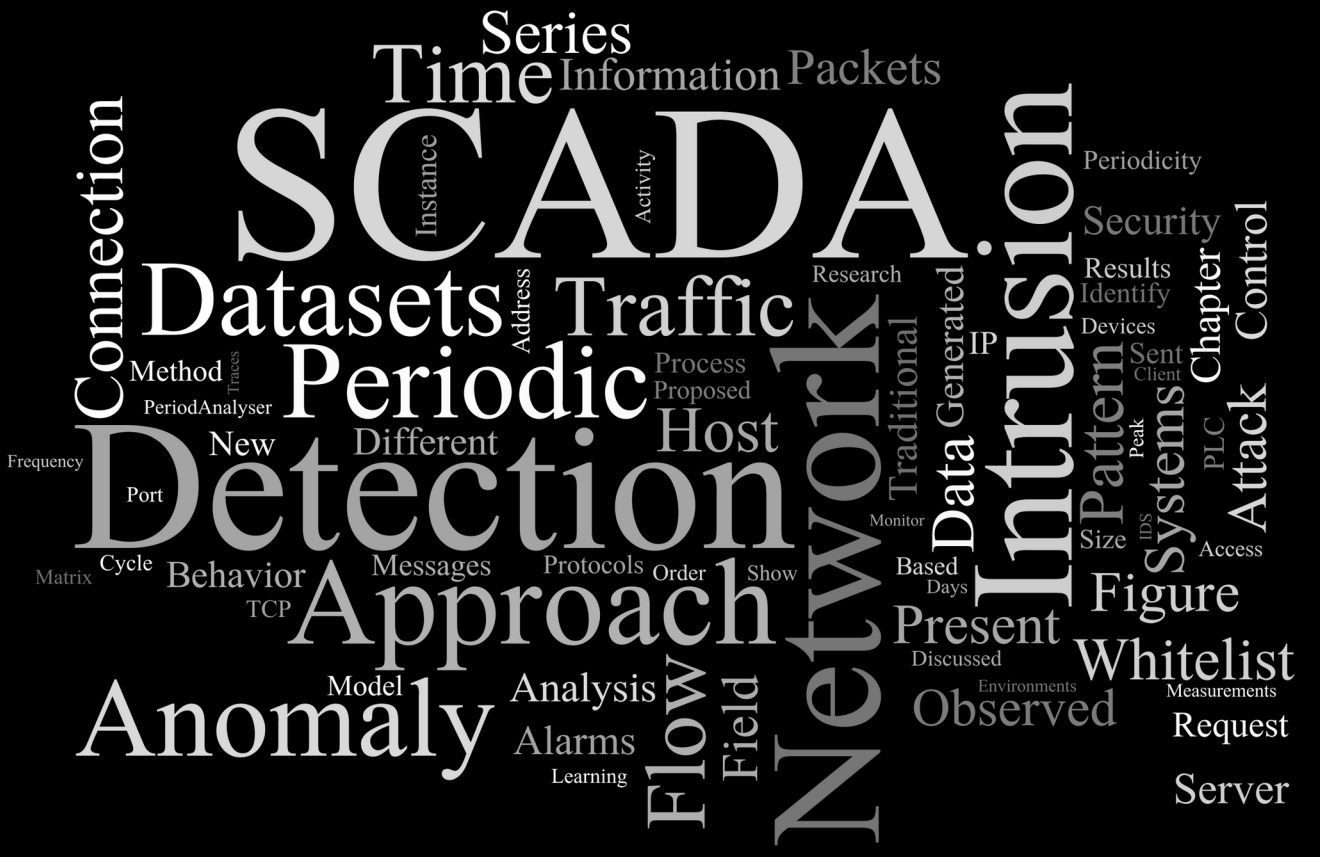
Intrusion Detection in SCADA Systems

A network based approach



Rafael Ramos Regis Barbosa

Anomaly Detection in SCADA Systems

A Network Based Approach

Rafael Ramos Regis Barbosa

Graduation committee:

Chairman:	Prof. dr. ir. A. J. Mouthaan
Promoters:	Prof. dr. ir. A. Pras
	Prof. dr. ir. B. R. Haverkort
Members:	
Prof. dr. G. Dreo Rodosek	Universität der Bundeswehr München
Prof. dr. O. Festor	University of Lorraine
Dr. R. Sadre	Aalborg University
Prof. dr. S. Etalle	University of Twente
Prof. dr. ir. L. J. M. Nieuwenhuis	University of Twente

Funding sources:

Hermes, Castor and Midas projects	Ministry of Interior and Kingdom Relations
PROSECCO project	University of Twente
IOP GenCom project SeQual	Agentschap NL
Network of Excellence project Flamingo	European Comission, Seventh Framework Programme

CTIT Ph.D. - thesis series No. 14-300
Centre for Telematics and Information Technology
University of Twente
P.O. Box 217, NL – 7500 AE Enschede

ISSN 1381-3617
ISBN 978-90-365-3645-5

Typeset with LaTeX. Printed by Ipskamp Drukkers B.V.

ANOMALY DETECTION IN SCADA SYSTEMS
A NETWORK BASED APPROACH

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties,
in het openbaar te verdedigen
op woensdag 02 April 2014 om 16.45 uur

door

Rafael Ramos Regis Barbosa

geboren op 19 november 1983
te Vila Velha-ES, Brazilië

Dit proefschrift is goedgekeurd door:
Prof. dr. ir. Aiko Pras (Promotor)
Prof. dr. ir. Boudewijn R. Haverkort (Promotor)

Acknowledgments

As one of my promoters and advisor, Aiko Pras, would often say to me: "you should now to take a step back and look at the big picture, you always focus too much on the details". The big picture, or what I learned looking back at these last four years, is that this thesis would probably not exist if I had worked alone. The work presented here was only possible due to the efforts of many.

My promoters Aiko Pras and Boudewijn Haverkort provided indispensable guidance during my Ph.D. program. Their input went beyond academic discussions, helping me shape my post-Ph.D. life. I would also like to thank Ramin Sadre for his constant advice. The weeks you received me in Aalborg, although few, were incredible productive. Also, thanks to all members of the committee for the providing interesting discussions and valuable feedback.

I would like to extend my thanks to my paranymphs Luiz Olavo, for the (not always) serious advice, and Giovane, roommate at DACS and always good company. I also grateful to all DACS colleagues, in particular to my other roommates, Idilio and Anja, who made this journey feel much shorter than it was, and to my M.Sc. thesis advisor, Pieter-Tjerk de Boer, with whom I kept having numerous technical discussions during my Ph.D program.

This research would also not be possible without the support received from several sources. The national Hermes, Castor and Midas projects provided founding and the equally important collaboration with industry partners. This collaboration facilitated the collection of network traffic at operational SCADA systems, which is central to the analysis described in this thesis. This work was also partially funded by the PROSECCO (Next Generation Protection and Security of Content) project from the University of Twente, by the IOP GenCom project SeQual from the Dutch agency Agentschap NL and by the Network of Excellence project Flamingo (ICT-318488) from the European Commission under its Seventh Framework Programme.

Leaving your home country is never easy, but it is certainly easier if you have amazing people supporting you along the way. For that I would to thank my family, specially my parents, Florencio and Angela: "Pais, obrigado por tudo!

O constante apoio de vocês foi fundamental." I also have to acknowledge that moving abroad was not my idea: thanks Ramon for literally expanding my horizons! My thanks also go to all my old friends in the new continent, but especially for my new friends in the old continent; these last years were awesome!

Finally, I would like to thank this wonderful woman I met just after arriving in Enschede, at this paradoxical place called Macandra. She had to poke a few times in the head until I *really* noticed her. But it worked. We are now together for 6 years, and I know many more to come. I love you Sanjka.

Abstract

Supervisory Control and Data Acquisition (SCADA) networks are commonly deployed to aid the operation of large industrial facilities, such as water treatment and distribution facilities, and electricity and gas providers. Historically, SCADA networks were composed by special-purpose embedded devices communicating through proprietary protocols. However, three main trends can be observed in modern deployments: (i) SCADA networks are becoming increasingly interconnected, allowing communication with corporate networks, remote access from engineers and system administrators, and even communication with the Internet; (ii) the use of commercial off-the-shelf devices, such as Windows desktops; and (iii) the adoption of the TCP/IP protocol stack. As a result, these networks become vulnerable to cyber attacks, being exposed to the same threats that plague traditional IT systems.

In our view, measurements play an essential role in validating results in network research, and can sometimes lead to surprising insights. Therefore, the first objective of this thesis is to understand how SCADA networks are utilized in practice. To this end, we provide the first comprehensive analysis of real-world SCADA traffic. We analyze five network packet traces collected at four different critical infrastructures: two water treatment facilities, one gas utility, and one (mixed) electricity and gas utility. We show exiting network traffic models developed for traditional IT networks cannot be directly applied to SCADA network traffic. In particular, SCADA networks do not present daily patterns of activity and self-similarity. We also validate two commonly held assumptions regarding SCADA traffic. First, we show that the SCADA connectivity matrix is stable, that is, the list of "who is communicating with whom" typically presents few and small changes. Second, we provide evidence that a large number of SCADA hosts, in particular all Programmable Logic Controllers (PLCs) in our datasets, generate traffic periodically.

Based on our analysis of real-world SCADA network traffic, the second objective of this thesis is to exploit the *stable connection matrix* and the *traffic periodicity* to perform anomaly detection. In order to exploit the stable connec-

tion matrix, we investigate the use of whitelists at the flow level. Despite the high level of protection that can be achieved by whitelists, a common problem with this approach is that maintaining a whitelist is burdensome to the user, as whitelists are commonly large and require manual updates. However, as changes in the connection matrix are rare, flow whitelisting becomes a promising solution for SCADA environments. We show that flow whitelists have a manageable size, considering the number of hosts in the network, and that it is possible to overcome the main sources of instability in the whitelists, therefore reducing the need for updates. In order to exploit the traffic periodicity, we focus our attention to connections used to retrieve data from devices in the field network (e.g., PLCs). As data is typically retrieved using a polling mechanism, such connections display periodic patterns. We show that the traffic in these connections can be modeled as a series of periodic requests and their responses, and propose *PeriodAnalyzer*, an approach that uses deep packet inspection to automatically identify the different requests belonging to each connection and the frequency at which they are issued. Once such normal behavior is learned, *PeriodAnalyzer* can be used to detect data injection and Denial of Service attacks.

Contents

CHAPTER 1

Introduction

1.1 Background

The operation of complex industrial processes, such as water distribution and electricity generation, requires managing information regarding a number of different components that compose an infrastructure, which potentially spread over hundreds of kilometers. Early control systems required operators to stay at, or to frequently visit, remote sites in order to ensure that the process is performing properly. Data gathered from field devices was displayed on large control panels, which also allowed operators to manually control the process [8].

With the advent of telemetry, it became possible to connect the devices used in these infrastructures. The term Supervisory Control And Data Acquisition (SCADA) refers to the technology that enables such infrastructures to be monitored and controlled from a centralized control room. For instance, in a water distribution facility, it can be used to: check the level in storage tanks and wells; monitor flows and pressure in pipes; monitor quality characteristics, such as acidity, turbidity and chlorine residual; control pumps and valves; and adjust the addition of chemicals.

SCADA systems provide operators with a real-time view of the whole process, by automating data collection from field devices in different remote sites. They also provide an alarm system that enables the field devices to report fault conditions. In addition, the system provides operators with means to react to changes in the process, by sending commands to the field. Again, using the water utility scenario as an example, such commands could be opening and closing valves, or changing set points, such as the capacity of a water tank. Operators are also able to change the algorithms that implement the control loops used in the process, e.g., the method used to forecast water consumption. One of the main advantages brought by SCADA systems was the reduction in the costs

of operating the infrastructure, by increasing process efficiency and minimizing the need of visits to remotes sites.

As industrial processes become more complex, with more devices and also more information managed per device, so grows the importance of these systems. Today, SCADA systems are considered a vital component of many nations' critical infrastructures [104, 96]. In the US alone, it is estimated that the control systems used by electric grid and oil and natural gas infrastructure represent an investment of $3 to $4 billion. The energy sector invests over $200 million each year for control system, network and related devices, and at least the same amount in personnel costs [53]. The importance of these infrastructures is enormous and failures can be catastrophic. Take for example the blackout that happened in Ohio in 2003, caused by trees brushing high-voltage transmission lines combined with a failure in the computer system responsible for generating the alarms. The incident left 50 million people without electricity for up to two days, contributed to at least 11 deaths and caused a damage estimated to be 6 billion US dollars [108].

Some recent incidents highlight the vulnerabilities of SCADA systems. The breach in Maroochy water services in Australia [131] by a disgruntled employee exposes the risk of insider attacks, the *slammer* worm infection at US Davis-Besse nuclear plant [17] shows that critical infrastructures can be affected by common Internet malware and the *Stuxnet* that attack targeted a specific industrial control system likely in Iran [55], demonstrated how much damage a resourceful attacker can cause.

It is important to stress that these are not isolated events. A survey with 200 industry executives from electricity utilities in 14 countries performed by Baker et al. [10] showed that 80% had faced a large-scale denial-of-service attack, and 85% had experienced network infiltrations. In fact, security incidents on industrial systems are on the rise. Data from the Industrial Security Incident Database (ISID) [25] which contains incident information since 1982, shows that 73% of the incidents happened between 2002 and 2007. The number of attacks reported to the United States' Department of Homeland Security (DHS) grew from 9 in 2009, to 198 in 2011 and 171 in 2012 [78].

Not surprisingly, the vulnerability of SCADA systems has received attention from both government and industry. World-wide, several industry sectors are developing guidelines to raise awareness regarding potential threats and improve their security practices. Examples of such efforts are the Dutch SCADA Security Good Practices for the Drinking Water Sector [102], the Recommended Guidelines for Information Security Baseline Requirements for Process Control, Safety and Support ICT Systems [113] by the Norwegian Oil and Gas Associ-

ation and the North American Electric Reliability Coorporation (NERC) reliability standards on critical infrastructure protection [112]. Although a common recommendation in these guidelines is the implementation of Intrusion Detection Systems (IDSs), research in SCADA specific IDS is still in its infancy [137, 87].

This brings us to the main problem addressed in this thesis: intrusion detection in SCADA networks. Given that intrusion detection in traditional IT networks has remained a prolific research area since its inception in the late 80's [48], one might question the need of new IDS solutions. Therefore, in Chapter 2 of this thesis, we present an extensive characterization of network traces collected in SCADA networks used in utility sector: water treatment and distribution facilities, and gas and electricity providers. The data collection was possible through the collaboration with industry partners, established in the context of the national Hermes, Castor and Midas projects[1]. The goal of this characterization is to expose the differences with traditional Information Technology (IT) networks, and thus motivate the need of new intrusion detection solutions.

We note that despite the increasing number of scientific publications in the area of SCADA networks, very little information is publicly available about real-world SCADA traffic. In fact, many publications on SCADA networks do not rely on empirical data, as obtained from real-world measurement (e.g., [37, 145, 126, 140]). We argue that a comprehensive analysis of real-world measurements is necessary to fully understand SCADA networks. Research on the field of traditional IT networks showed us that this type of analysis can lead to surprising insights, like the self-similar nature of network traffic [98, 119].

Based on our characterization of SCADA traffic, in the second part of this thesis we propose two complementary intrusion detection techniques that exploit regularities observed in the traffic to perform intrusion detection. More specifically, in Chapter 5 we propose the use of *flow whitelists* to exploit the *stable connection matrix*, and in Chapter 6 we propose an approach to model the normal traffic and detect anomalies that exploit the *traffic periodicity*.

In the following, we provide an introduction to SCADA terminology in Section 1.2. In Section 1.3, we then discuss how these systems evolved, becoming more similar to traditional IT networks, and how this evolution impacted the security of SCADA systems. In section 1.4, we introduce the problem of intrusion detection in SCADA networks and motivate high-level decisions made when designing the intrusion detection mechanisms proposed in this thesis. We then proceed to discuss the goal of this thesis, the tackled research questions and our

[1] `https://zeus.tsl.utwente.nl/wiki/hcm/ProjectDescriptions`

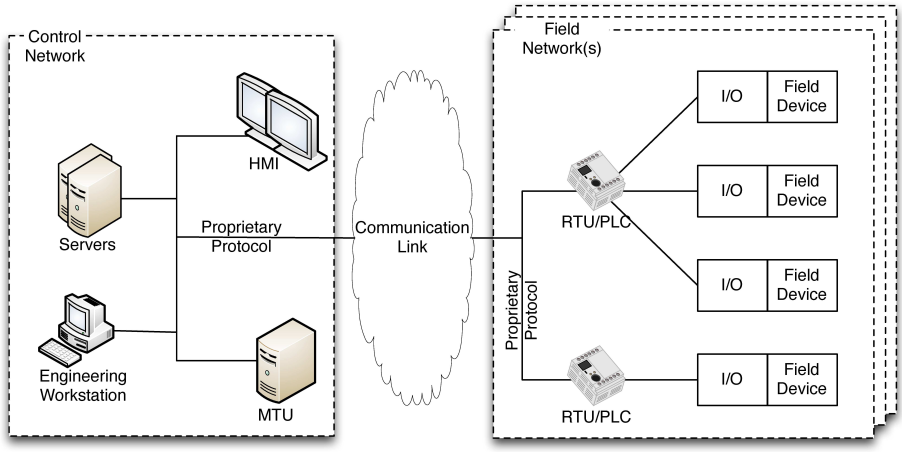


Figure 1.1: A generic traditional SCADA network architecture

approach to answer them (Section 1.5). Finally, in Section 1.6, we present the outline of this thesis.

1.2 What is SCADA?

The details of SCADA system implementations can largely vary, but some commonly used building blocks can be identified. Figure 1.1 depicts a generic network architecture for a traditional SCADA system. It can be divided into three parts:

- **Field network(s):** Represent the remote locations to be controlled. For instance, in the case of a electricity grid, a field network might be present in each substation. The field devices are instrumented by means of sensors and actuators. Remote Terminal Units (RTUs) provide a communication interface to these instrumentation devices. In many environments, the role of an RTU is played by a Programmable Logic Controller (PLC), a small embedded device that, besides providing the communication interface, also implements the *control loop* used in the process. In the power systems domain, PLCs are commonly referred to as Intelligent Electronic Devices (IEDs). In this thesis, we refer to these three equipments collective as *field devices*.

- **Control network:** Represents the control room. Data is collected by the Master Terminal Unit (MTU), which periodically polls the RTUs in the field. Operators have access to this data via a Human-Machine Interface (HMI), which commonly provides a graphical representation of the field process. The HMI is also used to report alarms and to issue commands. Additional servers that perform various tasks, such as storing data in databases or forecasting water consumption, can also be present. Finally, an engineering workstation is used to change the configuration of RTUs, for instance, setting a new maximum level for a water tank.

- **Communication link:** Connects the control and field networks. Any communication technology can be used, including wire, fiber optic, radio, telephone line, microwave or satellite [8].

Traditionally, a distinction is made between SCADA and Distributed Control System (DCS) systems [137, 8, 96, 102], depending on the distance covered by the *communication link*. The term SCADA is commonly used in industrial processes that are geographically distributed over long distances (e.g., a oil pipeline). As a consequence, SCADA networks were designed to deal with challenges imposed by Wide Area Network (WAN) communication, such as high delays and error rates, and low bandwidth. In contrast, the term DCS is used to refer to control processes within the same geographical area (e.g., a oil refinery), connected via a high-speed and more reliable Local Area Network (LAN) communications. This allowed a more tight integration between the DCS and the closed control loop used in the process.

Boyer [22] suggested that once long-distance, continuous and high-bandwidth connection between the control and field networks would be available, the system should no longer be referred to as SCADA, but as *very large* DCS instead. As WAN technology advances, we are very close to this scenario. Today, SCADA vendors offer powerful devices with functionalities that were previously only found in DCS solutions. As a consequence, the differences between these two systems become increasingly blurred [102]. In fact, many recent publications use the terms interchangeably (e.g., [2, 92, 145, 123]).

We note that, depending on the application, many other terms are used to refer to industrial networks, including but not limited to: Process Control System (PCS), Cyber-Physical System (CPS) [31], Process Control Network (PCN) [120], Industrial Automation Control Network (IACS) [34] and Networked Control System (NCS) [3]. Explaining the differences between these terms is out of the scope of this thesis. In the remainder of this dissertation,

with a slight abuse of terminology, we refer to any industrial network which follows the generic architecture shown in Figure 1.1 as a SCADA network, unless a specific reference is made necessary.

1.3 Evolution and Vulnerabilities

Historically, SCADA components were special-purpose embedded devices connected through a proprietary communication bus. Vendors would typically offer *turn key* solutions, which would be incompatible with competitors' systems [42]. Security was not a main concern in the design of these systems, instead major concerns regarded real-time processing, jitter limitation and event-notification [34].

Despite the lack of security features, SCADA vendors and operators believed they could rely on two forms of protection. The first was the *air gap*, that is, the fact that SCADA network would be physically isolated from any other networks, thus making it harder for an attacker to gain access. Secondly, they relied on *security through obscurity*, that is, vendors and operators believed that very little, if any, information was publicly available about their environments, and this lack of information made their systems secure. Security concerns focused on restricting access to unmanned field networks and on preventing configuration mistakes [120].

With the goal of reducing costs and increasing efficiency these systems are changing. Three main trends can be identified in modern installations: (i) increased interconnection, allowing communication with corporate networks, remote access from engineers and system administrators, and even communication with the Internet; (ii) the use of low cost Commercial Off-The-Shelf (COTS) devices, such as Windows computers; and (iii) the adoption of the TCP/IP protocol stack [87]. These changes have a deep impact on the security of these systems. An example of a modern SCADA network architecture is shown on Figure 1.2.

Despite the increasing interconnection, SCADA networks should still be (logically) isolated, that is, all connections between the corporate and SCADA networks should traverse a firewall [137, 96], which is responsible for blocking unauthorized traffic. However, that this isolation does not always happen in practice. In the end of 2012, the United States Industrial Control Systems Cyber Emergency Response Team published in their trimestral report [78] the results of the Project Shine, which included a list of approximately 7200 Industrial Control System (ICS) devices directly reachable via the Internet (see Figure 1.3).

Even if we assume the *air gap* exists, it is not a reliable security measure, as an attacker can use other vectors then the network for gaining access to the system. For instance, the *Stuxnet*, probably the most well-known attack to an industrial facility, used an infected USB stick as the attack vector; no direct network access was required [55]. Similarly, at least two US power generation facilities have been reported to be infected via USB sticks [65].

Relying on hiding systems details as a form of security is not a widely accepted form of security. Some argue that the opposite is true, that is, open systems are inherently more secure than closed ones [75]. Nonetheless, *security through obscurity* completely falls apart in SCADA networks, as special-purpose hardware is replaced by COTS servers and proprietary communication protocols by the TCP/IP stack. It is no longer reasonable to assume that the details about SCADA components are, by any reasonable definition, a secret. But more than only exposing details about components of SCADA systems, these changes make these systems vulnerable to the same threats that plague traditional IT systems. Consider, for instance, the *slammer* worm that exploited a vulnerability on Microsoft's SQL Server. In 2003, the worm infected at least 75000 hosts, causing network outages and unforeseen consequences, such as canceled airline flights, interference with elections, and ATM failures [110]. Another victim of *slammer* was the Davis-Besse nuclear power plant. The infection overloaded the plant's network, causing a safety-related system to be unavailable for almost 5 hours [17].

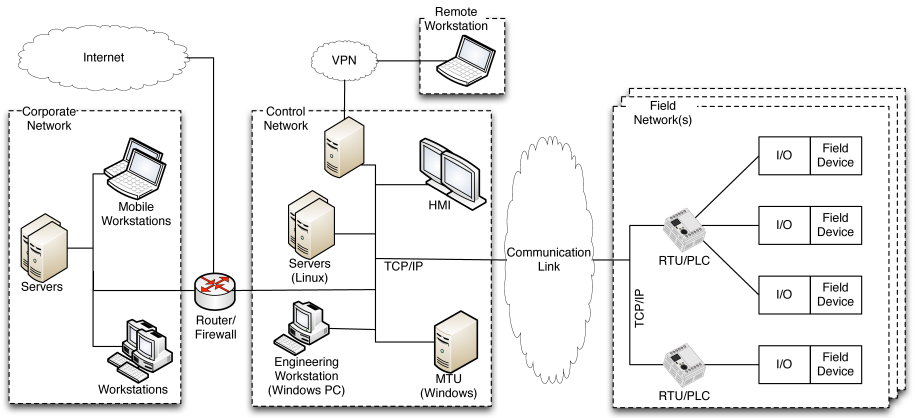


Figure 1.2: A modern realization of a SCADA network

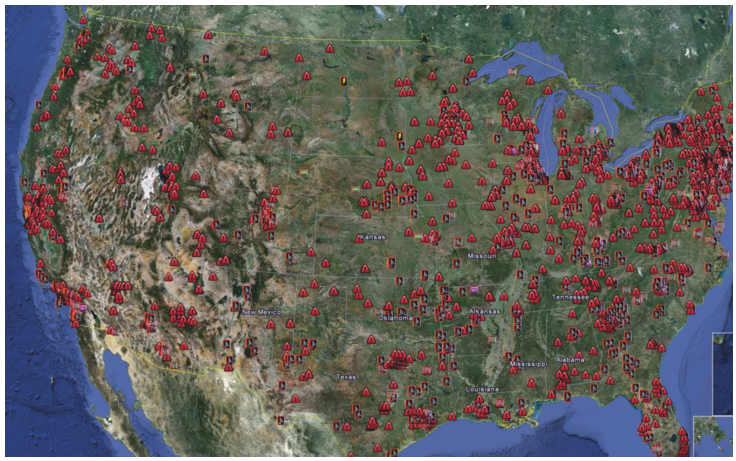


Figure 1.3: Approximately 7200 Internet facing devices in the US [78].

In addition, information about SCADA systems is becoming more widely available. Take for instance the work of Basnight et al. [15], which presents a proof-of-concept attack on how legitimate firmware can be modified and uploaded to a specific PLC model. The firmware used by the authors was obtained directly from the vendor's website. In addition, a number of SCADA protocols (running on top of the TCP/IP stack) are defined in open standards: Modbus [109], DNP3 and IEC 60870.5 [42]. In fact, it is relatively easy to find information even on SCADA-specific vulnerabilities, like exploits for the well-known penetration test tool *Metasploit*[2] or attacks signatures for Digital Bond's Quickdraw SCADA IDS[3].

The critical nature of the infrastructures that employ SCADA systems makes them a valuable target for criminal organizations, terrorists and nation states. *Stuxnet*, the infamous attack that targeted control systems likely in Iran, showed that groups with motivation, financial resources and skill to perform sophisticated attacks to critical infrastructures exist. In this context, it should be evident that more advanced security practices are necessary in SCADA networks.

[2] `http://www.metasploit.com/`
[3] `http://www.digitalbond.com/tools/quickdraw/`

1.4 Intrusion Detection in SCADA

As noted by Lunt [103], fixing all flaws of a system is not technically feasible and building one without vulnerabilities is virtually impossible. Current SCADA systems lack basic security services such as authentication and access control, and, although feasible, the costs of deploying such services will certainly delay their adoption. Besides, even secure system might have vulnerabilities caused by configuration errors or abuse by insiders. In order to deal with such limitations, intrusion detection is proposed as an complementary approach to secure systems. Mukherjee et al. [111] defines intrusion detection as the problem of identifying individuals who are using a computer system without authorization (i.e., "crackers") and those who have legitimate access to the system but are abusing their privileges (i.e., the "insider threat").

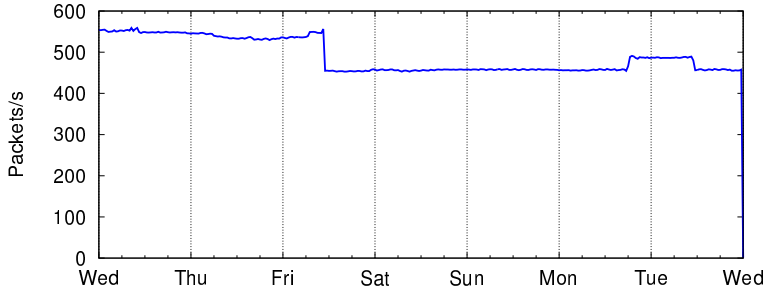
Since the seminal work by Denning [48] in 1987, intrusion detection remained an active area of research. An IDSs survey by Lunt [103] included around 20 approaches as early as 1993. As of 2007, a list compiled by Meier and Holz [106] contained an impressive 132 IDSs. Not surprisingly, many taxonomies have been proposed, from which the ones from Debar et al. [46, 47] and from Axelsson [5] being the most commonly used.

