name	AuditEvent_Acknowledge	Administration	Operator	OP04	✓
1	1	none	Network Messages	none	none	EN01	✗ A1
1	1	Name	AuditEvent_Acknowledge	Building/Street1	Operator	OP04	✓
1	1	Event List	AuditEvent_OperatorAction	Building/Street2	Operator	OP04	✓
1	1	Name	AuditEvent_Acknowledge	Building/Street1	Operator	OP04	✓
1	1	none	Network Messages	none	none	OP03	✓
1	1	none	Network Messages	none	none	OP01	✓
1	2	none	Network Messages	none	none	OP01	✓
1	3	none	Operation	none	none	OP01	✓
1	1	none	Operation	none	none	OP01	✓
2	1	Name	AuditEvent_Acknowledge	Building/Street1	Operator	OP04	✓
2	1	Name	AuditEvent_Acknowledge	Administration	Operator	OP04	✓
2	3	Control Module	AuditEvent_OperatorAction	[illegible]/Util	Operator	OP04	
2	2	none	Network Messages	none	none	OP03	
2	2	none	Network Messages	none	none	CS01	
2	3	none	Network Messages	none	none	OP01	
2	2	none	Network Messages	none	none	OP04	

Figure 4.4: MELISSA testing, day 4 (before introducing semantic knowledge)

We now describe the context of the pattern which was labelled as anomalous. Figure 4.4 represents the summary of results of the pattern analysis from this day. The table consists of 7 columns. The first column represents the pattern support count. The remaining columns represent the attributes used in the analysis. The wavy horizontal line represents the border between interesting and uninteresting patterns as decided by the process experts (maximum 20 events per day). On the righthand side of the table, the stakeholders labelled each pattern as either normal or anomalous (e.g., A1). For the anomalous pattern, circles imply why the pattern is unusual.

Anomalous pattern *A1* occurred only once (support count is 1). Node EN01 represents an engineering workstation. Shift 1 represents the night shift. For the experts, *A1* is anomalous because engineers are expected to work only during day shifts. While inspecting the complete log we found that, except this event, all activities performed by engineers or on engineering workstations did occur during day shifts only. After a thorough internal inspection, the stakeholders found a software emulator with a faulty automation script that remotely attempted to connect to the EN01 engineering workstation. We classify this event as an example of scripting threat (described in Chapter 3), and thus an operational mistake of an engineer which could have effect on system performance, as other actions could depend on it.

4.3.3 Inclusion of process context

During the preliminary analysis of results we note a shortcoming of our approach. Currently, we assume that all events equally impact the process. In reality, this is not true. When using the threshold of 20 events per day, our stakeholders acknowledged that in several cases some uncritical events were within the threshold while some severe (and suspicious) events were omitted due to the restriction on the number of selected events per day. Therefore, we decide to include the process knowledge to our algorithm and thus improve the quality of results. We do this by implementing a loose ordering on the algorithm output. The order is based on the process severity of specific events. By applying the new order we perform a fine-grained tuning of results so that the semantically more severe patterns appear within the usability threshold while less severe patterns (although with a low frequency of occurrence) tend to appear lower on the output list (and thus appear to be less interesting). The loose order is defined by evaluating the semantic meaning of values of one log attribute.

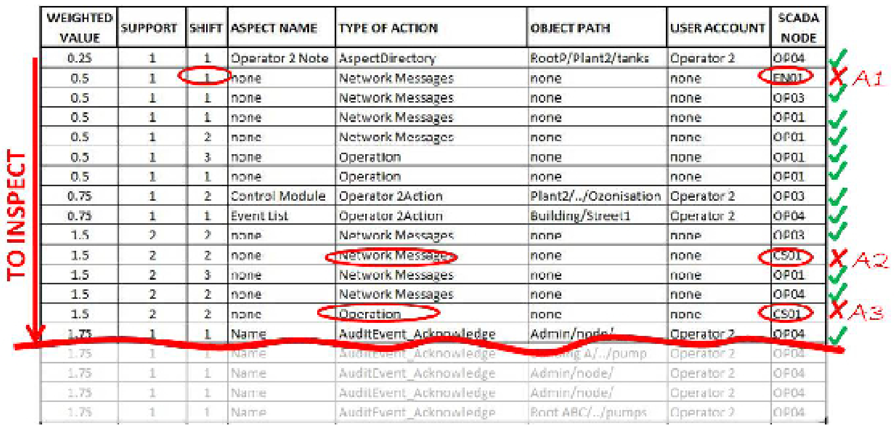
To choose the suitable attribute, we perform a semi-automated preselection of attributes. As we mentioned earlier, not all attributes are used in all entries. Thus we assume that only attributes used in the same entries as user actions (i.e., performed on user workstations) are semantically important for detecting threats identified in Chapter3.

The preselected attributes are: *Type of action, ICS node* and *Aspect of action*. We asked the stakeholders to compile the ranking of the values for each attribute. For example, for the attribute *Type of action* the stakeholders compiled the severity ranking list:

1. *AspectDirectory*,
2. *Network message*,
3. *Operation*,
4. *Operator action*,
5. *AuditEvent Acknowledge*.

Here, the order of the values implies how severe that specific action for the process is. For example, an action which includes *AspectDirectory* is more severe than *AuditEvent Acknowledge* as explained in Section 4.1.1). We run several experiments to generate different PE outputs using the selected attributes and the compiled lists. For each list we add weights to the attribute values. For example, we add a negative weight to severe actions (to increase the chances that the action is closer to the top of the pattern list) and a positive weight for noncritical actions (to decrease the chances that the action is close to the top of the pattern list). We then submit the results to the stakeholders. The stakeholders selected the attribute *Type of action* as the one whose ranking performed the most useful results within the extracted patterns. Thus we perform the final fine-grained re-ordering based on the severity weights of this attribute.

Finally, we use the sum of support value and the severity weight of each pattern entry to determine the final weighted value which is used for the final ranking of patterns.



WEIGHTED VALUE	SUPPORT	SHIFT	ASPECT NAME	TYPE OF ACTION	OBJECT PATH	USER ACCOUNT	SCADA NODE	
0.25	1	1	Operator 2 Note	AspectDirectory	RootP/Plant2/tanks	Operator 2	OP04	✓
0.5	1	1	none	Network Messages	none	none	EN01	✗ A1
0.5	1	1	none	Network Messages	none	none	OP03	✓
0.5	1	1	none	Network Messages	none	none	OP01	✓
0.5	1	2	none	Network Messages	none	none	OP01	✓
0.5	1	3	none	Operation	none	none	OP01	✓
0.5	1	1	none	Operation	none	none	OP01	✓
0.75	1	2	Control Module	Operator 2Action	Plant2/../Ozonisation	Operator 2	OP03	✓
0.75	1	1	Event List	Operator 2Action	Building/Street1	Operator 2	OP04	✓
1.5	2	2	none	Network Messages	none	none	OP03	✓
1.5	2	2	none	Network Messages	none	none	CS01	✗ A2
1.5	2	3	none	Network Messages	none	none	OP01	✓
1.5	2	2	none	Network Messages	none	none	OP04	✓
1.5	2	2	none	Operation	none	none	CS01	✗ A3
1.75	1	1	Name	AuditEvent_Acknowledge	Admin/node/	Operator 2	OP04	✓
1.75	1	1	Name	AuditEvent_Acknowledge	[illegible]g A/../pump	Operator 2	OP04	
1.75	1	1	Name	AuditEvent_Acknowledge	Admin/node/	Operator 2	OP04	
1.75	1	1	Name	AuditEvent_Acknowledge	Admin/node/	Operator 2	OP04	
1.75	1	1	Name	AuditEvent_Acknowledge	Root ABC/../pumps	Operator 2	OP04	

Figure 4.5: MELISSA testing, day 4 (after introducing process knowledge)

4.3.4 Final results

After providing the tuned results, the process experts labelled two more patterns as anomalous. These patterns also appear on day 4. Figure 4.5 shows the new ordering after introducing the process knowledge.

We now explain the context of the anomalous patterns. Anomalous patterns *A2* and *A3* occur twice. Node CS01 represents the primary Connectivity Server. *Network message* item typically reports problems in the network communications. *Operation* item reports system responses to a user action (such as input expression error messages, condition-triggered procedures). The stakeholders evaluate patterns *A2* and *A3* as anomalous because these patterns reflect network and operational errors on the main connection backbone node (CS01). After a thorough internal inspection, the stakeholders found out that all events from these two patterns occurred in the same minute of day 4. User Engineer 1 was logged in on CS01 during the time these errors happened. The stakeholders assume that Engineer 1 inserted a value which triggered an overflow in a device cache, which in turn generated an error report from the system. We verify that this is the only case, over two weeks of operations, that error messages were triggered on Connectivity Servers. The experts classify these patterns as misconfiguration threats where the user triggered cache overflows by inserting unexpected values. We note that this kind of error (e.g., an error reporting the input value is out of range) could be an indication of a masquerade attack. For example, an attacker with valid credentials would possibly be unaware of the working ranges for specific devices. Thus he might insert a value that would trigger a cache overflow which would be logged.

In summary, by applying our tool, the stakeholders detected and acknowledged three unexpected events. All detected events relate to an undesirable engineer operation on the system.

4.4 Related work

To detect anomalous behaviour in ICS systems, authors use approaches based on inspecting network traffic [18], validating protocol specifications [21], and analysing data readings [69]. Process-related attacks typically cannot be detected by observing network traffic or protocol specifications in the system. We argue that to detect such attacks one needs to analyse data passing through the system [18, 23] and include a semantic understanding of user actions. Bigham et al. [23] use periodical snapshots of power load readings in a power grid system to detect if a specific load snapshot significantly varies from expected proportions. This approach is efficient because it reflects the situation in the process in a case of an attack. However, data readings (such as power loads) give a low-level view on the process and do not provide user traceability data.

Salfner et al. [94] discuss the difficulties in processing logs with unstructured format. Lim et al. [65] present an approach for failure prediction in an enterprise telephony system. Authors propose to use context knowledge for efficient process visualization and failure prediction.

Several researches explore pattern mining of various logs for security purposes (e.g., alarm logs in [56, 74], system calls in [64], event logs in [50]). These authors use pattern mining on burst of alarms to build episode rules. However, pattern mining can sometimes produce irrelevant and redundant patterns, as shown in

Table 4.2: MELISSA results: size of input logs, size of tool output, number of first and second stakeholders' inspection

	Full log	**For inspection**	**Suspicious**	**Anomalous**
number of events	101,025	198	23	5
number of patterns	450	131	20	3

[56]. We use pattern mining algorithms to extract the most and the least frequent event patterns from ICS log.

Naedele [78] proposes to combine various log resources in a process control environment to detect intrusions. The detection is operator-assisted. To the best of our knowledge, only Balducelli et al. [18] analyse ICS logs to detect unusual behaviour. There, the authors use case-base reasoning to find sequences of events that do not match sequences of normal behaviour (from the database of known cases). The authors analyse sequences of log events that originate from a simulated testbed environment. In contrast, we analyse individual logs from a real ICS facility.

4.5 Discussion

In this section we discuss different aspects of tool performance (such as usability and timing), acknowledge limitations and present potential directions for future work.

Usability Table 4.2 summarizes the output of the performed log analysis through different phases. To inspect system behaviour in a currently running ICS system, the users would have to look at individual events (a few thousand per day). By transferring the level of analysis to patterns, instead of individual events, we help stakeholders in aggregating log information. To discard a large number of uninteresting patterns, we perform frequency pattern analysis. With the suggested "cut off" threshold, our stakeholders receive for inspection 131 unique patterns in 14 days. The number of patterns per day varies, but on average it is less than 10. Finally, after context analysis of suspicious patterns (i.e., an additional round of analyses on suspicious patterns), we estimate that the user had to inspect in average 11 patterns per day.

Table 4.3: System performance results and estimation of processing annual logs

Dataset information	**ICS log**	**“Accidents”**	**Estimated annual ICS logs**
number of instances	101,025	1,000,000	2,500,000
total number of items	69	500	70
avg size of itemsets	6	45	6
Data Preparator (s)	22.7	does not apply	1,080
Pattern mining (s)	0.97	100	200
Total MELISSA processing time (s)	23.6	does not apply	1,280

System performance Testing has been performed on a machine with an Intel Core 2 CPU at 2.4GHz and 2Gb of memory. Table 4.3 shows runtime results of testing. The table consists of three columns. The first column shows the results of our testing on ICS system logs. The second column shows benchmark results of the pattern mining algorithm by Grahne et al. [46] on the “Accidents” dataset [57]. We use these results to estimate the runtime of the expected size of system logs over a year (shown in the third column). The complexity of the preprocessing is O(n). Scalability of the used mining frequent patterns algorithm (in PE) is discussed in [46]. To estimate MELISSA’s performances on an annual ICS log, we consider benchmarks of the pattern mining algorithm of [46] on the “Accidents” dataset. We argue that this dataset is more complex than the dataset we use, due to the higher number of attributes. Thus, we take the results from [46] as our worst case. Based on this, we estimate that our tool would preprocess and mine patterns in size of approximately one year of work in the stakeholder facility in around 22 minutes.

Enhancing effectiveness and usability While performing the preliminary log analysis, we identified two interesting and challenging directions to improve the detection of anomalous behaviours:

1. derive an automated method to identify patterns that describe normal ICS behaviour,
2. build a self-calibrating threshold to distinguish between regular and unexpected patterns.

The first direction implies that we can determine which patterns occur with the same (or similar) frequency over a longer period of time. By knowing this, we can build a profile of normal behaviour in the ICS system over time [108]. For example, we can determine a set of patterns that are regular in their presence and frequency. If a pattern suddenly changes his "regularity", this can imply that a mimicry attack is taking place. On the other hand, if a regular pattern becomes less frequent, this can imply that a device is malfunctioning or has been reconfigured. Similarly, an operator can use the results of the rare pattern mining to compile rules for alerting similar events. This way the usual alarming system could be improves. By inferring models of normal and anomalous behaviour we can compile rules and thus turn our tool into an online mode.

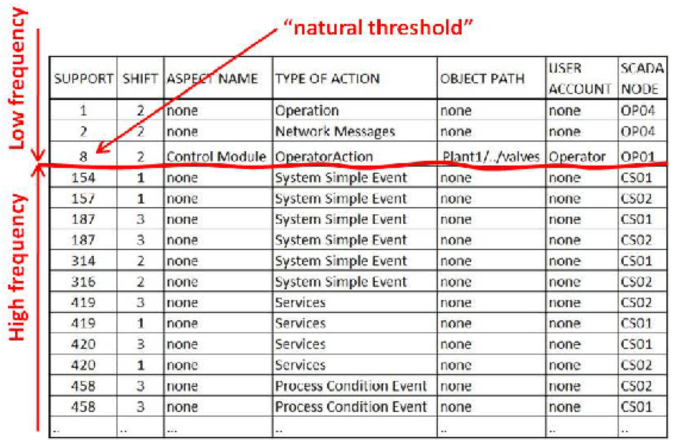
The second direction addresses the shortcoming of the manually-set output threshold in our solution. Currently MELISSA delivers up to 20 least frequent patterns to the security expert, by taking into account the process knowledge. We point out that there are drawbacks in this approach. For example, during a heavy workload day (e.g., a plant temporary increases the work flow to cover a larger area), applying this threshold can cause that some, potentially important patterns, are omitted. By contrast, during a low workload day, a number of semantically uninteresting patterns might be unnecessarily reported to the expert.

Having these in mind, we performed an additional analysis of the derived frequencies of event patterns. Our goal was to investigate if logs contain traces of regular behaviour.

Indeed, we discovered that logs do present certain regularities. For each day in the log, there is a "gap" which divides patterns with low and high frequency of occurrence. For example, Figure 4.6 shows an ordered list of pattern frequencies for day 6 of the analysed log. The first column in the table presents an ordered list of frequency support values. At the top of the list there are patterns with low frequency of occurrences. These patterns are then followed by patterns with a significantly higher frequency of occurrence. Interestingly, there is a "gap" in values between patterns with low and high frequency. We call the value that differentiates these groups of patterns as the "natural threshold". Together with the stakeholders we investigate the character of patterns on the list. The stakeholders agreed that bellow the "natural threshold" there are no events which can be interesting for security purposes. We argue that these patterns represent automated system (re)actions and periodical updates which are time-triggered and potentially inter-dependant (one event triggers another one(s)). For example, some patterns always occur with the same frequency over days. After analysing two weeks of log, we found out that the pattern with the type of action *Services* typically occurs 420 times per one working shift (Figure 4.6). For this, we suspect that the system is configured so that devices are sending time-triggered messages signaling the online status.

Salfner et al. [94] show that observed failures in logs tend to be described in many log entries that occur consecutively forming repetitive patterns. We verified that the patterns in our context behave differently. More specifically, the high frequent patterns that we observed do not occur as burst of events and are spread through the whole day.

Because of this we believe that the analysis of the log over a longer period of time can provide interesting insights in the content of the logs. For example, we could extract patterns that are present on every day, and occur with the same (or similar) frequency of occurrence. These observations would define a profile of regular ICS behaviour.



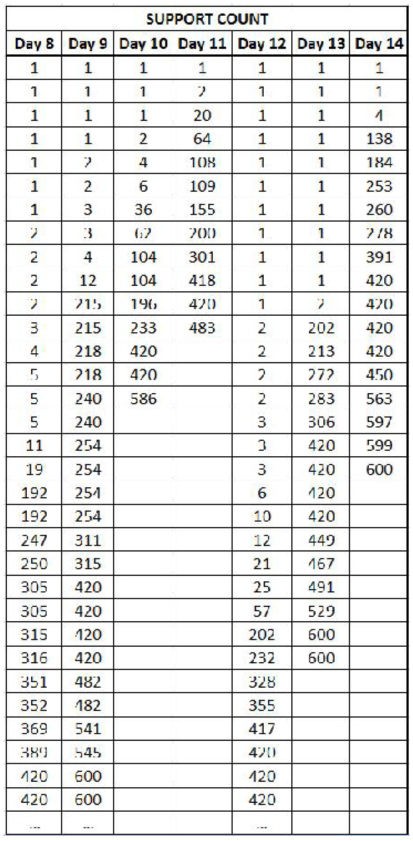
"natural threshold"
Low frequency
High frequency

SUPPORT	SHIFT	ASPECT NAME	TYPE OF ACTION	OBJECT PATH	USER ACCOUNT	SCADA NODE
1	2	none	Operation	none	none	OP04
2	2	none	Network Messages	none	none	OP04
8	2	Control Module	OperatorAction	Plant1/../valves	Operator	OP01
154	1	none	System Simple Event	none	none	CS01
157	1	none	System Simple Event	none	none	CS02
187	3	none	System Simple Event	none	none	CS01
187	3	none	System Simple Event	none	none	CS02
314	2	none	System Simple Event	none	none	CS01
316	2	none	System Simple Event	none	none	CS02
419	3	none	Services	none	none	CS02
419	1	none	Services	none	none	CS01
420	3	none	Services	none	none	CS01
420	1	none	Services	none	none	CS02
458	3	none	Process Condition Event	none	none	CS02
458	3	none	Process Condition Event	none	none	CS01
..	..	...	..	..	..	..

Figure 4.6: Day 6 pattern analysis

According to the stakeholders, the patterns above the "natural threshold" represent potentially interesting patterns for the inspection. These patterns are "incidental". They consist of regular (but unfrequent) user actions and potentially suspicious events. The threshold value varies for different days in the log. Figure 4.7 shows how the "gap" between patterns of low and high frequency changes over the second week in the log. For some days in the log (e.g., day 13 and 14) it seems easy to determine the threshold between "incidental" and regular patterns. However, for other days (e.g., day 11) it is hard to decide where the threshold is. Thus we argue that the threshold value should be determined dynamically.