One of the fundamental distinctions made by these taxonomies is based on the source of audit data, namely *host* or *network* based detection. The first relies on data collected in a host, such as system logs and system calls, and the later on data collected in the network, commonly traffic measurements made in a central monitoring location. The restricted resources and real-time requirements of RTUs constitute an obstacle for host based detection. These devices simply might not have the required resources to support the new functionality, while still meeting the required time constraints. In addition, changes in these devices might characterize a break in license agreements, as some vendors disallow third party applications to be installed [137], or may require re-certification of the entire system [94]. For these reasons, in this thesis we explore the network based approaches.

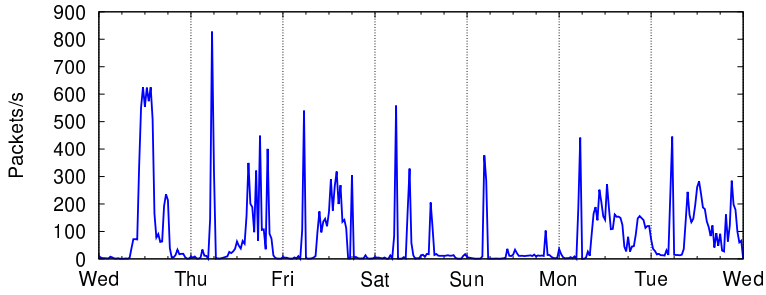
Another important distinction made is between *anomaly* and *signature* detection, with "the former relying on flagging all behaviour that is abnormal for an entity, the latter flagging behaviour that is close to some previously defined pattern signature of a known intrusion" [5]. In Debar's convention [46, 47], these are referred to as *behaviour based* and *misuse based* detection, respectively. Anomaly detection methods have the advantage of (potentially) being able to detect so called *zero-day* exploits, that is, previously unknown attacks, while the high false-alarm rate is commonly cited as a major drawback (e.g.,

[135]).

One of the causes attributed to the high false-alarm rate in traditional anomaly detection methods is the enormous variability of network traffic [133]. As we will show in Chapter 2, SCADA traffic patterns are rather *stable* when comparing to traditional IT networks. For example, take the two time series representing the number of transmitted packets per second over the course of a week displayed in Figure 1.4. The difference between time series representing a measurement in a water distribution infrastructure (Figure 1.4(a)) and the one representing a research institute (Figure 1.4(b)) is clear. In the SCADA, the packet transmission rate is fairly stable over large periods of time, spanning over days, while daily patterns of activity are clear in the traditional IT trace. This stability is due to the fact that most of the traffic in SCADA networks is



(a) Water distribution SCADA network



(b) Research institute IT network

Figure 1.4: Traffic variability

generated through an automated process, like the polling mechanism used by SCADA servers to retrieve information from the field devices.

Furthermore, SCADA components have a lifetime of tens of years [137] and changes in software are rare, if at all [34]. This should make SCADA networks even more stable, potentially reducing the high false-alarm rates common to anomaly detection methods developed for traditional IT system. For these reasons, in this thesis we explore the anomaly based approaches.

1.5 Goal, Research Questions and Approach

Our goal is to exploit characteristics of SCADA networks and develop anomaly detection techniques that are suited for these networks. First, we analyze measurements made in real-world environments to understand how SCADA networks are utilized in practice. Then, building up on this study, we exploit intrinsic characteristics of SCADA traffic to develop models that describe their "normal" behavior. These models can then be used to detect traffic anomalies, which represent potential security threats.

Given the large number of intrusion detection techniques in traditional IT networks, one could ask why is it necessary to develop new solutions. To motivate the need of specialized solutions, we first expose differences between SCADA and traditional IT networks. In other words, we address the following research question:

> **RQ 1:** *Does SCADA network traffic differ from the traditional IT network traffic? If yes, what are these differences?*

The first step we take to answer this question is to perform a literature study on SCADA networks, including a discussion of commonly used architectures, protocols, and security requirements. We then proceed to capture and analyse traffic measurements from several real SCADA infrastructures, such as water treatment and distribution facilities, as well as gas and electricity providers. Given the virtually infinite number of features that can be used to compare SCADA and traditional IT networks, we focus our analysis in a list of well-known "invariants", i.e., behaviours that are empirically shown to hold for a wide range of environments, described in [58].

In addition, we validate common assumptions regarding SCADA traffic. The first assumption we check is that changes in the network topology are rare [30, 37], that is, hosts and services are not frequently added to or removed from the network. By tracking changes in the IP-level connectivity, we verify if SCADA

networks have a *stable connection matrix*. Secondly, we verify the assumption that the polling mechanism used to retrieve data from the devices in the field network causes a large portion of the traffic to display periodic patterns [11, 145]. By means of a spectral analysis, we search for evidence of *traffic periodicity*.

The *stable connection matrix* and the *traffic periodicity* cause SCADA network traffic to be remarkably well-behaved when compared to traditional IT networks, which bring us to our second research question:

> **RQ 2:** *How to exploit SCADA traffic characteristics to perform Anomaly Detection?*

To answer this research question, we develop models for the *normal* traffic that exploit both the *stable connection matrix* and the *traffic matrix*; two characteristics that cause SCADA traffic to be predictable. We then describe techniques that detect deviations in these models (i.e., anomalies), which represent potential security threats. We also provide a discussion on how our models behave in the presence of realistic attack scenarios.

The first model uses *flow whitelists* to exploit the *stable connection matrix*. A flow whitelist is an exhaustive list of all connections which are allowed in the network. To evaluate the feasibility of this approach, we use real-world traffic measurements to verify if the size of such whitelists is manageable and verify the potential sources of instability in the whitelist.

The second model exploits the *traffic periodicity* by automatically learning which messages are sent in a periodical fashion. This method is able to deal with periodical traffic generated in different forms, e.g., periodically established connections or single connections with multiple periods. The proposed method is used to detect data injection and Denial of Service (DoS) attacks. Finally, we use real-world traffic to demonstrate the applicability of this approach.

1.6 Thesis Outline

The core topics discussed in this thesis are divided in two parts, which are closely related to the two proposed research questions (see Figure 1.5). We begin with the *"Understanding SCADA"* part, where the first research question is addressed. In this part, we present an introduction to SCADA networks and provide an extensive characterization of SCADA traffic. We focus on the differences between SCADA and traditional IT networks. This discussion is based on a literature study on different aspects of SCADA environments and an analysis of traffic measurements collected at real-world SCADA networks.

In the second part, *"Protecting SCADA"*, we address the remaining research question. In this part, we propose two complementary methods to model the normal traffic and detect anomalies which represent potential intrusion attempts. These models are based on two characteristics uncovered during the our traffic analysis: *stable connection matrix* and *traffic periodicity*.

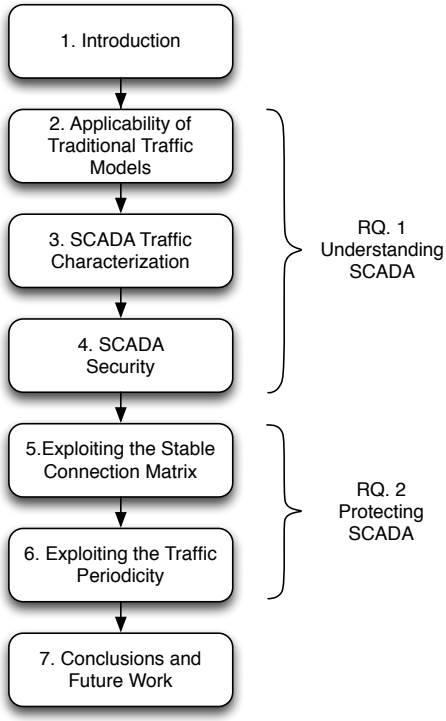


Figure 1.5: Thesis outline

The outline of this thesis is shown in Figure 1.5, which depicts how each of the chapters presented in this thesis fit in this organization. In the following, we present a short summary of each chapter, including the publication used as basis for it:

- **Chapter 2 - Appplicability of Traditional Traffic Models** provides an analysis of real-world datasets collected in SCADA networks operated

by Dutch companies in the utility sector: water treatment and distribution facilities, and gas and electricity providers. We verify whether SCADA traffic differs from traditional IT network traffic. Our analysis is based a list of "invariants", i.e., behaviors that are empirically shown to hold for a wide range of network environments, proposed in the well-known work by Floyd and Paxson [58]. This chapter is an extended version of the following conference publication, including a more detailed explanation of the tests performed and the analysis of two additional datasets:

> R. R. R. Barbosa, R. Sadre, and A. Pras. *Difficulties in Modeling SCADA Traffic: A Comparative Analysis*. Passive and Active Measurement: 13th International Conference, Pam 2012, Vienna, Austria, March 12-14, 2012.

- **Chapter 3 - Characterization of SCADA traffic** expands our analysis of real-world SCADA datasets. We verify the validity of two commonly held assumptions regarding SCADA traffic, namely, that traffic is generated in a periodic fashion and that the SCADA traffic connection is stable. This chapter is an extended version of the following conference publication, including a more detailed explanation of the tests performed and the analysis of two additional datasets:

  > R. R. R. Barbosa, R. Sadre, and A. Pras. *A First Look into SCADA Network Traffic*. In 2012 IEEE Network Operations and Management Symposium volume 17, pages 518–521. Springer, IEEE, Apr. 2012.

- **Chapter 4 - SCADA security** provides a literature study in security aspects of SCADA networks. We discuss differences to traditional IT network security, documented incidents, and industrial, governmental and academic efforts to secure SCADA systems, including a survey of SCADA intrusion detection mechanisms.

- **Chapter 5 - Exploiting the Stable Connection Matrix** discusses the feasibility of using of flow-level whitelists to protect SCADA networks. Our analysis focus on two aspects necessary for whitelists to be practical: a manageable size and stable entries. This chapter is a more detailed version of the following journal publication:

  > R. R. R. Barbosa, R. Sadre, and A. Pras. *Flow Whitelisting in SCADA Networks*. International Journal of Critical Infrastructure Protection 6(3-4):150–158, Dec. 2013.

- **Chapter 6 - Exploiting the Traffic Periodicity** presents *PeriodAnalyzer*, an approach to model SCADA traffic that exploits the periodicity generated by the polling mechanism used to retrieve data from the field devices. *PeriodAnalyzer* automatically learns this periodic behavior, and protects SCADA protocol traffic against data injection and DoS attacks. This chapter builds up on the lessons learned from the analysis discussed in the following conference publication:

  > R. R. R. Barbosa, R. Sadre, and A. Pras. *Towards Periodicity Based Anomaly Detection in SCADA Networks.* In Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation, pages 1–4. IEEE, Sept. 2012.

- **Chapter 7 - Conclusions** closes this thesis by summarizing our main findings and suggesting future work directions.

CHAPTER 2

Applicability of Traditional Traffic Models

2.1 Introduction

Intuitively, we expect SCADA networks to present traffic patterns to be different from traditional IT networks, for a number of reasons. First, SCADA networks are expected to present a nearly fixed number of nodes, in the sense that new nodes are not expected to join or leave the network frequently. Many SCADA systems are designed to work continuously, with virtually no changes, for tens of years [34]. In contrast, traditional IT components have a typical lifetime of 3 to 5 years [137]. Secondly, while traditional networks usually support a multitude of protocols, such as HTTP, instant messaging and Voice over IP, the number of supported protocols in SCADA networks is expected to be rather limited. Finally, most of the SCADA traffic is expected to be generated by polling mechanisms, which gather data from field devices. As a consequence, traffic patterns will not be as dependent on human activity, as it is the case in traditional IT networks.

Apart from the assumptions given above, little is publicly known about the behavior of SCADA traffic. This is partly caused by the sensitivity of the data. In fact, publications on SCADA networks generally do not rely on empirical data as obtained from real-world measurement [37, 95, 144]. In our view, however, measurements play an essential role in validating results in network research, and can sometimes lead to surprising insights.

As an example, consider the seminal work by Leland et al. [98] on the self-similar nature of Ethernet traffic. Based on real-world measurements, they show that Poisson processes and other models that were used to describe packet arrivals are not valid, and formalized the idea of traffic being "bursty" at different timescales. This finding resulted in models and tools that are today applied to

various tasks, such as the design and dimensioning of network equipment and the parameterization of management algorithms.

Connected to this result, studies on the presence of long-range dependency and heavy-tailed distributions [44, 50, 63, 101], contributed to a better understanding of the causes of self-similarity, and consequently a better understanding of network traffic in general. Naturally, the question arises whether the existing traffic models designed for traditional IT network, such as self-similarity, are also valid for SCADA networks.

The goal of this chapter is therefore *to verify whether SCADA traffic differs from traditional IT network traffic.* We achieve this by comparing traditional IT traffic with *real-world* SCADA measurements. The first challenge we face is that network behavior can be compared in a virtually infinite number of ways, starting from the above mentioned characteristic of self-similarity to topological properties [147] and application specific aspects [127, 14]. A second challenge is the enormous diversity observed in traditional IT networks: different topologies, different protocols and different usage patterns. Besides that, network traffic is continuously changing, and even a single network will exhibit different characteristics in different years [38]. For example, the amount of traffic tends to increase and the set of most utilized protocols tends to change. As a consequence, defining characteristics that are representative of traditional IT networks in general can be in itself a challenging task.

To cope with these problems, we base our analysis on a list of "invariants", i.e., behaviors that are empirically shown to hold for a wide range of network environments, proposed in the well-known work by Floyd and Paxson [58]. This list consists of seven invariants: diurnal patterns of activity, self-similarity, Poisson session arrivals, log-normal connection sizes, heavy-tailed distributions, invariant distribution for Telnet packet generation and invariant characteristics of the global topology. We revise this proposed list of invariants and test the ones applicable to our context.

The rest of this chapter is organized as follows. In Section 2.2, we describe the used datasets. In Section 2.3, we provide a description of the invariants, discussing the methods used to test their presence and motivating the reasons why some of them are not addressed in this work. The results are presented in Section 2.4. Finally, conclusions are given in Section 2.5.

2.2 Datasets

One of the difficulties of performing research in SCADA networks is the difficulty of obtaining real traffic measurements. For instance, all anomaly detection approaches surveyed by Garitano et al. [60] have their validation limited to either simulations or testbeds.

One of the contributions of this thesis is the use of datasets captured at operational infrastructures in the utility domain. This was possible through the collaboration with industry partners, established in the context of the national Hermes, Castor and Midas projects[1]. One of the goals of these projects was to devise new detection techniques, likely based on anomaly detection, which can monitor proprietary protocols' data and detect attacks.

The SCADA datasets that we analyze in this thesis consist of five network packet traces in `libpcap/tcpdump` format[2], collected at four different critical infrastructures: two water treatment facilities, one gas utility and one (mixed) electricity and gas utility. The networks used in these locations exhibit different topologies. First we describe these topologies and then describe how each dataset maps to these topologies.

These five critical infrastructures, besides covering different types of controlled physical processes and architectures, also provide diversity in the used SCADA protocols, three in total: Modbus, Manufacturing Message Specification (MMS) and International Electrotechnical Commission (IEC) 60870-5-104 (in this thesis, refereed to as IEC-104). For more information about these protocols, the reader is referred to Appendix A.

SCADA networks might be connected to *corporate networks*, which in turn are generally connected to the Internet. Although we include the *corporate network* in our description of SCADA topologies, none of the measurements performed in our work contains data collected in this segment of the network. Despite the fact that the *corporate network* is a potential source of intrusions, characterizing its traffic is out of the scope of this chapter, as it can be considered a traditional IT network. We discuss potential attack vectors in SCADA networks in Chapter 4.3. Between our datasets, we distinguish three different SCADA network topologies:

1. We refer to the simplest topology as *two-layer*. In this topology, the corporate network is separated from the SCADA network by a router/firewall. However, the SCADA network itself is a single domain, meaning that any

[1] `https://zeus.tsl.utwente.nl/wiki/hcm/ProjectDescriptions`
[2] `http://www.tcpdump.org`

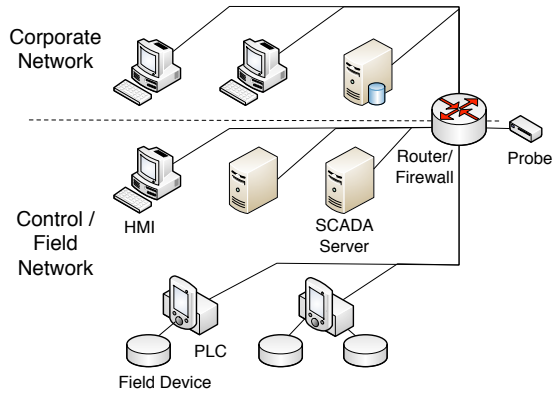


Figure 2.1: A two-layer SCADA topology

host (e.g., an operator workstation) is directly reachable from any other host (e.g., a PLC) in the network. This topology is depicted in Figure 2.1.

2. The second topology, referred to as *three-layer*, segregates the SCADA network in two subnetworks, a *control* network and a *field* network. The field network comprises the PLCs and RTUs that monitor (and potentially issue commands to) the field devices. The control network contains several servers for different purposes, such as automatically polling of field nodes and performing the access control, and the HMIs. In this topology, the communication between the control network and the field network passes through a single node, the connectivity server. This topology is depicted in Figure 2.2.

3. In the third and last topology, several *field* networks are controlled by a single *control* network. This topology it is depicted in Figure 2.3.

One of the measured SCADA networks, is organized in the *two-layer* topology. In this network, we only use a single probe connected to a router in the network, allowing us to capture all traffic. We refer to this dataset resulting from this measurement as *SCADA1*. This network uses *Modbus* as its SCADA protocol.

The dataset we refer to as *SCADA2* uses a *three-layer* topology. In order to be able to capture all traffic in both (sub)networks, we use two probes to perform simultaneous measurements, *probe 1* collecting data from the control network

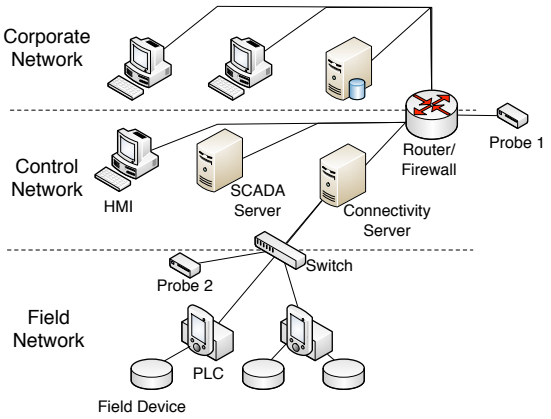


Figure 2.2: A three-layer SCADA topology

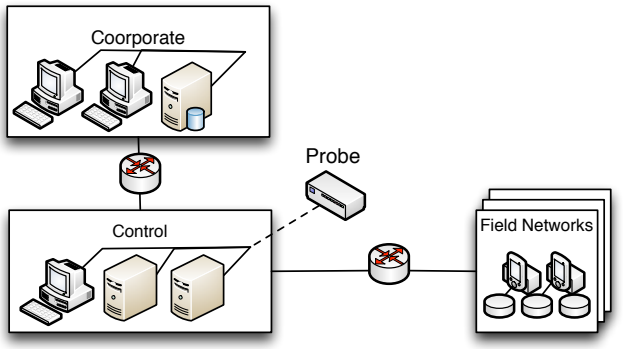


Figure 2.3: A three-layer SCADA topology with multiple field networks

and *probe 2* from the field network (see Figure 2.1). Therefore, we produce two datasets for this network, *SCADA2-control* and *SCADA2-field*. This network uses MMS as its SCADA protocol.

The *SCADA3* datasets is also organized in a *three-layer* topology and uses MMS as its SCADA protocol. However, differently from *SCADA2*, only one probe was used (*probe 2* in Figure 2.3), covering only the traffic from the *field* network.

The last dataset, *SCADA4*, is organized in the *three-layer* topology with mul-

Name	# hosts	Duration	pkts/s	KB/s	Protocol	Topo.
SCADA1	45	13 days	504.1	82.5	Modbus	2-L
SCADA2-control	14	10 days	28.7	5.1	MMS	3-L
SCADA2-field	31	10 days	75.7	28.2	MMS	3-L
SCADA3	11	1.5 days	137.8	24.0	MMS	3-L
SCADA4	215	86 days	245.6	547.8	IEC-104	M-L
IT	100	7.5 days	81.9	65.3	*NA*	*NA*

Table 2.1: Datasets overview

tiple field networks, and uses the IEC-104 protocol. The probe was connected to a switching element within the control network, allowing us to capture all traffic internal to the control network, including the traffic exchanged between the control network and five field networks.

Due to a non-disclosure agreement, we do not provide a map between the datasets and the locations where they were captured.

In order to provide a comparison with a traditional IT environment, we have selected a publicly available traffic trace from the network of an educational organization: Location 6 from the SimpleWeb Trace repository [12]. The organization is relatively small, consisting of around 36 employees and 100 students. Since the network at this location is comparable to the above SCADA networks regarding the number of hosts and the average bandwidth use, it is an adequate candidate for the analysis described in this chapter. We show in Section 2.4 that the traffic in this network exhibits characteristics that are in line with the proposed invariants for traditional IT networks. We refer to this dataset as *IT*.

An overview of all six datasets is given in Table 2.1. For each dataset, we present the number of hosts (**# hosts**), the approximate duration of the measurement in days (**Duration**), the average number of packets per second (**pkts/s**) and kilobytes per second (**KB/s**), the SCADA protocol used (**Protocol**) and which of the described network topologies is used (**Topo.**): two-layer (2-L), three-layer (3-L) or multi-layer (M-L). The last two items are not applicable for the *IT* dataset.

2.3 Invariants

In this chapter, our goal is to investigate differences between SCADA network traffic and traditional IT networks. Given the virtually infinite number of characteristics that could be used to perform this comparison, the first step is to

define a set of relevant characteristics.

In [58], Floyd and Paxson discuss difficulty in simulating and modeling Internet traffic. The problems faced when performing these tasks include constant changes in topologies, the large number of deployed protocols (exacerbated by multiple implementations of a single protocol) and the lack of realistic traffic generation models. In order to cope with these difficulties, one of the proposed approaches is "the search for invariants". In this context, invariants refer to characteristics that are empirically shown to hold for a wide range of network environments. Once these invariants are identified, they can be used to develop more realistic simulation models for the Internet.

The starting point of our analysis is the list of seven invariants proposed by Floyd and Paxson [58]. However, not all invariants are suitable for the datasets considered in this thesis. In Sections 2.3.1 through 2.3.3, we provide a description of the four invariants we test in our analysis, and explain the approach we use to test them. In Section 2.3.4, we discuss the remaining three invariants and explain the reasons why we do *not* consider them in our analysis.

2.3.1 Diurnal patterns of activity

Diurnal patterns of activity arise from the fact that network activity is strongly correlated with human activity. It has been widely observed that network traffic starts increasing around 8–9 AM local time, peaking around 11 AM. After a lunchtime drop, it starts increasing again around 1 PM, peaking around 3–4 PM and finally decreasing as business day ends at 5 PM. In addition, some measurements present a peak of activity during the evening and/or night, commonly attributed to home usage. Finally, traffic during the weekends and holidays should present a considerable decrease in comparison to normal weekdays.

The authors also acknowledge that these diurnal patterns may vary depending on the network protocol under study, explicitly mentioning protocols used in applications for which activity is not human-initiated. Such protocols are common in SCADA environments. So, in order to verify whether diurnal patterns of activity are also present in SCADA traffic, we study the time series comprising one week of traffic for three different metrics: the number of active connections, and the bandwidth measured in *packets/sec* and *bytes/sec*. Our results are discussed in Section 2.4.1.

2.3.2 Self-similarity

Informally, self-similarity refers to the quality that the whole resembles its parts. Mathematically, a *self-similar* time series is defined as follows [63, 117]. Let $X = X(i), i \geq 1$ be a *stationary* sequence and

$$X^m(k) = \frac{1}{m} \sum_{i=(k-1)m+1}^{km} x(i), k = 1, 2, 3, \cdots,$$

be a corresponding aggregated sequence where m is the aggregation level obtained by aggregating non-overlapping segments of size m. If X is self-similar:

$$X \stackrel{d}{=} m^{1-H} X^{(m)}, \text{for all } m \in \mathbb{N},$$

where the equality $\stackrel{d}{=}$ means equality in the sense of finite dimensional distributions and H is the Hurst parameter. We stress that self-similarity is only defined for time series which are at least *wide-sense stationary* [98, 117], which implies, among other properties, a constant mean over time.

In a *second-order self-similar* stationary sequence, $m^{1-H}X^{(m)}$ has the same variance and auto-correlation as X for all natural aggregation levels m.

Finally, the network traffic invariant proposed in [58] refers to *asymptotically second-order self-similarity.* This means that $m^{1-H}X^{(m)}$ has the same variance and auto-correlation as X, as $m \to \infty$. An *asymptotically second-order self-similar* process can be also called a Long Range Dependent (LRD) process, and these terms are often used interchangeably in the literature.

In practice this means that when observing network traffic time series, so-called *bursty periods*, i.e., extended periods above the mean, are present at different timescales, ranging from milliseconds to a few hours. This property violates the assumptions of traditional Markovian modeling for network traffic that predicts that long-term correlations are weak. Since the initial findings in the early 1990's [98, 119], self-similarity of network traffic has remained an active field of research (see, for instance, [101]).

In Section 2.4.2, we employ the same three popular visual methods to test for self-similarity used in [44, 98]: the R/S analysis, variance-time plots and periodograms. The visual representation of their results allows estimating the degree of self-similarity in the data.

All three methods start from a time series that represents the number of packets (or bytes) crossing a certain network segment sampled at equally spaced time intervals. The time series takes the form $X = X(t), t = 1, \cdots, N$, where

$X(t)$ is the number of packets (or bytes) at time t and N denotes the size of the time series. In the following we discuss the three tests in more detail. For a more comprehensive discussion of these methods, see [18].

Rescaled adjusted range (R/S) analysis

Consider a subset of the time series X with starting point t_i and size n. Let $\overline{X}(t_i, n)$ be the mean:

$$\overline{X}(t_i, n) = \frac{1}{n} \sum_{i=t_i}^{t_i+(n-1)} X_i,$$

and $S(t_i, n)$ the standard deviation:

$$S(t_i, n) = \sqrt{\frac{1}{n} \sum_{i=t_i}^{t_i+(n-1)} \big(X_i - \overline{X}(t_i, n)\big)^2},$$

of a subset of X calculated over the interval $[t_i, t_i + (n-1)]$.

We now define the partial sum $W(t_i, n, u)$ as:

$$W(t_i, n, u) = \sum_{j=t_i}^{t_i+u} \big(X_j - \overline{X}(t_i, n)\big).$$

Finally, the *rescaled adjusted range (R/S)* statistic [100] is then defined as:

$$R/S(t_i, n) = 1/S(t_i, n) \left[\max_{0 \le u < n} \Big(W(t_i, n, u)\Big) - \min_{0 \le u < n} \Big(W(t_i, n, u)\Big) \right]. \quad (2.1)$$

The construction of the *R/S plot* (also known as the *R/S pox diagram*) for a set of observations X with size N is outlined in Algorithm 1. First, one should select logarithmically spaced values of n. For each value of n, X is divided into K non-overlapping subsets of size $n = N/K$, with starting points $t_i = [0, n+1, 2n+1, \cdots]$. The *R/S* statistic is then calculated for each (t_i, n). Finally, the *R/S plot* is obtained by plotting $log(R/S(t_i, n))$ versus $log(n)$. The Hurst parameter can be estimated as the slope of a line fitted to the resulting curve using the least-square method.

Input : A set of observations $X = X(t), t = 0, \cdots, N$
Output: *R/S plot* for X
select logarithmically spaced values of n;
foreach n **do**
  slice X into K non-overlapping subsets of size n;
  calculate the slice starting points t_i;
  foreach t_i **do**
    plot $log(R/S(t_i, n)$ versus $log(n)$;
  end
end

Algorithm 1: Generating the R/S plot

Variance-time plots

A self-similar time series does not become "smoother" at larger time scales, i.e., the variance decreases only very slowly for increasing aggregation levels. This characteristic can be visualized with the *variance-time plot*[98], defined as follows.

For a given process $X = X(t), t = 1, \cdots, N$, let $X^{(m)}$ be the aggregated process, defined as

$$X^{(m)}(t) = \frac{1}{m} \sum_{t=1}^{t+(m-1)} X(t).$$

The first step to construct the variance-time plot is to calculate the aggregated process $X^{(m)}(t)$ for different aggregation levels (i.e., different values of m). Then the plot is obtained by plotting the variance of each aggregated process, $S^2(X^{(m)})$, versus the aggregation level m in a log-log scale. For a given aggregated process, the variance is calculated as

$$S^2(X^{(m)}) = \frac{1}{N/m} \sum_{i=1}^{N/m} \left(X_i^{(m)} - \overline{X^{(m)}}\right)^2$$

where the mean $\overline{X^{(m)}}$ is defined as

$$\overline{X^{(m)}} = \frac{1}{N/m} \sum_{i=1}^{N/m} X_i^{(m)}.$$

To obtain the Hurst parameter, a line is fitted to the resulting curve, utilizing the least-squares method and ignoring small values of m. Estimates for the line slope β between -1 and 0 suggest self-similarity. The Hurst parameter is then estimated as $H = 1 + \beta/2$.

Periodograms

The Discrete Fourier Transform (DFT) is a sequence of complex numbers X_k with $k = 0, \cdots, N-1$ such that:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}.$$

The DFT correspond to a linear combination of complex sinusoids, where each coefficient represents the amplitude and phase of these sinusoids. Although a naïve computation of the DFT using the definition above takes $O(N^2)$ operations, Fast Fourier Transform (FFT) algorithms are more efficient and can calculate the same DFT in $O(N \log N)$ operations [43].

A *periodogram* is then defined as

$$\mathcal{P}(X_k) = \|X_k\|^2, \text{ with } k = 0, 1, 2, \cdots, \lceil (N-1)/2 \rceil,$$

where$\|X_k\|$ denotes the "length" of the complex number X_k. Each value of $\mathcal{P}(X_k)$ represents an estimate for the power at frequency k/N or at period N/k. Note that, following the Nyquist theorem, the periodogram only contains information about periods that are at least twice the sampling period.

The periodogram method to estimate the Hurst parameter, consists of plotting $\mathcal{P}(X)$ against the frequency in a logarithmic scale, and then fitting a least-squares line to the low-frequency portion of the periodogram, typically the lowest 10%. The Hurst parameter is then estimated as $H = (1 - \beta)/2$, with β being the slope of the fitted line.

2.3.3 Log-normal connection sizes and heavy-tail distributions

In this section we present two distinct invariants. The first is that connection sizes have a log-normal distribution, i.e., the logarithm of the connection sizes obey a normal distributions. The second relates to the large number of network

related activities and objects that follow a *heavy tail distribution*, including sizes of Unix files, compressed video frames; and bursts of Ethernet and FTP activity.

Since the original list of invariants was published, a debate started over which of these models better describe connection size distributions. While the classical works such as [119, 44] make the case for heavy-tail distributions, Downey [50] argues that log-normal distributions provide better (or at least as good) fit to the empirical observations.

Recently, Gong et al. [63] argued that there is never sufficient data to support any analytical form summarizing the tail behavior; therefore the research efforts should focus instead on studying the complex nature of traffic generation and its implications.

In this thesis, we do not attempt to fit our measurements to theoretical distributions. We simply show, through widely used Complementary Cumulative Distribution Functions (CCDFs) [50], that measurements from the *IT* dataset generally match the results reported in the literature and point out the differences to the connection size distributions in SCADA networks. More precisely, we use CCDFs to show that the connection size distribution is always *positively skewed*, i.e, it has a body containing the majority of the values in the distribution and a tail with extreme values in the right.

The CCDF of a random variable X can be defined as $\bar{F}(X) = P(X \geq x)$. A CCDF plotted in a log-log scale for a Pareto-distributed random variable should approximate a straight line with negative slope, while the slope of the CCDF of a log-normal random variable increases along the *x-axis* (assuming a tail on the right side).

In this chapter, we define a connection as a set of packets aggregated according the traditional 5-tuple key consisting of protocol number, source and destination IP addresses and port numbers. We consider a connection to end by following the TCP state machine (i.e., after packets with RST or FIN flags have been received) or after 300 s of inactivity. We calculate the size of connections considering three metrics: duration (in seconds), number of packets and number of bytes.

2.3.4 Invariants not addressed in this work

In addition to the above four invariants, [58] also defines three invariants that are *not* considered in this chapter. In the following we described them and argue why they are not applicable to our context.

Poisson session arrivals

A "session" refers to the period of time a human uses the network for a specific task. Examples of such activities are users starting a FTP transfer, remote logins or web surfing. Strong evidence that the arrival process of such activities can be modeled as a Poisson process is provided in [119] for FTP and Telnet sessions; and in [56, 114] for HTTP traffic. Note that a Poisson process does not provide a good fit for the arrival of individual *connections* within a session [119].

While the start and end of FTP and Telnet sessions can be easily inferred from packet traces, this is not the case for all protocols. Since the concept session is protocol dependent, it is hard to develop a general method to group network packets into sessions. Therefore, the approaches for session identification generally rely on some kind of approximation. For instance, [56] approximates HTTP sessions from modem calls, while [114] uses a fixed timeout for HTTP activity. As acknowledged in [114], both methods might provide imprecise HTTP session information.

Moreover, while the concept of a session closely relates to user activity, we expect most of the traffic from the SCADA protocols observed in our datasets to be machine-generated. For these reasons, we do not attempt to test this invariant in this work.

Telnet packet generation

This invariant states that the interval between consecutive packets triggered by keystrokes in a Telnet session, obey a Pareto distribution. Since this invariant mostly concerns human behavior and a single specific protocol, we have not considered it in this work.

Characteristics of the global topology

The last invariant relates to behavior that appears due to characteristics of the Earth. For example, the delay in inter-continental connections is bounded by the propagation delay. Obviously, this invariant is not useful for comparing SCADA and traditional IT networks.

2.4 Analysis Results

In this section we discuss the results of our analysis regarding the four selected invariants described in the previous section. In Section 2.4.1, we show the

time series used to verify the presence of diurnal patterns in our datasets. In the sequence, in Section 2.4.2, we present the results for the three visual self-similarity tests: the *R/S analysis*, the *variance-time plots* and the *periodograms*. Finally, in Section 2.4.3, we show the CCDFs used to discuss distributional aspects of connection sizes.

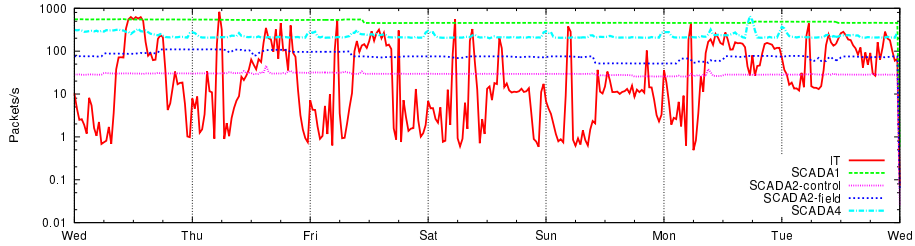
2.4.1 Diurnal patterns of activity

Diurnal patterns in network activity are widely reported in the literature [58]. In contrast, most of the traffic of a SCADA environment is expected to be machine-generated, for instance by the polling mechanism used to retrieve data from the field. As a consequence, we expect SCADA traffic to have a very regular throughput. To verify this, we plot three different time series: *packets/s*, *bytes/s* and *number of active connections*, calculated over 30-minute bins for our six datasets. Figure 2.4 show the results for all datasets, with the exception of *SCADA3*, as it is not long enough to reliably identify daily patterns. To ease the comparison, we plot only one week of data for each dataset, aligning the time series based on weekdays.

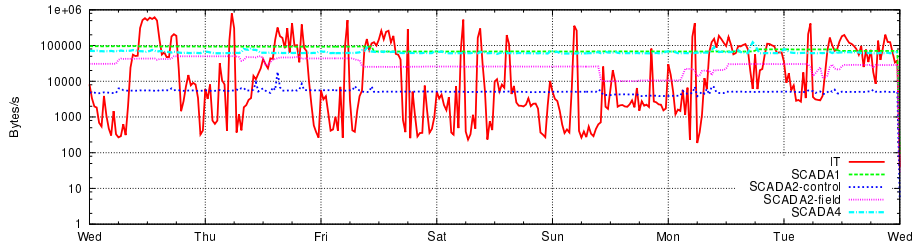
As expected, the *IT* dataset, shown in red, displays diurnal patterns of activity, with lower throughput during the nights in comparison to days (note that the *y-axis* is plotted using a logarithmic scale). We also observe less traffic during the weekend. The pattern is particularly clear when observing the number of active connections (Figure 2.4(c)). Another interesting pattern are the recurring peaks seen daily in the early morning (around 5:25 AM) for the *packet/s* (Figure 2.4(a)) and *byte/s* (Figure 2.4(b)) time series. After closer inspection, we verify that these peaks are caused by a large reoccurring connection between the same two hosts. We assume it to be related to some automated activity, such as backup, but we did not attempt to verify which.

The figure also shows that SCADA traffic does not present day and night patterns. Instead, for the SCADA datasets, the time series remain stable over large periods of time, to which we refer as *baselines*. Note, however, that the throughput is not always constant. Notably, datasets *SCADA1* and *SCADA2-field* present a considerable drop in the packet rate at around Friday noon and Sunday noon respectively.

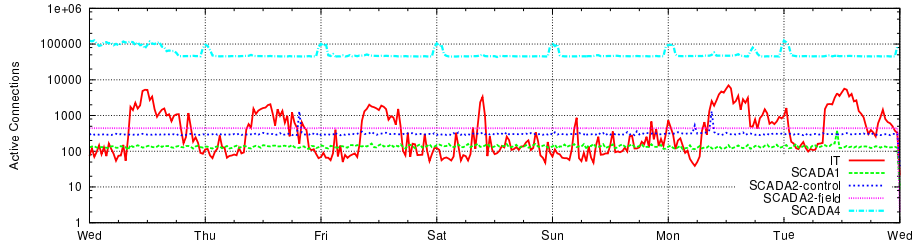
A closer inspection of the data reveals two major causes for the deviations from the baseline: the start (or the end) of connections with large throughput, and the increase (or decrease) in the number of variables polled by certain connections. We speculate that these deviations are mostly caused by certain changes in the physical process that the SCADA systems control, e.g., tanks



(a) Packets per second



(b) Bytes per second



(c) Active connections

Figure 2.4: Looking for diurnal traffic patterns

becoming full or an increase in the water demand. Another possible cause is a manual access to the PLCs, for either retrieving data or uploading a new configuration. Further research is necessary to establish if these changes can be predicted from network traffic information.

In addition, the *SCADA4* dataset has a slight increase in the amount of traffic during the business hours. This behavior is best visualized in the *pkts/s*

time series shows on Thursday, Friday and Monday (Figure 2.4(a)). We argue however, that the *baseline* behavior dominates the time series. Similar to the *IT* dataset, *SCADA4* also presents a visible pattern recurring daily, but around midnight instead of around 5 AM. Again, we speculate this behavior is due to some automated activity such as backup.

2.4.2 Self-similarity

One of the requirements for a time series to be self-similar is that it must be wide-sense stationary [98, 117], which implies, among other properties, a constant mean over time. A time series that presents seasonal trends or abrupt changes in the mean, therefore, is not self-similar. For this reason, Internet network traffic is not truly self-similar [63]. When observing sufficiently large samples of network traffic the already discussed daily patterns of activity become the dominating behavior. Following this reasoning, our traditional IT dataset (*IT*) is not truly self-similar.

In Section 2.4.1, we showed that the time series for the SCADA datasets have deviations from the *baseline* behavior, which we speculate to be caused by changes in physical processes or maintenance operations. Due to these deviations, these time series are also not stationary and, therefore, at a scale of days, SCADA network traffic also does not present self-similarity.

However, network traffic self-similarity is only valid for measurements with smaller durations, up to a few hours. In this section, we therefore verify whether our datasets present self-similarity at this time scale, by extracting subsets with the duration of approximately 1 hour from each of our datasets, and applying the tests described in Section 2.3.2. For each subset, a time series for the number of packets and bytes per second is calculated, using a 100-millisecond bin size.

Care has to be taken that the time series are stationary, as non-stationary time series are, by definition, not self-similar. When considering the *IT* dataset, we select a subset during a "busy" period (i.e., a period with higher throughput). The motivation for this choice is that periods with low throughput are less likely to present self-similarity, as observed by Crovella and Bestavros [44] when studying the self-similarity of HTTP traffic. This approach is further supported by the fact that "busy" traffic periods present a higher degree of self-similarity, as estimated by the Hurst parameter, when compared to less busy periods [98].

This approach, however, is not applicable to our SCADA datasets, as the *baseline* changes discussed above occur at seemingly arbitrary moments. We then manually inspect the time series to identify stationary subsets, i.e., subsets for which both packets and bytes time series do not present *baseline* changes.

Figure 2.5 depicts the *R/S pox diagram*, the *variance time plots* and the *periodogram* test for *IT* in the first column (Figures 2.5(a), 2.5(c) and 2.5(e) respectively) and *SCADA2-control* in the second column (Figures 2.5(b), 2.5(d) and 2.5(f) respectively). The results for the other SCADA datasets, with the exception of *SCADA4*, are similar, and therefore omitted here. In addition, the results for the *bytes/s* are similar to the ones from *pkts/s* and, therefore, are also omitted here. The results omitted here are shown in Appendix B.1. The *SCADA4* presents a peculiar behavior, which is discussed separately.

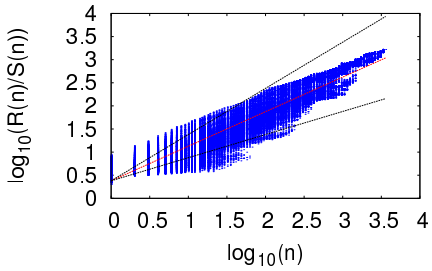
The R/S pox diagram of a self-similar random variable should have an asymptotic slope between 0.5 and 1. In the top row of Figure 2.5, these slopes are represented by the black dotted lines. The slope is typically estimated by the slope of a line fitted using the least-square method, in the figure represented by the red dotted line. It should be clear from Figures 2.5(a) and 2.5(b) that *IT* presents self-similar behavior, while *SCADA2-control* does not.

A comparable result is obtained with the variance-time plot. Remember that, in a self-similar time series, the slope of the resulting curve should be greater than -1. To aid visualization, we show a black dotted line in the middle row of Figure 2.5 with a slope of -1. The results show that the variance of the SCADA time series decays much faster than expected for a self-similar process. In contrast, the *IT* dataset result is consistent with the traditional network measurements results reported in the literature.

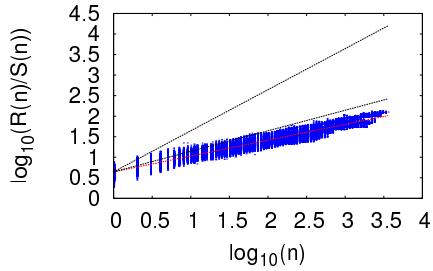
The same conclusion can be drawn from the periodogram. When applying this method, we obtain an estimative of $H = 0.79$ for the *IT* dataset and of $H = 0.44$ for *SCADA2-control*. Remember that the Hurst parameter of a self-similar process lies in the interval $H \in [0.5, 1)$.

We now discuss the results of our analysis for the *SCADA4* dataset. Figure 2.6(a) shows the *variance time* plot for the packet time series of a subset of this dataset. As the variance decays with a slope slightly higher than -1, the plot suggests that the considered subset presents self-similarity. However, upon close inspection of this subset, we observe that the reason for the slow decay of the variance with increasing aggregation levels is *not* self-similarity.

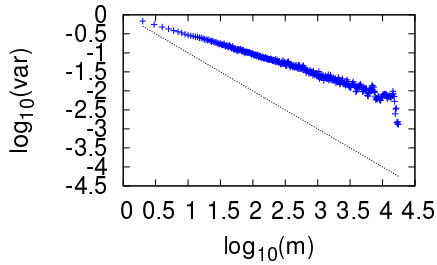
In Figures 2.6(b), 2.6(c) and 2.6(d), we show the considered subset at different aggregation levels $m = 1$, $m = 10$ and $m = 100$ respectively. These aggregation levels are equivalent to sampling the subset using bin sizes of $0.1s$, $1s$ and $10s$ respectively. Upon close inspection of this subset at these different aggregation levels, it is clear that the variance is dominated by the large peaks of periodic activity, recurring every 10 minutes. The presence of these peaks at every aggregation level is the reason why the variance decays so slowly, although the time series is clearly not self-similar.



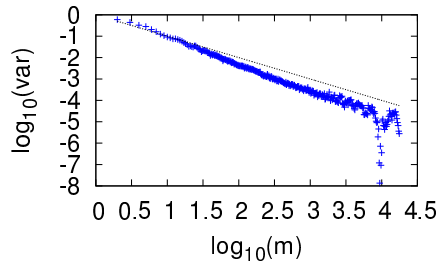
(a) R/S diagram (IT trace)



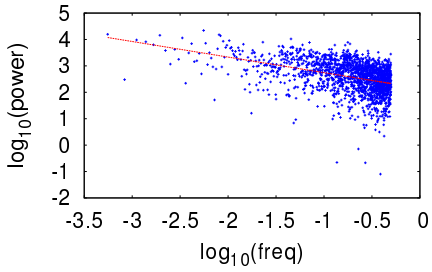
(b) R/S diagram (SCADA2-control trace)



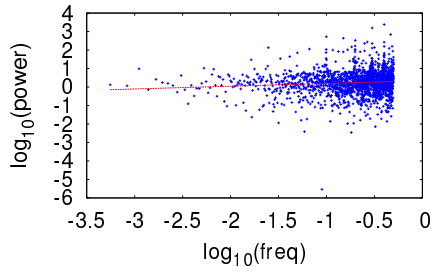
(c) Variance-time (IT trace)



(d) Variance-time (SCADA2-control trace)

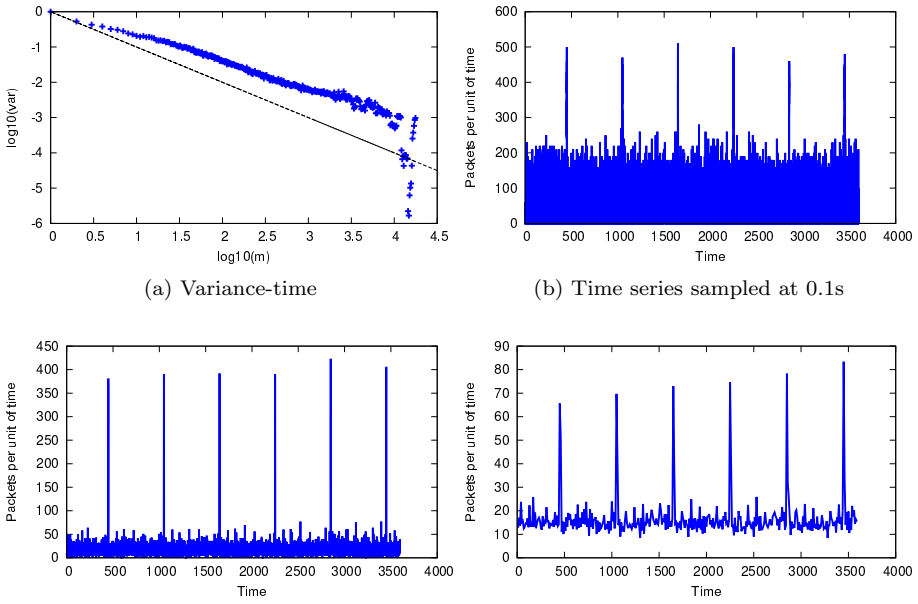


(e) Periodogram (IT trace)



(f) Periodogram (SCADA2-control trace)

Figure 2.5: Self-similarity tests on the *IT* trace and the *SCADA2-control* trace



(a) Variance-time

(b) Time series sampled at 0.1s

(c) Time series sampled at 1s

(d) Time series sampled at 10s

Figure 2.6: Looking for self-similarity in the *SCADA4* dataset

Table 2.2 summarizes the results of our self-similarity analysis, reporting the estimates for the Hurst parameter from the R/S analysis (*R/S*), variance-time plots (*vartime*) and periodograms (*period*). With the exception of *SCADA4*, all estimates for the SCADA datasets indicate a non-self-similar behavior, however, the estimates are not consistent between all tests. Although the *SCADA4* estimates for the Hurst parameter indicate self-similarity, we discard this hypothesis for the reasons discussed above.

In contrast, the *IT* dataset shows a more consistent estimate of the Hurst parameter, which is in agreement with a self-similar behavior. Note also that, while the R/S analysis and periodograms yield a single estimate, the variance-time plots produce a small range of estimates. This happens because for both small and large aggregation levels m there is a considerable amount of variance that should not be taken into account when performing the least-square fit. In our analysis we remove from 0% up to 15% of either end of the variance-time

dataset	bytes/s			pkts/s		
	R/S	**var-time**	**period**	**R/S**	**var-time**	**period**
IT	0.73	0.72	0.79	0.75	[0.71-0.72]	0.79
SCADA1	0.17	[0.09,0.11]	0.13	0.17	[0.32,0.42]	0.22
SCADA2-control	0.38	[0.38,0.44]	0.43	0.39	[0.36,0.37]	0.44
SCADA2-field	0.02	[0.27,0.31]	0.29	0.44	[0.35,0.42]	0.04
SCADA3	0.15	[0.36,0.54]	0.24	0.20	[0.17,0.21]	0.26
SCADA4	0.64	[0.48,0.55]	0.74	0.62	[0.38,0.51]	0.69

Table 2.2: Hurst parameter estimates

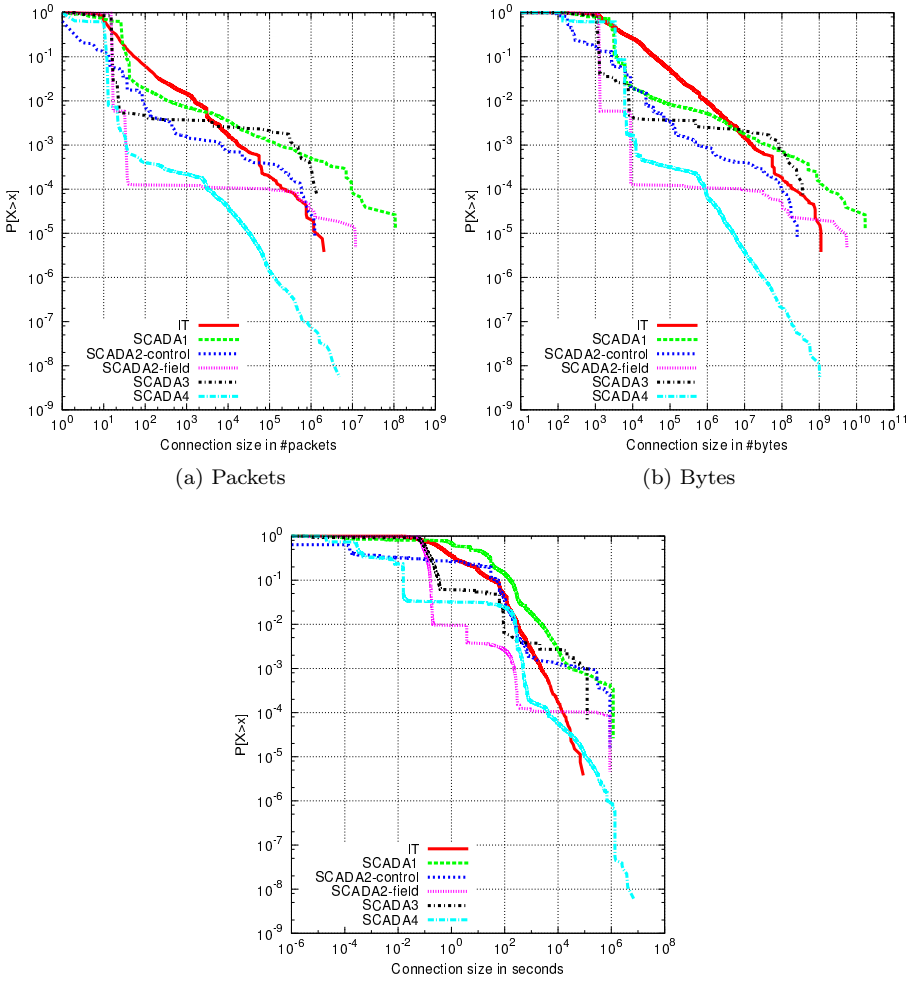
plot to obtain the Hurst estimates.

2.4.3 Distributional aspects of connection sizes

In this section we study aspect of the connection size distribution of our datasets. In Figure 2.7 we plot the CCDF for the size of the connections in number of packets and bytes, and in seconds. As explained in Section 2.3.3, there is a debate in the research community around which distribution best fits the tail behavior of connections sizes: Pareto (a heavy-tail distribution) or log-normal.

The distribution that best fits the connection sizes for the *IT* dataset depends on the used metric. In case of the number of packets and bytes per connection, plotted in Figures 2.7(a) and 2.7(b), respectively, the CCDF presents an almost constant slope (in a log-log scale), indicating that a Pareto model might provide a good fit. In case of connection duration in seconds, plotted in Figure 2.7(c), the behavior is closer to that of a log-normal distribution, with an increasing slope when approaching extreme values in the tail.

For the SCADA datasets, neither Pareto nor log-normal distributions generally provide a good fit. First, lets consider again the connection size in packets and bytes plotted in Figures 2.7(a) and 2.7(b). The tail for dataset *SCADA1* could be modeled as Pareto, if one considers the tail to consist of values above 10^2 and 10^4 for size in packets and bytes, respectively. In the case of *SCADA2-control*, the CCDF presents large variations and cannot be approximated by either model. The remaining datasets present a higher concentration of values in the head of the distribution, e.g., when considering the size in number of packets, the probability rapidly falls below 10^{-2}. The far CCDF tail of the *SCADA4* dataset, above 10^4 and 10^6, has a constant slope, thus also agreeing with a Pareto distribution.



(a) Packets

(b) Bytes

(c) Duration

Figure 2.7: Connection size complementary cumulative distribution functions

When considering duration in seconds (Figure 2.7(c)), the dataset that closer resembles the *IT* dataset is *SCADA1*. The increasing slope of the CCDF indi-

cates that log-normal might provide a good fit. For the remaining datasets, the slope is relatively small up to 10 s, it sharply increases in the interval $[10, 10^3]$ after which it sharply decreases. In addition, the CCDF for these datasets present different slopes at different duration ranges, with some large, nearly vertical drops, indicating that some values are fairly common.

Irrespective of which is the best model to represent the connection size distribution, all datasets share a common characteristic: the connection size distribution is always *positively skewed*, i.e, it has a body containing the majority of the values in the distribution and a tail with extreme values in the right. As a consequence, the distribution of the connection sizes present a large disparity between the mean, located in the tail, and the median, located in the head. A final interesting observation is that all SCADA datasets contain at least one connection that lasts almost the whole duration of the measurement.

2.5 Conclusions

The goal of this chapter was to verify whether models used to describe traditional network traffic can also be applied to SCADA traffic. To this end, we have analyzed five SCADA traffic traces collected at four different critical infrastructures: two water treatment facilities, one (mixed) gas utility and one electricity and gas utility. We compared characteristics of these networks with those of traditional network traffic. Our analysis is based on a list of network traffic invariants widely observed in network measurements.

We draw the following conclusions. First, SCADA networks do not present the diurnal patterns of activity common to traditional IT networks. The main reason for the absence of these daily patterns is that, in general, the generation of traffic is not dependent on human activity. Instead, traffic is mostly generated by automated processes, such as the polling used to retrieve data from field devices.

In addition, we show that SCADA traffic does not present self-similarity. The results from this chapter, combined with the periodicity analysis presented in Chapter 3 suggest that simpler *ON/OFF* models might provide a good approximation for the *pkts/s* and *bytes/s* time series.

Finally, neither heavy-tail nor log-normal distributions seem to provide a good fit for the connection sizes of SCADA traffic. However, as noted by Gong et al. [63], there is a fundamental problem in attempting to fit models to tail distributions based on finite amounts of data and it might be a better idea to focus attention to the *waist* of the connection size distributions.

In summary, our results indicate that the existing traffic models cannot be directly applied to SCADA traffic.

To our best knowledge, we provide the first study on real-world SCADA traces in this thesis. Since existing publications on SCADA networks generally do not rely on empirical data, we believe that our findings are a first step towards constructing realistic SCADA traffic models to support future research in the area. In future work, research should focus on improving the understanding of SCADA traffic, including the characterization of the packets and connection arrival processes.

CHAPTER 3

SCADA Traffic Characterization

3.1 Introduction

SCADA networks are implemented to coordinate and manage the actions of industrial infrastructures, possibly comprising a high number of devices spread over a large geographical area. HMIs provide human operators with real-time information about the field process and might provide control capabilities. Automated processes ensure that the infrastructure operates correctly and safely, by continuously polling field information and, possibly, sending control commands. Alerts are generated in case of severe problems, so human operators can intervene.

Much of the research in SCADA seems to be dedicated to security issues. In [87], the authors identify general threats faced by SCADA environments and provide a list of research challenges in the area. The lack of Intrusion Detection Systems (IDS) for SCADA is one of the challenges addressed [37, 144]. In [37], an IDS is proposed that is mainly based on the Modbus protocol specification [109], but also assumes regularity and stability in regard of topology, communication and configuration. A different approach is used in [144], where the communication patterns are the basis of the proposed IDS. A limitation of these studies is that they assume certain patterns without having empirical results to support them. We argue that measurements should be an essential part of IDS SCADA research, as they allow the validation of the traffic models used.