After inspecting the summarised ICS log, we argue that there are regularities which are suitable for building a self-calibrating pattern output threshold. Also, we believe that the log contains a number of patterns that describe normal plant work. Unfortunately, at this stage we cannot confirm our intuitions in a mathe-



SUPPORT COUNT						
Day 8	Day 9	Day 10	Day 11	Day 12	Day 13	Day 14
1	1	1	1	1	1	1
1	1	1	2	1	1	1
1	1	1	20	1	1	4
1	1	2	64	1	1	138
1	2	4	108	1	1	184
1	2	6	109	1	1	253
1	3	36	155	1	1	260
2	3	62	200	1	1	278
2	4	104	301	1	1	391
2	12	104	418	1	1	420
2	215	196	420	1	2	420
3	215	233	483	2	202	420
4	218	420		2	213	420
5	218	420		2	272	450
5	240	586		2	283	563
5	240			3	306	597
11	254			3	420	599
19	254			3	420	600
192	254			6	420	
192	254			10	420	
247	311			12	449	
250	315			21	467	
305	420			25	491	
305	420			57	529	
315	420			202	600	
316	420			232	600	
351	482			328		
352	482			355		
369	541			417		
389	545			420		
420	600			420		
420	600			420		
...	...			...		

Figure 4.7: Frequency of pattern occurrences over one week of ICS log

matically sound way. This is due to the fact that we only have two weeks of plant logs, which is a short time in process life or a stabile system, such as ICS facility.

Nevertheless, we believe that these observations should be further investigated. Also, we believe that the evidences found in the logs further corroborate the paradigm that ICS facilities are stable and repetitive environments [105].

As a final observation, the mining of logs can be provided in an unstructured manner. We remark that the ICS logs are structured better than some other types of logs, such as telecommunication system [93], console logs [120]. However, in Section 4.1.1 we decide to omit one attribute (*Message description*) from the analysis due to the redundant, unstructured and highly variable character. Although we believe that we did not loose (significantly) on the data quality by doing so, we acknowledge a concern that such solution in general may represent a tradeoff. A solution for this would be a log with structured information. Such log could easily be parsed into various (and consistent) attributes, become computer-readable and thus decrease the uncertainty about inferring important information.

Limitations of the approach We now describe the limitations of our approach.

Firstly, there is a threat scenario in which the ICS logs could be corrupted. For example, attacks performed on the devices in the field can produce erroneous input data for the ICS application and cause the generation of logs (and automated actions) which do not reflect the real situation in the field. Also, an attacker might manage to gain higher privileges (e.g., by exploiting a system-related vulnerability) and then prevent recording or erase some log entries. These attacks cannot be detected by observing ICS logs, since the log no longer represents a consistent data resource. For detecting these kinds of attacks, a complementary analysis of network data or field measurement is necessary.

Secondly, an important limitation of our approach is the possibility for an attacker to evade the detection by repeating the same command a number of times. To overcome this, we propose to enlarge the “knowledge window” and so learn what are normal patterns of behaviour over a longer period of time, as described in Section 4.5. Since our current log capture is limited, we could not validate this yet. This also applies to the limitation of the currently manually-set output threshold.

Thirdly, our approach for introducing the process knowledge highly depends (and thus can be biased) on the experts knowledge about the specific process. We acknowledge that we cannot do anything to overcome this fact (because attribute values are nominal and thus human-readable only).

Finally, our approach cannot provide reasoning to the operator about the character of a suspicious event (e.g., “This event is suspicious because user *A* never worked from node *B*”). Generally, all anomaly-based approaches have the same limitation. This is because the model of normal (i.e., expected) behaviour is typically described by a combination of attributes (i.e., implicitly). By inferring rules from the model, this limitation can be partly addressed. For example, by applying the algorithm for mining association rules to the identified patterns we can compile rules whose interpretation is more readable to humans.

Approach generalisation In this section we discuss the possibilities and the difficulties of applying our approach to other ICS environments. We distinguish two different problems: (1) applying the threat analysis and (2) applying the log processing.

First, we observe that the deviations identified during the threat analysis are compiled from the tailored ICS application software. Thus the set of undesirable user actions is not universal. This is because, due to the nature of plant process, a set of undesirable actions in one environment might only be a subset of all undesirable actions in another environment. However, after the discussion with the experts from four different facilities (two water treatment, one gas distribution and one power distribution company), we conclude that all environments operate in similar conditions (i.e., stabile number of components, operations and users, similar communication frequency). Because of this, we believe that the proposed approach is transferable.

Second, we believe that the log processing cannot be generalised. This is because log content and log format may differ in applications of various vendors. However, under the condition that the parsed log is either (1) provided by the vendor or (2) inferred during the mining process, and does represent a continuous ICS monitoring log, we hypothesize that our approach is applicable. We base this on the knowledge that ICS typically represents a "chatty" systems whose main task is process monitoring (and thus it continuously communicates with its components). Therefore, we expect the logs to be continuous in other environments also.

4.6 Conclusion

We propose the *MELISSA* tool that extracts non-frequent patterns from the ICS application log. These patterns are expected to be the result of an anomalous events such a undesirable user actions. We benchmarked the tool with real logs from a water treatment facility. Although no real security incident occurred in the log we took into account, at least three events were labelled by the process experts as anomalous. All detected events relate to an undesirable engineer operation on the system. These events could not be detected by applying common log filters targeting direct user actions. In fact, the entries were only indirectly related to user operation (and thus represent the consequence of an action). In our context, the actual action of activating a script that generated one anomalous event (i.e., *A1* in Figure §4.4) occurred before of the log capture time. Similarly, other two anomalous events (i.e., *A2* and *A3* in Figure §4.5) represent propagated errors on devices caused by a legitimate user action (which was logged and initially cleared as normal by the stakeholders).

In summary, our results show that ICS logs represent an interesting data resource which gives a new perspective on ICS behaviour and represents a complementary view to the traditional security mitigation strategies. The obtained results show that it is indeed possible to automate the detection of undesirable user activities in ICS application.

Chapter 5

N−gram Against the Machine: On the Feasibility of the N−gram Network Analysis for Binary Protocols

In Chapter 1 we present two attack vectors for implementing process input manipulations (user application and network infrastructure). In Chapters 3 and 4 we analyse threats carried through user application. In this and the next chapter we investigate network manipulations in ICS. In particular, we focus on threats that occur through the exchange of network messages that directly influence process operation (e.g., a network message that sends command to shut down a PLC). We perform an analysis of process network manipulations by describing the manipulations in the context of common network threats and discussing the feasibility of suitable detection techniques.

In Chapter 2 we give a brief overview of techniques for detecting network threats. In this context, techniques including payload-based packet analysis seem more suitable for addressing process network manipulations in ICS. This is because an attack targeting process disruption carries the critical content in the payload section of packets, rather than in flow statistics (since process attacks targets application logic of the process). In fact, process manipulations represent more covert threats than other types of content attacks, such as a data injection attack. This is mainly because the data injection attack carries content that is not legitimate (i.e., the injected code), while attacks targeting process disruption only carry legitimate content (e.g., commands that will misuse legitimate process operation). Distinguishing between malicious and benign message content is thus harder in case of process manipulations.

Most of the current commercial detector systems use a detection model that leverages *signatures*, i.e., they recognize an attack when it *matches* a previously defined signature. An anomaly-based detector raises an alert when the observed input *does not match* the behaviour that was previously observed. The main disadvantage of anomaly-based systems is a high rate of false positive alerts [101]. In principle, however, anomaly-based detectors have *one great advantage* over misuse-based ones: they can detect threats for which there exists no signatures yet, including zero-day and targeted attacks. Targeted attacks are so complex and evasive that *by definition* they cannot be detected by misuse-based systems (false negative problem). An example of such targeted, and process, threat is Stuxnet [77]. This malware targeted specific installations for uranium enrichment to modify, and damage, the enrichment process.

Process threats occurring through network manipulations display a higher diversity of possible forms compared to other network manipulations. For example, attack exploitations in process threats are specific for each different process environment, while the exploitation of an operating system (OS) related to a limited number of OS versions. Therefore, creating a comprehensive set of signatures for process manipulations is probably infeasible. More generally, given that misuse-based systems are ineffective against targeted and zero-day attacks and that most likely there exists no signature yet for the great majority of the attacks that one can buy on the black market, an effective anomaly-based detector remains the most promising solution.

Problem Although the field of anomaly detection is well established in research, to date there are only few actual deployments of anomaly-based detectors worldwide. A common reason used for explaining this is that such systems show poor performance with respect to false positive rate in real-life environments. More generally, Sommer and Paxson [101] argue that many machine learning approaches (which are typically used in anomaly-based intrusion detection systems) are not effective enough for real-life deployments.

To address **RQ3**, this chapter sheds new light on the detection capabilities of anomaly-based detectors for payload inspection. In particular, we assess the performance of selected engines employing a form of n–gram packet content analysis. For performing our benchmarks we choose state-of-the art algorithms that are conceptually different in the way the content analysis is performed. Unfortunately, our choice is somewhat limited by the availability of implementations and the level of details in algorithms descriptions. As a test set, we use attacks that are: *(i)* publicly available and *(ii)* target ICS environments. Although the obtained attack set does not include process manipulations, we perform an in-depth analysis of available attacks to understand the capabilities and estimate the performance of the presented solutions against the process manipulations as well.

The assessment focuses on binary protocols, a family of network protocols widely utilised in automated environments such as ICS. In practical terms, the network payload of a binary protocol is more compact when compared to text protocols, often unreadable by a human and may resemble to attack payloads (since malware packets often consist of binary fragments as well). Due to these reasons the challenge of detecting attacks in binary-based data is typically greater than in text-based data.

Contribution The main contributions of this chapter are:

- we perform thorough benchmarks using real-life data from binary-based protocols, which have been targeted lately by high-impact cyber attacks,
- we analyze and discuss the reasons why certain attack instances are (not) detected by the chosen approaches,
- we discuss the feasibility of deploying such approaches in real-life environments, in particular w.r.t. the false positive rate, an issue that is insufficiently discussed by authors in their work (except in works like [62]).

5.1 Preliminaries

In this section we introduce the concepts and terminology that will be used in the remainder of the chapter. In particular, we present *(i)* a survey of plausible attacks carried out on Modbus and *(ii)* a description of the state-of-the art content detectors.

5.1.1 Classification of network manipulations on Modbus TCP

To understand the specifics of different types of network attacks, we survey plausible attacks carried out in the context of Modbus protocol from literature. We group the attacks according to the targeted level of the OSI model [8] (presented in the first column of Table 5.1) and gained impact (e.g., system/process integrity, availability and confidentiality, presented in the second column of Table 5.1).

Attacks on the Transport level imply manipulations of message arrival and error checking mechanisms. Here, an attacker needs the knowledge about a vulnerability of the TCP stack implementation in a particular system. In the context of Modbus TCP, a crafted packet with different length than the defined value in packet parameters caused a buffer overflow [137]. This attack, targeting data integrity, triggered a vulnerability in an implementation of Modbus server and caused server failure.

To perform an attack on the Session level, an attacker hijacks the TCP session and performing an arbitrary set of commands. To perform an attack, an attacker needs the knowledge about protocol weaknesses (e.g., lack of authentication) or implementation flaws (e.g., inconsistent protocol specification compliance). Unfortunately, most ICS protocols today still lack security features (e.g., such as authentication or encryption) and robust protocol implementation (e.g., Byres et al. [29] show that PLCs behave unreliably on various fuzzing attempts). This makes them an easy target for the presented attacks. There are several known attacks targeting systems through the session level in Modbus TCP. For example, an attacker may used a man-in-the middle attack to perform an unauthorised use of administrative commands that will restart the TCP connection and thus influence system availability [128]. Similarly, an attacker may intercept the communication to explore the implemented functionalities (e.g., probe various function codes and listen for responses and exceptions [128]). This information can be used as a reconnaissance mechanism for crafting next stages of the attack.

Attacks on the Application level commonly refer to malware activity which, by exploiting a vulnerability in application logic, gains control over the system (e.g., manipulations of user interaction or application programs such as Adobe, Ms Office, etc.[125]). For an ICS, the most interesting misuse on the application level refers to the attacks targeting process disruptions. The attacks represent an activity that is legal at the protocol level, yet violates semantic constraints that a process imposes, including both semantically incorrect messages (e.g., conflicting commands) and operations that lead the site into an undesirable state (e.g., inserting inverted parameters [40] or a command to open a pump when it must remain shut). Here, the knowledge requirements for performing an attack are higher than for any other OSI level. More specifically, an attacker needs to be able to interpret the application semantics of a specific protocol. By knowing what kind of data is exchanged and how it is encoded, he can craft network messages to influence application program and thus disrupt ICS process. In the context of Modbus, Caracano et al. [30] present a proof-of-concept malware that aims at diverting process flow in ICS. The malware performs simple value manipulations and generates Modbus packets that constitute of legitimate commands. More specifically, the malware tracks the status of current values and crafts packets that will invert coil values or set registers to their maximum/minimum allowed value according to the protocol specification. Such attacks can potentially lead to fatal consequences on critical process variables. However, the precise effect on the process remains specific to each environment. The most relevant example of a targeted process

Table 5.1: Summary of attacks against Modbus implementations

OSI Level	Impact	Attack description	Example
Transport	System integrity	Corrupt integrity by adding or removing data to the packet.	Craft a packet that has a different length than defined in parameters or is longer than 260 bytes (imposed by the spec.) [128].
Session	System confidentiality	Explore implemented functionalities in PLC.	Probe various FC and listen for responses and exceptions [128].
	System integrity	Exploit lack of specification compliance.	Manipulate application parameters within spec. (e.g., offset) or outside of spec. (e.g., illegal FC) [29, 128, 137].
		Perform unauthorized use of an administrative command.	Use FC 8-0A to clear counters and diagnostics audit [128].
	System availability	Perform MITM to enforce system delay.	Send exception codes 05, 06 or FC 8-04 to enforce Listen mode [128].
		Perform unauthorized use of administrative command.	Use FC 8-01 to restart TCP communication [29, 128].
Application	Process confidentiality	Explore structure of memory map.	Probe readable/writable points and listen for exceptions to understand process implementation details [128].
	Process integrity	Perform unauthorized change of process variable.	Write inverted read values. Write maximal or minimal data values allowed per data point [30]. Tamper measurements to trigger undesirable reaction [43].

FC: Function code defining the type of functionality in Modbus.
MITM: Man-in-the-middle attack.

attack is Stuxnet [77][1]. By attacking PLCs, this malware crossed a boundary as the first publicly known malware that injected *semantically meaningful commands* into a highly-specific plant environment.

5.1.2 Description of analysed systems

In the remainder of the section we introduce four algorithms that we select for testing: PAYL, POSEIDON, Anagram and McPAD. These algorithms are

[1]Although the attack was not implemented on Modbus (but on Profinet), the concept of the attack is transferable.

1) general-purpose enough to be used with multiple application-level protocols, 2) proposed by often cited papers in the IDS community or 3) claiming to improve over the previous ones. Each algorithm requires as an input only the incoming network traffic, and does not perform any correlation between different packets.

5.1.2.1 PAYL

Wang and Stolfo in [115] present their 1-gram-based payload anomaly detector (PAYL). The system detects anomalies by combining 1-gram analysis algorithm with a classification method based on clustering of packet payload data length. The system employs a set of *models*: a model stores incrementally the resulting values of the 1-gram analysis for packet payloads of length l, thus each payload length has a different model. Each model stores two data series: mean byte frequency (i.e., relative byte frequencies span across several payloads of length l) and byte frequency standard deviation for each byte value (i.e., how relative byte frequencies change across payloads). During the detection phase, the same values are computed for incoming packets and then compared to model values: a significant difference from the model parameters produces an alert.

When PAYL fails to detect an attack Fogla et al. [38] show that PAYL's detection can be evaded by mimicry attacks. PAYL is vulnerable to mimicry attacks since it models only 1-gram byte distributions. By carefully crafting an attack payload, an attacker is able to deceive the algorithm with additional bytes, which are useless to carry on the attack, but match the statistics of normal models.

5.1.2.2 POSEIDON

Bolzoni et al. present POSEIDON [25], a system built upon a modified PAYL architecture. PAYL uses the data length field for choosing the right model. By contrast, POSEIDON employs a neural network to classify packets (and thus choose the most similar model) during the preprocessing phase. The authors use Self-Organizing Maps (SOMs) [61] to implement the unsupervised clustering. First, the full packet payload is analyzed by the SOM, which returns the value of the most similar neuron. That neuron model is then used for the calculation of byte frequency and standard deviation values, as in PAYL.

When POSEIDON fails to detect an attack Differently from PAYL, POSEIDON is more resilient to mimicry attacks due to the combination of SOM and PAYL. The SOM analyzes the input by taking into consideration byte value at i-th position within the whole payload. Thus, extra bytes inserted by the attacker would be taken into consideration as well, resulting in a different classification than normal traffic. However, the granularity of the classification done by the SOM is coarse. Thus, if the attack portion of the sample payload is small enough, then the sample could be assigned to one of the clusters containing models of regular traffic, and may go unnoticed because of a similar byte frequency distribution.

5.1.2.3 Anagram

Wang et al. [113] present Anagram. The basic idea behind Anagram is that the usage of higher-order $n-$grams (i.e., $n-$grams where $n > 1$) helps to perform a more precise analysis. However, the memory needed to store average and standard deviation values for each $n-$gram grows exponentially (256^n, where n is the $n-$gram order). For instance, 640GB of memory would be needed to store 5-grams statistics. To solve this issue, the authors propose to use a binary-based $n-$gram analysis and store the occurred $n-$grams efficiently in a Bloom filters [24]. The binary-based approach implies a simple recording of the presence of distinct $n-$grams during training. Since less information is stored in the memory, it becomes possible to effectively use higher-order $n-$grams for the analysis. Authors show that this approach is more precise than the frequency-based analysis (e.g., used in PAYL) in the context of network data analysis. This is because higher-order $n-$grams are more sparse than low-order $n-$grams, and gathering accurate byte-frequency statistics becomes more difficult as the $n-$gram order increases. When in detection mode, the current input is ranked using the number of previously unseen $n-$grams.

When Anagram fails to detect an attack There are two main reasons why Anagram may fail to detect attack attempt. Firstly, the Bloom filter could saturate during training. This is because the user may underestimate the number of unique $n-$grams and allocates a small Bloom filter, during testing any $n-$gram would be considered as normal. Secondly, Anagram will likely miss the detection if the attack leverages a sequence of $n-$grams that have been observed during testing.

5.1.2.4 McPAD

Perdisci et al. present “Multiple-classifier Payload-based Anomaly Detector” (McPAD) [84] with a specific goal of an accurate detection of shell-code attacks. The authors use a modified version of the 2-gram analysis, combined with a group of one-class Support Vector Machine (SVM) classifiers [111]. The 2-gram analysis is performed by calculating the frequency of bytes that are ν positions apart from each other. By contrast, a typical 2-gram analysis measures the frequency of 2 consecutive bytes. By varying the parameter ν, McPAD constructs several representation of the payload in different feature spaces. For example, for ν=*0..m*, McPAD builds *m* different representations of the packet payload. When in testing mode, a packet is flagged as anomalous if a combination (e.g., majority) of SVM outputs acknowledge the payload as anomalous.

When McPAD fails to detect an attack By design, McPAD tries to give a wide representation of the payload (i.e. add more context by constructing byte pairs that are several positions apart). This may represent a difficulty in two cases. First, this is an approximate representation and that may imply a poorly described payload in case of slight differences between the training sample and an attack [15]. This may lead to a high false positive rate and a low detection rate. Secondly, McPAD uses different classifiers that have to come into an “agreement” to decide if a particular packet is anomalous or not. A problem may arise when, due to an approximate payload representation, several classifiers are misled by the byte pair representation and result in outvoting “correct” classifiers. In such case, the system might miss the detection.