In the Chapter 2, we showed that SCADA traffic displays different patterns than those of traditional IT networks, and models used to describe the latter might not be directly applied to the former. The observed differences partially arise from the way most traffic is generated in these networks. While traffic patterns in traditional IT networks are strongly correlated with human activity, as

reflected by the *diurnal pattern of activities*, SCADA traffic is mostly generated by automated processes.

In this chapter, we extend our analysis of SCADA traffic by verifying two commonly held assumptions regarding SCADA traffic. More specifically, based on measurements done in SCADA infrastructures for water, gas and electricity utilities we address two research questions:

RQ 3.1: *Is SCADA traffic generated in a periodic fashion?*

The first assumption is that traffic displays strong periodic patterns, because a large portion of it is generated by automated processes, like the polling mechanism used to retrieve data from the field, [11, 145]. By means of time series and spectral analysis, we search for evidence of periodicity.

RQ 3.2: *Is the SCADA traffic connection matrix stable?*

To answer this question we verify whether the assumption that changes in the network topology are rare [30, 37], e.g., hosts are not frequently added to or removed from the network. By studying changes in the IP-level connectivity, we verify if SCADA networks have a stable connection matrix.

We acknowledge that a number of other tests could be performed, but we argue that the analysis presented in this chapter provides an overview of two important traffic characteristics. If valid, these two characteristics could be exploited to create models for normal SCADA traffic, and subsequently be used to detect anomalies.

Finally, SCADA operation seems very similar to management operations found in traditional IT networks, such as the ones provided by the Simple Network Management Protocol (SNMP) [32]. Probably the most widespread network management protocol, SNMP is used by automated applications, such as *Cacti*[1] and *MRTG* [115], to provide network managers with real-time information about the network. These applications ensure that the network infrastructure operates correctly, by continuously polling information from network devices in the field. Alerts can also sent in case of problems, so that network managers can intervene.

Given the usage similarities in the scenarios described above combined with the fact that the SNMP is a widely used and well-known protocol, we propose the following research question:

[1] `http://www.cacti.net/`

RQ 3.3: *Is SCADA traffic similar to SNMP traffic?*

This question is interesting since, real-world SCADA traffic captures are hard to obtain, due of the critical nature of the infrastructures they control. On the other hand, SNMP traffic traces might be more readily available for researchers given its widespread adoption. In case they are similar, SNMP traces could be used by researchers to emulate SCADA behavior. In this chapter, we investigate this possibility by performing the same analysis used for testing SCADA assumptions to a wide range of SNMP traffic traces.

The remainder of this chapter is organized as follows. In Section 3.2, we describe the datasets used in this chapter and the preprocessing steps performed. We then present our analysis of both assumptions, periodic traffic generation in Section 3.3 (RQ 3.1) and stable connection matrix in Section 3.4 (RQ 3.2). The tests are applied to both SCADA and SNMP traces, allowing us to compare the traces characteristics, thus addressing RQ 3.3. Finally, in Section 3.5, we present our conclusions.

3.2 Datasets

In this section we present our datasets. We provide some background information about the measurement setup (Section 3.2.1) and describe the preprocessing steps performed before our analysis (Section 3.2.2).

3.2.1 Data collection

In this study, we analyze eleven datasets: five SCADA and six SNMP network traffic traces. All datasets were collected with `tcpdump` and stored using the `pcap` format[2].

The SCADA traces were collected at critical infrastructures: two water treatment facilities, one gas utility and (mixed) one electricity and gas utility. The datasets are numbered from *SCADA1* to *SCADA4*, with dataset *SCADA2* being split in two: *SCADA2-control* and *SCADA2-field*, containing traffic from the control and field networks, respectively. A more detailed description of these datasets is provided in Chapter 2.

The SNMP traces are the same as studied in the work by Schönwälder et al. [129], and its extension [23]. As such, we adopt the same naming scheme: the first number in the name identifies the location where the trace was collected

[2] `http://www.tcpdump.org`

and the second number gives the trace number, e.g., the dataset *l01t02* is the second trace from location 1. The traces were collected at a number of locations, ranging from a networking laboratory network to national research networks. A summary of the description of the datasets is shown in Table 3.1.

trace	**description**	**duration**	**start**
SCADA1	control and field network	13.6 days	2011/01/22
SCADA2-control	control network	10.2 days	2010/10/18
SCADA2-field	field network	10.2 days	2010/10/18
SCADA3	field network	1.5 days	2011/01/13
SCADA4	control network	86 days	2011/04/14
l01t02	national research network	3.1 days	2005/07/26
l05t01	regional network provider	15.4 days	2006/04/19
l11t01	networking laboratory network	25.6 days	2006/01/07
l15t04	national research network	7.7 days	2006/10/17
l17t01	university network	5.0 days	2007/06/20
l18t01	national research network	10.0 days	2007/06/22

Table 3.1: Overview of the datasets

3.2.2 Preprocessing

Some of the datasets contain large *gaps*, that is, periods without any traffic. For these traces, we restrict our analysis to the longest continuous period without any gaps. We believe that these gaps were caused by problems during the data collection procedure, and, as such, do not represent the normal behavior of the networks. Furthermore, even after removing the gaps, the traces are still considerably long, with durations ranging from 1.5 days up to 86 days[3].

The SCADA datasets contain packets generated by a number of basic services that, although essential for the network operation, are not considered in our analysis. Since the goal of the chapter is to characterize only the traffic specific to the SCADA system, we removed the traffic of the following network services, identified by their transport port numbers: 53 (`DNS`), 67-68 (`DHCP`), 123 (`NTP`), 137-139 (`NetBIOS`), 161-162 (`SNMP`), 546-547 (`DHCP` for IPv6). In addition, all traffic at port 1010 in the dataset *SCADA1* was also removed, as it contained only single-packet rejected TCP connections. During the measurements, some of the locations were performing tests with IPv6 traffic, which we also removed

[3]The duration presented in Table 3.1 is calculated after removing the gaps.

from the data. Furthermore, we restrict our analysis to the IP level and do not attempt to decode application protocols.

Table 3.2 contains some general characteristics of the datasets after preprocessing, including the average amount of packets and bytes per second, the average packet size and the total number of active hosts (counted as the number of IP addresses that send at least 1 packet). Our SCADA traces generally contain less hosts than the SNMP traces, but each host generates, on average, more traffic.

dataset	**avg. pkts/s**	**avg. bytes/s**	**avg. pkt size (bytes)**	**#hosts**
SCADA1	251.96	40798.59	162.74	30
SCADA2-control	19.30	3889.49	173.37	19
SCADA2-field	75.76	28236.90	372.30	14
SCADA3	132.54	29505.52	216.33	11
SCADA4	242.89	68059.13	265.05	262
l01t02	118.33	11788.16	99.62	176
l05t01	13.06	1167.25	89.40	56
l11t01	2.97	261.48	88.04	22
l15t04	13.43	2154.00	160.34	140
l17t01	2.47	731.63	296.43	35
l18t01	27.93	2795.61	100.08	83

Table 3.2: General dataset characteristics

3.3 Periodicity

If traffic is indeed generated in a periodic fashion, the time series representing the number of transmitted packets should be fairly stable, assuming the bin sizes are sufficiently large. In addition, due to the increasing interest in Intrusion Detection Systems (IDS) for SCADA [145, 37], we consider time series analysis to be of particular importance, as it is widely used in IDS developed for traditional IT networks and the Internet [136]. In Section 3.3.1 we present an analysis of the *packets/s* and *bytes/s* time series for our datasets.

Next, in Section 3.3.2 we address the periodicity directly, by means of a spectral analysis. We generate and manually inspect periodograms for the packet time series for all (preprocessed) traffic and also for each active traffic source in

our datasets.

3.3.1 Time series

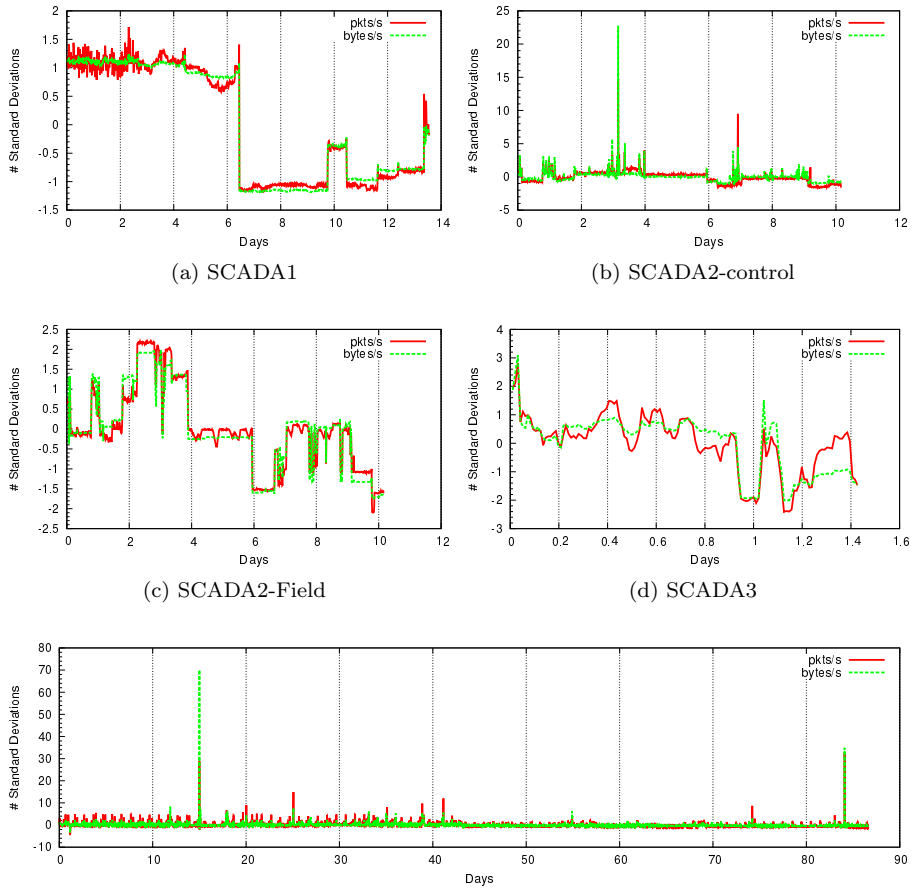
The expected periodicity for SNMP communications is 5 minutes (a typical polling time used in SNMP-based applications [146]) and, from our interviews with the operators, we expect SCADA polling periods to be in the order of a few seconds. Using a time bin smaller then the periods commonly found in these environments would result in a time series with periodic peaks. To reduce this effect, we decide to use 15 min bins, i.e., in the time series generated in this section, each point represents the total number of packets and bytes accumulated over a 15 min interval.

The results for the SCADA datasets are shown in Figure 3.1, while the results for the SNMP datasets are shown in Figure 3.2. In order to able to compare the results, we normalize each plotted point according to it's *standard score*, defined as $(x-\mu)/\sigma$, where x is the measured value, and μ and σ are respectively, mean and standard deviation of the time series. The *y-axis* represents the number of standard deviations the measured value differs from the mean.

The first observation we make is that the *packets/s* and *bytes/s* time series are strongly correlated, which is a common characteristic of network measurements (see for instance [135]). In general the time series are indeed fairly stable, consistent with the assumption that traffic is generated in a periodic fashion. Most datasets display long periods of nearly constant mean, to which we refer to as *baselines*. Clear examples of such baselines are days 4 to 6 in *SCADA2-control* and *SCADA2-field* (Figures 3.1(b) and 3.1(c), respectively) and days 20 to 25 in *l11t01* (Figure 3.2(c)).

Note however that changes do occur. A notable feature in our results is the change in the baseline, for example in *SCADA1* between days 6 and 7 (in Figure 3.1(a)). These baseline changes happen at seemingly arbitrary times, with no apparent pattern. By manually inspecting the traffic, we verify that the main cause for this phenomenon in the SCADA datasets is the start (or end) of high throughput multiple connections. Another cause for baseline changes, observed in *SCADA2-field* and *SCADA3* is the momentary increase/decrease in the amount of variables requested by the SCADA server and/or in the rate at which variables are requested within a single connection.

In [129], it was observed that changes in the physical environment (e.g., fiber cuts) can cause drastic changes in the regularity of the SNMP time series. We speculate that the changes observed here are similar in nature, i.e., they are due to changes in the infrastructure, such as water tanks becoming full and pipes



(a) SCADA1

(b) SCADA2-control

(c) SCADA2-Field

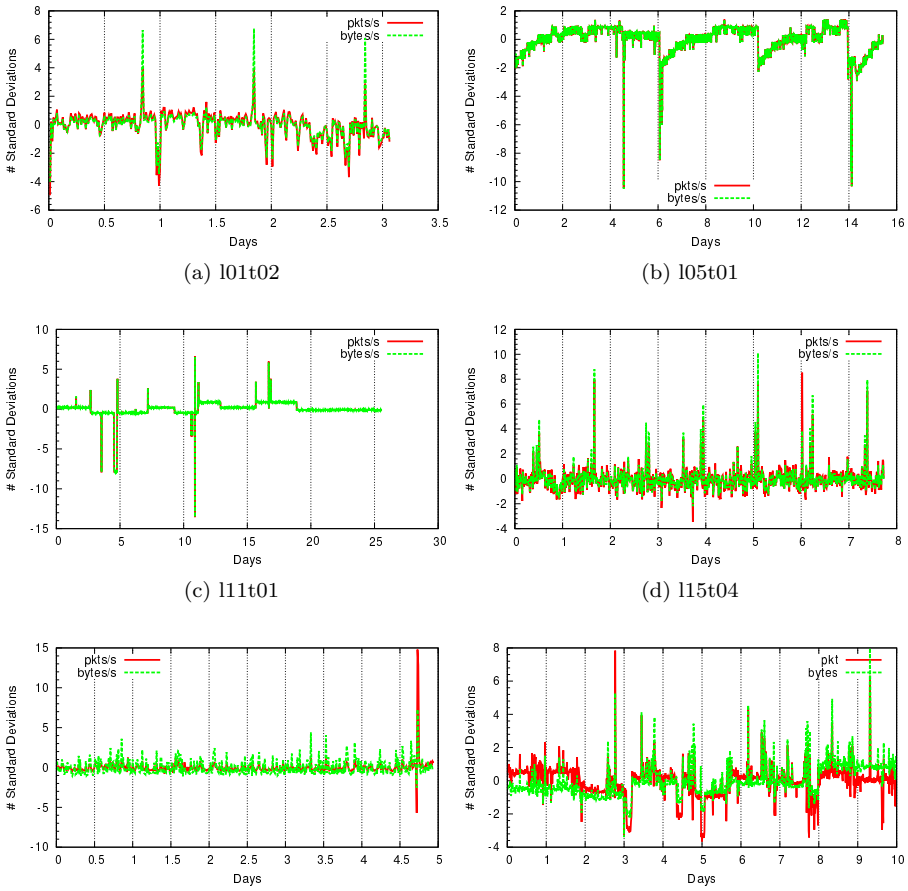
(d) SCADA3

(e) SCADA4

Figure 3.1: SCADA time series

being closed, which affect the traffic patterns.

Another common behavior is the presence of sharp peaks in the time series, for instance, in dataset *SCADA2-control* at day 3 (Figure 3.1(b)) and in dataset *SCADA4* at day 15 (Figure 3.1(e)). In an earlier measurement, where both `tcpdump/pcap` traces and SCADA activity logs (containing information such



(a) l01t02 (b) l05t01

(c) l11t01 (d) l15t04

(e) l17t01 (f) l18t01

Figure 3.2: SNMP time series

as user logins, application crashes and configuration changes) were available, we observed that human-related activities, such as uploading a new software configuration to the PLCs, caused a huge increase in traffic for a short moment. We speculate that the observed peaks are similar in nature.

It is important to stress that the baseline is not a constant value. For

instance, dataset *SCADA1* shows relatively high variation in the first 3 days of measurement. Even if the traffic would be generated in a perfectly periodic fashion, variation should be expected, as the number of periods observed in consecutive bins in the time series is not necessarily constant. For instance, consecutive bins will have a different number of periodic peaks if the bin size is not a multiple of the of the traffic period. In practice, network delays also cause variation.

In the case of *SCADA4*, the variance is high through the whole measurement. In this case however, peaks seem to be caused by periodic activity. We observe periodic peaks with a 10 min period (confirmed by the spectral analysis presented in Section 3.3.2) and peaks with a longer 24 h period.

The SNMP traces present, in general, a similar behavior: time series with a stable throughput, baseline changes and short-term peaks. The variation in the SNMP datasets is generally within 2 standard deviations from the mean. Baseline changes are particularly common in *l11t01* (Figure 3.2(c)). Dataset *l01t02* (Figure 3.2(a)) shows some regular peaks with the period of approximately 1 day. This pattern is discussed in Section 3.4.1.

Dataset *l05t01* (Figure 3.2(b)), however, does not presents any clear baseline, but is seems to be periodic in its unique way. Traffic slowly increases for approximately four days, when it suddenly drops. The reason for the absence of the baseline is unknown to us.

It is important to note that none of our datasets, SCADA nor SNMP, present the diurnal patterns of activity discussed in Chapter 2, i.e., network traffic increasing in the morning, peaks at noon and decreasing in the evening. Clearly, human activity does not have a major impact on the analyzed datasets.

3.3.2 Spectral analysis

The results presented in the previous section support the assumption that traffic is generated in a periodic fashion. But in order to examine it in more detail, we carry out a spectral analysis that consists of the following steps. For each dataset, we generate one time series per IP address. Each time series represents the number of packets sent by that IP address. We also generate an aggregated time series with the number of packets sent by all IP addresses. We then generate a periodogram for each time series and manually inspect the periodograms for high power spectrum components, which indicate periodicity.

The first step in calculating a periodogram is to obtain the DFT. Given a time series $x_0, x_1, \cdots, x_{N-1}$ (typically called *input signal*), the DFT is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}, \text{ with } k = 0, 1, 2, \cdots, N-1.$$

The DFT correspond to a linear combination of complex sinusoids, where each coefficient represents the amplitude and phase of these sinusoids.

The DFT can be efficiently calculated by means of a FFT [43]. Before applying the FFT, we subtract the mean value of the time series from each entry in order to reduce the DC component (or "zero frequency"). Zero-padding is used to adjust the length of the time series to a power of two to increase performance.

The *periodogram* is defined as:

$$\mathcal{P}(X_k) = \|X_k\|^2, \text{ with } k = 0, 1, 2, \cdots, \lceil (N-1)/2 \rceil,$$

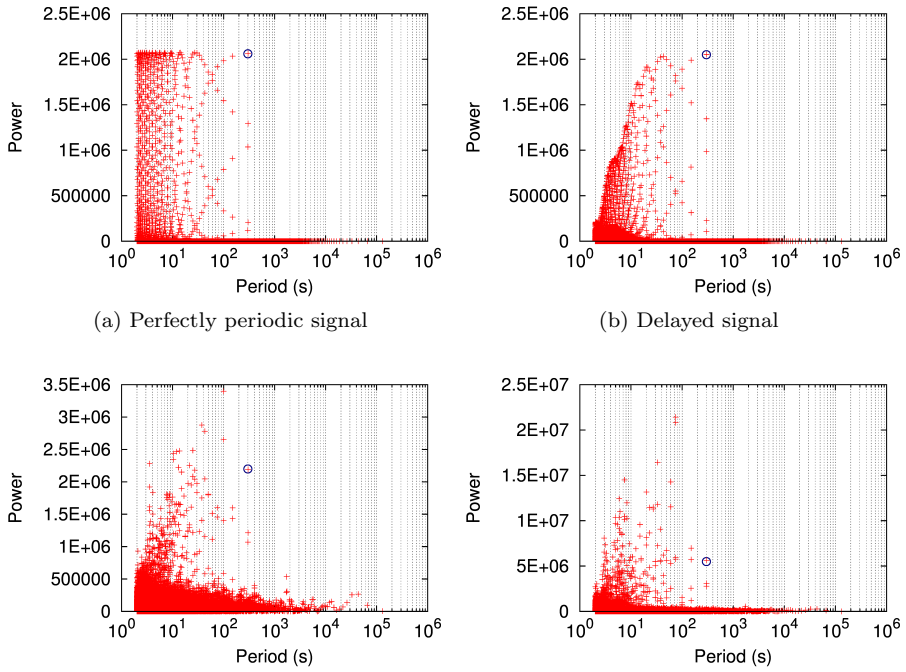
where$\|X_k\|$ denotes the *absolute value* (i.e., the "length") of the complex number X_k. Each value of $\mathcal{P}(X_k)$ represents an estimate for the power at frequency k/N or, equivalently, at period N/k. Note that, following the Nyquist theorem, the periodogram only contains information about periods that are at least twice the sampling period.

Finally, a figure is constructed by plotting $\|X_k\|^2$ (or the "power components") versus the period, measured in seconds. In a periodogram, a high power component indicates periodicity. Note, that the *harmonics* of the period will also display a high power component. Harmonics are found at $p/(n+1)$, where p is the fundamental period and n is the harmonic index. For instance, an input signal with 1 s period has its first harmonic at 0.5 s, second harmonic at 0.33 s, third harmonic at 0.25 s, and so on.

An illustrative example

In Figure 3.3, we provide an example on how periodograms can be used to detect periodicity. We start with perfectly periodic traffic: a burst of 10 packets sent every 300 s. We sample this time series using 1 s bins. We show the effect of three changes in this time series: delays, non-periodic traffic and multiple signals.

Figure 3.3(a) shows the periodogram corresponding to the perfectly periodic time series described above. The high power component at 300 s period is highlighted with a blue circle. As expected, peaks are also present at fundamental period's (300 s) harmonics, e.g., 150 s, 100 s and so on. Note also that the



(a) Perfectly periodic signal

(b) Delayed signal

(c) Delayed signal and noise

(d) Mulitple delayed signals and noise

Figure 3.3: Observing traffic periodicity with periodograms

power is not concentrated at a specific period. The power components adjacent to the fundamental period and its harmonics are also high. This phenomenon is called *spectral leakage*, and is can be caused several factors such as the sampling frequency, the size of the sample and the "format" of the signal.

However, this example is not a realistic representation of network traffic, where delays are common. We create a *delayed periodic signal* by either anticipating or postponing the burst of packets by 1 bin. That is, instead of having a burst of 10 packets every 300s, we allow the period to vary in the range $[299, 301]$ s. Even such a small delay has a large impact on the periodogram, as it can be seen on Figure 3.3(b). The most visible change is the reduced power at the harmonics. The noise level, i.e., the power outside the fundamental period and its harmonics, also slightly increases (although the effect is barely visible

in the figure). The larger the delay, the less clear the periodic pattern will be visible in the periodogram.

In Figure 3.3(c), we add a *random signal*, with amplitude uniformly distributed in the range $[0, 5]$. This simulates the presence of non-periodic traffic in the time series. The periodic pattern is further deformed. The noise level is greatly increased, and, in this case, the power at the first harmonic is actually higher than at the fundamental period.

In our last example, shown in Figure 3.3(d), we show the effect of multiple periodic sources. The time series given as input for the periodogram consist of the random signal presented above and five delayed periodic signals. In addition, the delayed signals are not synchronized, i.e., the instants at which the burst occurs are different for each signal. Again, the power component at one of the harmonics is higher that the one at the fundamental period, however, the difference is even larger.

In summary, the presence of periodicity in a time series (or periodic traffic) can be detected by the presence of peaks in the periodogram. We note, however, that the largest power component does not necessarily represent the fundamental period of the input signal, as it could represent of its harmonics. To find the fundamental frequency, one should identify the right-most peak. Noise and *spectral leakage* further conceal the precise location of the fundamental period.

Real traffic data

An important parameter in a spectral analysis is the sampling frequency f_s used to create the time series, as it determines the maximum observable frequency in the considered signal. According to the Nyquist-Shannon sampling theorem, the sampling frequency must be at least twice the frequency of the signal, so that no information is lost. Ideally then, one would use the highest sampling frequency possible, as to avoid any aliasing artifacts. However, higher sampling frequencies also incur more data points, and as a consequence, high computational requirements.

Given this trade-off, we decided to perform the analysis selecting 1 h subsets from our datasets and sample them using 10 ms bins ($f_s = 100$ Hz). Theoretically, this parameter allows us to observe evidence of periodicity ranging from tens of milliseconds to several minutes, while keeping the required computation time and space acceptable. A limitation of using a 1 h subset to perform our analysis is that not all IP address in our datasets are analyzed, as not all addresses are active throughout the whole duration of the datasets.

The use of higher sampling frequencies would allow us to observe smaller

periods. However, considering that (i) the typical polling period for SNMP applications is 300 seconds, (ii) from our interviews with the operators we expect SCADA polling periods to be in the order of a few seconds, and (iii) our analysis comprises more than 500 source hosts, the chosen sampling frequency appears reasonable.

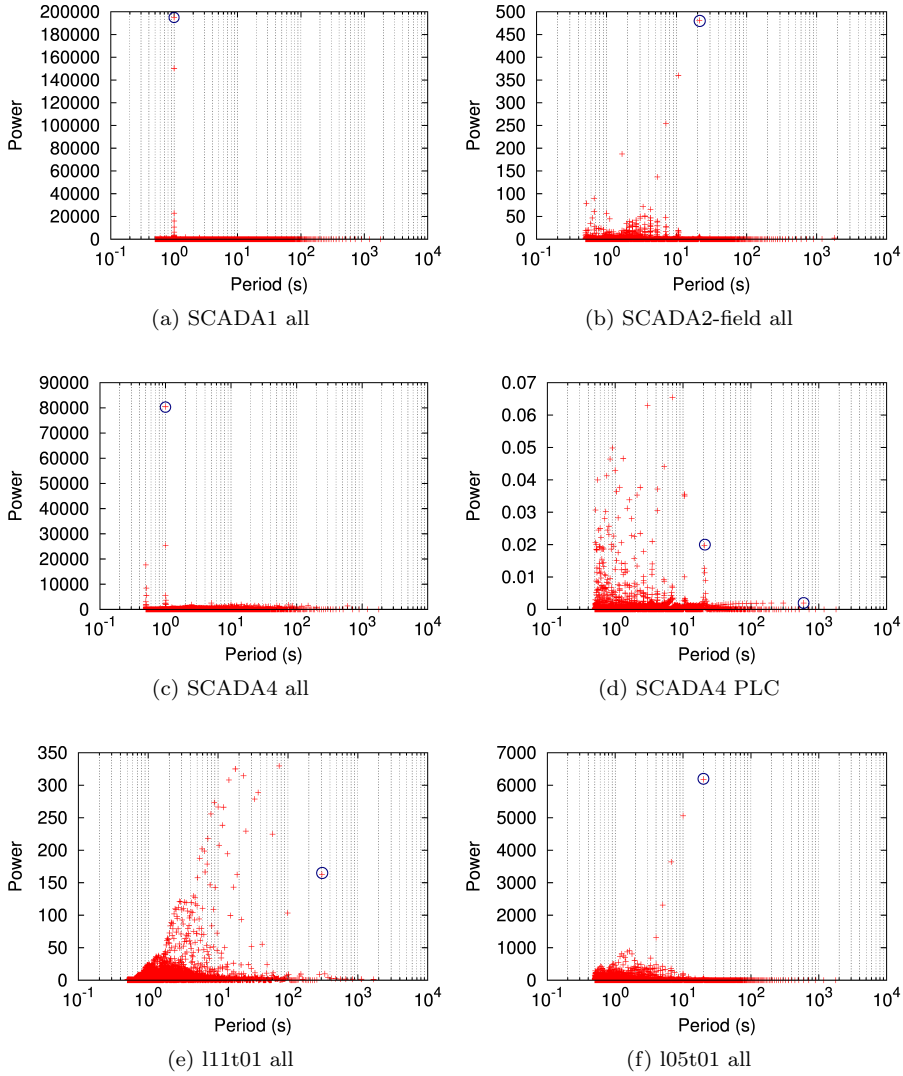
Our results are shown in Figure 3.4. To avoid repetition, we present six periodograms that are representative of the behavior we observe in our datasets. We show the remaining periodograms for the aggregated time series in Appendix B.2. As expected, we find evidence that both SNMP and SCADA traces exhibit periodical behavior, although at different scales.

SCADA1 (Figure 3.4(a)) and *SCADA3* present a dominant period at $1s$, while *SCADA2-field* (Figure 3.4(b)) shows periodicity at $21s$. In these datasets, the PLCs are the largest source of traffic. They generate 71% of the traffic in *SCADA1*, 91% in *SCADA2-field* and 92% in *SCADA3*. Not surprisingly, the period observed in the aggregated time series is the same observed in PLCs' time series.