5.2 Approach

We believe that one of the main reasons for poor performance of anomaly-based NIDS lies in the intrinsic limitation of commonly applied algorithm for content analysis: *$n-$gram analysis*. Since performing a comprehensive test to verify the ability of an IDS of identifying (all) attacks and to spot its weaknesses is unfeasible [70], we proceed to experimentally address our claim. We present a comparative analysis and evaluate the effectiveness of anomaly-based algorithms that analyse network payloads by using some form of $n-$gram analysis.

To verify the effectiveness of different algorithms we execute a number of steps: 1) collect network and attack data, 2) obtain a working implementation of each algorithm, 3) run the algorithms and analyse the results.

Obtaining the data In general, optimal conditions for evaluating the performance of an IDS consist of running tests on unprocessed data from real networks [16]. Thus we first collect real-life data from different network environments, which are currently being operated. The past research is typically focused on benchmarking the algorithms with the HTTP protocol, although the authors do not explicitly restrict the scope of their algorithms to this protocol. We focus on the analysis of binary protocols. In particular, we analyse an example of a binary protocol found in a typical LAN (such as a Windows-based network service) and an example of a binary protocol typically found in an ICS.

Windows is heavily used OS in the world, and every instance runs by default certain network services that are often used within LANs. For instance, the SMB/CIFS protocols [131] are used to exchange files between two computers, while other services (e.g. RPC) run on the top of it to provide additional functionalities. Although Windows systems are usually secured against abuses of such service from the Internet, corporate users take advantage of this feature quite often. An attacker that would develop an exploit for a zero-day vulnerability leveraging this protocol could potentially affect a large number of systems. In the last decade several malware [77, 126] exploited SMB/CIFS to operate botnets and carry on other malicious activities.

We collect attacks in two different datasets. Our focus is on data injection attacks that have a high impact (see [133]).

Obtaining the implementations Secondly, to carry out the benchmarks, we need working prototypes of all the algorithms we want to test. We could obtain an implementation of POSEIDON and McPAD from the authors. For the other two algorithms, we write our own implementation based on the description found in the papers. To be sure that our implementations are correct, we verified that our results resemble the ones shown in the benchmarks of the respective original papers.

Analysing the results The last step of our evaluation is the analysis of results with a focus on the identification of reasons for (un)successful detection.

5.2.1 Evaluation criteria

The effectiveness of an IDS is mainly determined by the detection and false positive rates. The detection rate indicates the number of attack instances correctly identified by the IDS (true positives), w.r.t. the total number of attack instances. The false positive rate indicates the amount of samples that the IDS flags as attacks when they are actually not. False positives are a major limiting factor in this domain because, differently from other classification problems, their cost is high [17].

Detection rate To provide a detailed overview of the detection capabilities of each algorithm, we consider both the number of correctly detected packets in the attack set and the number of detected attack instances. In fact, not all attacks show malicious payload within one single packet. Although an algorithm that exhibits a high per-packet detection rate has a higher chance of detecting attack instance, we do not argue that a low per-packet detection rate implies an equivalently low per-instance detection. In summary, we consider an alarm as a true positive if the algorithm is able to trigger at least one alert packet per attack instance.

False positive rate The usual approach to document the performance of an IDS is to relate the false positive rate with the detection rate. This is done by drawing so called Receiver Operating Characteristic (ROC) curves. The benchmarks from the original papers of the proposed algorithms express the false positive rate as a percentage. However, such number has little meaning to the final users. A better way to express the false positive rate is in terms of the number of false positives per time unit. We establish two different thresholds: 10 false positives per day and 1 false positive per minute. The former value is proposed in [68] as a reasonable number for a user to maintain trust in the system. The latter is, in our opinion, the highest rate at which a human can verify alerts generated by an IDS. It is worth noting that anomaly-based IDSes, unlikely misuse-based ones, do not provide information regarding the attack classification. Thus, the user might require additional time to investigate whether the alert is a true or a false positive. For each data set we compute these two thresholds based on the number of actual packets included in the verification sub data set after having split the original data set.

Since we do not make the data sets attack-free before hand, and thus some "noise" could have been collected as well, we need a way to verify that the alerts generated while processing the verification sub data set are actual false positives. To do that, we use a misuse-based IDS (the popular open-source Snort), which is automatically fed with the network stream for which an alert was triggered during verification.

Commonly, in IDS evaluation papers the authors stop their analysis by reporting on the true and false positive rates. We believe that inspecting which attacks are detected, which are not detected and why, would provide useful information to fully understand when an algorithm could perform better than others (and for which threats). This kind of analysis can provide insights for future improvements. Finally we aim at evaluating the effectiveness of combining diverse algorithms to boost the detection rate.

5.3 Description of network data

In this section we describe in detail the background data set and attack data sets that will be used for benchmarking the detection algorithms. The chosen data sets comprise network traffic taken from two environments. We use the publicly available vulnerabilities and high impact exploits to run three rounds of tests.

First, to validate that the implemented and tuned algorithms have comparable detection and false alert rates with the tests reported in the research papers, we perform initial testing with common (HTTP) test data sets. Second, to assess the detectors we use different content attacks coming from IT environment (LAN network data with SMB-based content attacks). Third, we use ICS environment to test known Modbus attacks. Based on the latter two analyses we discuss the capabilities of the known detectors to capture process attacks.

We now describe the used datasets.

5.3.1 Data sets for implementation verification

DS_{DARPA} The DARPA 1999 data set [68] is a standard data set used as reference by a number of researchers, and offers the possibility of comparing the performance of various systems. Despite being anachronistic (and criticized in several works [72]), three of the algorithms we test have used this data set to compare their performance to previous works. Thus, we use the DARPA data set to verify that our own implementations of PAYL and Anagram offer comparable detection and false alert rates with the tests reported in the research papers of the detection algorithms.

The DARPA 1999 data set is a synthetic set of network dumps (thus it does not comply with our requirements). The data generated for the evaluation span over 5 weeks and can be divided in training and testing data. Training data (week 1 and 3 of traffic) is intended to be completely free of attacks, while testing data (week 4 and 5) is intended to consist entirely of attack scenarios. An additional week of traffic (week 2) is provided with labelled attacks, i.e., attacks are clearly marked with temporal timestamps and classified. The process used to generate training data is not deeply presented. The data is claimed to be similar to that observed during several months of sampling data from a number of Air Force bases (see Lippman et al. [68]), but the data set lacks of statistics to evaluate and establish similarities.

AS_{HTTP} This attack set is presented by Ingham and Inoue in [54], and has been used also by the authors of McPAD for their benchmarks. It comprises 66 diverse attacks, including 11 shellcodes, which were collected from public attack archives. The attacks are instances of buffer overfows, input validation errors (other than buffer overflows), signed interpretation of unsigned values and URL decoding errors.

5.3.2 LAN data set

DS_{SMB} This data set includes network traces from a large University network. Samples have been collected through a week of observations. The average data rate of incoming and outgoing packets is ~40Mbps.

In particular, we focus on the SMB/CIFS protocol, and even more on SMB/CIFS messages which encapsulate RPC messages (see Section 5.4.2). The average packet rate for this traffic is ~22/sec. Based on this we calculate the false positive rate threshold for obtaining 10 alerts per day as 0.0005% and the one for obtaining 1 alert per minute as 0.073%.

AS_{SMB} This attack data set is made of seven attack instances which exploit four different vulnerabilities in the Microsoft SMB/CIFS protocol: *ms04-011*, *ms06-040*, *ms08-067* and *ms10-061* [127].

ms04-011 is a vulnerability of certain Active Directory service functions in LSASRV.DLL of the Local Security Authority Subsystem Service (LSASS) of several Microsoft Windows versions. We select this vulnerability because it is used by the worm Sasser [135]. We collect two different attack instances for this vulnerability. One trace is downloaded from a public repository of network traces [129] where the attack payload is split in three fragments and contains a shellcode of 3320 bytes. The shellcode is made of a number of NOP instructions (byte value `0x90`), followed by valid x86 instructions and a sequence of the ASCII character '1'. The second instance is generated through the Metasploit framework [132]. The attack payload is split into three fragments and contains a shellcode of 8204 bytes to remotely launch a command shell in the victim host.

m06-040 is a vulnerability of the Microsoft Server RPC service. In particular, the vulnerability allows a stack overflow during the canonicalization of a network resource path. The specified path can be crafted to execute arbitrary code after the exploitation. We collect the attack instance from a public repository of network traces [129].

ms08-067 is a vulnerability of the Microsoft Server RPC service which exploits a similar weakness as the one described in *m06-040*, with the same effects. We select this vulnerability because it is used by Conficker [126] and Stuxnet, two high-impact pieces of malware. We collect two different attack instances for this vulnerability. One instance was downloaded from a public repository [129], while the other one was generated by us using the metasploit framework. In the first instance the payload contains a shellcode of 684 bytes, while in the second instance the shellcode is 305 bytes long only.

ms10-061 is a vulnerability of the PrintSpooler RPC service. When printer sharing is enabled, the PrintSpooler service does not properly validate spooler access permissions. Remote attackers can create files in a system directory, and consequently execute arbitrary code, by sending a crafted print request over RPC. We select this vulnerability because it was used by Stuxnet to successfully propagate in both regular backoffice LANs as well as in industrial control system environments. We collect two different attack instances for this vulnerability, both of them are generated through the metasploit framework. In one instance the attack payload is a binary file (the meterpreter executable), which accounts for 69832 bytes spanned over 18 fragments, while in the other instance the payload consists of a DLL file, which accounts for 1735 bytes spanned over three fragments.

5.3.3 ICS data set

DS_{Modbus} To test the anomaly detection algorithms on ICS networks we collect a data set of traces from the industrial control network of a real-world plant over 30 days of observation. The average throughput on this network is ~800Kbps.

This data set includes network traces of one of the most common protocols used in such environments, Modbus/TCP [119]. Modbus was developed more than 30 years ago initially as a protocol used in serial channels, while the TCP/IP variant was introduced approximately 15 years ago to allow the serial protocol to be used in TCP/IP networks. Modbus/TCP features basic instructions and functions. Its structure is relatively simple, and operators of critical infrastructures usually repeat a limited set of operations, thus reducing the variability of the transmitted data. In fact, the maximum size of a Modbus/TCP message is 256 bytes. Thus, a Modbus/TCP message is always contained in one single TCP segment. Observations on DS_{Modbus} reveal that the average size of Modbus/TCP messages is 12.02 bytes (in DS_{SMB} it is 535.5 bytes). In the observed DS_{Modbus}, the number of duplicated TCP segment payloads is high (96.08%), in contrast to the other data set (e.g., DS_{SMB} has 31.37%). In other words, more than 9 TCP segments out of 10 carry a Modbus/TCP message that is a perfect duplicate of some other message already observed. The average packet rate for Modbus incoming traffic is ~96/sec. Based on this observation, we calculate the false positive rate threshold for generating 10 alerts per day as 0.00012% and the threshold for generating 1 alert per minute as 0.017%.

AS_{Modbus} The attack data set is made of 163 attack instances, which exploit diverse vulnerabilities of the Modbus/TCP implementations. There are fewer publicly known attacks against Modbus/TCP devices than SMB/CIFS attacks. Network traces for a good deal of these attacks can be downloaded from the website of an ICS security firm [128]. The exploited vulnerabilities can be categorised in two large families: *unauthorised use* and *protocol errors*.

Unauthorized use consist of two attack types: (a) "weird" clients talking to the Modbus server and (b) messages used only for diagnostics and special maintenance, which are thus seldom seen in the network traffic. By issuing these special messages, the attacker is often able to achieve a complete take over of the device.

Protocol errors are mainly fuzzing attempts against a device. For instance, these attacks are carried out by sending data not compliant with the protocol specifications (e.g. a too short protocol data unit). The outcome of such attacks can range from the unavailability of the device up to the control of the execution flow (see Cui and Stolfo [33] for a more detailed discussion).

5.4 Benchmarks

In this section we show the results of our benchmarks and compare the performances of the algorithms for each data set.

Setting up and tuning the algorithms For each dataset we split the background traffic into two sets, one for training and one for verification. The splitting is performed randomly, by processing the network streams. Each split sub data set accounts for nearly 50% of the original data. The training sub data set is used to build the detection profiles for each algorithm, while the verification one is used to evaluate the number of false positives raised by the algorithm. Finally, we run the algorithms on the attack data set.

For performing the benchmarks we need to set up several starting parameters for each algorithm.

PAYL As introduced in the original paper, the size of the PAYL model can be reduced by merging profiles when their number becomes too large. For each data set we perform several runs using different values of the *merging* parameter. Finally, for the DS_{DARPA} data set we set the parameter value to 0.12. For the remaining data sets we do not merge profiles, as the total number of profiles remains low (up to 150, compared to 480 in DS_{DARPA}). As the "smoothing factor", we use the same value (0.001) suggested by Ingham and Inoue in [54]. We apply the algorithm to individual TCP segment payloads.

Anagram For each data set we run tests with different n–gram sizes (*n* size of 3, 5, 7, 9 and 12). Since we obtain the best results with the 3-grams, we set this as the standard n–gram size. As in the original paper, we set the size of the Bloom filter to 2MB. We do not use the "bad content model" proposed by the authors because it would be ineffective as they build it with virus samples, and our attack data sets do not include viruses.

POSEIDON For all tests we use a SOM with fixed number of neurones (96). Also, we set the number of instances for training the SOM at 10000.

McPAD For all tests we use the best performing parameters as proposed in [84]. Those are: number of clusters k = 160, desired false positive rate for each SVM classifier set to 1% and *maximum probability* as combination rule for the output of the SVM classifiers.

For each algorithm we vary the value of the threshold to observe how the false positive and detection rates change.

5.4.1 Implementation verification

We use this test to verify that the re-implemented algorithms behave as expected in terms of detection and false positive rates (as described in research papers). To verify the correctness of our implementations we run initial tests using publicly available test sets (i.e., DS_{DARPA} data set for training and AS_{HTTP} for testing).

For training we choose DS_{DARPA} as that was the only common data set used in 3 original algorithm benchmarks. Instead of using DARPA attack dataset, we use the AS_{HTTP} for testing because: (1) the original attack instances of the DARPA data set do not reflect at all modern attacks, and (2) not all the algorithms have been benchmarked against the DARPA attack set (Anagram is the exception). Thus, it would be impossible to faithfully reproduce the previous experiments. In Table 5.2 we summarize the results of this first round of benchmarks: for each algorithm we report the highest detection rate we achieve, and the corresponding false positive rate.

Table 5.2: Test results on DS_{DARPA} and testing with AS_{HTTP}

	FPR (packet-based)	DR (packet-based)
PAYL	0.00%	90.73%
POSEIDON	0.004%	92.00%
Anagram	**0.00%**	**100.00%**
McPAD	0.33%	87.80%

All the algorithms show high detection and low false positive rates. When compared to the original papers, these results match our expectations, thus we can be reasonably sure that our re-implementations are not (too) dissimilar from the original ones in terms of completeness and accuracy.

5.4.2 Tests with LAN data set

We first perform the test on DS_{SMB} by using all the SMB/CIFS packets directed to the TCP ports 139 or 445. However, none of the algorithm can perform well enough under these conditions. For example, the Bloom filter used by Anagram saturates with 3-grams during the training phase. Consequently, no attack instance is further detected, even with the lowest threshold. Increasing either the size of n–grams or the size of the Bloom filter cannot solve the issue of undetected attacks. In the former case, the Bloom filter saturates even with fewer training packets. In the latter case, even with a filter size of 20MB (10 times bigger than the size suggested by the authors) which does not cause full saturation, no attack is detected with false positives rates lower than 4%. Other algorithms exhibit similar detection problems, with at least one attack instance detected only with false positive rates higher than 1%.

This result alone would imply a complete failure for this protocol. We believe the reason why the algorithms poorly perform on SMB/CIFS is because of the high variability of the analyzed payload. In fact, SMB/CIFS is mainly used to transfer files between Windows computers. Such files are contained in the request messages processed by the algorithms and can be of any type, from simple text files to compressed binary archives or even encrypted data.

We observe that all attack instances publicly available exploit vulnerabilities of the Windows RPC service by leveraging SMB/CIFS, which can encapsulate RPC messages. Thus, we re-run the test on a filtered data set. In particular, we extract only SMB/CIFS messages that carry RPC data (~1% of the original SMB/CIFS traffic). By doing this, we are implicitly providing a semantical hint to the algorithms. We expect this to improve both the detection and false positive rates.

The results of this round of tests are summarized on Table 5.3. Anagram achieves a 0.00% false positive rate while still being able to detect 3 attack instances. Anagram also achieves the lowest false positive rate among the tested algorithms when detecting all of the attack instances, a rate lower than the adjusted false positive of 1 alert per minute. McPAD generates the highest false positive rate, and it is impossible to lower that no matter which combination of parameters we choose. We believe that the main reason for this lies in the fact that McPAD implements the approximate payload representation, that, in such variable conditions, provides poor payload description.

There is no need to evaluate how a combined approach, i.e., using all the algorithms simultaneously, would perform since Anagram is already outperforming the other algorithms.

Finally, we process all false positives with SNORT to verify that none of them is actually a true positive.

Table 5.3: Test results on DS_{SMB} and testing with AS_{SMB}

	FPR (packet-based)	DR (packet-based)	DR (instance-based)
PAYL	0.004%	1.43%	2/7
	0.007%	3.35%	3/7
	0.01%	6.65%	4/7
	0.05%	23.92%	5/7
	4.41%	66.51%	6/7
	11.05%	85.02%	7/7
POSEIDON	0.007%	6.22%	2/7
	0.007%	10.04%	3/7
	1.27%	37.32%	4/7
	2.28%	50.23%	6/7
	3.32%	58.37%	6/7
	5.39%	66.98%	7/7
Anagram	0.00%	0.95%	2/7
	0.00%	22.48%	3/7
	0.02%	37.32%	7/7
	2.34%	55.50%	7/7
	8.39%	63.64%	7/7
McPAD	19.02%	60.38%	7/7
	20.62%	60.86%	7/7
	22.31%	61.35%	7/7
	27.61%	65.21%	7/7
	97.39%	90.00%	7/7

Analysis of detected and undetected attacks By considering which attack instance is detected the most, we observe that all the algorithms can detect an attack instance exploiting the *ms04-011* vulnerability. In particular, of the three segments composing the attack payload, only one is always identified as anomalous. Here we report a fragment of it:

```
0230   59 35 1c 59 ec 60 c8 cb cf ca 66 4b c3 c0 32 7b  Y5.Y.`....fK..2{
0240   77 aa 59 5a 71 76 67 66 66 de fc ed c9 eb f6 fa  w.YZqvgff.......
0250   d8 fd fd eb fc ea ea 99 da eb fc f8 ed fc c9 eb  ................
0260   f6 fa fc ea ea d8 99 dc e1 f0 ed cd f1 eb fc f8  ................
0270   fd 99 d5 f6 f8 fd d5 f0 fb eb f8 eb e0 d8 99 ee  ................
0280   ea ab c6 aa ab 99 ce ca d8 ca f6 fa f2 fc ed d8  ................
0290   99 fb f0 f7 fd 99 f5 f0 ea ed fc f7 99 f8 fa fa  ................
02a0   fc e9 ed 99 fa f5 f6 ea fc ea f6 fa f2 fc ed 99  ................
02b0   90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
```

```
02c0   90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ...............
02d0   90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ...............
02e0   90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ...............
02f0   90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ...............
0300   90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ...............
...
0580   90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ...............
0590   90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ...............
05a0   90 90 90 90 90 90 90 90                          ........
```

We verify that this particular segment is flagged as anomalous by all the algorithms because of the long sequence of the byte value `0x90`, which corresponds to the NOP instruction of the shellcode. Although there are individual bytes with value `0x90` in the training set, we verify that there is never a sequence of three bytes with this value. This explains why Anagram can identify as anomalous a vast majority of the 3-grams in the aforementioned payload. Also PAYL and POSEIDON will identify an anomalous byte frequency distribution, in which the byte value `0x90` peaks above all the others. Similarly, the fact that the payload consists of continuous `0x90` bytes implies that McPAD classifiers will be able to recognize the peak in the frequency of these 2-grams.

We also observe that both PAYL and POSEIDON fail to detect one attack instance exploiting the *ms06-040* vulnerability, when the false positive rate is below 2%. A fragment of the payload of such attack instance is reported below:

```
0170   52 df 47 2c 0c 86 de fe fe b9 f6 68 ae 23 4f a5  R.G,.......h.#O.
0180   81 53 79 43 fc fc 31 58 af ad 6e 30 29 f7 50 8a  .SyC..1X..n0).P.
0190   4a e1 78 43 30 6a 55 75 58 72 6e 4f 42 48 4c 42  J.xC0jUuXrnOBHLB
01a0   36 34 33 4a 51 38 69 42 52 37 39 46 59 49 79 71  643JQ8iBR79FYIyq
01b0   7a 62 38 48 4e 47 68 48 7a 52 59 6e 38 76 55 78  zb8HNGhHzRYn8vUx
01c0   41 4d 57 61 57 66 68 30 48 4c 30 61 76 73 61 6b  AMWaWfh0HL0avsak
01d0   7a 56 65 4d 32 42 76 64 64 43 64 41 45 75 75 53  zVeM2BvddCdAEuuS
01e0   4f 7a 41 4f 70 6b 30 37 4c 45 70 66 73 44 73 49  OzAOpk07LEpfsDsI
01f0   66 57 39 65 31 59 45 6e 43 38 52 62 76 57 65 50  fW9e1YEnC8RbvWeP
0200   59 63 54 77 7a 63 32 4f 50 4f 52 6b 71 4c 33 4b  YcTwzc2OPORkqL3K
0210   65 7a 69 62 72 57 4e 6d 58 33 4b 56 70 50 6c 45  ezibrWNmX3KVpPlE
```

This fragment contains a large majority of printable characters, thus one would expect that, since RPC over SMB/CIFS messages are mostly binary, a detection algorithm based on byte frequency distribution would be able to detect such payload. However, RPC over SMB/CIFS is also used to feed remote print spoolers with files to print. For example, in the filtered data set used for training we can identify the following fragment which is part of a PDF file sent to the spooler:

```
0200   09 2f 40 6f 70 53 74 61 63 6b 4c 65 76 65 6c 20  ./@opStackLevel
0210   40 6f 70 53 74 61 63 6b 4c 65 76 65 6c 20 31 20  @opStackLevel 1
0220   61 64 64 20 64 65 66 0d 0a 09 09 63 6f 75 6e 74  add def....count
0230   64 69 63 74 73 74 61 63 6b 20 31 20 73 75 62 0d  dictstack 1 sub.
0240   0a 09 09 40 64 69 63 74 53 74 61 63 6b 43 6f 75  ...@dictStackCou
0250   6e 74 42 79 4c 65 76 65 6c 20 65 78 63 68 20 40  ntByLevel exch @
0260   64 69 63 74 53 74 61 63 6b 4c 65 76 65 6c 20 65  dictStackLevel e
0270   78 63 68 20 70 75 74 0d 0a 09 09 2f 40 64 69 63  xch put..../@dic
0280   74 53 74 61 63 6b 4c 65 76 65 6c 20 40 64 69 63  tStackLevel @dic
0290   74 53 74 61 63 6b 4c 65 76 65 6c 20 31 20 61 64  tStackLevel 1 ad
```

Similar to the previous fragment, this fragment also contains a vast majority of printable characters. Due to the high variability of data, the threshold for detecting such fragment had to be set in such a way that many normal samples were classified as anomalous.

5.4.3 Tests with ICS data set

The results of this round of tests are summarized on Table 5.4. We report obtained combinations of false and true positives for each detector with respect to different tuning parameters. Anagram shows outstanding results in this test, and this does not come unexpected. The messages in this data set are rather short and the number of duplicates is also high (more than 95%). This is a perfect combination for Anagram and its binary-based approach. Anagram detects most of attack instances with a false positive rate that is lower than the adjusted false positive rate of 10 alerts per day. When detecting all of the attack instances, the false positive rate is still one order of magnitude lower than the adjusted false positive rate of 1 alert per minute.

McPAD also performs well with respect to the false positive. This is expected because the analysed Modbus traffic is expressed in messages of fixed length structure and with a limited range of values in used bytes. This results in McPAD accurately modeling relationships in the message structure.

PAYL seems to have a better packet-rate detection rate than POSEIDON. However, POSEIDON always performs better with respect to the instance-based detection rate as well as lower false positive. When the two algorithms are tuned to detect all of the attack instances, they both generate an overwhelming number of false positives, triggering on almost every packet.

With respect to the false positives generated during the verification phase, no raised alert turned out to be a true positive when processed with Snort. This is largely expected due to 1) the small number of available signatures for the Modbus protocol, and 2) the fact that the industrial control network from which we collected traffic from is highly isolated from other networks, with only a fixed number of hosts communicating. Thus, the chance of “noise” is substantially low.

Similarly to the previous test, there is no reason to test how a combination of algorithms would perform, because Anagram outperforms all the other algorithms in terms of detection and false positive rates.

Analysis of detected and undetected attacks To understand why Anagram works so well with Modbus traffic consider the following two Modbus messages. The first one is a valid read request (identified by the 8th byte with value `0x03` which corresponds to the request “function code”):

```
35 ae 00 00 00 06 00 03 0c 7f 00 64
```

The following fragment is an attack instance attempting to corrupt the PLC memory by invoking a vulnerable diagnostic function (byte value `0x08`) with invalid data (bytes `0x00 0x04 0x00 0x00`):

Table 5.4: Test results on DS_{Modbus} and testing with AS_{Modbus}

	FPR (packet-based)	DR (packet-based)	DR (instance-based)
PAYL	0.00%	62.57%	101/163
	0.00%	92.63%	150/163
	9.25%	95.70%	155/163
	95.00%	100.00%	163/163
POSEIDON	0.04%	3.07%	4/163
	0.07%	77.30%	125/163
	0.37%	95.09%	154/163
	7.65%	97.54%	158/163
	97.81%	100.00%	163/163
Anagram	0.00%	3.68%	5/163
	0.00005%	10.42%	16/163
	0.00007%	18.40%	29/163
	0.00007%	96.93%	157/163
	0.002%	100.00%	163/163
McPAD	0.041	6.31%	10/163
	0.044	96.93%	157/163
	0.045	96.93%	157/163
	0.046	96.93%	157/163

```
00 00 00 00 00 06 0a 08 00 04 00 00
```

We first observe that the anomalous value in this payload are the byte value of the function code, and the subsequent four bytes (never observed in the training set on that same positions). There are 6 3-grams over 10 (60%) which are not present in the valid request. The number of distinct 3-grams observed during training is not much bigger than the one observable in the aforementioned request, due to the large number of duplicated payloads. Thus, with such a small packet size, even a few bytes with unusual value can make a big difference.

On the other hand, from the results in Table 5.4 we see that for all the algorithms there is always a significant increase in the amount of false positives raised when the threshold is adjusted to detect all attack instances. We observe that the attack instances that do not get detected before the threshold is adjusted are similar to the one in the following example:

```
00 00 00 00 00 02 0a 11
```

This 8 bytes long message is the smallest possible Modbus message allowed by the protocol specification. We point out that this request is not unusual only because of its size, but also because the function code value `0x02` (corresponding to the request "report slave ID") was never observed during training. We have verified that the only 3-gram in this payload not observed during the training is the last one `0x02 0x0a 0x11`. Thus, in spite of the small size of this payload, the threshold has to be lowered in order to detect it. Detecting such anomalous packet (with only one anomalous n−gram) in a bigger message would be much more difficult.

5.5 Related work

To the best of our knowledge, Ingham and Inoue describe the most recent framework for testing the performance of IDS algorithms [54]. The authors focus on the HTTP traffic. The framework is based on the general principle that testing different IDS algorithms on the same network environment and with the same network and attack data allows a better comparison of the algorithms' performance, which would be impossible by re-using the results of the unrelated, individual tests run by the algorithm developers. They collect background web traffic from four different websites and create a publicly available set of network traces. The traces contain instances of web attacks generated by running exploitation tools downloaded from popular vulnerability repositories (e.g., BugTraq, SecurityFocus and the Open Source Vulnerability Database).

Other IDS evaluation frameworks, as reported in [55], were developed by Puketza et al. [87, 88], Wan and Yang [112] and by IBM Zurich [35], but are all quite dated.

Song et al. [102] show that polymorphic behaviour in shellcodes is too greatly spread to model effectively.

5.6 Discussion

We now highlight interesting observations from the obtained results.

Despite the fact that the attack instances on the SMB/CIFS protocols are correctly detected, all studied algorithms incur a high penalty in terms of false positives they raise. Concretely, it would be expensive to deploy them independently in a real environment. On the other hand, if we restrict the field to the Modbus protocol alone, Anagram detects almost every attack instance with a rate of false positives lower than the 10 alerts per day threshold. We believe that such performance of the algorithm would not be too distracting for the operators when deployed in a real environment. When looking into details of detection results, we see a limited performance in the detection of process threats. For example, Anagram successfully detects Level 2 attacks (FC 8-04, illegal FC, see Table in Chapter 3). This is mainly because the presented attacks use message types that were not seen during training (e.g., FC 8 message has different format than usual FC 3). Since the detection is based on the score that enumerates "known" against "unknown" $n-$grams in a particular network message, the system has good chances in detecting these threats. However, in case of process threat (e.g., an implementation of application level attack by Carcano [30]), where the attack uses the same message type as common traffic, the threat instance would carry content that differs in only few consecutive bytes from the training trace. In this case, due to the fact that most $n-$grams in the message are already seen, the detector has small chances in capturing the malicious message.

We observe that all studied algorithms trigger on the exploitation payload. We can observe this by selecting two different attack instances that exploit the same vulnerability, but using two different attack payloads. In several cases, while one instance is detected even with a low threshold, to detect both attacks one needs to increase the threshold significantly. The previously missed attack instance usually contains a small-size attack payload, and thus "blends" more easily with the normal payload data, thereby avoiding detection.

5.7 Conclusion

In this chapter we present a thorough analysis of several $n-$gram-based algorithms for network-based anomaly detection. We investigate the performance of state-of-the-art detection algorithms when analyzing network traffic from two binary protocols. Our analysis shows that the detectors can unlikely capture the semantics required for detecting process attacks.

In general, there is no absolute best algorithm among the ones we studied. Anagram performs slightly better than the rest when analyzing the filtered SMB/CIFS and the Modbus protocols, but it is also the one performing worst when the filter is not applied. Technically, this is due to the fact that the unfiltered SMB/CIFS traffic contains several $n-$grams that are also present in the attack payloads. This supports the intuition that the variability of the network traffic has an impact on the performance of these systems. Indeed, *every* studied algorithm is affected by this, allowing us to conclude that, rather than the single implementation, it is the underlying principle of capturing regularity in the packet payload that is unsuitable for unrestricted application.

In particular, this applies to the indiscriminate application of $n-$gram analysis on network streams: our results show such $n-$gram analysis quickly becomes incapable of capturing relevant content features when analysing moderately variable traffic. This problem could be partly alleviated by deploying the detection system in combination with some other sensor that will verify the correctness of alerts [113].

We believe that a more promising approach is the one focusing on identifying chunks of payload (that represent some kind of semantic unit) and applying the n−gram analysis in them. For example, several authors propose to exploit the syntactical knowledge of the HTTP protocol to improve the overall performance of anomaly-based systems, e.g. in [103]. We foresee that a similar approach could be applied to binary protocols as well.

Chapter 6

Through the Eye of the PLC: A Network Approach for Monitoring PLC Communication

In Chapter 5 we showed that the common techniques for network intrusion detection cannot effectively address attacks carried over packet content. In particular, the common underlying approach for analysing packet payloads, n-gram, has limited performance against different classes of network manipulations. In case of process threats, we showed that the current approaches cannot capture process information required for distinguishing between benign and malicious content.

This chapter addresses **RQ4**. In particular, we present an approach that uses process context information to address network manipulations in ICS. We focus on detecting network attacks representing process threats: malicious actions that drive an ICS into an unsafe state (e.g., process halt) without exhibiting any obvious network-level red flag (e.g., exploitation of a protocol vulnerability). We focus our analysis on the attacks that, for a successful execution, need to deviate at least one variable controlled by a PLCs whose information is exchanged in the network communication. Generally, by performing a process attack, an attacker can do the following damage to the process: *(i)* impact operator awareness and *(ii)* divert the process. Attacks on operator awareness typically involve sending forged alarms/events, or incorrect measurements (e.g., corrupt the flow and content of commands sent towards HMI to prevent the normal reaction). Attacks targeting process divert may involve actions such as sending a "stop" command to a PLC, or changing control variables (e.g., change critical parameters that will deviate the process).

A reliable detection of these network attacks requires addressing the key challenge: an interpretation of analysed network messages within the context of the current process state. The process state refers to a set of characteristics (i.e., field measurements and process parameters), that describe the current process condition. Our hypothesis is that, due to a narrow focus, a relatively small and homogeneous set of actors, and mostly automated activity in ICS, process characteristics display an overall regularity that we can exploit for finding a stable baseline separating common activity from attacks.

Problem The extraction of process characteristics describing the process state from a network trace is difficult. More specifically, there are two problems. First, the current format of the payload in network packets does not allow a direct analysis of process parameters and measurements (since the content is encoded within a binary network protocol). Second, a real life environment inevitably carries noise due to different environmental reasons (e.g., communication drops, sensor problems). This leads to difficulties in generation of reliable ICS models that can be used to evaluate the character of the analysed messages. In addition, other practical challenges include the need for gaining access to tapping points that provide broad coverage, technical ambiguities with interpreting protocols, and semantic context required for interpreting observations.

We propose an approach to monitor the network traffic of programmable logic controllers (PLCs), by extracting updates of process variables from their communication with other devices. We choose this communication because PLCs represent the interface between a plant's operators and the field devices (e.g., a pump), thus any state changes—including malicious commands—go through them to take effect. Their network traffic hence provides an ideal vantage point for monitoring. At a high-level, our approach derives behavioural models of a plant's state from past activity over the network, which then facilitates monitoring of future changes for unexpected deviations in the process.

We present a prototype implementation of our approach that focuses on *Modbus TCP*. In Modbus, each PLC defines a *memory map* representing an internal table of process variables (typically a few thousand). The data model of Modbus proves particularly challenging due to its flat structure and the frequent programmer practice of misusing its simple structure (e.g., arbitrary addressing of variables), which makes it harder to characterize process variables accurately.

Our approach differs from previous analysis techniques in two ways. First, compared to usual content analysis techniques (which consider the statistical features of byte sequences), our approach captures and performs the interpretation of specific byte sequences in the packet payload. The interpretation refers to a consistent reconstruction of actual process values (current measurements and process parameters) from the network trace which can then be analysed in the context of previous process behaviour. Second, compared to other techniques focusing on PLC analysis (e.g., McLaughlin [76] uses PLC code to generate process malware, Liu [69] use process measurements to detect false process data), we use a passive, non-intrusive, approach to extract, reconstruct and analyse communicated process variables.

Contribution The main contributions of this chapter are:

- we propose a new approach for extracting and analysing process information from ICS network messages,
- we perform experiments on a testbed to validate and discuss detector capabilities,
- we perform experiments on network traces coming from real-life process environments to validate and discuss approach limitations.

We structure the remainder of this chapter as follows. In §6.1 we present our approach along with a testbed scenario (§6.1.1) that we use for demonstrating the capabilities of the presented techniques. §6.2 summarizes implementation details, and in §6.3 we present our results. Finally, in §6.5 we discuss our findings, and in §6.4 we present related work. Section §6.6 concludes the chapter.

6.1 Approach

We now present our approach for detecting attacks that aim at manipulating process variables in ICS. The core of our work is based on the assumption that one can infer process semantics from ICS network traffic. We support this assumption based on the characteristics of usual ICS communication. In particular, PLCs exchange comprehensive status information with the ICS server (that updates HMI) on a regular basis, and in turn the HMI issues control commands to the PLCs to initiate process changes. We find both activities reflected at the network level in the form of requests and replies that report and manipulate PLC process variables, encoded in their corresponding network representation. Hence, a network monitor following this communication can derive an understanding of the controlled process that, in principle, is a superset of the information available in the HMI's perspective. For example, except exchanging values of relevant process parameters from the field, HMI and PLCs exchange a set of internal variables that are critical for process infrastructure but are not explicit part of the field process (e.g., program counters, timers, process stages). By contrast to the process field parameters, these variables are not directly monitored by operators. Crucially, malicious commands have to traverse the network. This implies that typically either their cause or their consequences will likewise be reflected as changes to process variables. As a trace of the attack cause, we will see the attack stage when a command modifies variables that control the process. As a trace of the attack consequence, variables tracking the current process state will begin reflecting the malicious update.

Our approach consists of the three main phases: *(i) extraction* distills current variable values out of network traffic; *(ii) characterization* divides the observed process variables into three categories that we examine separately; and *(iii) modelling and detection* derives behavioural models for each variable and reports when new observations deviate from what they predict. We discuss these phases individually in 6.1.2–6.1.4, after first introducing a small testbed setup in §6.1.1 that we use for illustration.

6.1.1 Testbed Scenario

To illustrate our approach, we set up a small testbed environment consisting of one PLC and one HMI workstation. We base the design on a demonstration kit from a known ICS vendor that models a simple water tank setup. The controlled process comprises six plant components (see Figure 6.1): a tank, a heater, two valves, a level sensor and a temperature sensor. The process consists of three operations which are repeated continuously: tank filling, water heating, and tank draining.

Although in a real-world environment an ICS server would collect data from the PLC and the HMI would use the process data collected by the ICS server, to simplify our architecture we configure the HMI to request process updates directly from the PLC once per second. The HMI collects nine variables: two for measurements (i.e., tank level and water temperature), five for control (i.e., valve 1 and 2 status, heater status, tank level setpoint and water temperature setpoint), and two for reporting (i.e., tank level and water temperature high/low alarms).

Table 6.1 shows the relevant parts of the PLC's memory map. The PLC's code implements the following logic:

```
while ( true )
  V1On        := ( TankLevel < TankLevelSP && !V2On);
  HeaterOn    := ( Temp < TempSP && !V1On && !V2On);
  V2On        := ( Temp >= TempSP && TankLevel > 0 && !V1On );
  TankLevelAl := 0;
  if ( TankLevel > hthreshold)    -> TankLevelAl := 1;
  if ( TankLevel > hhthreshold )  -> TankLevelAl := 2;
  if ( TankLevel > hhhthreshold ) -> TankLevelAl := 3;
  TempAl      := 0;
  if ( Temp > hthreshold )        -> TempAl      := 1;
  if ( Temp > hhthreshold )       -> TempAl      := 2;
  if ( Temp > hhhthreshold )      -> TempAl      := 3;
```

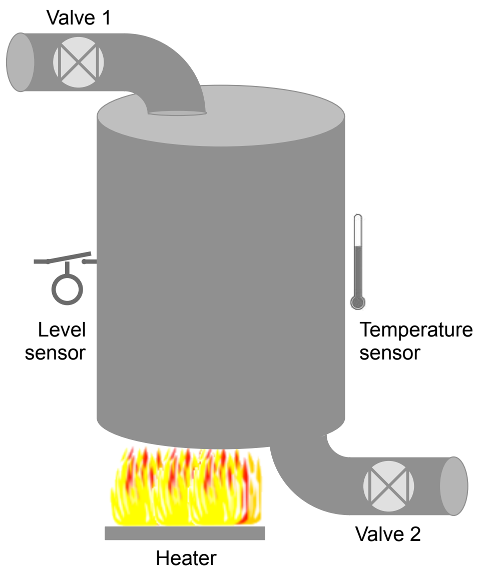


Figure 6.1: Process setup of the testbed environment

For the sake of simplicity we did not include safety constraints in our process logic, except for the alarming system. This also simplifies the illustration of our two attack scenarios: *(i)* changing the level setpoint to overflow the tank; and *(ii)* sending tampered measurements information to PLC to trigger process changes.