In contrast, in *SCADA2-control* and *SCADA4* most traffic is generated within the control network. In the former, 54% of the traffic is exchanged between the SCADA server and operator workstations (34%) and between the SCADA server and a server related to authentication (20%). While in the latter most of the traffic is exchanged between the two SCADA servers and the SCADA servers and HMIs (70% combined). The periodogram of the aggregated traffic reflects the periodicity exhibited by these connections. For instance, *SCADA4* (Figure 3.4(c)) presents a strong component at $1s$. The PLCs in this dataset however, display periodicity at $21s$ and $600s$, as it can be observed in Figure 3.4(d).

Most of the SNMP traces show periodicity at a larger period, $300s$, consistent with the typical polling interval of common applications, such as *Cacti* and *MRTG* [115]. The typical SNMP behavior is exemplified with dataset *l11t01*, in Figure 3.4(e), where a high power component can be observed at $300s$. The only SNMP datasets that exhibits a different behavior is *l05t01*. This dataset shows a strong $20s$ period, as shown in Figure 3.4(f).

A summary of the periodicity analysis is presented in Table 3.3. The table reports the main period of the aggregated traffic in seconds, the number of IP addresses that generate traffic in periodic (**#p**) and non-periodic fashion (**#np**). The last column (**na**) reports the number of not analyzed hosts. Given that we analyze a 1 h subset of the datasets, we also analyze only a subset the total number of hosts reported in Table 3.2. The dataset with the largest number of not analyzed hosts is *SCADA4*. Note, however, that in this datasets many



(a) SCADA1 all

(b) SCADA2-field all

(c) SCADA4 all

(d) SCADA4 PLC

(e) l11t01 all

(f) l05t01 all

Figure 3.4: Periodograms

hosts have their IP address changed: 149 in total (of which 60 are present the analyzed subset). These hosts are (logically) relocated to different subnetworks. This relocation event is further discussed in Chapter 5.4.4.

dataset	period (s)	#p	#np	#na
SCADA1	1	7	0	14
SCADA2-control	1	12	1	6
SCADA2-field	21	16	0	7
SCADA3	1	12	0	0
SCADA4	1	83	0	279
l01t02	300	171	5	0
l05t01	60	54	2	0
l11t01	300	19	0	3
l15t04	300	55	21	64
l17t01	300	26	1	8
l18t01	300	60	10	13

Table 3.3: Summary of the periodicity analysis

From the results we conclude that not only the aggregated traffic displays periodicity, but also many source addresses generate traffic in a periodic fashion. Particularly, all PLCs in our SCADA datasets do. Among the analyzed addresses, there is only one non-periodic source in the SCADA traces, in *SCADA2-control*. It corresponds to a server that stores user information and performs access control. This server only needs to be accessed when an operator workstation is in use, thus traffic patterns are dictated by human interaction, and are not expected to be periodic. The unexpected behavior is that the operator workstations in this dataset *do* generate periodic traffic, probably connected to some automated function (e.g., collecting real-time information from the infrastructure). The majority of the workstations present a strong $1s$ period component.

When considering the SNMP datasets, only on *l11t01* all IP addresses display a clearly periodic pattern. For the remaining datasets, at least one periodogram does not display periodic behavior. After closer inspection, we observe three main types of non-periodic traffic (at the time scales considered in our analysis):

- **Inactive hosts:** Hosts that generate little traffic, or become inactive for

extended periods of time. This is observed in datasets *l01t01*, *l05t01*, *l15t04* and *l17t01*.

- **Erratic hosts:** Host that generate traffic with no distinguishable pattern. Observed in datasets *l01t02* and *l18t01*
- **Daily periods:** Three hosts in *l01t02* present recurring connections every night. However, this period is excessively large to be captured by our analysis.

3.3.3 Discussion

The results discussed in this section show that both SCADA and SNMP traffic display periodicity. The stability in the throughput expected from sources generating periodic traffic is confirmed in our time series analysis. However, it is important to note that the throughput is not constant. The occurrence of baseline changes, peaks and variation is common. These sources of instability present a challenge for anomaly detection methods based on time series. For instance, simply comparing the current packet load to historical values, as proposed in [145], might generate a non-negligible number of (false) alarms. Further research is necessary to establish the cause of sources of instability in the time series, so they can be taken into account in the design of realistic models for packet generation for these environments.

Stronger evidence for traffic periodicity is provided by the spectral analysis. Strong power components are present in all datasets, implying traffic periodicity. In particular, our analysis shows that all PLCs in our datasets present this characteristic. Finally, SCADA and SNMP datasets present periodicity at different time scales. While the former present dominating periods in the order of a few seconds (typically 1 s or 21 s), the latter typically exhibit a 300s period.

The presence of periodicity suggests that SCADA traffic patterns should be predictable. In Chapter 6, we exploit this characteristic to learn traffic patterns generated by SCADA protocols, more specifically, Modbus and MMS, and to detect anomalies in their behavior.

3.4 Connection Matrix

The objective of second research questions is to determine whether SCADA traffic presents a *stable connection matrix*. In SCADA networks, nodes are not expected to commonly leave or join the network as is the case in traditional IT

networks. For instance, while visitors connecting their laptops to the network might be common in a traditional IT network, they are not expected to do so in a SCADA network.

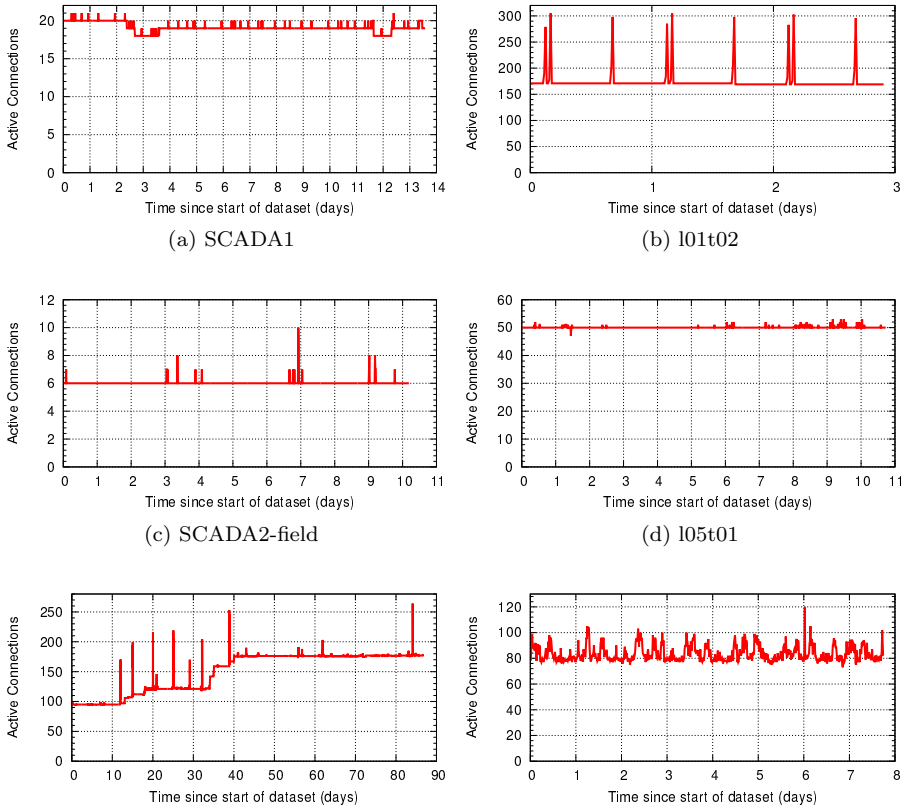
In order to answer this research question, we create an IP-level connection matrix, which shows "who is connecting to whom". In this context, we say there is an active connection between host A and host B if a packet is sent from one host to the other within the duration of a bin, regardless of the direction. The goal of the tests presented in this section is to verify whether the set of communicating hosts, or connections, remains stable over time.

We divide the datasets into non-overlapping bins of size n seconds, and perform three different, albeit related, tests. First, in Section 3.4.1 we generate time series for the number of active connections, which allows us to observe at a glance if there are changes in the number of connections. In Section 3.4.2, we present the second test, which quantifies the number of added and removed connections, by comparing the connection matrix of consecutive bins. For the last test, discussed in section 3.4.3, we monitor unique connections, i.e., we create a time series where each connection is only counted once, in the bin at which its first packet is exchanged.

3.4.1 Time series

For our first test, we build the time series of the total number of active IP-level connections. As in the time series analysis in Section 3.3.1, we use bins of 15 minutes. The time series allows us to observe at a glance whether the number of connections stays constant over time. Figure 3.5 shows the results for six datasets. To avoid repetition, we present the remaining time series in Appendix B.2.

The results present some of the same characteristics from the time series for the number of packets and bytes presented in Section 3.3.1. All SCADA datasets present a stable number of active connections. Also, baselines changes and peaks of activity are present. *SCADA1* shows a baseline of 19 connections, with frequent shifts to 20 connections, as shown in Figure 3.5(a). In *SCADA2-field* (Figure 3.5(c)), the baseline of 6 active connections represents the connections between the SCADA server and 5 PLCs, plus those between the SCADA server and its backup. With exception of a few short-lived the peaks, the time series for this dataset is remarkably stable. In Figure 3.5(e) we show the results for *SCADA4*. This dataset presents two large changes responsible for most peaks and baseline changes in the time series: the relocation of hosts and the use of a redundant SCADA server. These events are described in details in



(a) SCADA1 (b) l01t02

(c) SCADA2-field (d) l05t01

(e) SCADA4 (f) l15t04

Figure 3.5: Current IP connections

Chapter 5.4.4. This dataset also shows a gradual increase in the number of connections, as new nodes are added to the network. Some of the large peaks represent nodes having their IP address changed, or the backup SCADA server taking over tasks from the SCADA server.

The results for this test divide the SNMP datasets in two groups. The first group, formed by *l01t02*, *l05t01* and *l11t01*, have a behavior much closer to that of SCADA, i.e., long stable periods with a few peaks and sporadic

baseline changes. While the second group, *l15t04*, *l17t01* and *l18t01*, present considerably more variation.

During the time series and periodicity analysis, we observed that *l01t02* contains traffic with a strong daily pattern. We observe the same pattern in Figure 3.5(b). Apart from the clear baseline, *l01t02* presents periodic peaks. After closer inspection, we verified that one of the monitors is only active during three moments, around 11 h, 22 h and 23 h every day, causing the peaks in the number of connections. Dataset *l05t04* presents a very clear baseline of around 50 connections, as can be observed in Figure 3.5(d). Interestingly, the unusual behavior observed in the packet and byte time series (Figure 3.2(b)), where traffic slowly increases for a few days then suddenly drops is not observed here. Therefore this variations the throughput of existing connections, and not by new connections.

Although the datasets in the second group present more variation, a baseline can still be seen. The result for dataset *l15t04* is shown as an example, in Figure 3.5(f). It presents a baseline of approximately 93 connections, but with high variation in the range $[73 - 120]$. Note once again that diurnal patterns of activity are not present.

3.4.2 Connection matrix changes

The time series does not provide much insight on *how* the changes in the connection matrix occur. To study this characteristic a second test is proposed. First, we build a list of connections that are active in each bin. We then compare consecutive bins, creating two lists: one containing *added* connections, i.e., connections present in the latter bin but not in the former; and the other containing *removed* connections, i.e., connections present in the former bin but not in the latter. Finally, we study the size of these lists over time. We note that when using small bin sizes, such as 15 min used in the previous section, we observed many intermittent connections, i.e., connections were *removed* in a bin and *added* in the following one. Therefore, for this analysis we use bins of 1 hour, as our objectives here is to study the long-term stability the connection matrix.

The results for this test are summarized in Table 3.4. The column *total* contains the total number of connections in a dataset. We present the *mean*, standard deviation (*std*) and maximum (*max*) for the number of *added* and *removed* connections. The last column, *changed*, contains the percentage of bins with at least 1 change (either *added* or *removed*). We use $< 1\%$ to denote a percentage that is below 1% but not 0. Finally, we use the traditional IT

dataset (*IT*), presented in Chapter 2 to put our results into perspective.

dataset	total	added			removed			changed
		mean	std	max	mean	std	max	
SCADA1	32	1%	2%	10%	1%	2%	10%	31%
SCADA2-control	31	1%	4%	29%	1%	4%	29%	15%
SCADA2-field	17	1%	5%	40%	1%	4%	40%	11%
SCADA3	10	0%	0%	0%	0%	0%	0%	0%
SCADA4	507	< 1%	3%	46%	< 1%	3%	45%	15%
l01t02	305	4%	11%	43%	4%	12%	44%	26%
l05t01	92	1%	1%	11%	1%	1%	9%	28%
l11t01	21	< 1%	1%	33%	< 1%	1%	33%	2%
l15t04	497	17%	13%	47%	17%	13%	47%	97%
l17t01	35	3%	3%	14%	3%	3%	15%	87%
l18t01	126	3%	3%	17%	3%	3%	17%	94%
IT	23292	73%	8%	96%	73%	8%	92%	100%

Table 3.4: Connection matrix changes

As expected, our results show that the connection matrix for the SCADA datasets present small changes over time. On average, not more than 5% of IP-level connections are added or removed. In general, the standard deviation is also small. Despite the low mean, with exception of *SCADA3*, all SCADA datasets present a large maximum change, ranging from 10% to 46%. A surprising behavior is that, in spite of the small number of changes in each bin, the occurrence of changes is not rare, i.e., the percentage of bins with at least 1 change is not negligible.

Among the SCADA datasets, only *SCADA3* presents *no* changes at all; the remaining datasets present at least 11% of bins with at least one change. The occurrence of changes is more common in *SCADA1*, where a surprisingly 31% of bins contain at least one change. Both datasets collected at *SCADA2* present a very similar distribution. The 40% maximum of added and removed connections in *SCADA2-field* can be explained by the small amount of active connections in the network. The maximum of 40% represents an increase (or decrease) of only 4 connections. Dataset *SCADA3* is the only dataset that does not present any change, however, it should be noted that the duration of this dataset is also very small, only 1.5 days. While in *SCADA4*, large maximum is caused by the (logical) relocation of many hosts, which we later discuss in Chapter 5.4.4.

The two SNMP groups identified in the time series analysis are also visible here. While the first group presents a maximum of 28% of changes, the second group, presents a minimum of 87% of changes. In *l01t02* the high standard

deviation, and maximum can be explained by the periodical peaks at 11 h, 22 h and 23 h, as discussed in the previous section.

When comparing the results to *IT*, the connection matrices of our SCADA datasets presents a small number of changes over time. The *IT* dataset presents by far the highest mean and maximum of added (and removed) connections. Also, in this datasets every bin presents at least 1 change.

3.4.3 Unique connections

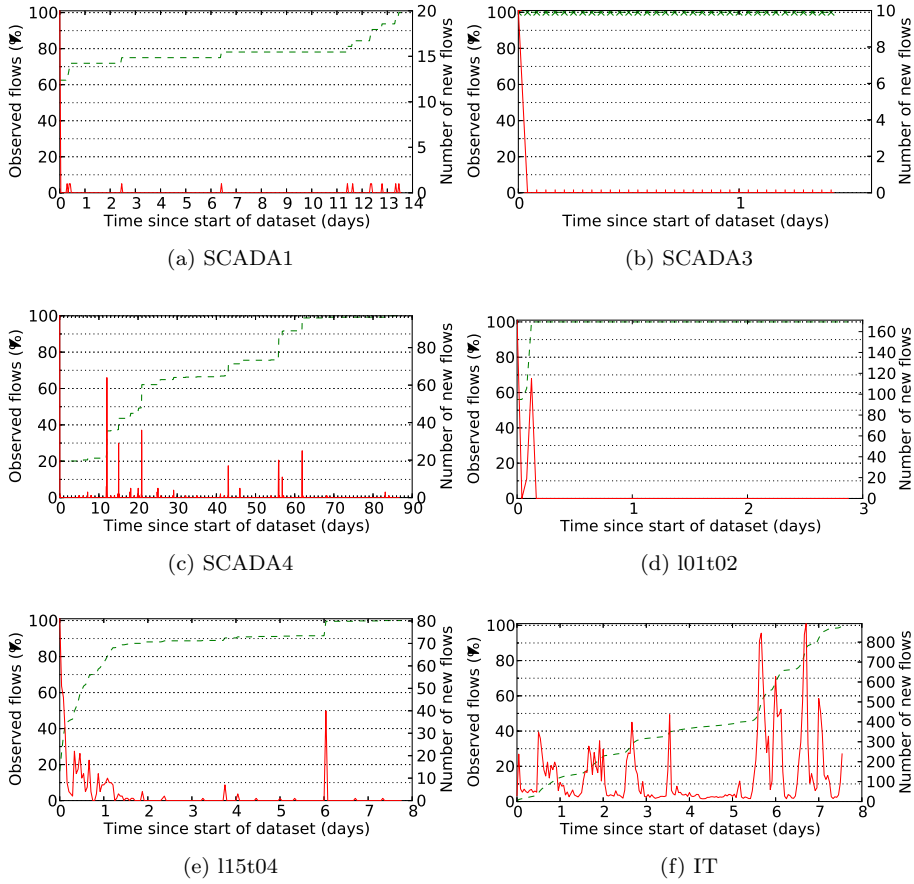
The last characteristic of the connection matrix we study is when a given connection is observed for the first time. The idea behind this test is to verify if the changes observed in the previous test are mostly caused by previously unseen connections, or if the set of connections is stable. To that end, we construct a time series of unique connections, that is, each connection is only counted once, in the bin at which its first packet is exchanged.

We present the results for six datasets in Figure 3.6. Each plot in the figure shows two curves: the full red line shows number of connections first observed in a given bin, and the dashed green line shows the percentage of the total number of connections that have been observed up to that bin. For ease of visualization, we count the amount of unique connections introduced in each 1 h bin. In this test the use of smaller bin sizes simply increases the resolution of the *x-axis*. Again, we make use of the *IT* dataset for comparison. The results for the remaining datasets can be found in Appendix B.2.

For both SCADA and SNMP datasets, a significant number of connections is observed within the first day of the dataset. In fact, all connections are observed within the first day for *SCADA3* and *l01t02*, Figures 3.6(b) and 3.6(d) respectively. After the first day, new connections tend to appear in isolated peaks.

Many of the large variations observed in the previous tests are not observed here. For instance, the majority of small peaks in the number of active connections observed in *SCADA1* (Figure 3.5(a)) are not present in Figure 3.6(a). Also, and the daily peaks of active connection observed in *l01t02* (Figure 3.2(a)) are not present in Figure 3.6(d). This means that these variations are caused by previously observed connections.

In *SCADA4* (Figure 3.6(c)) the peaks of new connections are considerably larger than in the other SCADA datasets; the larger peak (at day 12) consists of more than 60 new connections. The high number of connections in this dataset is attributed to two events that occur in the network: hosts having their IP address changed and a redundant server taking over the activity of a



(a) SCADA1

(b) SCADA3

(c) SCADA4

(d) l01t02

(e) l15t04

(f) IT

Figure 3.6: Number of new IP connections over time as absolute number (right scale) and relative percentage (left scale)

main server in the network. These events will be discussed in more detail in Chapter 5.

The results for dataset *l15t04*, depicted in Figure 3.6(e), show that connections are continuously introduced until the end of the second day. A large peak is also present on the sixth day of measurement, however, these connections are

not long-lived (Figure 3.5(f)), thus probably relate to some manual activity.

Finally, we note that *IT*, shown in Figure 3.6(f), presents a very different pattern. Unique connections appear following a diurnal pattern, rather then in isolated peaks. Even after a week of measurements, new unique connections are still common.

3.4.4 Discussion

From the analysis presented in this section, we can conclude that changes in the connection matrix are not rare. This observation needs to be incorporated in models that describe SCADA operation. For instance, the IDS proposed in [144] generates alarms for each newly observed (transport-level) connection, reporting it as an anomaly. The underlying assumption is that new connections are a rare event. If this IDS solution was to be deployed in the SCADA networks considered in our analysis, a considerable number of (false) alarms would be generated, as changes in the connection matrix, although small, are common.

Half of our SNMP traces present the same behavior as SCADA traces: a connection matrix with a small number of changes. However, the other half display a rather unstable connection matrix, where the occurrence of changes is the rule rather than exception.

Finally, our SCADA datasets display a considerably more stable connection matrix than that of a traditional IT environment. The time series for the number of active connections show clear baselines, changes in the connection matrix are generally small, and a significant number of connections is observed within the first day of our measurements. This relative stability suggests that keeping track of connections over time might be a useful approach to detect unauthorized activity in SCADA environments. This topic is further explored in Chapter 5, where we investigate how the connection matrix stability can be exploited for anomaly detection.

3.5 Conclusions

In this chapter we continued our characterization of SCADA traffic, and performed a comparison with SNMP, a widely used protocol from the (traditional) IT world. In RQ 3.1 we address the question whether the SCADA traffic is periodic, as it is mostly generated by automated processes. In Section 3.3 we used time series and spectral analysis to confirm that traffic indeed presents strong periodic patterns. The packets and bytes time series present clear base-

lines, and the periodograms reveal that a large number of hosts, in particular all PLCs in our datasets, generate traffic periodically. However, one should observe that changes do occur. Time series present baseline changes at seemingly arbitrary time intervals and some of the hosts do not generate traffic in a periodical fashion.

In RQ 3.2 we investigate the stability of the connection matrix. In Section 3.4, we find that the time series for the number of active connections is very stable, and changes tend to be small. However, the occurrence of changes is not rare. Most of our datasets contain a significant number of changes over time.

Finally, in RQ 3.3 we investigate the possibility of using SNMP traffic traces to emulate SCADA behavior. Despite the fact that our SCADA and SNMP traces share some characteristics, there are also important differences. SNMP typically displays periodicity at larger time intervals (typically 300 s) when compared to SCADA traffic (1 s or 21 s). Furthermore, half of our SNMP traces display a rather unstable connection matrix, where the occurrence of changes is the rule rather than exception. Therefore, we do not recommend emulating SCADA behavior using SNMP traffic traces.

Although our analysis showed interesting results, more research is needed to fully understand the particularities of SCADA traffic. Nonetheless, the results presented in this chapter are the motivation of the two anomaly detection approaches proposed in this thesis. In Chapter 5, we use whitelists to exploit the connection matrix stability, and, in Chapter 6, we propose *PeriodAnalyzer*, an approach that automatically learns traffic patterns generated by SCADA protocols and detects anomalies in their behavior, by exploiting traffic periodicity.

CHAPTER 4

SCADA Security

4.1 Introduction

SCADA systems were originally developed using special-purpose embedded devices, which communicated using proprietary protocols. These systems were designed to operate in isolation from other systems, so vendors and operators believed they could rely on the *air gap* for protection. That is, as the SCADA network would be physically isolated from any other networks, attackers could not gain access to it. Also, vendors and operators relied on *security through obscurity*, that is, the belief that the lack of publicly available information regarding SCADA system made them secure.

However, in modern SCADA deployments we observe that interconnection with other systems is becoming commonplace. SCADA networks can be accessed from corporate networks, engineers have remote access via Virtual Private Networks (VPNs) and even access from the Internet is possible in some instances. In addition, the special-purpose embedded devices are being replaced with COTS devices and the proprietary communication technology with the TCP/IP stack. These changes have a deep impact in the security of SCADA networks [87].

In this chapter, we review the literature of SCADA security. First, in Section 4.2, we discuss the differences with traditional IT security. Next, in Section 4.3, we discuss documented security incidents in SCADA. In Section 4.4, we provide a review of industry standards and academic research on the topic. Finally, in section 4.5, we summarize our findings.

4.2 Differences with Traditional IT

The security requirements for traditional IT networks are commonly summarized as the triple: Confidentiality, Integrity and Availability (CIA) [139]. In fact, for a typical business enterprise these requirements are ordered in importance:

1. **Confidentiality:** the property of a computer system whereby its information is disclosed only to authorized parties, has the highest priority;
2. **Integrity:** the characteristic that alterations to a system's assets can be made only in a authorized way, follows as second priority;
3. **Availability:** the property that a system is ready to be used immediately, has the lowest priority.

Safety has the highest priority in a SCADA environment. While in a traditional IT environment outages typically cause only monetary losses, in SCADA systems, a system outage can also bring risk to human and public safety, and damages to equipment. Therefore, the priority order is reversed: availability has the highest priority; integrity comes as close second, as safe operation depends on the correctness of the data; and confidentiality has the lowest priority, as data has to be analyzed within context to have any value [82].

In addition to different priorities, there are several other key differences between these environments, including:

- **Constrained resources:** SCADA equipment typically has constrained resources making it a challenge to implement traditional cryptographic algorithms developed for traditional IT environments. This problem is exacerbated by the real-time requirements of SCADA applications.
- **Information vs. assets:** Traditional IT environments have as priority protecting the information during storage, transmission and computation. However, in SCADA environments the focus lies on protecting the devices in the field network, e.g., PLCs and RTUs, as those are responsible for controlling the physical process. Protecting the SCADA server is also important, as it can influence the edge devices.
- **Patching problem:** Applying security patches is one of the primary mechanisms to prevent the exploitation of vulnerabilities in traditional IT

environments. However, this practice is not common in SCADA environments, as every change in the systems requires comprehensive testing. In extreme cases, changes in the environment might require renewed certification [94].

- **Component Lifetime:** The lifetime of SCADA components is typically much longer then that of traditional IT systems components. Changes in IT are fast paced, in the order of 2 to 5 years, due to constant innovation. In contrast, changes in SCADA are slower, in the order of 15 to 20 years, as solutions are developed for specific scenarios. Connected to the patching problem, this long lifetime commonly results in legacy components that are operational but no longer maintained by vendors.

A summary of the differences between the environments is shown in Table 4.1 (adapted from [34]). For a detailed description of the differences between SCADA and traditional IT environments, the interested reader is referred to [137].

System Characteristics	SCADA	Traditional IT
Number of users	low	high
Multi-vendor	limited	common
Lifetime (years)	15 to 20	3 to 5
Outage tolerance	low/none	medium/high
Delay tolerance	low (real-time)	medium/high
Maintenance and Upgrade		
Patching	rare	common
Unsupported soft/hardware	common	rare
Soft-/hardware releases	rare (small changes)	frequent (extensive changes)
Frequency soft/hardware update	very low	medium/high
Security Practices		
Security awareness	low (but rising)	high/very high
Availability of security expertise	low	high/very high
Adoption of security audits	rare	frequent
Real-time security checks	rare/unavailable	common
Security Countermeasures		
Use of Antivirus	rare/unavailable	common
Physical security	difficult in remote sites	high
Use of firewalls and IDSs	rare/unavailable	common

Table 4.1: Differences between SCADA and traditional IT (based on [34])

4.3 Documented Incidents

A survey with 200 industry executives from electricity utilities in 14 countries performed by Baker et al. [10] showed that 80% had faced a large-scale denial-of-service attack, and 85% had experienced network infiltrations. According to a report from the American DHS, the number of attacks to SCADA networks grew from 9 in 2009, to 198 in 2011 and 171 in 2012 [78].

While most of these attacks are not published, some cases are well-documented. In this section, we review some of these cases, with the intent of showing how fragile the ideas of the air gap and security through obscurity are.

4.3.1 Maarochy water breach

In March 2000, the Maarochy Water Services on Queensland's Sunshine Coast in Australia was hacked by a former contractor, after his job application for the local council was rejected. The attacker managed to gain access to the field network, successfully taking control of 150 sewage pumping stations. The attacker proceeded to spill over one million liters of untreated sewage into local parks, rivers and private properties [131].

During a period of three months, communications sent by radio links to wastewater pumping stations were lost, pumps were not working properly, and alarms put in place to alert the staff of faults were not going off. Only after an engineer started monitoring all network traffic, the attack was discovered. The attacker was later discovered to have gained access to the network via wireless communications. This attack shows that the air gap might not exist in practice, and demonstrated a typical threat for security through obscurity: an insider attacker might know the details of the system, even if such details have never been made public.

4.3.2 Slammer at Davis-Besse

In 2003, the *slammer* worm infected at least 75000 hosts, exploiting a vulnerability on Microsoft's SQL Server. The worm caused numerous network outages and a series of unforeseen consequences, such as canceled airline flights, interference with elections, and ATM failures [110].

One of the victims of *slammer* was the Davis-Besse nuclear power plant in Ohio, United States. The infection overloaded the plant's network, causing a safety-related system to be unavailable for almost 5 hours, while a plant process

computer remained unavailable for over 6 hours. Fortunately, the impact of the worm was limited, and operators did not loose control over the infrastructure. The attack vector was a link between the corporate and the SCADA networks which bypassed the existing firewall [17]. The slammer infection at Davis-Besse shows that SCADA networks might be vulnerable to the same threats of traditional IT networks, and specialized knowledge is not necessary to attack then, exposing another problem with security through obscurity.

4.3.3 Blaster at CSX Corp

Also in 2003, the *blaster* worm infected over 100000 systems exploiting a vulnerability on Window's Remote Procedure Call (RPC) interface. A surprising characteristic of the blaster infection was its persistence, i.e., a large number of hosts remained infected after 1 year since the initial outbreak; despite clean up efforts [9].

In August 2003, CSX Corp, the third-largest railroad company in North America, reported an outage caused by the blaster worm. At about 1:15 AM, a variant of the blaster virus was found to be interfering with a train signaling system. According to reports, the systems were "substantially restored" (whatever that may be) to normal by about 9 AM. Several freight and passenger trains faced delays of 40 minutes to several hours due to this incident [68, 73]. Like the slammer infection at Davis-Besse, the blaster infection at CSX Corp is an example of an attack without specialized knowledge regarding the SCADA system.

4.3.4 Stuxnet

Stuxnet is considered the most advanced attack to SCADA environments to date, and one of the most complex ever created. The attack components included four zero-day exploits[1], the first known PLC rootkit[2], antivirus evasion techniques, peer-to-peer updates and stolen certificates from trusted Certification Authorities (CAs) [55]. It has been speculated that the objective of the attack was to damage centrifuges used in the Iranian uranium enrichment project [16, 36, 57].

After the initial infection, probably via an infected USB drive, the stuxnet worm started to propagate itself looking for its initial target: engineering work-

[1] A zero-day exploit is an exploit that makes use of a previously unknown vulnerability.

[2] A rootkit is a software that exploit some vulnerability with the goal of obtaining privileged access to a system.

stations used to program PLCs. The propagation was performed by exploiting vulnerabilities in other computers connected to the same LAN as the infected host. The worm also copied itself to removable USB drives, as the target might be located in a different, "air gapped" LAN. Both LAN and USB propagation techniques made use of zero-day exploits. The attack payload was only extracted when the target workstation was identified. Once this happened, Stuxnet infected the software used to program PLCs. The malware then waited for some specific PLC models to be connected. It then uploaded malicious code to these PLCs, effectively leaving them under the attacker's control.

During the whole process, Stuxnet used different mechanisms to evade detection. The malicious binaries were signed with valid certificates. Once infected, the binary used a rootkit to hide itself. A similar approach was taken to hide the modified code on the PLCs. Finally, the malware sent modified messages to the SCADA server emulating the "normal" PLC behavior, with the objective of hiding the effects of the attack from operators.

In the report describing the attack, Symantec claims that to develop Stuxnet it would be necessary to construct a large test infrastructure that "mirrored" the target environment, a very costly and time consuming effort [55]. The complexity of this attack has led to speculation that only a nation-state would have the capabilities to develop it [16, 36, 57]. In June 2012, the New York Time published an article claiming Stuxnet was developed by the United States government, with help from the Israeli government, as part of a larger cyberattack operation named "Olympic Games" [128]. Stuxnet shows that, even if the air gap exists, techniques can be used to circumvented it. The attack also exposes the ineffectiveness of security through obscurity against a resourceful attack.

4.4 Securing SCADA

In this section, we review standards and recommendation documents proposed by the industry to secure SCADA systems in Section 4.4.1. We then proceed to discuss the academic research in the area, in Section 4.4.2, we discuss general aspects of SCADA security, followed by Section 4.4.3, in which we focus on intrusion detection approaches.

4.4.1 Standards and recommendations

The International Organization for Standardization (ISO) has created two widely used standards concerning computer security: ISO/IEC 15408 [90], com-

monly referred to as "common criteria", and the ISO 27000 series [91]. Although developed for the traditional IT domain, some of the concepts discussed in these documents can be applied to the SCADA domain. In fact, a draft document from the ISO 27000 series, ISO 279019, has the descriptive title: "Information security management guidelines based on ISO/IEC 27002 for process control systems specific to the energy utility industry".

International Electrotechnical Commission (IEC) has applied the common criteria standard to the context of power substation communications. The results of this study are published as IEC 62210 [80]. This document was later made obsolete by the IEC 62351 series [83], which describe several aspects of confidentiality, integrity and authentication in the power substation context. This series addresses the security of different protocols, such as TCP/IP, MMS and IEC 60870-5, describing topics such as attack scenarios, key distribution and mitigation.

Together with IEC, the International Society of Automation (ISA) addresses security of SCADA environments in two equivalent series of standards, ISA 99 and IEC 62443 [82]. The topics discussed in these series include compliance metrics, establishing and operating security programs, patching, security controls and specific requirements for equipment.

The Institute of Electrical and Electronics Engineers (IEEE) has also published three standards relating to SCADA security. IEEE 1402 [84] provides a guide to security issues related to both physical and cyber intrusions at power substations. The two remaining standards deal with very specific problems. IEEE 1686 [86] describes detailed functions and definitions for Intelligent Electronic Devices (IEDs) and IEEE 1711 [85] describes a retrofit cryptographic protocol for communications between RTUs and SCADA servers.

Worldwide, several national organizations also have developed guidelines and best-practices documents. Examples of such efforts are the Dutch SCADA Security Good Practices for the Drinking Water Sector [102], the Recommended Guidelines for Information Security Baseline Requirements for Process Control, Safety and Support ICT Systems [113] by the Norwegian Oil and Gas Association, and the NERC reliability standards on critical infrastructure protection [112].

4.4.2 General aspects

Several aspects of SCADA security are also being addressed by the (academic) research community. For instance, attack taxonomies have been proposed for two popular SCADA protocols, *Modbus* [77] and *DPN3* [52]. The problem of

threat analysis, that is, evaluating the impact that attacks and failures have in the controlled infrastructures, has been studied in [6, 7, 31, 140]. In addition to threat analysis, Baiardi et al. [7] also discuss mitigation strategies. Ten et al. [140] proposes an even more complete solution, including monitoring, detection, threat analysis and mitigation. A similar approach is presented in [31], however, in their paper the authors focus on the behavior of the physical process controlled by the system.

Another interesting area of research is how to implement security services, such as authentication, integrity and confidentiality for SCADA systems [33]. Given the limited resources of devices in the field network, a challenging task is to address the conflict between the real-time requirements of these systems and the additional delay caused by encryption algorithms [149, 150, 143, 121, 39]. A common solution to circumvent this limitation is to encrypt the communication link between the field and control networks using a "*Bump-In-The-Wire (BITW)*" [149, 150, 143]. The solution, depicted in Figure 4.1, consists of adding a BITW gateway in both field and control networks, which is responsible to encrypt all data traversing it.

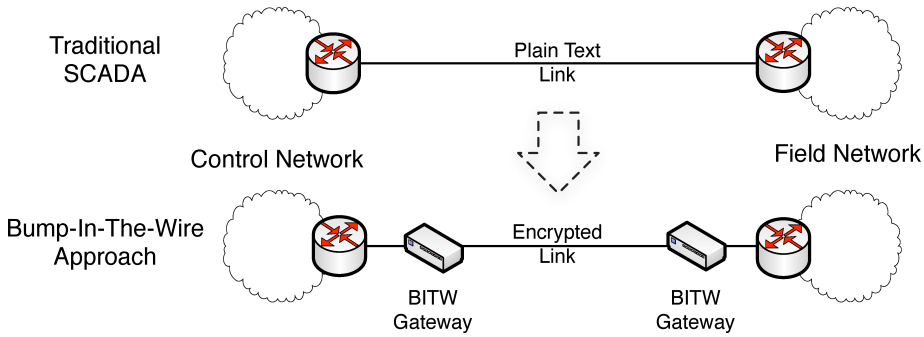


Figure 4.1: The "bump-in-the-wire" approach

A related problem deals with the management of cryptographic keys. Some of the challenges in this area are discussed by Pi and Sitbon [121]. Choi et al. [39] provide a SCADA specific solution that supports message broadcasting and secure communications.

4.4.3 SCADA IDS

In Figure 4.2 we show the four dimensions used to classify the existing approaches towards SCADA IDS: *source of audit data*, *detection method*, *validation method* and *process-aware detection*. The first two dimensions are commonly used [5, 47], and the last two provide practical information regarding the surveyed SCADA IDS.

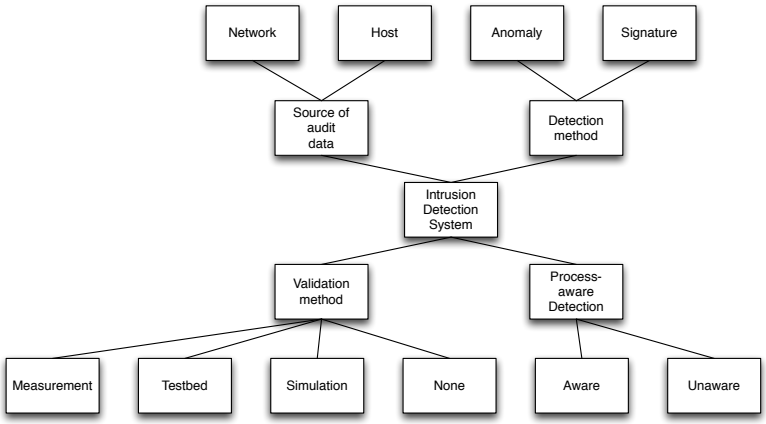


Figure 4.2: SCADA IDS classification dimensions

Regarding the source of audit data, an IDS use be either host or network-based detection [5, 47]. The first relies on data collected in a host, such as system logs and system calls, and the latter on data collected on the network, commonly traffic measurements made at a central monitoring location. As discussed in the previous section, devices in the field network typically have limited resources, therefore, host-based detection is challenging. Most devices simply do not have the resources necessary for supporting new functionality. Also, such new functionality could imply additional delay to the tasks performed, which is undesirable given the real-time requirements of a SCADA system.

Regarding the detection method, an IDS system can be either signature-based, or anomaly-based [5, 47][3], with "the former relying on flagging all behavior that is abnormal for an entity, the latter flagging behavior that is close

[3]In Debar's convention [47], signature and anomaly methods are referred to misuse and behavior methods, respectively.

to some previously defined pattern signature of a known intrusion". In traditional IT environments, anomaly detection techniques tend to display a large number of false-alarms due to the enormous variability of network traffic [133]. On the other hand, SCADA traffic is expected to be more *well-behaved*, due to characteristics such as the traffic periodicity and a stable connection matrix (discussed in Chapter 3), making anomaly-based methods particularly attractive. Not surprisingly, network anomaly-based approaches dominate the SCADA literature. Among the 25 surveyed papers, only two present a host-based technique [70, 152], and a single one presents a signature-based approach [122].

Regarding the validation method, we observe four possibilities among the surveyed approaches. The first is the use of *real-world* audit data, that is, data collected in a operational SCADA environment. The second is the use of a *testbed* environment, which typically emulates a simplified version of a real system. Third, is the use of *simulation*. Typically this approach is used to study the physical process itself, while the infrastructure (e.g., PLCs, servers and network) are abstracted. Finally, some papers limit themselves to the description of the detection mechanism, without any validation whatsoever.

One of the fundamental differences between traditional IT and SCADA environments is that the latter is closely related to a physical process. This opens a new possibility: the detection of intrusions based on process information. If a mathematical model of the process is available (or can be created), one can theoretically evaluate the impact of commands issued in the SCADA system on the controlled process. Alarms can be raised if a command causes a system to reach an undesirable state. Note that the use of process information is not related to the source of audit data. The information can be both directly polled from hosts (e.g., SCADA server and PLCs) or extracted from network data. Incidentally, all surveyed process-aware approaches also happen to be network-based.

A summary of our survey is presented in Table 4.2. In the following subsections, we describe each of the surveyed approaches in more detail.

Host/Anomaly based

In [70], an anomaly detection approach based on log information collected from SCADA servers is proposed. These logs contain timestamped information, such as configuration changes in a PLC. The proposed system requires domain knowledge to assign weights (relating to risks) to the logged events. This information is used by a pattern matching algorithm, which order the logs according to their "severity". The system is validated using 2 weeks of data from a real water treatment facility. The data was captured in the context of the Hermes,

Reference	Source of audit data	Detection method	Validation method	Process-aware detection
Bigham et al. [20]	network	anomaly	testbed	aware
Carcano et al. [28, 29, 59]	network	anomaly	simulation	aware
Cárdenas et al. [31]	network	anomaly	simulation	aware
Cheung et al. [37]	network	anomaly	testbed	unaware
D'Antonio et al. [45]	network	anomaly	none	unaware
Di Santo et al. [49]	network	anomaly	simulation	aware
Düssel et al. [51]	network	anomaly	measurement	unaware
Goldenberg and Wool [62]	network	anomaly	measurement	unaware
Gonzalez and Papa [64]	network	anomaly	testbed	unaware
Hadeli et al. [69]	network	anomaly	testbed	unaware
Hadiosmanovic et al. [70]	host	anomaly	measurement	unaware
Hoeve [76]	network	anomaly	testbed	unaware
Linda et al. [99]	network	anomaly	testbed	unaware
McEvoy and Wolthusen [105]	network	anomaly	simulation	aware
Oman and Phillips [116]	network	anomaly	none	unaware
Premaratne et al. [122]	network	signature	testbed	unaware
Rrushi et al. [125, 126]	network	anomaly	none	aware
Valdes and Cheung [145]	network	anomaly	testbed	unaware
Xiao et al. [151]	network	anomaly	none	aware
Yang et al. [152]	host	anomaly	testbed	unaware

Table 4.2: Overview of surveyed IDS approaches

Castor and Midas projects [4], which also supported the work described in this thesis.

In the work by Yang et al. [152], an anomaly detection mechanism based on host level information such as CPU and I/O utilization is presented. The approach assumes all computation is related to the control activities, therefore an unusual load represents a security threat. A statistical method is used to verify if the current measurements deviates from historical observations. Alarms are raised if the deviation is exceeds a certain threshold. Using a testbed composed of COTS devices simulating a SCADA environment, the authors show that DoS attacks and insider attacks (simulated by running an "unusual process" in the servers) can be detected using the proposed approach.

[4] `https://zeus.tsl.utwente.nl/wiki/hcm/ProjectDescriptions`

Network/Signature based

Premaratne et al. [122] proposes, to the best of our knowledge, the only signature-based approach focused on SCADA environments. By observing traffic generated by real and simulated IEDs, the authors manually derive rules characterizing attack behavior (e.g., ICMP packets larger than 100 bytes indicate a Ping DoS). However, the authors do not present a methodology to generate signatures nor discuss how the method could be extended to other scenarios.

Network/Anomaly based

D'Antonio et al. [45] propose the use of flow-level information (e.g., IPFIX [41]) to detect anomalies using a "context extraction algorithm". However, only the general architecture is described and details about the context extraction algorithm are not given.

Similar methods, based only on TCP/IP header fields, are proposed in [99] and [145]. In [99] features, such as the number of IP addresses and the number of packets, are calculated over a sequence of N packets, which is referred to as a window. A neural network is proposed to learn the normal contents of such window. Results from experiments performed in a testbed achieve 100% detection without a single false positive. However, the used testbed consisted of a single PLC, and only a small window of $N = 20$ packets was used during the tests, so it is unclear whether the solution would scale to a larger SCADA network.

In the approach described in [145], flow-level metrics extracted from network traffic are compared to historical values. If the difference between current and historical values is too large, the flow is marked as anomalous. In addition, alarms are also defined for "new" flows, i.e., flows for which no historical data is known, and "missing" flows, i.e., flows not observed after a certain time. The approach is implemented in a testbed environment.

The approach proposed in [37] is based on a model of valid Modbus interactions. For instance, although Modbus allows for 256 different function codes, only a subset is used in a real deployment. The authors proposed to create models characterizing the allowed Modbus interactions, and raise alarms when disallowed interactions are observed. The authors also propose to create a whitelist of hosts that are allowed to communicate. An offline implementation is tested using data collected in a testbed environment.

Oman and Phillips [116] propose a hybrid of an IDS and a configuration tool.

The IDS internally uses models representing the allowed traffic patterns (e.g., which hosts are allowed to communicate, which commands can be sent, etc.). The authors propose the use of the *telnet* protocol to both test connectivity to devices in the field network and to configure them. The configuration of these devices is periodically retrieved and stored, so it can later be restored in case of operator errors or to recover from attacks. The authors motivate the use of *telnet* because of its widespread adoption in SCADA environments. A proof-of-concept is implemented and deployed in a testbed environment. However, as *telnet* does not provide any security features (all commands, including authentication related, are sent over plain text) the resulting solution is not secure.

Another anomaly detection approach based on allowed connections is proposed in [69]. In this case, the normal behavior is received as input through a configuration file augmented by "implicit information" obtained from a system expert. Besides allowed connections, these files also contain additional information, such as the rate at which specific request commands are issued. The approach consists of parsing such configuration file to generate a communication model and monitor network traffic for violations. A proof of concept is implemented in a testbed environment.

While [37, 116, 69] assume the allowed traffic patterns are known *a priori*, the idea of learning these patterns by passively monitoring traffic is introduced in [64]. However, no specific mechanism for learning is proposed in this work. In [62], an automated learning approach that exploits periodicity of Modbus traffic is proposed. Modbus traffic is modeled as a Deterministic Finite Automaton (DFA) representing a series of requests and replies sent periodically. The authors propose a method to automatically learn the DFA from traffic measurements. The viability of the approach is shown using data captured at a university campus power grid. This approach will be discussed in more detail in Section 6.3.

A number of classical machine learning methods are tested in [51]. The basic approach consists of extracting traffic information using Bro [118], then applying a different combination of feature extraction methods, similarity measures and anomaly detection methods. Despite extensive experiments, using HTTP and RPC traffic, the best combination is scenario dependent and the authors do not provide a methodology to select a combination for deployment. We also note that it is not made clear whether the measurements are performed in a real-world deployment or in a testbed.

All approaches described so far assume that traffic is transmitted unencrypted or that the decryption keys are available to the IDS. Hoeve [76] studies the feasibility of detecting anomalous activity from encrypted traffic in the smart

metering domain. The proposed approach learns normal traffic patters based only on packet sizes and timestamps. The approach is tested using Modbus and IEC 60870-5-104 traffic captured in a laboratory environment.

Process-aware approaches

To the best of our knowledge, Bigham et al. [20] was the first work to suggest the use of process level information in the context of intrusion detection. This work describes two methods to detect anomalies based on network measurements. Both methods process *power flow* data to create a model for the normal behavior. The first method learns and monitors measurement traffic using a customized *n-grams* algorithm capable of processing floating-point data. The second method is based on invariant reduction, which verifies if mathematical relations between different *power flow* measurements remain constant. For instance, the authors observe that some of the measurements (P_1 and P_2) have a linear relationship, i.e., are in the form $P_1 = kP_2 + C$, where k and C are constants learned from measurements. More complex relationships are not discussed in the paper. The methods are evaluated using a "load flow program, real and reactive power flow measurements", which we assume to be collected in a testbed environment.

In a series of works, [28, 59, 29] Carcano et al. propose the idea of detecting *critical states*, defined as "the set of system configurations which might cause system stops, damages etc.". The approach is based on an accurate simulation model and a set of critical states for the physical process controlled by the SCADA network. The IDS consists of identifying commands sent over the network and feed these commands to the simulation model. The system then calculates the distance between the current state of the simulated model and the set of critical states, raising an alarm if the distance falls below a certain threshold. A proof of concept is described in [28], a prototype capable of parsing Modbus and DNP3 traffic is described in [59], and the results of experiments in a testbed simulating a boiling water reactor are presented in [29].

Xiao et al. [151] describe the idea of *workflows*, which are responsible for evaluating all commands issued in the SCADA system. All commands sent to the SCADA system are first routed to the workflow, which contains a simulation model for the physical process where the impact of commands can be evaluated. If the workflow deems the command malicious, it will not be executed.

Di Santo et al. [49] describe an IDS for the power system domain, in which the system is mathematically modeled. The IDS then monitors traffic to be able to compute the current state of the physical process, predict if *contingencies* are

likely to occur and determine their impact on the overall system. The authors acknowledge that the proposed system is computationally intensive and parallel processing is necessary to evaluate the models fast enough. In the simulated power system tested, the model takes several minutes to be evaluated by a cluster of 12 workstations. However, it is not made clear whether this processing time is acceptable in a real system.

Process-aware approaches have also been proposed for nuclear power plants [125] (later tested in a simulated advanced boiling water reactor [126]), for a pasteurization process in McEvoy and Wolthusen [105] and for a chemical process known as Tenesse-Eastman process in [31]. An interesting aspect of [31] is the evaluation of the impact of different realistic attack scenarios and the discussion of responses to these attacks.

Cárdenas et al. [31] also identifies the major challenges for realistic IDS based on process information. Although some models of physical processes are available, for the majority of process control systems the development of such models is difficult and hard to justify economically. Finally, some models might even be impossible to be evaluated in reasonable time, due to the complex nature of many systems and process.

4.5 Summary

SCADA environments are moving from special-purpose embedded devices communicating through proprietary protocols to COTS devices communicating through standard network protocols (such as TCP/IP), as such, SCADA environments start adopting the same devices and protocols as used in traditional IT environments. Despite these similarities, in Section 4.2, we showed that the security requirements for SCADA environments are considerably different. For instance, while confidentiality plays a central role in traditional IT environments, it is of lesser importance in SCADA environments, where availability and integrity have precedence due their impact on safety.

In Section 4.3, we described a number of recent incidents showing that threats to SCADA networks are real. These incidents expose the problem of relying on the air gap for security. The air gap might not actually exist, as in the Maroochy Water breach or might be circumvented, as in the Stuxnet incident. These incidents also demonstrate problems with security through obscurity. The worm infections in the Davis-Besse nuclear power plant and CSX Corp train signaling system show that these networks are exposed to some of the same threats that plague traditional IT networks, and no specialized knowl-

edge is necessary to attack them. Finally, Stuxnet exposes the ineffectiveness of security through obscurity against a resourceful attack.

Attention from industry and research community to SCADA security issues has increased over the last 10 years. In Section 4.4 we review industry and academic efforts in the area. Our survey shows that very little research is supported by empirical data obtained from measurements at real-world SCADA networks.

CHAPTER 5

Exploiting the Stable Connection Matrix

5.1 Introduction

In Chapter 3, we confirmed the assumption that the connection matrix of SCADA networks does not change considerably over time. In that chapter, we defined the connection matrix as the pairs of communicating hosts, which are identified by their IP addresses, and study how it changes over time. Our results show that the connection matrix is stable, i.e., it does not present any change over periods of time longer than a day, and changes tend to be small. In this chapter, we investigate how this characteristic can be exploited to detect potentially malicious activity in SCADA networks.

A straightforward method to detect malicious activity is to define an access control list that determines which entities may access which resources. An entry on such a list represents an entity that is *allowed* to access a specific resource, or conversely, an entity that should be *denied* access to the resource. The first approach is commonly referred to as a *whitelist*, and the latter as a *blacklist* [141].

The *flow whitelist* is a list that describes all legitimate traffic patterns in the network based only on information extracted from packet headers (as opposed to their contents). The main motivation for the use of whitelists is that a large portion of SCADA network traffic is generated by automated processes, like the periodic polling of field devices. Besides that, SCADA networks are closed, with very limited external access, if any. Finally, changes in SCADA systems are rare, that is, hosts and services are not expected to be frequently added to or removed from the network.

In fact, the idea of whitelisting can be commonly found in recommendations for SCADA security. For instance, the Norwegian Oil and Gas Association suggests that "all access requests shall be denied unless explicitly granted" [113].

The American National Institute of Standards and Technology (NIST) recommends to "block all communications with the exception of specifically enabled communications" [137]. However, to the best of our knowledge, the feasibility of flow whitelists was never studied in real-world SCADA environments. It should be noted that, although flow-level whitelists are not commonly used in traditional IP networks because the number of legitimate connections is too large to be manageable, whitelisting has been proposed to some specific domains, such as reducing SPAM [27], avoiding phishing [54], guaranteeing access to important customers during DDoS attacks [153], and preventing certain attacks in VoIP infrastructures [35].

In this chapter, we investigate the use of flow-level whitelists to protect SCADA networks. Given the connection matrix stability, it should be possible to define the allowed behavior ("who can communicate to whom"). We discuss the feasibility of implementing a flow-level whitelisting approach in SCADA environments to assist the network administrator in the task of detecting illegitimate network traffic. The first problem we investigate is how flow whitelists can be created and maintained.

After that, we study its feasibility. To be feasible the whitelist should present two characteristics. First, its size should be manageable. A very large list with millions of entries, as it would occur in traditional IT networks, would make the approach hard to implement and to manage. Second, the whitelist should be stable. If the list is unstable, i.e., it changes frequently, it either requires continuous updating by the network administrator or it results in a large number of false alerts, therefore becoming an unpractical solution. We evaluate our approach using real-world traffic measurements.

In summary, we address the following research questions:

RQ 5.1: *How can we perform flow whitelisting at the network level?*

RQ 5.2: *Is the size of a flow whitelist for a SCADA network manageable?*

RQ 5.3: *What are the sources of instability in SCADA flow whitelists?*

The remainder of this chapter is organized as follows. To address research question 5.1, we first motivate the flow whitelist entry format in Section 5.2, and then describe our flow whitelisting approach in Section 5.3. In order to answer research questions 2 and 3, we apply our approach to real-world traffic traces. The results of this feasibility evaluation is presented in Section 5.4. Section 5.5 discusses four different aspects toward a real-world deployment of our approach:

dynamic port allocation, attack scenarios, automatically blocking traffic, as well as limitations of the proposed learning phase. Finally, in Section 5.6 we present our conclusions.

5.2 Flow Whitelisting

Whitelists (and blacklists) can be defined based on different information. Broadly speaking, when utilizing network data, one could perform Deep Packet Inspection (DPI), i.e., parsing the application data, or use flow-level information only, i.e., not parsing the application data. One of the drawbacks of using DPI is that individual parsers are necessary for each protocol in use in the network (e.g., Modbus, MMS, IEC 60870-5-104 and etc.). In contrast, flow-level information might not be sufficient to detect invalid commands (i.e., application data), such as an HMI that commonly only issues "read" commands, suddenly attempting to re-write a PLC configuration.

As an intrusion detection system, flow whitelisting presents several advantages over deep packet inspection [37] and host level [71] IDS. The most obvious advantages are simplicity and efficiency [21]. In addition, by not depending on the packet payload, flow whitelisting should be able to handle proprietary protocols. Furthermore, it only employs passive network measurements, that is, it does not add, remove or change traffic and it does not require that additional software is installed in the existing hosts. The only necessary modification in the network is the addition of a monitoring host with access to *all* traffic, for instance a host connected to an existing switch or router, in which the measurements and analysis will take place. This also overcomes a common resistance of SCADA operators to make changes in their environment.

A main design decision made in this chapter is not to parse application data, that is, we only decode protocol information up to the transport layer. Therefore, we are limited to information that can be obtained from link, network and transport layers. In our discussion we assume that the protocols are Ethernet in the link, IP in the network and TCP or UDP in the transport layer. The motivation for this choice is that these protocols are extensively used, particularly, representing most of the data in our datasets. Furthermore, our approach should be easily extensible to other protocols, as they should contain information similar to that used in these protocols.

We choose not to consider the Ethernet addresses as they can typically be directly mapped to network addresses. Multiple Ethernet addresses being mapped to the same IP address could indicate a spoofing attempt, however, we

consider this situation out of the scope of this chapter.

When studying the connection matrix stability in Chapter 2, we investigated the changes at the IP level connectivity over time. We observed that the pairs of communicating hosts (servers and clients) do not present many changes over time. Based on this information, we could create a whitelist where each entry is a pair of hosts which is allowed to communicate. Any connection attempt between host pairs not present in the whitelist would raise an alarm.

However, this aggregation level is to coarse for the task of identifying illegitimate connection attempts. The reason for this is that a client might be allowed to access a specific service provided by a server, but no other. For example, an operator workstation might be allowed to gather information directly from a certain PLC using the Modbus protocol, however, remote logins to the same PLC (e.g., Telnet or SSH) are only allowed from an engineering workstation. If we construct our whitelist based solely on host pair information, we would not be able to make this distinction and generate an alarm when a remote login session is set up.

Therefore, in order to obtain a fine grain view on the traffic, we propose to add the service information. A service can be identified by using the protocol field in the IP header and the server side transport port[1]. More specifically, we propose each whitelist entry to contain the following information:

(Server Address, IP Protocol, Server Port, Client Address).

These fields are inferred from packet headers. The server and client addresses are IP addresses, the IP protocol is a field in the IP header and the server port is a field in the TCP or UDP header.

Such 4-tuple is akin to an access control list at the flow level. It describes, for each server, which services are available for each client, where the server and client are identified by their IP address and the service by the pair: *Server Port and IP Protocol*. We refer to this 4-tuple as a flow aggregation key. A flow is defined as the set of packets that have the same key information.

It is important to note that the information regarding which of the hosts represents the server (or client) in a connection is not explicitly available from a packet header, as it contains only source and destination addresses. Consequently, to match a given packet to a flow it is necessary to infer the server (and client) side information. We present an approach to perform this task in Section 5.3.1.

[1]In contrast, the client side port should change in every connection attempt.