6.1.2 Data extraction

The data extraction phase is a preprocessing step that distills the values of process variables out of the ICS network traffic. It consists of two subparts: *(i)* parsing the application-layer network protocol to extract the relevant commands, including all their parameters; and *(ii)* constructing *shadow memory maps* inside the analysis system that track the current state of all observed process variables, providing us with an external mirror of the PLCs' internal memory.

For this work we focus on parsing Modbus, in which each command comes with a set of parameters as well as a data section. Basic parameters include function/sub-function codes that define the operation, an address reference specifying a memory location to operate on, and a word size giving the number of memory cells affected. The data section includes the actual values transmitted, i.e., the current value for a *read* operation and the intended update value for a *write*. We maintain shadow memory maps by interpreting each command ac-

Table 6.1: Testbed PLC memory map

Register	Name	Type	Description
HR0010	V1On	bool	Status of valve 1
HR0011	V2On	bool	Status of valve 2
HR0012	HeaterOn	bool	Status of the heater
HR0020	TankLevelSP	fixpoint	SP tank level (L)
HR0021	TankLevel	fixpoint	Level of the tank (L)
HR0022	TempSP	fixpoint	SP water temp.
HR0023	Temp	fixpoint	Water temp (celsius)
HR0030	TankLevelAl	enum	Alarms tank level
HR0031	TempAl	enum	Alarms water temp.

SP: setpoint

cording to its semantics, updating our current understanding of a PLC's variables accordingly.

The following (simplified) commands from our testbed setup (see §6.1.1) illustrate the extraction step.

```
Time 1: PLC 1, UID: 255, read variable 10, value: 1
Time 1: PLC 1, UID: 255, read variable 11, value: 0
Time 1: PLC 1, UID: 255, read variable 12, value: 0
Time 1: PLC 1, UID: 255, read variable 20, value: 500
Time 1: PLC 1, UID: 255, read variable 21, value: 10
Time 1: PLC 1, UID: 255, read variable 22, value: 800
Time 1: PLC 1, UID: 255, read variable 23, value: 150
Time 1: PLC 1, UID: 255, read variable 30, value: 0
Time 1: PLC 1, UID: 255, read variable 31, value: 0

Time 2: PLC 1, UID: 255, read variable 10, value: 1
Time 2: PLC 1, UID: 255, read variable 11, value: 0
Time 2: PLC 1, UID: 255, read variable 12, value: 0
Time 2: PLC 1, UID: 255, read variable 20, value: 500
Time 2: PLC 1, UID: 255, read variable 21, value: 12
Time 2: PLC 1, UID: 255, read variable 22, value: 800
Time 2: PLC 1, UID: 255, read variable 23, value: 150
Time 2: PLC 1, UID: 255, read variable 30, value: 0
Time 2: PLC 1, UID: 255, read variable 31, value: 0

Time 3: PLC 1, UID: 255, read variable 10, value: 1
Time 3: PLC 1, UID: 255, read variable 11, value: 0
Time 3: PLC 1, UID: 255, read variable 12, value: 0
Time 3: PLC 1, UID: 255, read variable 20, value: 500
Time 3: PLC 1, UID: 255, read variable 21, value: 14
Time 3: PLC 1, UID: 255, read variable 22, value: 800
Time 3: PLC 1, UID: 255, read variable 23, value: 150
Time 3: PLC 1, UID: 255, read variable 30, value: 0
Time 3: PLC 1, UID: 255, read variable 31, value: 0
```

After processing the commands, the shadow memory map will report 18 as the current value for variable 21. Looking more closely at variable 21, we know from the testbed configuration that it corresponds to the *tank level* and, hence, will reflect three distinct types of behaviour: an increasing trend during filling, a constant value during heating, and a decreasing trend during the draining phase. Indeed, our extraction step confirms this expectation: The following list represents a small excerpt from variable 21's values, as extracted from actual network traffic in the testbed:

```
...490,492,494,496,498,500,500,500,500,500,500,
...496,492,488,484,480,476,472,468,464,460,456, ...
```

By using an independent PLC simulator, we have confirmed that these extracted values indeed match what the PLC stores internally over time.

6.1.3 Data characterization

Next, we perform a *characterization* phase that separates variables into different categories based on the knowledge obtained during focus groups sessions with plant engineers. In general, PLC process variables fall into four groups: *(i) control*: variables for configuring plant operation (e.g., device setpoints, configuration matrix); *(ii) reporting*: variables for reporting alarms and events to operators through HMI or other PLCs (e.g., pump load is too high); *(iii) measurement*: variables reflecting readings from field devices and sensors (e.g., current tank level, current water flow), *(iv) program state*: variables holding internal PLC state such as program counters, clocks, and timeouts. While the character of variables varies significantly with their groups, we observe three cases that suggest specific models for predicting future behaviour: most variables either *(i)* change continuously, and gradually, over time; *(ii)* reflect attribute data that draws from a discrete set of possible values; or *(iii)* almost never change. The first is typical, e.g., for sensor measurements, program state and reporting tend to use the second, while the third is typical for process settings (e.g., setpoints). Unfortunately there is no definite resource to directly tell what type of data a variable reflects—recall from Chapter 2 that memory maps are specific to each PLC instance. Thus, we apply heuristics to categorize process variables according to the behaviour we observe. In our testbed we can directly cross-check if the results indeed match the configuration. In the actual environment we examine later in §6.3, we compare our results with labels extracted from PLC project files.

During discussions with engineers, we learned that reporting variables are encoded in bitmaps which, depending on the number of distinct reporting events, appear as a discrete set of 2^k values. We use this information to build heuristics that allows us to distinguish between attribute, continuous and constant series. For us, a series that consists of only 2^k, where $k = 0..8$ discrete values in the whole training set is considered as an attribute series. A special case of an attribute series with $k = 0$ represents constant series (i.e., the whole dataset consists of only one distinct value). A series that consists of more than 2^k distinct values is considered as a continuous series. To characterize different types of parameters in our testbed environment, we set the parameter $k = 3$ (i.e., series with up to 8 values are considered as attribute). In practice, a high value of *k* leads to too specific characterisation while a low value of *k* leads to too coarse classification. We run the characterization on the network traffic of over 2h of operation and classify the 9 variables as 4 constant, 3 attribute and 2 continuous series.

6.1.4 Data modeling and detection

Once we distinguish between constant, attribute and continuous time series, we can proceed with building behavioural models.

Modeling. To model constant and attribute data, we derive a set of expected values (e.g., enumeration set for attribute data, one observed value for constant series). To model continuous data, we leverage two complementary techniques, (*autoregression modelling* and *control limits*, to capture the behaviour of a series and understand operational limits. An autoregressive model is a common technique to capture the behaviour of correlated series, such as successive observations of an industrial process [116]. An autoregressive model of order *p* states that x_i is the linear function of the previous *p* values of the series plus a prediction error term [26]:

$$x_i = \phi_0 + \phi_1 x_{i-1} + \phi_2 x_{i-2} + ... + \phi_p x_{i-p} + \epsilon_i$$

Here $\phi_1, ..., \phi_p$ are suitably determined coefficients and ϵ_i is a normally distributed error term with zero mean and non zero variance σ^2. There are several techniques for estimating autoreregressive coefficients (e.g., least squares, Yule Walker, Burg). We choose to use Burg's method, as it has proven as a reliable choice in control engineering, a field closely related to ICS processes [52]. To estimate the order of the model, we use the common Akaike information criterion [106]. Using the autoregressive model, we can make one step estimation for future values of the process variable underlying the time series. This way, the model can be used to detect stream deviations. However, as for any regression, a set of small changes can take the stream outside of operational limits without exhibiting regression deviations [116]. To address this, we use a complementary strategy, namely Shewart control limits [116]. This is a common technique used for controlling mean level and preventing the system shift from normal operation. The control limits represent a pair of values $\{L_{min}, L_{max}\}$ that define the upper and lower operation limit of the process variable. Typically, the limits are calculated as values that are three standard deviations from the estimated mean.

Detection. For constant and attribute series we raise an alert if a value in a series reaches outside of the enumeration set. To detect deviation in continuous series, we raise an alert if the value *(i)* reaches outside of the control limits or *(ii)* produces a deviation in the prediction of the autoregressive model. More specifically, for estimating the deviation in the autoregressive model, we compare the residual variance with the prediction error variance. The residual variance describes the deviation of the real stream from the stream predicted by the model during training. The prediction error variance describes the deviation of the real stream from the stream predicted by the model during testing. A prediction error variance that is significantly higher than the residual variance implies that the real stream has significantly deviated from the estimated model, thus we raise an alert. We apply this techniques since it is commonly used for the detection of anomalies during instrument operation in control engineering [52]. To estimate the "significance" of deviation we use hypothesis testing (see §6.2).

To illustrate the detection capabilities we test our approach on two semantic attacks crafted for the process operating in our testbed. The first attack effectively consists of a command that changes the tank level setpoint (HR0020 in Figure 6.2). As a result, the tank filling phase (HR0021 in Figure 6.2) continues until the water level overflows the tank capacity. Results show that this attack is detected as *(i)* a deviation in setpoint variable and *(ii)* a value reaching maximal control limit $Lmax$. The second attack consists of a set of commands that set tampered information about the temperature level measurement (HR0023 in Figure 6.3). As a result, the tank filling phase is terminated early, the heating process starts and then immediately stops and the draining process starts. As a consequence, both the heater and the boiler may get damaged. In this second scenario no alarm is generated by the PLC and thus presented to operators. Our results show that this attack is detected as a deviation in autoregressive model.

Limitations. The obtained results demonstrate known capabilities of both approaches [116]. In particular, an autoregressive model is effective for detecting sudden changes (e.g., detection of the second attack). However, a sufficiently slow deviation (i.e., slower than the model order p), can still take the system beyond specification limits without triggering an alarm (e.g., the first attack was not detected by the autoregressive model). On the other hand, control limits are generally a good strategy for maintaining the process mean level (thus, can detect the process drift in the first attack). However, control limits cannot detect a deviation that is within the defined limits of operation (e.g., cannot detect the second attack).

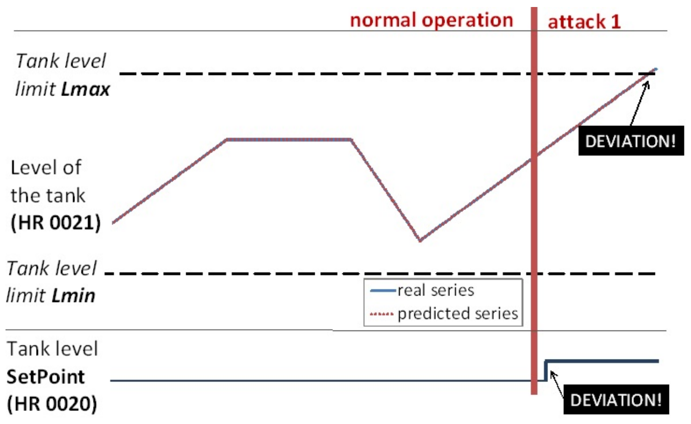


Figure 6.2: Illustration of the configuration change

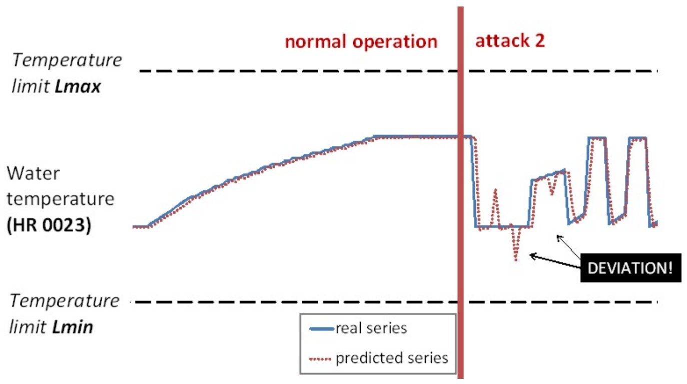


Figure 6.3: Illustration of measurement tampering

6.2 Implementation

We implement a prototype of our approach using a combination of Bro [83] and custom C++ code. We name our prototype as *Sonics (Semantic Network Monitoring in ICS)*. Bro performs the initial *data extraction* step. We develop a Modbus analyser for Bro that extracts the main protocol commands from network traffic and makes them available to scripts written in Bro's custom policy language. We leverage Bro's BinPAC [82] parser generator to automatically generate much of the Modbus-specific code from a corresponding grammar. Our Modbus analyser is fully integrated into Bro, and is now part of the recently released version Bro *2.2*. We also add a custom analysis script to Bro that records each Modbus command into an ASCII-based log file that we then process with external code implementing the subsequent characterization, modelling, and detection phases.

For the *data characterization* we test different values for k in the range $[2..8]$ to distinguish between attribute and continuous series. For our tests with real environments we choose $k = 3$ since for this value our preliminary analysis showed the least number of mismatches for attribute series. For the *data modelling* of continuous series, we build the autoregressive model and derive process control limits. We leverage an open source implementation of the autoregressive model[1]. To derive control limits, we implement Shewart control limits following the description in [116]. For each continuous series we derive an estimate of the behaviour on both the autoregressive and control limit model. Finally, for the *detection* of deviations in continuous series by using the autoregressive model, we use two variance hypothesis tests (commonly known as F-test). For both tests we set $p = 0.05\%$ as significance level.

6.3 Evaluation

Our work represents a first step towards accurately modelling ICS processes from a network vantage point. As such, we are primarily interested in understanding properties of the setting that impact security monitoring at the semantic level, and less in specific true/false positives rates. With that perspective, we evaluate our memory map modelling with two overarching objectives: *(i)* understand the degree to which our approach can successfully predict typical process behaviour; and *(ii)* gain insight into the underlying activity that improves, or weakens, accuracy. In the following we first present the two real-world environments in §6.3.1 that we use for the evaluation. We evaluate the characterization and modelling

[1]Available at https://github.com/RhysU/ar.git

phases independently and describe the corresponding methodology and results in §6.3.2 and §6.3.3 respectively. We do not analyse in depth the data extraction here as it constitutes primarily a pre-processing step which proved to work without problems also with the real-world traffic.

6.3.1 Environments and data sets

Our data comes from two real-life water treatment plants that serve a total of about one million people in two urban areas. They are part of a larger system of over 30 sites controlled by one company. The two plants are of comparable size, and they perform semantically similar tasks (e.g., water pumping, purification, ozone treatment). However, their setups still look quite distinct. They deploy different numbers of PLCs (3 vs. 7), and chose different strategies to divide processes among them. The PLC memory maps differ both between the two environments, and also among PLCs of the same site. While both plants use equipment from the same (well-known) vendor, they deploy different software versions.[2]

We have access to *(i)* 3 day long packet trace from *plant A* and *(ii)* 14 day long packet trace from *plant B*. Both traces contain the complete network traffic captured from the mirroring port of the switches that connect the different PLCs and the ICS servers. The traces include 64GB and 101GB of network traffic, respectively, with bandwidths varying between 9 and 360 packets/sec during the recorded periods. We find two ICS protocols and six non strictly ICS protocols in use. The ICS protocols are Modbus, which is used for communication between PLCs and from PLCs with ICS servers and a vendor proprietary protocol which is used for communication between ICS server and HMI. The non ICS protocols are VNC, SSL, FTP, HTTP SMB and DCOM and are used by the servers and workstations in the network. The non ICS traffic is an negligible fraction of the overall network trace. Of the 20 and 28 hosts active in the two traces, 7 and 11 receive or send Modbus messages. While we see the vendor proprietary protocol in use among hosts that are part of the supervisory infrastructure, we observe only Modbus for all communication involving PLCs. We see three types of Modbus messages in the traces: *read multiple registers* (function code 3), *write multiple registers* (16), and *parallel reading and writing on multiple registers* (23). In the following, we focus our discussion on the Modbus traffic. According to the plant operators, there were no security or operational incidents during out measurement periods. To present our test results, we select 5 PLCs taken from both plants, namely: all 3 PLCs from the first plant, and 2 PLCs from the second plant. The second plant operates with 7 PLC in total, of differing complexity: The number of process variables goes as low as 135 for three of them and as high as 3500

[2]Due to legal constraints, we cannot name the equipment vendor.

in one; the remaining four PLCs operate on approximately 2200 variables. We select two PLCs that representative the most complex and the most simple PLC setup, respectively. We base our choice on the fact that 5 PLCs in the second plant exhibit very regular behaviour, implying that the overall results of modelling per PLC would, in case of including all available PLCs, be unfairly optimistic.

In addition to the traces, the plant operators provided us with *project files* that describe each PLC's memory map layout, exported by PLC programming environments in the form of CSV files holding information on addressing, data type, and process role for each process variable defined by a PLC. In practice, such project files closely resemble the information shown by the example memory map in Table 6.1.

6.3.2 Evaluation of data characterization

Recall that the goal of variable characterization is to apply to each variable (register) the most appropriate technique in the modelling step based on the variable semantics. For example, we want to use a set of values for setpoints and alarms, while we want to use autoregressive model and control limits for measurements and counters. In §6.1.3 we propose a characterization heuristic that separates registers into constant, attribute and continuous time series based on the value of the variables over time. Another approach to data characterization leverages the semantic information contained in the project files. In fact, project files contain as human readable text the semantics of a specific process variable (e.g., variable X is the "throughput rate of pipe Y"). Under the assumption that project files are available, this approach would in theory achieve the characterization goal without the need to process the network traffic. In this section we aim to evaluate the practical applicability and quality of results of our heuristic-based approach compared to the extraction of the same information from project files. To this end, we translate the semantic information of project files into labels that define what we expect the characterization phase to return. As the manual analysis is infeasible (since each PLC in the plant comprises of several thousand of addresses), we assign the labels semi-automatically. We construct a table that maps keywords commonly found in the descriptions into the three categories. For example, we consider descriptions including "measurement", "counter", or "usage" to indicate variables holding continuous values. On the other hand, words like "command" or "alarm" suggest attribute data, and "configuration" indicates a process variable of generally constant value. In total, we identify 24 keywords, which allow us to classify all PLC variables defined in the project files. We then run our heuristic-based algorithm on the network traffic exchanged by three PLCs from the first plant during three days and we calculate an average percentage of variables belonging to the specific series across three PLCs. Our results show that the characterization phase classifies 95,5% of all variables as constant series, 1,4% of variables as attribute, and 3,1% as continuous series. Further analysis with different batches shows that these results remain consistent over time for intervals longer of one day. Table 6.2 shows the comparison with the classification we derive independently from the PLC project files. Our analysis shows that the retrieved information from project files covers only 35% of all observed variables in all three PLCs. As we found out, the main reason for such small coverage are implicit definitions of multiple variables (e.g., PLC programmers use tailored data structures to define a range of variables by only defining the starting variable in project files). We now analyse the results of the comparison. We see an excellent match for constant variables. However, only about half of the continuous variables match, and even less in the attribute category. Closer inspection reveals two main reasons for the discrepancy. First, ambiguities in the project file mislead the keyword-based heuristic. Generally, the descriptions are not standardized but depend on the PLC programmer, and hence keywords sometimes overlap. For example, one PLC has several fields that include the description “ControlForAlarm”. Yet, we consider the keyword “control” to indicate a constant variable, and “alarm” to suggest an attribute series. While this example could be addressed easily, similar ambiguities would remain. This difficulty shows that in practice it is not so easy to extract meaningful semantic information from project files, as initially assumed.

Table 6.2: Comparison of obtained characterizations against the labels from project files

Type of data stream	**Matched process variables (in %)**		
	PLC1a	*PLC1b*	*PLC1c*
Constant	96.2	95.0	97.0
Attribute	33.3	20.0	40.1
Continuous	44.3	56.7	68.3

The second cause of mismatches is that variables that, according to the PLC configuration, contain attribute or measurement data, in practice exhibits a constant behavior. For example, in PLC1b a variable describing the measurement level of a specific tank always remains constant, and hence the characterization step classifies it as such.

Although both approaches (heuristic and configuration-based) show advantages and pitfalls, our heuristic-based approach is the only one that allows us to characterize all the variables. In general, we would be in favour of combining the two approaches, providing our heuristics additional context information. However, for this initial work, we chose not to do so in order to *(i)* understand the step's capabilities on its own, and *(ii)* use the PLC information as a cross-check. Furthermore, as our analysis shows, integration would raise a different challenge due to the inherent ambiguities.

6.3.3 Evaluation of data modeling

To evaluate our modelling approach, we examine how well its predictions capture common plant behaviour. In the first step, we measure the number of deviations that the models report on network traffic representing typical plant operations. In the second, more interesting step, we then dig deeper into these results and focus on understanding the underlying reasons and situations in which our approach *(i)* indeed models process activity correctly; and *(ii)* fails to capture the plant's behaviour, flagging benign deviations as alarms. Our objective here is to gain insight into the capabilities and limitations of our approach, as well more generally into potential and challenges of modelling process activity at the semantic level.

We point out that we do not evaluate the detection rate (i.e., true positives), due to the inherent difficulty of achieving meaningful results in a realistic setting. As actual attacks are rare, we cannot expect our traces to contain any malicious activity (and as far as we know, they do not). However, it also remains unrealistic to inject crafted attacks into the traces; we would be limited to trivial cases like those already demonstrated in §6.1.4 (which our detector would find for the same reasons as discussed there). On the other hand, we cannot inject more complex attack data sets, like from simulations carried out elsewhere, as any ICS activity has little meaning outside of its original environment (e.g., recall how Stuxnet tailored its steps to its specific target setup; interpreting that activity inside a different setting would make little sense). Hence, we see more value in our semantic analysis of capabilities and limitations than measuring detection rates on unrealistic input.

We now start with measuring the number of deviations that the models report. Generally, we consider our approach to generate an "alert" on a process variable when, at any time during testing a batch of data, an observation deviates from the prediction—i.e., when observing an unexpected value for constants and attributes, or a value outside the autoregressive/control limit models for a continuous time series. We perform 3-fold cross validation using the *rolling forecasting procedure* [53] on a set of 3-day batches of data extracted from the two plant's network traces. The rolling forecasting procedure is a common technique for performing cross validation in time series and implies two key modifications compared to the traditional cross validation procedure: *(i)* the training length is increasing through different folds and *(ii)* the training set never includes data occurring after test set (i.e., the model should not train on the data that is *later* than test data). The first two batches of data come from the first plant, each representing a randomly chosen continuous range from the first and the second weeks, respectively. The third batch represents the complete trace from the second plant (recall that we only have 3 days of network trace available from that plant). At a technical level, a 3-day batch size gives us a reasonable volume of data suitable for processing repeatedly with our implementation. At an operational level, operators confirm to us that one day matches a typical PLC work cycle.

In Table 6.3 we summarize the results of the testing across different behaviour models. For each pair of category and PLC, we compute the percentage of variables deviating, showing mean and standard deviation over the batches.

In the following sections, we discuss the three categories separately.

Table 6.3: Testing model capabilities

	Deviating variables across different types of series (mean %/ st.dev)		
	Constant	*Attribute*	*Continuous*
PLC 1a	0.5 / 0.29	19.05/0.2	57.49/5.78
PLC 1b	0.31 / 0.04	19.80/1.4	44.64/5.41
PLC 1c	0.14 / 0.02	19.55/4.0	37.58/2.98
PLC 2a	0.64 / 0.0	26.92/0.0	63.63/0.0
PLC 2b	0.0 / 0.0	0.0/0.0	0.0/0.0

6.3.3.1 Constant series

Our results show that by far the most variables that we classify as constant indeed stay stable over time. Examining the small number of deviating variables in this category, we observe two main causes for false positives: *(i)* configuration changes, and *(ii)* misclassifications of the variable type. The former typically relates to a previously unobserved status change of specific field device. For example, in PLC1 we find a pump device that is enabled only after about 40 hours of normal operation. In another similar, but more extreme case, we observe a burst of alarms: 60 variables all trigger at the same time even when the training was longer than two days. Upon closer inspection we find them all to belong to a "configuration matrix", a large data structure that defines an operation mode in terms of a set of values controlling multiple devices simultaneously. As it turns out, it is a single packet from the HMI that performs a "multiple register write", triggering the deviation for all of them. We verified that this occurrence represents the only time that the operators change the matrix over the two weeks interval that our traces cover. As such, it is a significant yet rare change that one could either whitelist or decide to keep reporting as a notification.

The second cause for false alarms represents shortcomings of our data classification phase. For example, during one of the folds in PLC2a we find that the tool misclassifies 9 out of 15 measurement variables representing aggregated flow information as constant due to a lack of activity during the training period. Similarly, in the same fold we find that 7 out of 18 device statuses (thus assumed attribute data) are misclassified as constants. This is because in both cases the values remain constant for more than 20 hours, yet then change during the testing period. Interestingly, we see several such situations that first trigger an alert for a configuration change (e.g., the status of a filter in PLC1b changes for 15mins after it has spent 21 hours in a previous state), followed by a burst of further ones reflecting the change being in effect now (i.e., variables representing activity linked to that filter start to deviate from constant behaviour: status, volume, throughput/hour, total throughput). In this case, the two main causes of errors are hence related.

Discussions with the plant operators confirmed that daily *cleaning* activities on that PLC might cause such sudden changes for a short amount of time. The misclassification in this case was avoided in the next fold with a longer training interval, confirming that when training spans the corresponding work cycle, the modeling of constants indeed works as expected.

6.3.3.2 Attribute data series

Our test show a stable, but reasonably high number of deviating attribute series across all tested folds. By sampling a subset of deviating variables, we find that the main cause for mismatches in this model concerns continuous variables misclassified in the data characterization phase: due to slow process character some of them exhibit only a limited number of distinct values during a training interval, and are thus wrongly labelled as attribute data (e.g., variables describing time information in the form of date and hour). Apart from this scenario, the targeted variables (thus commands and alarms) are captured correctly for training longer than one day. However, we note that a blind spot for our current attribute models are alarm/command *sequences*. With attribute data, sequences carry important semantic information as such variables often encode the current process state within a series of steps (e.g., alarm type X raised to operator, operator acknowledged, alarm cleared, state normal). Since some alarms require operator acknowledgement, a sequence that, e.g., omits that feedback would be suspicious. For such variables, we attempted to apply the continuous models as an alternative, but they only further reduced the accuracy. That however is not surprising: for attribute data ordering matters, yet typically not the actual timing (e.g., an operator may acknowledge an alarm at any time). Hence, we consider a sequence-based analysis of process states as a promising extension of our current attribute model.

When examining the variables that describe attribute data in detail, we discover further structure that our modelling does not currently key on, yet which we consider a promising venue for exploiting in the future. During focus group sessions with plant engineers, we learnt that alarms and commands are typically encoded in bitmaps, and we indeed find this reflected in the network traffic. For example, for a variable that the PLC project file refers to as *"various status notifications from PLC3 to server"* , we observe a series of what, at first, appears like an arbitrary set of values: 40960, 36864, 34816. However, when aligned in binary format, the values map to:

```
1010 0000 0000 0000
1001 0000 0000 0000
1000 1000 0000 0000
```

This representation reveals patterns of bits that are constant (e.g., the first bit indicates that PLC1c is active). If we integrated this structure into the characterization step, we would be able to refine the attribute modelling significantly. In other words, some variables require a different granularity than just their numerical value for capturing their semantics.

6.3.3.3 Continuous data series

We now summarize results from the two models considering continuous time series: control limits and autoregression. We observe that autoregressive model generally alerts more frequently than control limits. In fact, the control limits contribute to only 28% of all deviations in continuous series.

Control Limits model Our results show that, apart from the overlap with autoregressive model, control limits report additional 3% series as deviating. Our inspection reveals that these variables represent series that are increasing trends during the whole available trace. For such variables, approaches in statistical process control commonly accept that the series should be modelled according to their regression nature only, and not on the control limits. This means that these variables should be whitelisted in this model. An alternative approach would be to obtain absolute process limits (e.g., from process engineers) and enforce those limits for series control.

Autoregressive model Our results show that the autoregressive model has no difficulty in modelling internal process stage and counter variables. Differently from alarms and commands, which occur in relation to human interaction, these variables are connected to the automatic process behaviour with highly correlated and regular sequences of values which are straight-forward to capture. Of all the deviations, we only find one related to a counter variable (which delayed an increment for 2 seconds): since the behaviour of this counter was extremely regular in the training data, the model detector was not tolerant against the delay.

The remaining deviations refer to measurement variables. By inspecting them more closely, we distinguish three groups. A first group of deviations refers to variables that autoregression fails to model well, independently of the training interval. This group accounts for ~ 80% of the deviations. We believe the autoregression model fails to model the behaviour of these variables because we observe the same variables reported consistently across all folds and batches. By sampling deviating variables, we find out that 70% of them have a presumably random behaviour with high oscillations. The remaining 30% behave as series that are nearly constant (or slow trends) whose deviation is captured when a sudden peak occurs. To understand the semantics of this behaviour, we look into project files. It turns out that, according to the project files, all these variables correspond to *floating point* measurement values of the same set of field devices (e.g., measurement from devices concerning purification in PLC2a). In Modbus, floating point values are represented by a set of registers. The vendor specification for our PLCs states that a single precision floating point is encoded according to the IEEE 754 standard [10] in *two* registers, which represent the actual value as a product of sign, exponent and mantissa. In Figure 6.4 we show how these three components are projected onto a pair of registers. To understand how this specification relates to our series, we find a pair of registers in project files that are labelled as a higher and lower register of the same floating point value. When reconstructed, the resulting value represents a value with an increment in range of 10^{-4}. Independently, the two variables describing the behaviour of the two registers look quite different. In particular, while one variable looks pseudo random (this illustrates the noisy fraction behaviour of RegisterB from Figure 6.4), the second variable looks nearly constant (this illustrates the exponent part of the RegisterA in Figure 6.4). We point out that, depending on the measurement noise, some registers containing (half of a) floating point variable are not suitable to be modelled raw. Our current tests show that, in the analysed environments, this refers to approximately %50 of all measurements (since the same variables are consistently alerted over all batches). To address this problem, we would need to reconstruct the floating-point values as part of the data extraction phase. Unfortunately, it is technically challenging to find a unique approach to identify the two halves of a floating point variable. Vendors use different approaches and even within the same vendors, programmers might follow different conventions. For example, in the analysed PLC project files we observe the use of at least three different conventions: use consecutive pairs of registers with *(i)* the register with even address as the upper register, *(ii)* the register with the odd address as the upper register, or *(iii)* for a set of variables, put all upper registers first and then all the lower registers.

By observing the peaks in the percentage of deviating variables across different folds, we find the second group of deviations. In more detail, we wanted to understand if the deviating variables refer to multiple field devices (e.g. one variable per field device) or to a few field devices (e.g. multiple variables per field device). We find out that in all analysed cases, all the deviating variables correspond to a limited set of devices (e.g., a peak of 9 deviating variables in PLC 2a semantically describe different aspects of only one field device, a pump). We also find several situations in which multiple variables are linked together and hence exhibit similar (even identical) behaviour. For example, we see 10 ozone filters whose flow is described by the same autoregressive model, and whose deviation occurred at the same time and thus resulted in a peak of deviations in one PLC.

In either case, a more sophisticated model could aggregate variables by incorporating more process context information into the detection approach, for example by extracting information from project files or performing a vertical analysis of variables, scanning for patterns of similar behaviour or grouping together variables that refer to the same device.

Finally, the third group of deviations is related to variables that behave differently over time. For example, the value remains nearly constant for 20 hours, then it fluctuates for 15 minutes and then it reverts to a constant value. We validate that this group of variables is the same group which was mischaracterised as constant series, as we described in §6.3.3.1. The data series of these variables is not stationary in a wide sense, and thus it is generally not well suited for autoregressive modelling. To address this, we envision the adoption of multivariate modelling approaches.

6.4 Related work

Statistical process control is a well established field which focuses on modelling and monitoring parameters in industrial processes. Researchers use various techniques (e.g., time series analysis, outlier detection, control limits, feedback adjustments) to model and validate safety of industrial processes [26, 116]. Cardenas [31] analyses process measurements and evaluates the process tolerance

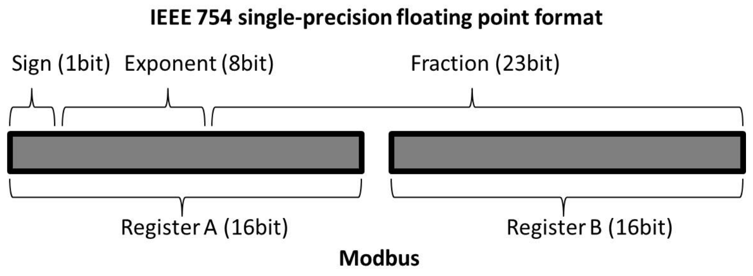


Figure 6.4: Representation of single precision floating point in Modbus

towards attacker efforts. Liu [69] analyses how false data from the field influences controllers. Different from these approaches, we aim at reconstructing process behaviour from network trace.

In computer science, a set of prior work focuses on understanding ICS communication patterns, showing that communication flows indeed reflect the regular and (semi-)automated character of process control systems [19]. Other efforts focus on analyzing security threats in ICS. For example, some authors analyze protocol vulnerabilities [11, 21, 122], explore the lack of compliance to protocol specifications in different PLCs [29, 107] and the feasibility of device fingerprinting [2]. To address security threats some efforts exploit communication patterns for anomaly-detection [109, 67] However, the effects that one can find at the flow-level remain limited; detecting semantic process changes requires inspection of the application layer. Consequently, some authors propose to parse network protocols for extracting information that can highlight changes to the process environment. For example, authors perform partial protocol parsing to enumerate functionality that Modbus clients use, aiming to detect unexpected deviations in requests sent to PLCs [32] and interpret events on a higher level [45], fingerprint and monitor current device configuration remotely [81, 99]. Düssel et al. [37] propose using application syntax (not *semantics*) for network-based anomaly detection; they use Bro to parse RPC, SMB, NetBIOS services inside process control networks, but do not further examine ICS-specific protocols. In terms of classic IDS signatures, DigitalBond provides Snort preprocessors that add support for matching on Modbus/DNP3/EtherNetIP protocol fields [128]. McLaughlin in [75] proposes a host-based approach for analyzing PLC behavior; they use a set of methods to reconstruct PLC configuration and process safety interlocks from PLC program code to build semantically harmful malware.

To the best of our knowledge there are only two prior efforts that extract and analyze process data values from network traffic. First, Fovino et al. [40] track current values of selected critical process parameters and thereby maintain a virtual image of the process plant that they then use to detect predefined undesirable system states. Their method of annotating critical parameters and states requires manual, intensive involvement of plant experts and thus remains expensive and likely incomplete. Second, Gao et al.[43] use neural networks to classify between normal and tampered process variables that are under injection attacks. Both works validate their approaches in controlled testbed environments. By contrast, we perform an unsupervised modelling on real world environment. We analyse all available process variables and provide in depth discussion of semantics that influence our results.

6.5 Discussion

In this section we discuss the evaluation results as well as other aspects that relate to the applicability of our approach, namely the threat model and generalization to the domain and other industrial control protocols.

At a higher level, our findings provide a perspective on ICS environments that may be unintuitive to security researchers. We find a common assumption that ICS activity follows regular patterns that should be straight-forward to model with approaches like the one we deploy in this work. However, we show that when looking at the core of the process control, inevitably, the real world is more complex than one might assume, exhibiting plenty of irregularities, semantic mismatches, and corner-cases that need care to get right. This is a well known challenge in the process control community: operational safety typically requires intensive manual work on understanding and estimating the process behaviour before enforcing any controls.

In relation to our threat model, we acknowledge that our approach does not explicitly detect PLC code updates. However, a PLC code update is an unusual event which involves issuing special commands to the PLC (function codes in Modbus). It is therefore trivial to detect such events by extracting the command from application layer messages and whitelisting the ones that are used. We also note that some process manipulation attacks remain outside of what our approach can conceptually find. By gaining control of a PLC, Stuxnet recorded the value of measurement variables during normal operation and replayed the recorded values after triggering the process variation to hide its traces. If the replayed values emulate the normal pattern over time perfectly, it will not be possible to detect the anomaly by any of our models. However, in case that the tampered measurement is not accurate over a long period of time (e.g., because the period of sampling was too short), our approach would still have a chance to detect the attack.

We argue that more extensive tests could be conducted with data coming from other environments, and in particular from other industrial domains. We are confident that our approach is applicable to other environments since we do not use any assumptions that are specific to the water treatment only. The choice of focusing on the Modbus specification for designing our approach is beneficial for extending its use to other (more recent) industrial control protocols. In fact, the data model of Modbus is generic and only defines two types of process variables (registers and coils). This makes decoding Modbus messages easy, yet renders it hard to extract meaningful semantics (see the problem with floating point values discussed before). Other industrial control protocols (e.g. DNP3, MMS and IEC104) define a much more structured data model, with a complete set of variable types (booleans, integers, floating points, etc.) and more fine grained variable semantics (e.g. measurements, setpoints, alarms, etc.). With more information extracted from protocol messages the impact of errors in the characterization step would decrease, and in some cases it might not be needed at all.

6.6 Conclusion

In this Chapter we propose an approach for monitoring ICS process by extracting and modelling the behaviour of process parameters from network traffic. We evaluate our approach as a tool, *SONICS*, and evaluate the performance of the tool on a small testbed and using real world data from a water plant. Putting our findings into perspective, we show that we can reliably monitor 98% of the process control variables used in two real-world plants. 95% of these variables are configuration settings, and according to plant operators changes in configurations indeed represent one of the most direct threats for plant control. In fact, our tests confirmed that we can detect a (although legitimate) configuration change happening at one of the plants during the monitoring period. The remaining 2% of the variables are still challenging to model with the presented approaches. When analysing the causes, we can isolate a number of reasons for the deviations, rooted not only in the models themselves but sometimes also in the data characterization and extraction phases. Specifically, we see mismatches between *(i)* training periods and activity cycles; *(ii)* data representation and process semantics; and *(iii)* chaining and cluster effects that cause individual deviations to propagate to a large number of variables.

Summarizing, our approach lays the ground for detecting critical attacks on industrial control systems that current approaches can fundamentally not find.

Chapter 7

Concluding remarks

We now summarize the contributions of the thesis, in relation to the main Research Question discussed in Chapter 1. We also put our findings in a broader perspective and highlight future research directions in the area of ICS cyber security.

7.1 Summary

The security of industrial control systems has gained an increasing attention in recent years. This was mainly due to *(i)* the public release of vulnerabilities in current ICS (e.g., via Basecamp [122] and individual activists [123]) and *(ii)* real-world incidents (e.g., Stuxnet [77]). Most efforts in the cyber security literature focus on adapting best practices from traditional IT to the ICS environment. These works often consider *information* as the main security asset (e.g., improve security of ICS by segmenting network, improving access control). By contrast, this thesis focuses on the *process* as the main security asset. In the introductory chapter we formulate the following research question:
"How to design techniques for the detection of process attacks in ICS?"

We argue that the main requirement for building an effective cyber security technology in the ICS domain is the inclusion of process knowledge. This is because the lack of process context information is the main limitation for evaluating if the observed activity (e.g., in log, network, sensor data) is going to affect the underlying process. For example, an observation about a change in one of the input parameters, captured over the network, does not mean much for the monitoring device if we cannot interpret the observed value (i.e., by knowing what that parameter does for the process and what is the normal value range).