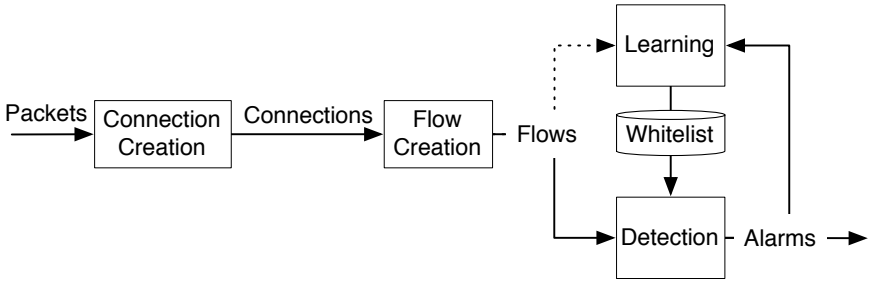


Figure 5.1: Outline of the flow whitelisting approach

Finally, we note that flow whitelists could be defined based on different information. For instance, the network administrator might decided to accept only non-fragmented IP packets (e.g., [155]). Alternatively, one could raise alarms in case a certain TCP option is observed, when it is known it is not supported by the communicating hosts. In this chapter, rather than proposing the definitive whitelist format, we are interested in studying the feasibility of flow-level whitelists, or, more precisely, whether flow whitelists present both manageable size and stability.

5.3 Approach

Our approach to flow whitelisting is outlined in Figure 5.1. First, the traffic in the SCADA network is captured, aggregated in bi-directional *connections*, and finally aggregated to *flows*. These phases are described in Section 5.3.1. In theory, flow whitelists could be (manually) defined by network administrators or even by the SCADA system vendor, however, we assume this information is not known. Accordingly, in Section 5.3.2, we propose a *learning* phase, where the flows observed over a certain period of time are used to create an initial flow whitelist. After the whitelist is created, flows are analyzed in the *detection* phase (see Section 5.3.3). All network traffic matching a whitelist entry is considered legitimate. Every flow not matching an entry generates an alarm, that, in case of a false-positive, can be added by the network manager to the whitelist.

5.3.1 Connection and flow creation

Since our approach relies on information from the IP packet header, packet headers have to be captured in the (sub)networks to be monitored by the whitelist. The *connection creation* consists in aggregating the captured packets to *connections*. We define a *connection* as all packets with the same source and destination IP addresses, source and destination transport port numbers and IP protocol field, regardless of direction. For instance, consider a UDP connection between endpoint A (IP address IP_1 at transport port P_1) and endpoint B (IP address IP_2 at transport port P_2). All UDP packets sent from A to B and from B to A are part of the same *connection*.

The end of a connection is determined either using the TCP state machine or an inactivity timeout of $300s$. In our experiments (see Section 5.4), we perform this task with the open source tool `argus`[2].

The *flow creation* phase identifies the client and server sides of the connections and further aggregates the connections according to our 4-tuple key defined in Section 5.3. The pseudo-code for this procedure is outlined in Algorithm 2. It consists of sequentially applying four rules to identify the server side:

- Rule 1 applies to all TCP connections for which we observe the 3-way handshake. The server is set to be the host which received the SYN packet or sent the SYN/ACK packet. The use of TCP handshake to infer the server and client sides of a connection is a common practice, used for instance in tools such as `BRO` [118], `TCP-REDUCE`[3] and `argus`.

- Rule 2 is applied if a well-known port (below 1024) is observed: the host using such port is set as the server. The use of well-known ports is also a common practice, used for instance in [132].

- Rule 3 is a heuristic. If the same protocol and port is re-used by a host in multiple connections, this host is set as the server and we use this protocol/port combination to identify the service. For instance, if a host A establishes two (or more) UDP connections to host B at port P, host B is set as the server. Host B would also be set as server, if the connections are originated from distinct hosts but using the same port.

  We argue that Rule 3 is useful to infer the server side of high-numbered port UDP (above 1024) connections and TCP connections for which the

[2] `http://www.qosient.com/argus/`
[3] `http://ita.ee.lbl.gov/html/contrib/tcp-reduce.html`

3-way handshake was not observed. These connections are particularly common in the beginning of the measurement. We rely on the fact that client ports normally vary with each connection, and are less likely to be repeated. This rule makes it necessary to keep information about every connection not classified by rules 1 or 2 in memory until a second connection with a repeated host address, protocol and port is observed, potentially indefinitely delaying the analysis. For an online implementation, we would recommend the use of a timeout, after which the connection should be classified according to Rule 4. In this work, we have implemented an offline analysis with an infinite timeout, i.e., Rule 4 is only triggered once we process every connection in a dataset.

- Finally, for flows which do not match any of the previous rules, Rule 4 sets the server to be the destination of the first packet observed in the connection.

There are two exceptions to our rules. First, in the case of Active FTP, the data connection is initiated by the server, so we invert our decision. That is, if any of the hosts uses port 20 (ftp-data) as source port, we set this host to be the server. Second, in some protocols such as NTP, both hosts use the same (well-known) port, making rule 2 inapplicable. In this cases, we use either rule 1 or 4 for classification.

Using Netflow/IPFIX records

The *flow creation* described above can be easily modified to deal with *flow-level* records, such as generated by Netflow [40] or IPFIX [41] exporters, instead of connections generated in the *connection creation* phase. In this context, a *flow* is defined as a set of IP packets passing an observation point in the network during a certain time interval, that share a certain set of common properties (e.g., fields in the packet header), termed *flow key*. *Flow records* contain measured information about a specific *flow* (e.g., total number of packets) [41].

Typically, a distinction is made between unidirectional, i.e., one record for each connection direction (e.g., traffic from client to server and from server to client), or bidirectional flows, i.e., a single record for both directions. Bidirectional exporter implementations might identify the server side of a connection, by using the direction field [142]. `argus` is an example of a flow exporter that performs this task. In such cases, the exporter effectively implements *Rule 1* from Algorithm 2. When the export fails to identify the connection direction,

```
Input  : A connection
Output: A flow
srcAddr, srcPort, dstAddr, dstPort, proto = parse(connection);
/* Active FTP exception */
if one host uses port 20 (ftp-data) :
    Host is the server

/* rule 1 */
if 3-way handshake (partially) seen :
    SYN destination or to SYN/ACK source is the server

/* rule 2 */
if only one host uses a well-known port :
    Host is the server;          /* if both well-known use rule 4 */

/* rule 3 */
if one host uses a port seen before :
    Host host is the server;

/* rule 4 */
Destination of first packet in the connection is the server;
```

Algorithm 2: Pseudo-code implementation for determining connection server side from packet headers

e.g., the 3-way handshake is not observed, one can still apply the remaining rules.

In the cases where unidirectional records are exported, *Rule 1* cannot be applied, as flow records do not contain enough information to identify which endpoint initiated the connection according to the 3-way handshake. For these cases, we propose Algorithm 3, a slight variation of the technique described in [132] to reconstruct TCP connections from NetFlow version 5 records. The client side (termed *originator* in [132]), and, as a consequence, the server side and the service port of a connection are determined by this technique.

if *(SYNs from both hosts and*
SYNs in earliest record of both hosts and
hosts' earliest packets differ) **:**
Host with earliest record is the client

if *(SYN only from one host and*
SYN is in connection's earliest record) **:**
Host is the client

if *one host uses port 20 (ftp-data)* **:**
Host is the server

if *only one host uses a well-known port* **:**
Host is the server

if *one host uses a port seen before* **:**
Host is the server

if *start of hosts' first packets differ* **:**
Host with earliest record is the client

Arbitrarily choose server

Algorithm 3: Pseudo-code implementation for client and server-side identification from unidirectional flow records (based on [132])

5.3.2 The learning phase

Ideally, the network administrator knows all services deployed in the network and the clients that commonly access them. Therefore, a flow whitelist could be constructed based on this knowledge. In practice, however, complete information is rarely available, partly due to the involved proprietary protocols.

The goal of the *learning* phase is to automatically create an *initial* whitelist from a given period of traffic, the *learning time*. This whitelist contains the entries for all flows observed during the learning period. We make two assumptions: (i) all flows in the learning period are legitimate, and (ii) most legitimate flows can be observed in the learning period. We argue that the first assumption will usually be valid as anomalous or malicious events are much rarer in SCADA networks than in traditional IT networks. In fact, no attacks were re-

ported during the capture of our datasets. The second assumption is based on the expectation that most of the traffic in SCADA networks is automated, thus flows should be repeated fairly often. We discuss how to set the duration of the learning phase in Section 5.4.3.

We stress that we do not expect to see *all* legitimate flows in the learning phase, but only creating an *initial* whitelist. For example, manual changes in the configuration of PLCs could, depending on the setup, only happen rarely, hence flows related to this activity will probably not be present in the whitelist. Hence, the whitelist can be extended by the network administrator during the detection phase.

5.3.3 The detection phase

The whitelist created by the learning phase is used in the detection phase to identify anomalous flows, e.g., flows without a corresponding entry in the whitelist. If the flow is whitelisted, then nothing happens, otherwise an alarm is raised. In a real-world deployment, an administrator would have either to add the flow that caused the alarm to the whitelist (in case of a false positive) or to block future occurrences (in case of a true positive). Note that, differently from traditional IT networks, where hosts are commonly put in quarantine in case of malicious activities, an automatic blocking is *not* advised for SCADA environments, as blocking legitimate traffic could have severe consequences, such as preventing legitimate commands to be sent during an emergency situation [31, 69]. This topic will be discussed further in Section 5.5.

5.4 Evaluation

In this section, we propose four tests to evaluate the feasibility of flow whitelists in SCADA networks. As discussed in Section 5.3.3, the network administrator needs to make a decision when a connection triggers an alarm, either recognizing the connection attempt as a false-alarm (i.e., a legitimate connection without a corresponding whitelist entry) or blocking future occurrences. In our feasibility evaluation we need to simulate the administrator's intervention. As our goal is to understand the sources of instabilities in our datasets, rather then to detect anomalous flows (remember that no attacks were reported during the capture of our datasets), we do this by always adding the flow which caused the alarm back to the whitelist, and at the same time storing the alarm for post processing. This means that an alarm is never repeated, which allows us to focus our analysis on

the nature of alarms, rather than their absolute number.

Before discussing these tests, we present our datasets, in Section 5.4.1. In the first test, discussed in Section 5.4.2, we discuss whether the size of the whitelist is manageable by comparing the size of the complete whitelist with the number of hosts and communicating pairs in a network. Our discussion regarding the stability of the whitelists is divided in three parts. In Section 5.4.3, we determine the ideal *learning time* to be used in the learning phase of our approach. Then, in Section 5.4.4, we present the classification method used to discuss the nature of the alarms, i.e., the potential sources of instability in a flow whitelist. Finally, in Section 5.4.5, we apply the classification method to provide an overview of the distribution of the number of alarms over the classes for our datasets.

5.4.1 Datasets

We use four packet `tcpdump/libpcap`[4] traces collected at three different SCADA environments: *SCADA1*, *SCADA2-control*, *SCADA2-field* and *SCADA4*. These datasets are discussed in more detail in Chapter 2. For comparison, we use two additional traditional IT networks datasets. One is a publicly available `tcpdump/libpcap` trace captured at an educational organization: "Location 6" (referred to as *IT*) from [11]. The last dataset consists of 15 days of NetFlow[5] records collected at an internal router in a university campus, referred to as *UNI*. An overview of the datasets is presented in Table 5.1, where we show the size of each dataset according to four different metrics: duration and number of packets, bytes and connections.

Table 5.1: Datasets overview

Name	Hosts	Duration	Packets	Bytes	Conn.
SCADA1	45	13 days	591M	96GB	76K
SCADA2-control	14	10 days	26M	4GB	131K
SCADA2-field	31	10 days	67M	24GB	215K
SCADA4	388	86 days	2G	511GB	179M
IT	93	7.5 days	53M	53GB	264K
UNI	22685	15 days	161G	126TB	1G

[4] www.tcpdump.org
[5] www.cisco.com/go/netflow

5.4.2 Whitelist size

The first characteristic we study is whether the whitelist size is manageable. In other words, we verify if the connection matrix is sparse, i.e., the number of acceptable flows should be small in comparison to the number of possible flows.

We test this characteristic by setting the *learning time* to the full duration of each trace and count the number of flows observed. This allows us to estimate the size of a trace's complete whitelist, assuming no attacks are present in the dataset. While no attacks were reported during the capture of our SCADA datasets, malicious activities such as network scans are so common in traditional IT networks that they are most probably present in the *IT* and *UNI* datasets. We attempt to reduce this bias by only considering flows for which traffic is observed in both directions, thus greatly reducing the number of observed flows caused by network scans and other types of network anomalies.

Table 5.2 shows the results. The column *internal hosts* gives the number of observed hosts located inside the monitored networks. In the column *whitelist*, we show the number of entries in the whitelist and in the column *host pairs*, the number of communicating host pairs. In order to make the different traces comparable, we express these metrics both as absolute values and as a ratio to the number of internal hosts (in parenthesis).

For most cases, the whitelist size for the SCADA datasets is in the same order of magnitude as the number of internal hosts, suggesting that flow whitelisting might be feasible in these environments. In comparison, the traditional IT counterparts present a whitelist orders of magnitude larger than the number of internal hosts, illustrating why the approach does not scale in these environments. Due to the excessively large size of the whitelist for the traditional IT datasets, we do not consider them in the tests performed in the remainder of this section.

Another observation is that, for the SCADA datasets, the difference between the host pairs and the whitelist ratios is not very large, meaning that on average we have one or two services per server. This means that a whitelist without the service information, which is less restrictive and, thus, less secure, would *not* greatly reduce the size of the whitelist.

The only exception to the results presented here is the dataset *SCADA2-control*, in which the whitelist size is one order of magnitude larger than the number of communicating host pairs. However, we show in Section 5.4.4 that this difference is mostly caused by a traffic anomaly.

Table 5.2: Whitelist size ratios

Dataset	Internal Hosts	Host Pairs	Whitelist
SCADA1	51	58 (1.1)	81 (1.6)
SCADA2-control	22	40 (1.8)	542 (24.6)
SCADA2-field	14	20 (1.4)	23 (1.6)
SCADA4	388	542 (1.4)	1188 (3.1)
IT	93	23 322 (250.8)	26 759 (287.7)
UNI	22 685	56 425 836 (2487.4)	141 744 206 (6248.4)

5.4.3 Training set size

In the following we study the influence of the learning phase duration on the size of the learned whitelist. Figures 5.2 to 5.5 show the size of the learned whitelist as the percentage of the total number of flows as function of the learning time. For the datasets *SCADA1* and *SCADA2-field*, over 50% of the flows are observed within the first hour of traffic, with a few other additions during the first day. This percentage is much lower for datasets *SCADA2-control* and *SCADA4*, around 10% and 15%, respectively. The *SCADA2-control* dataset shows a significant jump in the list size after around 7 days. The whitelist for the *SCADA4* dataset grows steadily from day 10 to day 40. The reasons for this behavior are explained in the next section.

Despite the relatively high number of flows observed within the first hour, commonly a few connections are still added in the course of the first day of the measurements. Due to this observation, we set the learning time to 1 day in the following experiments. As expected, not all flows will be observed in the learning time, however there would be little improvement in extending this period for a few more days. We also note that selecting any day other than the first would yield similar results, with exception of days with large amount of anomalies, such as day 6 at dataset *SCADA1*.

5.4.4 Nature of alarms

In this section, we present our post-processing analysis of the alarms. Its goal is to determine the sources of instability in the whitelists, that is, the nature of the flows not observed during the learning phase.

During our analysis we identified four main alarm classes:

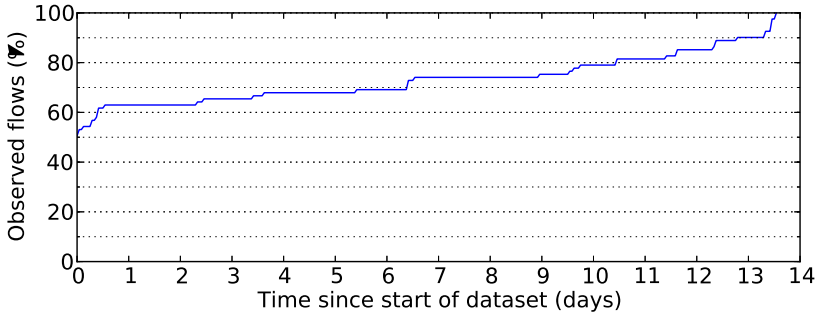


Figure 5.2: Number of learned flows over time: *SCADA1*

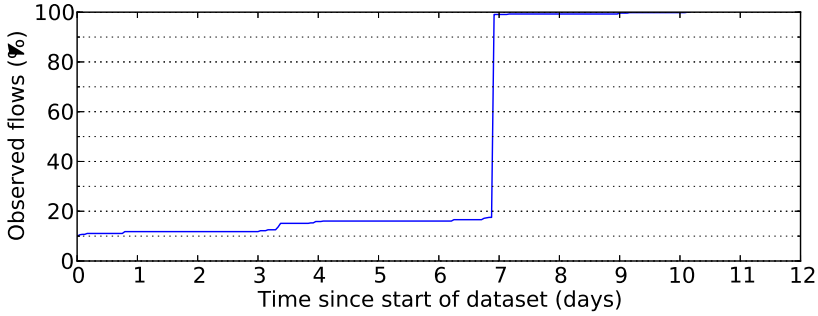


Figure 5.3: Number of learned flows over time: *SCADA2-control*

1. *Dynamic Port Allocation (DPA) Anomaly:* Our flow definition implicitly assumes that every transport port and IP Protocol combination used by a server uniquely identifies a service in the SCADA network. This definition turned out to be particularly problematic with network services that use Dynamic Port Allocation (DPA), such as Microsoft's Active Directory. In this service, high ports (above 1024) are dynamically allocated for Remote Procedure Calls [107].

   We did not attempt to uncover all services using dynamic ports, but we identified anomalies which are most likely triggered by it. The datasets *SCADA2-control* and *SCADA4* present moments in which several sequential TCP connections are made by the same hosts in short time interval,

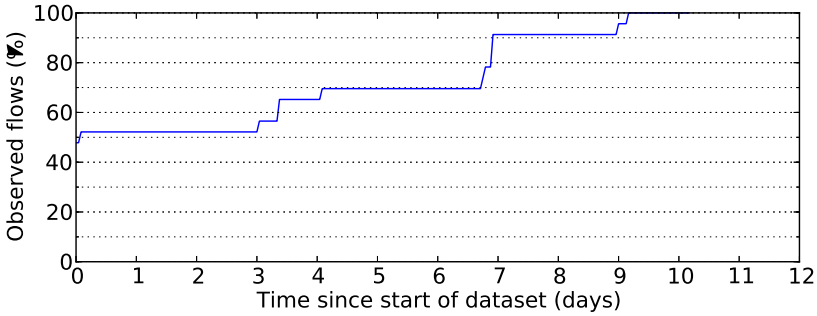


Figure 5.4: Number of learned flows over time: *SCADA2-field*

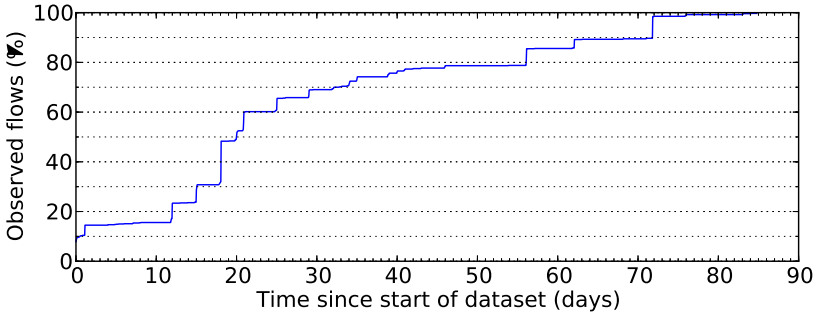


Figure 5.5: Number of learned flows over time: *SCADA4*

with transport port numbers monotonically increasing on both client and server side. Table 5.3 shows an excerpt of such moment.

Table 5.3: DPA anomaly example

StartTime	Proto	Sport	Dport	Pkts	State
09:26:50.944328	tcp	3714	1178	16	FIN
09:26:50.960961	tcp	3715	1178	2	RST
09:26:50.976884	tcp	3716	1180	16	FIN
09:26:50.990740	tcp	3717	1180	2	RST
09:26:51.007886	tcp	3718	1183	16	FIN
09:26:51.021606	tcp	3719	1183	2	RST

2. *Manual Activity:* This class consists of human triggered flows. All flows which used the following services, identified by IP protocol and transport port number, fall in this class: *telnet* (TCP-22), *ssh* (TCP-23), *http* (TCP-80), *https* (TCP-443), *shell* (TCP-514), *kshell* (TCP-544), *rdp* (TCP-3389), *vnc* (TCP-5800 and TCP-5900) and *x11* (TCP-6000 to TCP-6007).

   In addition, for datasets *SCADA1* and *SCADA2*, we have a list of operator workstations. If the client side of a flow is on this list, such flow is also classified as manual.

3. *New Host:* This class contains all flows for which at least one host (either server or client) did not communicate during the training period and, obviously, can not be present on the whitelist.

4. *Other:* A *catchall* class for all flows that do not fit any of the other classes.

We map each flow to a single class, and the membership of a class is tested in the same order presented here. For instance, consider an alarm for a flow where the client is not present in the whitelist and where the service is *ssh*. In this case, the flow is classified as *manual activity*, as this class has precedence over the *new host* class.

When analyzing the *SCADA4* dataset, we observed two events that deserve to be studied separately. In SCADA networks, it is very common for most functions in the network to be replicated, including duplicating servers, in order to increase reliability.

The first event consists of a single *redundant* host taking over tasks of one of the main servers in the network, the SCADA server responsible for polling the field devices. Just before the change occurs, we observe *telnet* traffic to some of the PLCs, issuing a reboot command. We do not observe telnet to all PLCs, however, as all changes occur in a relatively small time interval, we presume they are related. Besides the flows involving the PLCs several other long-lived flows present the same behavior, for example, *ssh*, *x11* and some high port services. We discussed this behavior with the operators responsible for this network and they informed us that changes like this one are routinely performed in order to verify if the redundant hosts work properly. In our analysis, we adopt the following procedure to identify flows belonging to this event. If one of the hosts in the flow is the SCADA server, we look for another flow with a similar key, where only the SCADA server address is changed to its backup or vice-versa.

The second event consists of the *relocation* of many hosts in the network, mostly PLCs. At times, a continuous range of IP addresses have their address

Table 5.4: Alarm breakdown

Dataset	DPA Anomaly	Manual	New Host	Other
SCADA1	0	14 (47%)	15 (50%)	1 (3%)
SCADA2-control	437 (91%)	16 (3%)	6 (1%)	19 (4%)
SCADA2-field	0	5 (45%)	6 (55%)	0
SCADA4	358 (35%)	269 (26%)	274 (26%)	136 (13%)
redundant	0	13 (5%)	16 (6%)	75 (56%)
relocation	0	14 (5%)	148 (54%)	1 (0%)
remaining	358 (100%)	242 (90%)	110 (40%)	60 (44%)

changed to (logically) separated subnetworks. For example, all hosts in the IP address range X.Y.Z.61 to X.Y.Z.71 have their addresses changed to the range X.Y.A.61 to X.Y.A.71. We observe *telnet* commands being issued to perform the address change, but not to all hosts. Again, the small time interval between the changes, suggests that they are related. The operators confirmed the behavior. A large subnetwork was split in several smaller ones. After the change, the logical address better represent the geographical location of the hosts.

We identify flows belonging to this this event simply by verifying if either host in the flow (client or server) is part of one of the newly created networks. Its important to note that we do not classify the *telnet* access to these hosts leading to these events as part of them. *telnet* connections are always classified as *manual activity*.

5.4.5 Frequency of alarms

We apply our alarm classification method to all SCADA datasets to provide an overview of how frequent each class of alarm is. As discussed in Section 5.4.3, the *learning time* is set to be the first day of the dataset. The results of the classification are presented in Table 5.4. This table presents the number of alarms and approximate percentages for each class. We present additional results for the dataset *SCADA4*, by breaking down the alarms using the *redundant* and *relocation* events identified in the previous section. To illustrate how these alarms are distributed over time, Figures 5.6 to 5.9 show time series for the number of alarms observed daily, also divided by class.

In the *SCADA1* dataset, the *new host* alarms consist of a few short *snmp* connections, likely due to testing; one *ntp* connection that seems to repeat once a week; and one real anomaly: several single packet TCP connection attempts at port 1010. The single *other* flow seems to be caused by DPA; a few moments

before it starts, a flow involving the same hosts, but different server port, ends. Finally, a few *http(s)* and *x11* connections and a connection originated from a operator's workstation compose the *manual activity* alarm class. A time series representing the number of alarms per class over time is shown in Figure 5.6.

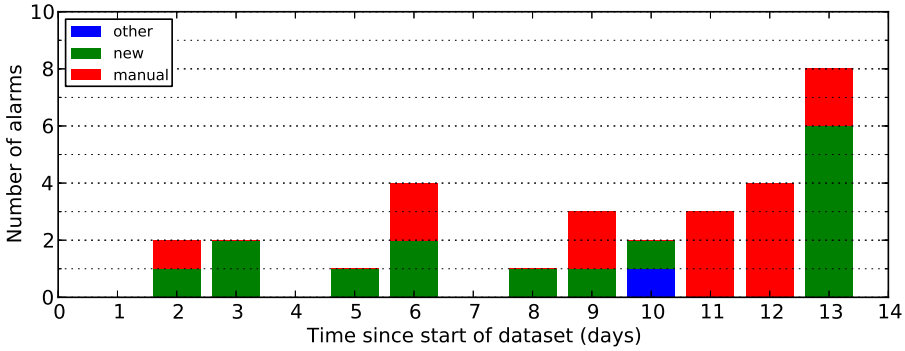


Figure 5.6: Nature of alarms: *SCADA1*

The leading cause of alarms in the *SCADA2-control* dataset is a DPA anomaly, being responsible for around 91% of the alarms. This anomaly is responsible for the majority of the flows which compose the jump present in Figure 5.3. In fact, if we remove the flows generated by this anomaly, over 60% of the flows would be present in the whitelist (i.e., be observed in the training period), much like in the other *water* datasets. In addition, the ratio between the size of the whitelist and the number of internal hosts would be considerably smaller, 4.7 instead of 24.6, i.e., in the same order of magnitude as the other SCADA datasets.

In the *SCADA2-control* and *SCADA2-field* datasets, most *new* and *other* alarms involve a server which, according to the network administrator, relates to user authentication and thus, probably are generated due to manual activity. An unexpected behavior is that some connections are made from this authentication server, which is in the control network, directly to PLCs, which are in the field network. According to the network administrator, this type of connection should not be allowed. All connections from the control network to the field should pass through the SCADA server. The remaining alarms involve hosts foreign to the control and field networks where the data collection was performed. It is not clear if these connections should be allowed. Figures 5.7 and 5.8 show the

time series for the number of alarms over time for datasets *SCADA2-control* and *SCADA2-field*, respectively. Note that the former graph presents a discontinuity in the *y-axis*, as the number of alarms caused by the DPA anomaly is extremely high in comparison to a typical day.

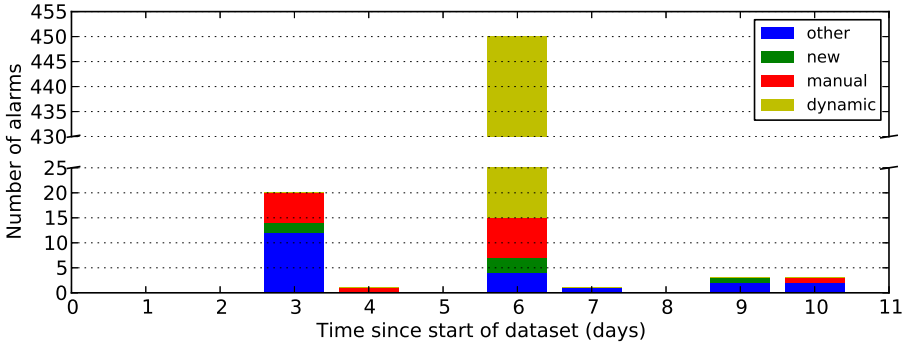


Figure 5.7: Nature of alarms: *SCADA2-control*

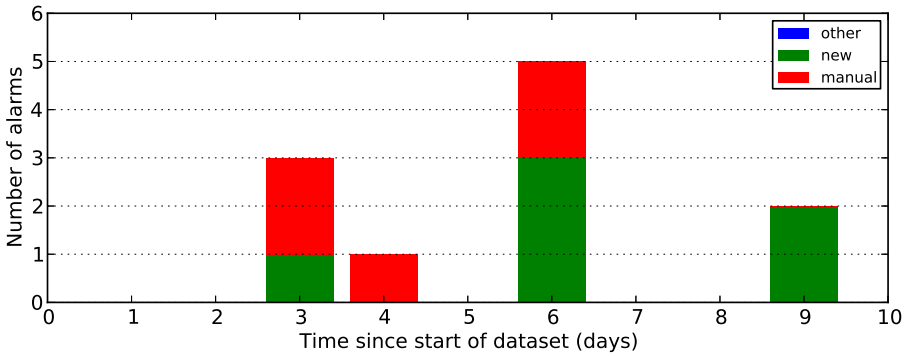


Figure 5.8: Nature of alarms: *SCADA2-field*

In comparison to the other datasets, the *SCADA4* dataset contains a considerably larger number of alarms: 1037 of the flows are not observed in the training period. Like *SCADA2-control*, the largest class is DPA anomalies, accounting

for 35% of the total. The redundant and relocation events are responsible for over half of the *other* and *new* alarms, respectively.