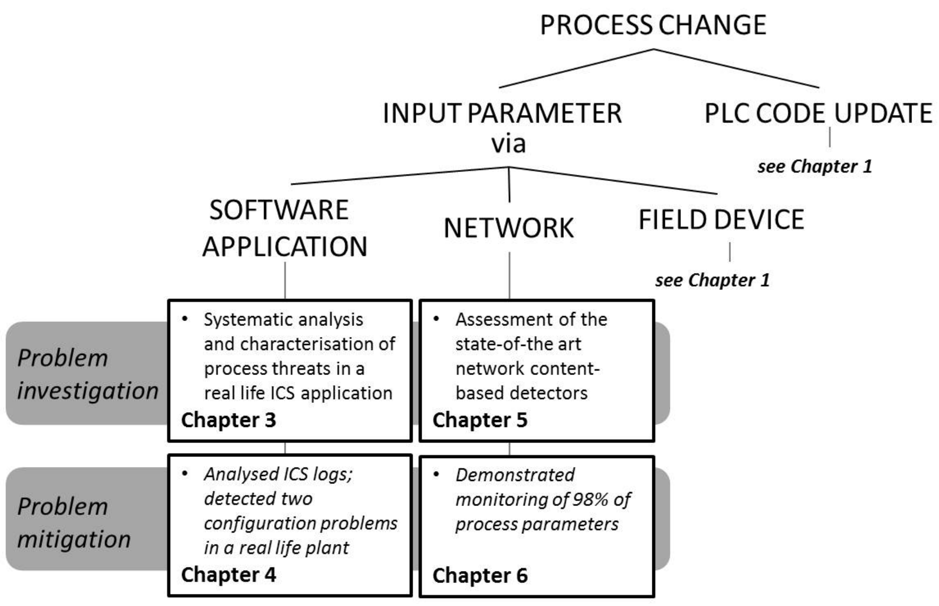

PROCESS CHANGE
INPUT PARAMETER via
PLC CODE UPDATE
see Chapter 1
SOFTWARE APPLICATION
NETWORK
FIELD DEVICE
see Chapter 1
Problem investigation
• Systematic analysis and characterisation of process threats in a real life ICS application
Chapter 3
• Assessment of the state-of-the art network content-based detectors
Chapter 5
Problem mitigation
• Analysed ICS logs; detected two configuration problems in a real life plant
Chapter 4
• Demonstrated monitoring of 98% of process parameters
Chapter 6

Figure 7.1: The outline of thesis results

The ultimate goal of our work is to improve detection techniques for addressing process attacks. This is challenging as the concept of process threats was unknown in the context of IT security so far. As a consequence, IT experts generally lack the necessary knowledge for building suitable techniques for the ICS domain.

To tackle the challenge, we perform two types of studies. First, in Chapters 3 and 5 we perform problem investigation. In particular, we extensively analyse and characterise how different attack vectors (i.e., attacks via user application and network) become manifest in the system (i.e., identify discriminative clues in the data that can be used to distinguish between benign and malicious activities). Second, in Chapters 4 and 6, we develop two approaches for addressing process threats (*MELISSA* and *SONICS*). More specifically, *MELISSA* is designed to analyse event logs and highlight unusual user activities. *SONICS* monitors process indicators to detect deviations in daily operations. Figure §7.1 highlights the results of the thesis per chapter. In particular:

- We identify and analyse different classes of process threats occurring via misuse of ICS application software. We validate the findings during focus groups sessions with ICS experts (Chapter 3)

- We present the *MELISSA* approach which analyses ICS logs to highlight suspicious user activities. During validation, we identified two configuration problems in a real-life ICS plant that were previously unknown (Chapter 4).
- We assess the capabilities of the state-of-the art network-based detectors and perform an in depth analysis of results to understand why different attacks can(not) be addressed by the proposed approaches (Chapter 5).
- We develop the *SONICS* method to monitor process behaviour by analysing network traces. We demonstrate that 98% of process parameters can be consistently monitored by using our approach. In addition, we discuss why the remaining 2% of process parameters stay as a challenge for our approach (Chapter 6).

In relation to the main research question, the work in the thesis can be subsumed in four main steps. First, we decompose the problem of process attacks. More specifically, we analyse how a malicious activity can affect an industrial process and what are potential attack scenarios for accomplishing this. We identify two general scenarios that may cause a process change: the use of an *input parameter* or *PLC code update* (depicted in Figure §7.1). By considering the location of the input, we further distinguish three subscenarios for accomplishing an input process manipulation: an attack via *software application*, *network* or *field device*. Second, to characterise the targeted scenarios, we perform problem investigation aiming to understand how process attacks are manifest in the system behaviour. Here we empirically show that process attacks cannot be addressed by the state-of-the art approaches for network content analysis. The main cause of this is the lack of understanding of process context. Third, we derive two approaches, as a proof of existence, to demonstrate how process attacks can be detected. In particular, we show that it is feasible to infer the process context information from network communication in ICS. Also, we identify patterns in user behaviour that can be leveraged to detect undesirable activities. Finally, we analyse the performance of the proposed approaches to identify limitations and gather insights that can be valuable for future works. For example, we find that *(i)* an unified modelling of the behaviour of process parameters is challenging (since the parameter behaviour origins from different causes such as: field measurement, program counter, sporadic event sequences), *(ii)* the extraction of process information might be inconsistent (since vendor implementations might differ in the way how the data is encoded) and *(iii)* the success of the data interpretation depends on human factor (e.g., we find three different policies of addressing process variables in one PLC).

From an engineering perspetive, the thesis explores *the use of independent tool suites*. The choice of designing an independent tool, against an integrated

ICS software, has two practical advantages. First, an independent monitoring can provide an orthogonal view to the existing safety monitoring mechanisms and thus improve the overall situational awareness. Safety mechanisms protect the system against hazards (potential process problems during normal operation). These methods do not consider malicious activities over the IT equipment, and thus cannot solely be used to protect an ICS against malicious activity. Although safety monitoring represents an excellent starting point for performing the security analysis as well, we argue that the safety monitoring should, to some extent, be decoupled from the security monitoring. Such design, as a result, also lowers the possibility of manipulating the process by compromising both safety and security protection mechanisms at the same time. Second, the deployment of independent tools is more suitable in the ICS context. This is because process components seldom change and have a life cycle of 10-20 years. In this setup, any upgrades from the vendor side (such as improving monitoring mechanisms) are hard to implement. As a practical result, the thesis implements two proof-of concept tools that can be deployed in ICS environments. The current implementations are indeed suitable for only specific vendors (i.e., that support the used log format) and a protocol (i.e., Modbus TCP). However, we argue that both approaches can be translated to other ICS protocols (e.g., by implementing PLC data model representation) and vendor environments (e.g., by adjusting the log format and keywords for evaluating the event severity). We also acknowledge a potential concern regarding the choice of building independent techniques for process analysis. Basically, any additional processing in the sensitive ICS environment can potentially interfere with normal ICS communication and operation. To avoid this, our approaches use passive, non-intrusive techniques for data gathering and analysis.

Summarizing, this thesis performs pioneering work in exploring suitable techniques for detecting process attacks in ICS. Various challenges from the areas of IT security and process monitoring play an important role in designing a suitable approach that is capable of detecting process attacks. We present approaches that tackle these challenges and, more importantly, provide insights into problems, opportunities and pitfalls that are relevant for the area and can be leveraged as valuable knowledge in future works.

7.2 Future research questions

We see several interesting research directions that relate to the topics discussed in this thesis:

Relationships amongst process parameters In Chapter 6 we show that process parameters can consistently be extracted from network. The challenge we identify here relates to the further analysis of parameter semantics (e.g., what is the relation between two extracted parameters). We see two possible research directions.

First, the information about the relationship amongst different process parameters can be found in PLC code (since this code holds the logic responsible for operating different process parameters). By feeding the information on relationships between different parameters into a network monitoring tool, we can derive models that will be capable of monitoring *(i)* if the process parameters behave over time as expected and *(ii)* if the PLC code has changed (by checking if the extracted parameter relationships still hold in the current parameter measurements). A possible drawback of this approach is the need for extracting and interpreting the PLC code into usable knowledge (i.e., the code might not be accessible).

Second, the relationships amongst process parameters can be learnt by a "code-agnostic" manner. The current analysis in Chapter 6 models the behaviour of each process parameter over time. We call this analysis as "horizontal". By performing a "vertical" alignment of process parameters (i.e., comparing values of two parameters at the same point in time), we can infer the relationships amongst different variables. The drawback of this approach is the fact that the derived models may consist of false relations (e.g., relationships which accidentally appeared in data, but do not represent true process semantics).

Analysis of sequences In Chapter 4 we analyse user activity. In this initial work we focus on detecting *single* undesirable user actions. As the next step, we see the analysis of sequences in user behaviour as a promising research direction. The sequence analysis is particularly suitable for the ICS domain since the user behaviour is constrained by the nature of the process: strictly defined, repetitive procedures. As a result, we foresee a set of patterns that represent usual user activity on a particular system. In practice, we see two potential challenges for performing sequence analysis of user activity. First, the logging system might be configured in such a way that some some user actions are not recorded. This would influence the completeness of the results. Second, the practical policies in some plants might allow operators to use a shared user credentials (as we saw it in several plants we had access to). This means that the analysed behaviour consists of the activity of several users, which could be misleading for sequences analysis.

Safety in security Intrusion detection is an established area of traditional IT security. However, the number of works that extend the intrusion detection to the ICS domain, by fully taking into considerations the specifics of the ICS field, is low. As a practical consequence, the deployment of techniques that address process attacks in the real world is at an early stage. On the other hand, process control safety has been researched for over 50 years in the field of control engineering [22]. These works ultimately aim at designing a safe and reliable system that is capable of maintaining optimal working conditions and recovering from failures (e.g., hazards due to equipment failures, human error, natural catastrophes). However, the safety domain does not explicitly cover threats that include malicious activity. This means that the designed safety models are not necessarily capable of addressing process threats with malicious intent. For example, a knowledgeable attacker can take the system into an undesirable state by performing actions that will make different process component fail at the same time, a situation that is highly unlikely to occur in normal working conditions (e.g., considering only the likelihoods of component failures under normal working conditions, as used in safety).

We argue that the experience from the safety domain can be exploited for securing ICS since safety models carry valuable semantic knowledge about the process. In a broad perspective, this thesis does pioneering work on including process monitoring into ICS security analysis. We believe that future efforts in this direction can bring useful results. For example, models used for safety monitoring (e.g., mathematical models describing the behaviour of process parameters) can enrich the context of information extracted in Chapter 6 and thus improve the evaluation of the parameter behaviour.

Anomaly-based intrusion detection in the ICS domain In general, an ICS environment is more suitable for the deployment of anomaly-based approaches than a traditional IT environment. For example, due to the limited number of functionalities in ICS, the behaviour of the system can easier be modelled.

To the best of our knowledge, there are no studies that comprehensively analyse and evaluate the usability of anomaly-based solutions in the ICS domain. To some extent, this thesis considers different aspects of usability. For example, we analyse implications of *operator time* required to analyse the results (through the concept of false positives in Chapters 4–6). Also, we consider the *level of knowledge* required for interpreting the results. We acknowledge that the current outputs of *SONICS* still does not provide actionable information to the operators (e.g., *SONICS* alerts a parameter behaviour without translating the parameter ID to an actual process component).

Nevertheless, we argue that a more comprehensive analysis of usability aspects is needed as it can highlight operational requirements in an ICS and represent a guideline for future works in this area.

Alongside with research directions that relate to the topics of this thesis, we see several, in our opinion important, open issues in the ICS domain.

7.2.1 Common open issues in ICS cyber security

We point out that one of the main limitations of the work presented in this thesis is the lack of comprehensive validation for the proposed techniques (i.e., the performed experiments used only benign data traces from real life plants). We see two issues that, in our opinion, limit the current cyber security research in the ICS domain:

Public dataset There are only few public datasets for testing security solutions in ICS. The most used attack dataset by Digital Bond [128] comprises attacks exploiting vulnerabilities in protocol implementations. However, this dataset does not consist of process attacks. In fact, even if such attack would exist, it would unlikely be applicable for any other environment (since processes differ in each environment).

To the best of our knowledge, there are no public datasets capturing ICS operation longer than few minutes. We acknowledge a similar problem in traditional IT (e.g., a lack of public datasets due to privacy concerns in enterprise systems). This is a problem as it limits the possibilities to compare results of different approaches and have measurable improvements in the field.

As a potential solution to the lack of attack datasets, we see a framework that can translate different threats to specific plant instances. For example, a generic attack (e.g., an override of a user command) via the framework could be translated to a tailored process attack (e.g., an overflow or an explosion).

Classification of ICS We acknowledge that there are no widely accepted classifications of ICS environments and components.

The most intuitive classification we found relates to the domain of application (e.g., water, energy, oil, building). Although this classification highlights similarities in components and general features in one domain (e.g., processes in water domain are generally less dynamic than processes in energy domain), the character of processes often differ in one domain (e.g., water treatment compared to water distribution).

Another classification relates to different types of ICS with respect to system architecture (e.g., SCADA, DCS, PCS). We describe the differences amongst these terms in Chapter 2. This characterisation is too general and thus inadequate because: *(i)* the architecture of each ICS type evolutes over time resulting in a number of mixed solutions, *(ii)* different vendors design own versions of common architectures and *(iii)* plant operators adjust default vendor architectures creating hybrid types.

We argue that a comprehensive classification is necessary for performing *(i)* interpretation and evaluation of proposed solutions (e.g., a high false positive rate of a detector can be manageable in water treatment, but not in smartgrid) and *(ii)* generalisation of knowledge across different domains (e.g., it is hard to reason if an approach can effectively be applied to another environment).

Author references

Journal publications

[1] D. Hadžiosmanović, D. Bolzoni, and P.H. Hartel. A Log Mining Approach for Process Monitoring in SCADA. *International Journal of Information Security*, 11(4):231–251, 2012. **(Subsumed by Chapter 3 of this thesis)**.

Refereed conferences

[2] M. Caselli, D. Hadžiosmanović, E. Zambon, and F. Kargl. On the Feasibility of Device Fingerprinting in Industrial Control Systems. In *Proc. International Conference on Critical Information Infrastructures Security*, LNCS. Springer Berlin Heidelberg, 2013. (to appear).

[3] D. Hadžiosmanović, D. Bolzoni, and P.H. Hartel. MEDUSA: Mining Events to Detect Undesirable uSer Actions in SCADA. In *Proc. Recent Advances in Intrusion Detection*, volume 6307 of *LNCS*, pages 500–501. Springer Berlin Heidelberg, 2010. [extended abstract] **(Subsumed by Chapter 4 of this thesis)**.

[4] D. Hadžiosmanović, D. Bolzoni, P.H. Hartel, and S. Etalle. MELISSA: Towards Automated Detection of Undesirable User Actions in Critical Infrastructures. In *Proc. of the European Conference on Computer Network Defense, EC2ND*, pages 41–48, USA, 2011. IEEE Computer Society. **(Subsumed by Chapter 4 of this thesis)**.

[5] D. Hadžiosmanović, L. Simionato, D. Bolzoni, Z. Zambon, and S. Etalle. N-Gram Against the Machine: On the Feasibility of the N-Gram Network Analysis for Binary Protocols. In *Proc. Research in Attacks, Intrusions,*

and Defenses, volume 7462 of *LNCS*, pages 354–373. Springer Berlin Heidelberg, 2012. **(Subsumed by Chapter 5 of this thesis)**.

International workshops

[6] D. Hadžiosmanović, D. Bolzoni, S. Etalle, and P.H. Hartel. Challenges and Opportunities in Securing Industrial Control Systems. In *Proc. of the IEEE Workshop on Complexity in Engineering, COMPENG*, pages 26:1–26:6, USA, June 2012. IEEE. **(Subsumed by Chapter 3 of this thesis)**.

Technical reports

[7] D. Hadžiosmanović, R. Sommer, E. Zambon, and P.H. Hartel. Through the Eye of the PLC: A Network Monitoring Approach for ICS. Technical Report TR-13-003, International Computer Science Institute, Berkeley, 2013. **(Subsumed by Chapter 6 of this thesis)**.

General references

[8] ISO/IEC 7498-1:1994 Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model, 1994.

[9] Control Systems Cyber Security: Defense in Depth Strategies. Technical Report EXT-06-11478, DHS - Idaho National Laboratory, 2006.

[10] IEEE Standard for Floating-Point Arithmetic, IEEE 754. `http://dx.doi.org/10.1109/ieeestd.2008.4610935`, 2008. [acessed October 2013].

[11] Common cybersecurity vulnerabilities in industrial control systems. Technical report, U.S. Department of Homeland Security, Control Systems Security Program, 2011.

[12] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. 20th International Conference on VLDB*, pages 487–499. Morgan Kaufmann, 1994.

[13] American National Standards Institute. BACnet - a data communication protocol for building automation and control networks specification. `http://www.bacnet.org/Addenda/Add-2001-135d.pdf`, 2004. [acessed October 2013].

[14] J. Anderson. Computer security threat monitoring and surveillance. Technical report, Fort Washington, Pennsylvania, 1980.

[15] D. Ariu, R. Tronci, and G. Giacinto. HMMPayl: An intrusion detection system based on Hidden Markov Models. *Computers and Security*, 30(4):221 – 241, 2011.

[16] N. Athanasiades, R. Abler, J. Levine, H. Owen, and G. Riley. Intrusion Detection Testing and Benchmarking Methodologies. In *IWIA '03: Proc. 1st IEEE International Workshop on Information Assurance*, pages 63–72. IEEE Computer Society Press, 2003.

[17] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security*, 3(3):186–205, 2000.

[18] C. Balducelli, L. Lavalle, and G. Vicoli. Novelty detection and management to safeguard information-intensive critical infrastructures. *Int. J. Emergency Management*, 4(1):88–103, 2007.

[19] R. Barbosa, R. Sadre, and A. Pras. Difficulties in modeling SCADA traffic: a comparative analysis. In *Proc. of the 13th international conference on Passive and Active Measurement*, PAM'12, pages 126–135, Berlin, Heidelberg, 2012. Springer-Verlag.

[20] K. Begnum and M. Burgess. Principle components and importance ranking of distributed anomalies. *Machine Learning*, 58:217–230, February 2005.

[21] C. Bellettini and J. Rrushi. Vulnerability analysis of SCADA protocol binaries through detection of memory access taintedness. In *Proc. 8th IEEE SMC Information Assurance Workshop*, pages 341–348. IEEE Press, 2007.

[22] M. Best and D. Neuhauser. Walter A Shewhart, 1924, and the Hawthorne factory. *Qual Saf Health Care*, 15:142–143, 2006.

[23] J. Bigham, D. Gamez, and N. Lu. Safeguarding SCADA systems with anomaly detection. In *Proc. 2nd International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security*, LNCS 2776, pages 171–182. Springer Verlag, 2003.

[24] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[25] D. Bolzoni, E. Zambon, S. Etalle, and P.H. Hartel. POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System. In *IWIA '06: Proc. 4th IEEE International Workshop on Information Assurance*, pages 144–156. IEEE Computer Society Press, 2006.

[26] G. Box and G. Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[27] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu. MAFIA: A maximal frequent itemset algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 17:1490–1504, 2005.

[28] L. Burns, J.L. Hellerstein, S. Ma, C.S. Perng, D.A. Rabenhorst, and D.J. Taylor. Towards discovery of event correlation rules. In *Proc. IEEE/IFIP International Symposium on Integrated Network Management*, pages 345 –359, 2001.

[29] E. Byres, D. Hoffman, and N. Kube. On shaky ground - a study of security vulnerabilities in control protocols. In *5th American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Controls, and Human Machine Interface Technology*, 2006.

[30] A. Carcano, I. Fovino, M. Masera, and A. Trombetta. Critical information infrastructure security. chapter Scada Malware, a Proof of Concept, pages 211–222. Springer-Verlag, Berlin, Heidelberg, 2009.

[31] A. Cardenas, S. Amin, Z. Lin, Y. Huang, C. Huang, and S. Sastry. Attacks against process control systems: risk assessment, detection, and response. In *ASIACCS*, pages 355–366. ACM, 2011.

[32] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Sinner, and A. Valdes. Using model-based intrusion detection for SCADA networks. In *Proc. SCADA Security Scientific Symposium 2007*. Digital Bound Press, 2007.

[33] A. Cui and S.J. Stolfo. Defending Legacy Embedded Systems with Software Symbiotes. In *RAID '11: Proc. 14th International Symposium on Recent Advances in Intrusion Detection*, volume 6361 of *LNCS*, pages 358–377. Springer, 2011.

[34] M. Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843–848, 1995.

[35] H. Debar, M. Dacier, A. Wespi, and S. Lampart. An Experimentation Workbench For Intrusion Detection Systems. Technical Report RZ 6519, IBM Research Division, Zurich Research Laboratory, 1998.

[36] M. Dekker, C. Karlsberg, and M. Lakka. Annual Incident Reports 2012. Analysis of Article 13a incident reports, ENISA, Athens, 2013.

[37] P. Dussel, C. Gehl, P. Laskov, J. Busser, C. Störmann, and J. Kästner. Cyber-critical infrastructure protection using real-time payload-based

anomaly detection. In *Proc. 4th International conference on Critical information infrastructures security*, CRITIS'09, pages 85–97, Berlin, Heidelberg, 2010. Springer-Verlag.

[38] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proc. 15th USENIX Security Symposium*, pages 241–256. USENIX Association, 2006.

[39] S. Forrest, S.A. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 40(10):88–96, 1997.

[40] I. Fovino, A. Carcano, M. Masera, A. Trombetta, and T. Delacheze-Murel. Modbus/DNP3 state-based intrusion detection system. In *Proc. of the IEEE International Conference on Advanced Information Networking and Applications*, AINA '10, pages 729–736, Washington, DC, USA, 2010. IEEE Computer Society.

[41] F.P.Lees. *Less' Loss Prevention in the Process Industries*. Butterworth-Heinemann, 3 edition, 2005.

[42] A. Goguen G. Stoneburner and A. Feringa. *Risk Management Guide for Information Technology Systems, NIST Special Publication 800-30*. National Institute of Standards and Technology, 2002.

[43] W. Gao, T. Morris, B. Reaves, and D. Richey. On SCADA Control System Command and Response Injection and Intrusion Detection. In *eCrime Researchers Summit (eCrime)*, 2010.

[44] B. Goethals and M. Zaki, editors. *FIMI '03, Frequent Itemset Mining Implementations*, volume 90 of *CEUR Workshop Proceedings*, 2003.

[45] J. Gonzalez and M. Papa. Passive scanning in Modbus networks. In *Critical Infrastructure Protection*, volume 253 of *IFIP International Federation for Information Processing*, pages 175–187, 2007.

[46] G. Grahne and J. Zhu. Fast algorithms for frequent itemset mining using FP-Trees. *IEEE Transactions on Knowledge and Data Engineering*, 17:1347–1362, 2005.

[47] DNP Users Group. DNP3 Specification- Application Layer. `http://www.dnp.org/About/DNP3%20Primer%20Rev%20A.pdf`, 2007. [acessed October 2013].

[48] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proc. 16th USENIX Security Symposium (Security'07)*. USENIX Association, 2007.

[49] J. Han and M. Kamber. *Data mining concepts and techniques*. Morgan Kaufmann Publishers, 2 pap edition, 2006.

[50] J. L. Hellerstein, S. Ma, and C.-S. Perng. Discovering actionable patterns in event data. *IBM Syst. J.*, 41:475–493, July 2002.

[51] J. Hieb, J. Graham, and J. Guan. An ontology for identifying cyber intrusion induced faults in process control systems. In *Critical Infrastructure Protection III*, volume 311 of *IFIP Advances in Information and Communication Technology*, pages 125–138. Springer Boston, 2009.

[52] M. Hoon. Parameter Estimation of Nearly non-Stationary Autoregressive Processes. Technical Report RF-86, Delft University of Technology, 1995.

[53] R. Hyndman and G. Athanasopoulos. Forecasting: principles and practice; an online textbook. `https://www.otexts.org/fpp`. [accessed October 2013].

[54] K.L. Ingham and H. Inoue. Comparing Anomaly Detection Techniques for HTTP. In *RAID '07: Proc. 10th International Symposium on Recent Advances in Intrusion Detection*, volume 4637 of *LNCS*, pages 42–62. Springer, 2007.

[55] K.L. Ingham, A. Somayaji, J. Burge, and S. Forrest. Learning DFA representations of HTTP for protecting web applications. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51(5):1239–1255, 2007.

[56] K. Julisch and M. Dacier. Mining intrusion detection alarms for actionable knowledge. In *Proc. 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 366–375, New York, NY, USA, 2002. ACM.

[57] T. Brijs K. Geurts, G. Wets and K. Vanhoof. Profiling high frequency accident locations using association rules. In *Proc. 82nd Annual Transportation Research Board, Washington DC (USA)*, pages 123–130. Transportation Research Board, 2003.

[58] J. Kassakian, R. Schmalensee, W. Hogan, H. Jacoby, and J. Kirtley. The Future of the Electric Grid. Technical report, Massachusetts Institute of Technology, 2011.

[59] M. Keeney and E. Kowalski. Insider Threat Study: Computer System Sabotage in Critical Infrastructure Sectors. Technical report, United States Secret Service, May 2005.

[60] K. Kent and M. Souppaya. Guide to Computer Security Log Management, NIST Special Publication 800-92. Technical report, 2006.

[61] T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, 1995.

[62] C. Kruegel, G. Vigna, and W. Robertson. A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5):717–738, 2005.

[63] R. Langner. *Robust Control System Networks*. Momentum Press, 2011.

[64] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In *Proc. 7th conference on USENIX Security Symposium - Volume 7*, pages 6–6, Berkeley, CA, USA, 1998. USENIX Association.

[65] N. Lim, N. Singh, and S. Yajnik. A log mining approach to failure analysis of enterprise telephony systems. In *Proc. the IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, pages 398 –403, june 2008.

[66] H. Lin, A. Slagell, C. Di Martino, Z. Kalbarczyk, and R. Iyer. Adapting Bro into SCADA: Building a Specification-based Intrusion Detection System for the DNP3 Protocol. In *Proc. of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, CSIIRW '13, pages 5:1–5:4, New York, NY, USA, 2013. ACM.

[67] O. Linda, T. Vollmer, and M. Manic. Neural network based intrusion detection system for critical infrastructures. In *Neural Networks, IJCNN International Joint Conference on*, pages 1827 –1834, 2009.

[68] R.P. Lippmann, J.W. Haines, D.J. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 34(4):579–595, 2000.

[69] Y. Liu, P. Ning, and M. Reiter. False data injection attacks against state estimation in electric power grids. In *Proc. 16th ACM conference on Computer and communications security*, CCS '09, pages 21–32, New York, NY, USA, 2009. ACM.

[70] P.A. Loscocco, S.D. Smalley, P.A. Muckelbauer, R.C. Taylor, S.J. Turner, and J.F. Farrell. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In *NISSC '98: Proc. 21st National Information Systems Security Conference*, pages 303–314, 1998.

[71] M.V. Mahoney and P.K. Chan. Learning non-stationary models of normal network traffic for detecting novel attacks. In *KDD '02: Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pages 376–385. ACM Press, 2002.

[72] M.V. Mahoney and P.K. Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In *RAID '03: Proc. 6th Symposium on Recent Advances in Intrusion Detection*, volume 2820 of *LNCS*, pages 220–237. Springer, 2003.

[73] M. Majdalawieh, F. Parisi-Presicce, and D. Wijesekera. DNPSec: Distributed Network Protocol Version 3 (DNP3) Security Framework. In *Advances in Computer, Information, and Systems Sciences, and Engineering*, pages 227–234. Springer, 2006.

[74] S. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz. A data mining analysis of RTID alarms. *Comput. Netw.*, 34:571–577, October 2000.

[75] S. McLaughlin. On dynamic malware payloads aimed at programmable logic controllers. In *Proc. of the 6th USENIX conference on Hot topics in security*, HotSec'11, pages 10–10, Berkeley, CA, USA, 2011. USENIX Association.

[76] S. McLaughlin and P. McDaniel. SABOT: specification-based payload generation for programmable logic controllers. In *ACM Conference on Computer and Communications Security*, pages 439–449. ACM, 2012.

[77] L. Murchu N. Falliere and E. Chien. W32.stuxnet Dossier, 2011.

[78] M. Naedele and O. Biderbost. Human-assisted intrusion detection for process control systems. 2004.

[79] N.H. Narayanan and N. Viswanadham. A methodology for knowledge acquisition and reasoning in failure analysis of systems. *Systems, Man and Cybernetics, IEEE Transactions on*, 17(2):274 –288, march 1987.

[80] A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *Proc. 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 575 –584, june 2007.

[81] P. Oman and M. Phillips. Intrusion detection and event monitoring in SCADA networks. In *Critical Infrastructure Protection*, volume 253 of *IFIP International Federation for Information Processing*, pages 161–173. Springer US, 2007.

[82] R. Pang, v. Paxson, R. Sommer, and L. Peterson. binpac: A yacc for Writing Application Protocol Parsers. In *Internet Measurement Conference*. ACM Press, 2006.

[83] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31:2435–2463, December 1999.

[84] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee. McPAD: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, 53(6):864 – 881, 2009.

[85] PI Profibus Profinet. Overview and guidance for PROFINET specifications, v2.3. `http://www.profibus.com/nc/download/specifications-standards/downloads/profinet-io-specification/`. [acessed October 2013].

[86] J. Pollet. Electricity for Free? The Dirty Underbelly of SCADA and Smart Meters. Black Hat Briefings, 2010.

[87] N. Puketza, Y. Chung, R.A. Olsson, B. Mukherjee, and C. Ronald. A Software Platform for Testing Intrusion Detection Systems. *IEEE Software*, 14:43–51, 1997.

[88] N.J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R.A. Olsson. A Methodology for Testing Intrusion Detection Systems. *IEEE Transactions On Software Engineering*, 22:719–729, 1996.

[89] R. Rantala. Cybercrime against businesses. Technical report, U.S. Dept. of Justice, Office of Justice Programs, Bureau of Justice Statistics, Washington, D.C., 2004.

[90] A. Rege-Patwardhan. Cybercrimes against critical infrastructures: a study of online criminal organization and techniques. *Criminal Justice Studies*, 22(3):261–271, Sep 2009.

[91] J. Rouillard. Real-time log file analysis using the simple event correlator (sec). In *Proc. 18th USENIX conference on System administration*, pages 133–150, Berkeley, CA, USA, 2004. USENIX Association.

[92] J. Rrushi. An exploration of defensive deception in industrial communication networks. *International Journal of Critical Infrastructure Protection*, 4(2):66 – 75, 2011.

[93] F. Salfner and S. Tschirpke. Error log processing for accurate failure prediction. In *Proc. 1st USENIX conference on Analysis of system logs*, WASL'08, pages 4–4, Berkeley, CA, USA, 2008. USENIX Association.

[94] F. Salfner, S. Tschirpke, and M. Malek. Comprehensive logfiles for autonomic systems. In *Proc. 18th International Symposium on Parallel and Distributed Processing*, page 211, april 2004.

[95] D. Savio, B. DeRemer, and S. Karnouskos. *From Data Access to Unified Architecture*, chapter Application: SAP and OPC-UA, pages 384–390. Number 978-3-8007-3242-5 (ISBN). VDE Verlag, 2010.

[96] K. Scarfone and P. Mell. Guide to Intrusion Detection and Prevention Systems (IDPS), NIST Special Publication 800-94. Technical report, 2007.

[97] Schneider Electric. *Modbus Protocol and Register Map for ION Devices*, 70022-0124-00 edition, 2009. `http://www.powerlogic.com/literature/ION_Devices_Modbus_Register_Map.pdf` [accessed October 2013].

[98] W.T. Shaw. *Cybersecurity for SCADA systems*. PennWell Corp., 2006.

[99] R. Shayto, B. Porter, R. Chandia, M. Papa, and S. Shenoi. Assessing the integrity of field devices in Modbus networks. In *Critical Infrastructure Protection II*, volume 290 of *IFIP International Federation for Information Processing*, pages 115–128. Springer Boston, 2009.

[100] J. Slay and M. Miller. Lessons Learned from the Maroochy Water Breach. In *Critical Infrastructure Protection*, volume 253 of *IFIP International Federation for Information Processing*, pages 73–82. Springer Boston, 2007.

[101] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning For Network Intrusion Detection. In *Proc. IEEE Symposium on Security and Privacy*. IEEE Computer Society, May 2010.

[102] Y. Song, M. Locasto, A. Stavrou, A. Keromytis, and S. Stolfo. On the infeasibility of modeling polymorphic shellcode. In *Proc. of the 14th ACM conference on Computer and Communications Security*, CCS '07, pages 541–551, New York, NY, USA, 2007. ACM Press.

[103] Y. Song, S.J. Stolfo, and A.D. Keromytis. Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic. In *NDSS '09: Proc. 16th ISOC Symposium on Network and Distributed Systems Security*. The Internet Society, 2009.

[104] T. Srivatanakul, J. Clark, and F. Polack. Effective security requirements analysis: Hazop and use cases. In *Information Security: 7th International Conference*, volume 3225 of *LNCS*, pages 416–427. Springer Berlin Heidelberg, 2004.

[105] K. Stouffer, J. Falco, and K. Scarfone. *Guide to Industrial Control Systems (ICS) Security, NIST Special Publication 800-82*. National Institute of Standards and Technology, 2011.

[106] N. Sugiura. Further analysts of the data by Akaike' s information criterion and the finite corrections. *Communications in Statistics - Theory and Methods*, 7(1):13–26, 1978.

[107] A. Treytl, T. Sauter, and C. Schwaiger. Security measures for industrial fieldbus systems - state of the art and solutions for IP-based approaches. In *Proc. International Workshop on Factory Communication Systems*, pages 201 – 209. IEEE Computer Society, 2004.

[108] R. Vaarandi. *Tools and technigues for event log analysis*. PhD thesis, Tallinn University of Technology, 2005.

[109] A. Valdes and S. Cheung. Communication Pattern Anomaly Detection in Process Control Systems. In *Proc. of International Conference on Technologies for Homeland Security*. IEEE, 2009.

[110] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[111] V.N. Vapnik and A. Lerner. Pattern Recognition using Generalized Portrait Method. *Automation and Remote Control*, 24, 1963.

[112] T. Wan and X.D. Yang. IntruDetecto: A Software Platform for Testing Network Intrusion Detection Algorithms. In *ACSAC '01: Proc. 17th Annual*

Computer Security Applications Conference, pages 3–11. IEEE Computer Society Press, 2001.

[113] K. Wang, G. Cretu, and S.J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation. In *RAID '05: Proc. 8th International Symposium on Recent Advances in Intrusion Detection*, volume 3858 of *LNCS*, pages 227–246. Springer, 2006.

[114] K. Wang, J.J. Parekh, and S.J. Stolfo. Anagram: a Content Anomaly Detector Resistant to Mimicry Attack. In *RAID '06: Proc. 9th International Symposium on Recent Advances in Intrusion Detection*, volume 4219 of *LNCS*, pages 226–248. Springer, 2006.

[115] K. Wang and S.J. Stolfo. Anomalous Payload-Based Network Intrusion Detection. In *RAID '04: Proc. 7th Symposium on Recent Advances in Intrusion Detection*, volume 3224 of *LNCS*, pages 203–222. Springer, 2004.

[116] G. Barrie Wetherill and Don W. Brown. *Statistical process control : theory and practice*. Chapman and Hall, 1991.

[117] R. Winther, O. Johnsen, and B. Gran. Security assessments of safety critical systems using hazops. In *SAFECOMP '01: Proc. 20th International Conference on Computer Safety, Reliability and Security*, LNCS 2187, pages 14–24, London, UK, 2001. Springer-Verlag.

[118] IEC 60870-5-104 - Protocol specification; Transmission protocols, 2006.

[119] MODBUS TCP/IP messaging implementation guide. `http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf`, 2006. [acessed October 2013].

[120] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. Mining console logs for large-scale system problem detection. In *Proc. 3rd conference on Tackling computer systems problems with machine learning techniques*, SysML'08, pages 4–4, Berkeley, CA, USA, 2008. USENIX Association.

Web references (acessed October 2013)

[121] Oxford Dictionaries Online. http://www.oxforddictionaries.com/definition/english/process?q=process.

[122] Project Basecamp. http://www.digitalbond.com/tools/basecamp/.

[123] L. Auriemma. Advisories. http://http://aluigi.altervista.org/.

[124] S. Bernstein and A. Blankstei. Two deny hacking into L.A.'s traffic light system. http://seclists.org/isn/2007/Jan/50.

[125] Adobe Security Bulletin. Security updates for Adobe Reader and Acrobat APSB13-25; CVE-2013-5325. http://www.adobe.com/support/security/bulletins/apsb13-25.html.

[126] Microsoft Security Response Center. Conficker Worm: Help Protect Windows from Conficker. http://technet.microsoft.com/en-us/security/dd452420.aspx.

[127] Microsoft Security Response Center. Microsoft Security Bulletin. http://technet.microsoft.com/en-us/security/bulletin/.

[128] Digital Bond, Inc. QuickDraw SCADA IDS. http://www.digitalbond.com/tools/quickdraw/.

[129] Mu Dynamics. pcapr. http://pcapr.net.

[130] K. Higgins. Security Incidents Rise In Industrial Control Systems . http://www.darkreading.com/attacks-breaches/security-incidents-rise-in-industrial-co/224400280, 2010.

[131] MSDN Library. [MS-CIFS]: Common Internet File System (CIFS) Protocol Specification. http://msdn.microsoft.com/en-us/library/ee442092(v=prot.13).aspx.

[132] Metasploit Penetration Testing Software. http://metasploit.com/.

[133] NIST: National Institute of Standards and Technologies. National Vulnerability Database. http://nvd.nist.gov.

[134] K. Poulsen. Slammer worm crashed ohio nuke plant network. http://www.securityfocus.com/news/6767.

[135] Symantec Security Response. W32.Sasser.Worm. http://www.symantec.com/security_response/.

[136] P. Roberts. Zotob, pnp worms slam 13 daimlerchrysler plants. http://www.eweek.com/c/a/Security/Zotob-PnP-Worms-Slam-13-DaimlerChrysler-Plants/.

[137] Common Vulnerabilities and Exposures. CVE-2010-4709 - heap-based buffer overflow in automated solutions Modbus/TCP Master OPC Server. http://www.cvedetails.com/cve/CVE-2010-4709/.

[138] Wireshark. http://www.wireshark.org.

Titles in the IPA Dissertation Series since 2008

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development*. Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and

tions. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components*. Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. Middelkoop. *Capturing and Exploiting Abstract Views of States in OO Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-18

M.F. van Amstel. *Assessing and Improving the Quality of Model Transformations*. Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. Tamalet. *Towards Correct Programs in Practice*. Faculty of Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. Basten. *Ambiguity Detection for Programming Language Grammars*. Faculty of Science, UvA. 2011-21

M. Izadi. *Model Checking of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. Kats. *Building Blocks for Language Workbenches*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

S. Kemper. *Modelling and Analysis of Real-Time Coordination Patterns*. Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. Wang. *Spiking Neural P Systems*. Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. Khosravi. *Optimal Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. Middelkoop. *Inference of Program Properties with Attribute Grammars, Revisited*. Faculty of Science, UU. 2012-02

Z. Hemel. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference*. Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation*. Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software*. Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective*. Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development*. Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. Kop. *Higher Order Termination*. Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain*. Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

ISBN 978-90-365-3604-2