In Figure 5.9, we consider the *redundant* and *relocation* examples as two additional classes. They contain all alarms pertaining to each of these events, regardless of class. DPA connection bursts happen at 4 distinct times, accounting for the largest peaks. The *redundant* event happens at day 15, and a portion of the *manual activity* alarms present at the same day represent the *telnet* connections used to reboot the PLCs. Interestingly, a larger peak classified as *redundant* appears before, at day 11. All flows in this peak represent single packet connections, sent by the *redundant* host to several PLCs. We speculate that this was a test or was caused by a configuration error.

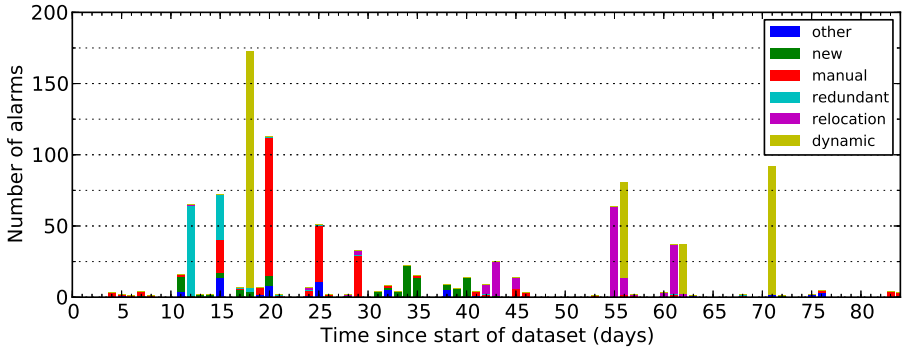


Figure 5.9: Nature of alarms: *SCADA4*

Large peaks of *manual activity* happen at days 20, 25 and 29. At each of these days a large portion of the address space is sequentially accessed via *telnet*, probably for maintenance reasons. For instance, some connections are configuring hosts for the first time, which appear later as *new*, around days 35 and 40. More importantly, the majority of the connections causing these alarms involve hosts in the subnetwork that is later split, suggesting a relation with the *relocation* event. Most of the hosts are relocated around day 55 and 61. Note that no peak of manual activity happens around theses days. The *telnet* connection used to change the hosts' addresses were accounted for in the previous peaks of manual activity (alarms for the same flow are not repeated).

The alarms discussed so far account for the majority of alarms in the dataset. The remaining alarms consist mostly of some manual *ssh, x11* and *http* connec-

tions; a few *samba* related ports (e.g., TCP/UDP 137-139 and 445); and several high port flows, which might be caused by DPA.

5.5 Discussion

In the following we discuss practical issues network administrators will face when implementing flow whitelists in real-world environments: dynamic port allocation, real-world attack scenarios, automatically blocking traffic, and the limitations of the learning phase.

5.5.1 Dynamic port allocation (DPA)

By far, the largest class of alarms identified in our analysis is due to DPA, and we only identified bursts in the change of ports. In practice, there are more services using DPA in our datasets. For a flow-level whitelisting approach to work with this type of service it is necessary to whitelist the whole range of transport ports that might be used by the service. This is not an ideal solution, as it makes the whitelist more permissive.

One of the main advantages that security experts have in protecting SCADA environments is that traffic patterns are rather predictable, when comparing to traditional IT environments. DPA reduces this predictability. We argue that SCADA systems should be designed without making use of DPA or, at least, DPA should be restricted to a non-critical segment of the network.

5.5.2 Dealing with real-world attack scenarios

Since our datasets contained no attack data, we could not test the efficacy of whitelisting against realistic attack scenarios. Based on a list of real-world attack types presented in a previous work [13], we motivate below how these attacks can be detected by our approach.

The first type is formed by *information gathering attacks*, such as network scans. These are normally performed by injecting several requests into the network, with the objective of discovering which services and/or hosts are available. At the flow level, these attacks resemble the DPA anomalies identified before, and as such, should be easily identified by our approach, as non-whitelisted connections are likely to be made. The other three types of attacks are: *denial of service attacks* which prevent a legitimate user to access a service or reduce its performance, *network attacks* used to manipulate the network protocols, and

buffer overflow attacks which attempt to gain control over a process or crash it by overflowing its buffer. These would only be observed if attempted from a host which is not allowed to access a given server/service or if attempted to a server/service not existent in the network.

In general, an attack will only remain undetected if either the whitelist is incorrectly constructed (i.e., it contains entries representing illegitimate traffic), or if the attack misuses whitelisted traffic (e.g., an operator machine that is normally used to access a PLC sets an invalid parameter). In the latter case, the connection itself is legitimate, but its contents are not. Note that a malicious host masqueraded as such HMI, by spoofing its IP address, could perform the same attack. Interested readers are referred to [1] for a discussion of mechanisms that can protect against such attacks.

5.5.3 Blocking or flagging

In a traditional IT environment it is a common practice to take a host offline in case it is suspected to be under attack [156, 130]. This is done to limit the impact of the attack, and prevent a possible spread. Taking a SCADA server offline might have dire consequences, as critical infrastructures might depend on it.

The same reasoning can be applied to blocking traffic, the cost of false positives might be too large. Whitelists, as any other system can suffer from configuration problems. In our analysis, we observed a number of alarms due to rare activities, such as manual access to PLCs and hosts switching to backup servers (or being accessed by backup clients), which might be overlooked while building a whitelist. We recommend that, when whitelists are first deployed in a real world scenario, flows that are not present in the whitelist should only be flagged (raise an alarm). The network administrator should then decide if it is an real anomaly or if it was a flow overlooked during configuration. Only after administrators are confident that the configuration mistakes are solved, they should consider using the whitelist to block traffic.

5.5.4 Limitation of the learning phase

Many of the observed alarms are the effect of a limitation of the technique used for learning the initial whitelist, for example, the ones generated by manual activities. These are connections which do not happen regularly, and it would be impractical to extend the *learning time* in order to include them. The larger

the *learning time*, the larger is the chance of including an anomalous flow to the whitelist.

In addition, some alarms were caused by the presence of new hosts, not observed in the learning phase. Although changes in the topology are not common, they should be taken into consideration when deploying our approach. Every change in the network incurs an extra task of updating the whitelist accordingly, either manually or by triggering a new learning phase. Note, however, that this problem is not exclusive to our approach. Most, if not all, anomaly-based intrusion detection systems would require a similar update after a change in the network, as the "normal" behavior has changed.

The limitation of the learning phase shows that network administrator's (and/or SCADA vendor's) knowledge is necessary to build a complete flow whitelist. However, relying only on this knowledge can also be dangerous, as mistakes are likely to happen. For instance, the addition of flows representing backup server connections or infrequent *ssh* connections might be overlooked. Presenting a list of flows learned from network measurements as proposed in the *learning* phase, would help administrators in identifying *all* acceptable flows.

5.6 Conclusions

An important contribution of this chapter, is the use real-world measurements to study the feasibility of flow whitelists to detect potentially malicious connections in SCADA networks.

Our first research question focused on *how can we perform flow whitelisting at the network level*. We proposed the use of a "natural" flow aggregation key that consists of the server and client IP address and a service, identified by the IP protocol field and transport port number. The whitelist then resembles an access control list at the network level. An entry on the list allows a certain client to access a specific service running in a given server. In addition, we proposed an approach to aid operators in creating an initial whitelist from a given period of traffic. The goal of this *learning* phase is to aid operators to identify *all* legitimate flows.

In RQ 5.2, we investigate whether flow whitelist sizes are manageable in SCADA networks. We showed that differently from traditional IT networks, where the number of legitimate connections is too large for the flow whitelisting approach to be feasible, the connection matrix in SCADA networks is rather sparse. After discarding the DPA anomalies, the size of the whitelists for the analyzed datasets is manageable, considering the number of internal hosts.

In RQ 5.3, we investigated the sources of instability in SCADA flow whitelists. Undoubtedly, services using dynamic port allocation were the main source of alarms in our analysis. These alarms can be eliminated by adding to the whitelist the complete range of ports that could be allocated by these services. However, we argue that a better solution would be removing them altogether from these networks. These services reduce predictability of traffic patterns, which can potentially be exploited to identify intrusion attempts. Most of the remaining alarms are caused by a limitation the approach used to construct whitelists, which, in real-world implementations, would be overcome by using the knowledge of network administrators and system vendors when creating them.

In summary, we have shown that our approach is a practical solution to increase the security of SCADA networks. However, there is still room for improvement. For instance, the flow definition could be extended using time intervals. Remote sessions to PLCs could be further restricted, by allowing these only during business hours. In addition, whitelisted flows could be monitored based on the contents of their packets. An operator workstation might be only allowed to issue *read* commands to PLCs and not *write* commands. In fact, the approach we propose to detect anomalies in the periodic behavior in Chapter 6 implicitly defines a whitelist for the type of queries each client is allowed to issue.

CHAPTER 6

Exploiting the Traffic Periodicity

SCADA networks are deployed to monitor and control devices in a field network. To accomplish this goal, data is continuously retrieved from these devices, so that a real time view of the infrastructure's processes can be established. Typically, data is retrieved through an automated polling process, in which requests are sent by the SCADA server(s) every predetermined interval, triggering responses from the field devices. Moreover, manual interventions, such as sending commands to field devices, are rare. In Chapter 3, we have confirmed that SCADA traffic exhibits a strong periodic pattern. In particular, all PLCs in our datasets generate traffic in a periodic fashion.

In Chapter 5, we have shown that it is possible to exploit the connection matrix stability to detect illegitimate connections in a SCADA environment. However, the methods described in that chapter do not protect against attacks that send malicious commands over legitimate connections. In this chapter, we exploit the periodicity of SCADA traffic to also protect against such attacks. The idea is to complement the flow whitelisting approach by monitoring the contents of whitelisted connections, and, as a consequence, detecting attacks such as the one described above.

The main contribution of this chapter is *PeriodAnalyzer*, an approach that is able to learn periodic patterns in SCADA traffic. The learned model can be used to protect SCADA protocol traffic against data injection and DoS attacks. Although we focus on two protocols present in our datasets, Modbus [109] and MMS [89], the principles presented here are applicable to other protocols as well.

The remainder of this chapter is organized as follows. First, we present the attack scenario and research questions addressed in this chapter in Section 6.1. In Section 6.2, we discuss the communication model used to describe the periodic traffic patterns and present the SCADA protocols considered in our analysis. In section 6.3, we review literature with the objective of inves-

tigating whether existing approaches are applicable to learning periodic traffic patterns. In Section 6.4, we discuss *PeriodAnalyzer*, a novel approach to learn and detect changes in periodic traffic. Next, in Section 6.5, we evaluate the proposed approach using real-world traffic measurements. In Section 6.6, we discuss practical aspects of *PeriodAnalyzer*: how it can be used as real-time IDS, how real-world attacks could be detected and how to optimize it. Finally, in Section 6.7 we present our conclusions and discuss future work.

6.1 Attack Scenario and Research Questions

Consider the following scenario. An attacker gains control over (or manages to impersonate) the SCADA server that is responsible for polling data from a certain PLC in the field. The goal of the attacker is to send malicious commands to the PLC over a Modbus connection. For instance, consider the *diagnostic register reset* attack described in [77]. This attack consists of sending a message with a specific function code, causing the target to clear all counters and its diagnostic register, potentially causing the target to misbehave. When observing this attack attempt from a flow perspective, all that is observed is a client (i.e., the SCADA server) establishing a connection to a server (i.e., the PLC) over Modbus. The flow-whitelisting approach discussed in Chapter 5 is not able to detect this attack, since this connection has the same "signature" as a whitelisted connection.

Leveraging this observation, we focus our attention to the connections used to retrieve data from field devices. Based on the assumption that these periodic request-responses form the majority of the data exchanged between SCADA server and field devices, our goal is to learn the periodic patterns generated by these connections, and then detect changes in the periodic behavior, which represent potential security threats. To achieve our goals, we tackle the following research questions:

> **RQ 6.1:** *How can periodic traffic patterns be learned?*

We review the existing scientific literature for methods that can be used to learn periodic patterns in time series. We show that they are not directly applicable to our problem, and therefore, a new approach is required.

> **RQ 6.2:** *What fraction of the SCADA protocol traffic displays periodic patterns?*

Despite the fact that connections used to retrieve data from field devices exhibit periodic patterns, it would be naive to expect that all traffic in these connections is perfectly periodic. For instance, an operator might manually retrieve data in a non-periodic fashion, causing new connections to be established. The objective of this research question is to establish if a considerable fraction of connections can be accurately modeled as periodic request-response exchanges, and, as a consequence, verify if our approach is viable.

RQ 6.3: *How can changes in the periodic pattern be detected?*

Once we establish a method to learn periodic patterns and determine which portion of traffic can be modeled, we investigate how changes in the periodic behavior can be detected. We note that some changes might not be relevant from a security perspective. For instance, packet loss can cause retransmissions, thus breaking periodicity, however, packet loss and retransmissions are not a security concern. Therefore, in addition to detecting changes in the periodic behavior, we need to define which changes are relevant from a security perspective.

6.2 Communication Model

In this section, we describe a generic SCADA protocol communication model used to motivate the requirements for an algorithm that can be used to learn the periodic patterns generated by SCADA protocols. We also shortly describe Modbus and MMS SCADA protocols, focusing on the aspects that are important for our analysis.

6.2.1 SCADA protocol communication model

Arguably, the most important function of a SCADA system is to give operators a real-time view of a remote physical process. This task is typically done by periodically polling the PLCs in the field. Each polling instance basically consists of a series of requests for different services performed by a PLC. Example of such services are read and write commands for values monitored by these PLCs, for instance, the pressure in pipes, the level of tanks, the pH of a solution and the status of a valve (e.g., open or closed).

Our starting assumption is that the vast majority of the packets sent to and from the PLCs will consist of a series of *requests* sent at regular intervals and their respective *responses*. When observing the connections involving the PLCs, such periodic patterns should be clearly observable. Note that we do

not expect that *all* network connections will necessarily show periodic patterns. Some protocols define more complex services such as file operations, starting or stopping programs and defining new data types, which are likely to be manually triggered by operators, thus generated in a non-periodic fashion.

In order to motivate the requirements for an algorithm that can learn the periodic behavior generated by SCADA protocols, we consider how an application that generates this behavior can be implemented. When implementing the application used to retrieve data from PLCs, a developer can make a number of different choices, e.g., opening a connection each time a requests is sent or sending all requests over a long lived connection, and these choices dictate how periodicity is generated.

The application defines one or more cycles. Each cycle consists of all requests that are sent with the same period. For instance, consider that requests R_1, R_2, R_3 and R_4 are generated respectively at every 1 s, 1 s, 1 s and 2 s. In the application, R_1, R_2 and R_3 form cycle C_1, which has a 1 s period, and R_4 forms cycle C_2, which has a 2 s period.

Let us now consider now how these the messages that form such cycles are sent over the network. The application can either establish a new connection each time a new cycle iteration starts, or open a single connection and use it for all iterations. In other words, when monitoring the traffic sent by this application, one can either observe a set of "short-lived" connections being made at periodic intervals, or a single "long-lived" connection in which requests are sent periodically. It is also possible that the same connection is used to send requests pertaining to multiple cycles, e.g., C_1 and C_2 being sent over the same connection.

The order at which requests are sent in a cycle iteration is not necessarily fixed. For example, consider again C_1. In the first iteration the application might generate requests in the following order: $[R_1, R_2, R_3]$. However, in the second iteration it might generate $[R_3, R_1, R_2]$. This lack of order can be caused, for instance, by a multithread implementation. In the most extreme case, the program defines one thread for each request. In each iteration, multiple threads will then compete for the open connection, and whichever thread gains access will send its requests first.

Regardless of how the application is implemented, small variations in the measured period of a cycle should be expected. Delays and jitter can be either introduced by the network [154] or by the server generating the requests, for instance the application can be preempted if the load on the server is high. Besides delays, packets can be lost, either by the network (e.g., due to electromagnetic interference) or by the measurement equipment.

In summary, the algorithm should be able to deal with the following characteristics:

- **Multiple cycles**: sets of requests can be sent at different intervals.
- **Periodicity at different aggregation levels**: periodic patterns can be observed as periodically made "short-lived" connections or "long-lived" connections with requests sent periodically.
- **Request reordering**: multi-threading might cause requests within a cycle to be reordered.
- **Timing variations**: variations in the measured period can be caused by the network and application, and scheduler implementation.

6.2.2 Modbus and MMS

Modbus is a simple master-slave protocol originally designed to use serial lines. The Modbus over TCP (or Modbus/TCP) standard [109] defines how Modbus messages can be used on top of the TCP/IP stack. This is the version considered in our work, and in this work we refer to it simply as Modbus.

In Modbus, the master always starts the communication by issuing requests to slaves for read or write operations. Each request is processed and replied independently by the slave, that is, Modbus has no concept of a session.

MMS [89] is a more complex protocol than Modbus, offering several capabilities such as file operations, starting and stopping programs, and defining new data types. In addition to these more advanced operations, MMS also defines application-level sessions. Finally, MMS is not a master-slave protocol: message exchanges can be initiated by either of the communicating hosts.

The MMS standard defines fourteen PDU types. Here we focus on the seven PDU types that we observed in periodic traffic exchanges: `confirmed request`, `confirmed response`, `initiate request`, `initiate response`, `unconfirmed request`, `conclude request` and `conclude response`. The remaining PDU types, such as, `cancel request` and `error`, should only be used in case of abnormal situations, and should not be exchanged periodically.

A typical MMS message exchange is shown in Figure 6.1. Arrows from the client to server represent request messages, while arrows in the opposite direction represent response messages. After a TCP connection is established (not shown in the figure), the `initiate` type messages are exchanged to establish an MMS session. After the MMS session is established, `confirmed` and `unconfirmed`

service messages are exchanged. Confirmed service implies a request-response exchange, while unconfirmed service does not trigger a response (e.g., reporting an alarm condition). Once all requests are issued, the session is torn-down with the `conclude` exchange.

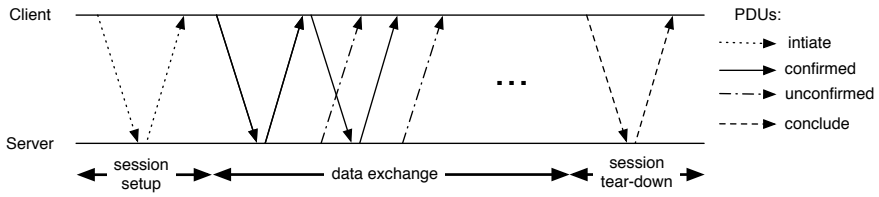


Figure 6.1: Typical MMS message exchange

In case the application sets up a new connection for each cycle iteration (see Section 6.2.1), all messages are exchanged periodically including the initiate and conclude. In contrast, in the case of a single connection for all iterations, the initiate and conclude exchange should only happen once, possibly outside the interval in which data is being captured.

The interested reader is referred to Appendix A for more detailed information regarding Modbus and MMS protocols.

6.3 Related Work

In this section, we discuss why the approaches described in the literature cannot be directly applied to our particular problem. We limit our survey to approaches that are able to learn periodic patterns automatically.

6.3.1 Spectral analysis

Spectral analysis has been used to uncover periodicity in network traffic (e.g., [4, 24, 66, 67]). In this section we describe the lessons learned from a previous work [13], in which we investigated the idea of detecting anomalies in the periodic behavior of SCADA traffic based on well-known mathematical tools such as the DFT and the Autocorrelation Function (ACF).

In the approach proposed in [13], the first step consists of generating a time series representing the packets sent by a host, aggregated by transport port.

The time series is then taken as the input signal for a spectral analysis, which has as objective detecting the dominant frequencies at which packets are sent. Subsequently, in the anomaly detection step, we track changes, such as new cycles, missed iterations and increased noise level, which indicates increased non-periodic activity.

In [13], we propose the use of *AUTOPERIOD* approach [148] to learn the periodic behavior. *AUTOPERIOD* automates periodicity detection using the periodogram and ACF of the input signal. The periodogram is used to extract *periodicity hints*, i.e., periods for which the power component of a certain frequency is above a certain threshold. The ACF is then used to either discard or confirm these hints, and also provide a more precise estimate of the period.

Alternatively, the periodic patterns could be learned using the method proposed by Argon et al. [4]. This method iteratively partitions the peaks observed in the ACF, and returns the set of $(\hat{P}, \xi)$, where $\hat{P}$ is the inferred period and ξ is a confidence value that quantifies the probability that the signal is indeed periodic with the inferred period. Besides the input signal, the method takes three input parameters: *max_slices*, the maximum number of periods the algorithm will attempt to find; *min_peaks*, the minimum period of peaks in a slice for testing for periodicity; and *min_prob*, a threshold for peak identification.

The next step in the approach described in [13] is to study changes in the learned periodic behavior. For this task, we propose the use of the discrete-time Short-Term Fourier Transform (STFT), i.e., applying a sliding window to the input signal and performing a FFT on each window. A spectrogram can be created by plotting time on the *x-axis*, frequency on the *y-axis* and the squared magnitude of the FFT (i.e., the "power") is encoded using a gray or color scale.

Such spectrogram provides a visualization of how the energy content of the different frequency bands vary over time, making it possible to observe: (1) changes in the set of high energy frequency bands, indicating changes in periodic burst intervals; (2) changes in the amount of energy in the same set, indicating changes in the periodic burst size; and (3) increases in the amount of noise, indicating increased non-periodic traffic.

The use of a sliding window introduces two parameters: window size and step. The window size determines the number of samples within a window. The choice of this parameter is a trade-off between frequency and time localization. In an FFT, the number of samples given as input is the same as the number of the discernible frequency bands in the output. Also, the spectrogram has a single data point on the time axis per FFT. Therefore, the use of large windows results in good frequency localization (high number of discernible bands), but bad time localization (small number of data points in the time domain).

The step size parameter determines the number of samples the window moves per FFT. By using small steps one can cope with large windows, as it increases the number of data points in the time domain. Note however, that small steps also mean that the time series for two consecutive FFTs are very similar, as they will have a large overlap.

To understand how the spectrogram can be used to track changes in periodic behavior, consider the following polling scenario. A client runs a process that periodically requests some data from a server. A time series is constructed by recording the number of packets flowing from server to client. The spectrogram for this time series is shown in Figure 6.2.

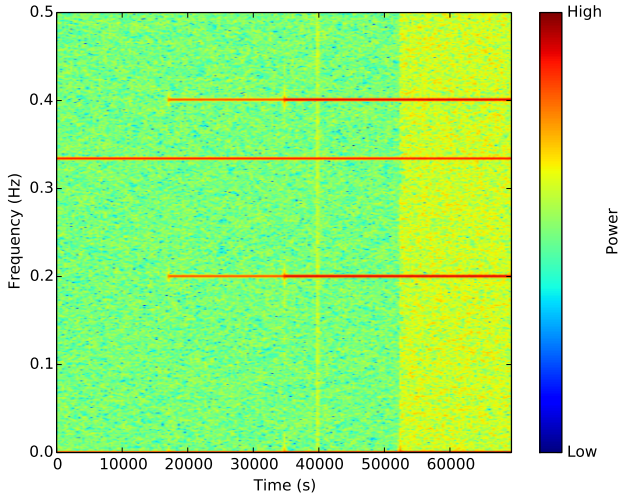
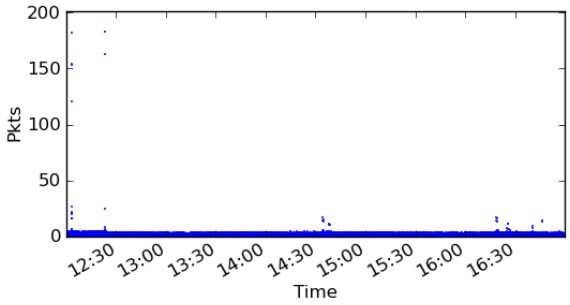
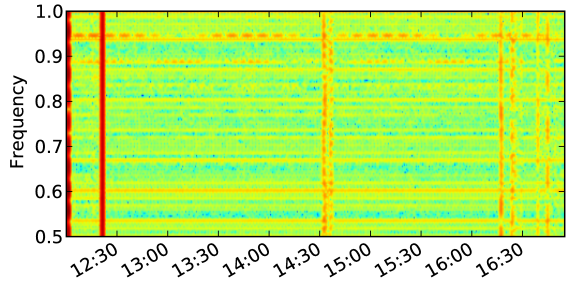


Figure 6.2: Observing changes in periodic behavior through a STFT

For the first fourth of the time series, a single polling period of 3 seconds is used. The corresponding frequency component is present at $\frac{1}{3}$ Hz, which can be observed as a horizontal red line in the spectrogram. After some time, a second polling series starts, with a period of 5 seconds. In the spectrogram, the corresponding frequency (0.2 Hz) and its first harmonic (0.4 Hz) become apparent, as red horizontal lines. In the second half, the amount of responses produced by the second polling instance is doubled, which can be observed as a



(a) Timeseries - service 1



(b) Spectrogram - service 1

Figure 6.3: Detecting anomalies in *service 1*

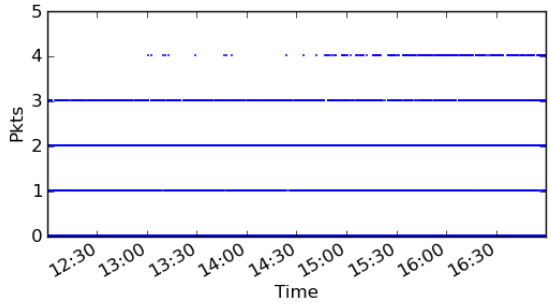
darker shade of red in the corresponding frequencies. A large burst of packets happens around 40000 s, which can be observed as a short increased amount of energy across the whole frequency spectrum (note the yellow vertical line). Finally, the amount of non-periodic data in the link is increased for the last fourth of the sample. This can be observed by an increase in energy in all frequencies (note the change from green to yellow across the *y-axis* range).

In Figure 6.3 and 6.4, we show the results of applying the approach to two time series generated from real traffic traces. These figures use the same "power" scale as in Figure 6.2. The time series represent all packets sent by a client to a specific service. Two different services are considered. In this test, we use a 10 Hz sampling frequency (1 sample per 0.1 s), as it seems a good trade-off between processing time and accuracy.

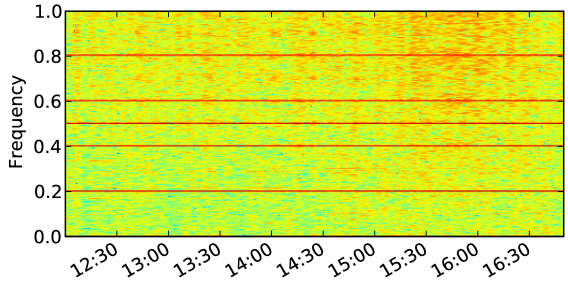
Figure 6.3(a) shows the time series for the traffic sent to *service 1*. A few peaks are present in the time series. These peaks in the time domain cause, as any other non-periodic activity, an increase in noise in the frequency domain, as the energy is spread over the whole frequency spectrum. The effects of these peaks are clearly visible in the spectrogram for this time series as red and orange vertical lines in Figure 6.3(b). Another interesting behavior is the intermittent activity at around 0.95 Hz. The spectrogram suggests a periodic burst of packets at this frequency, but not for the whole duration of the sample. We are not certain of the causes of this behavior. This intermittent activity exemplifies the first major shortcoming of detecting changes in the periodic behavior using this method: there is a *semantic gap* between the behavior that can be observed in the spectrogram and what causes it. Approaches based on the FFT and ACF, like the one discussed in this section, simply identify the presence of periodic activity, however they do not provide any insight which traffic *causes* this periodicity. These techniques treat the time-series as an inseparable flow of values [72]. In other words, these tools do not identify which packets contribute to the periodic pattern and which do not.

To elaborate on this point, consider the example of a signal composed by one pulse (e.g., one packet) every second. Now consider how this behavior can be generated. It could consist of a single request being issued every second. Alternatively, it could be a set of 4 different requests, each being sent every 4 seconds. In fact, any set of n requests being sent every n seconds could generate the same signal, as long as they are equally spaced. Regardless of the value of n, the period (correctly) identified by either the periodogram or by the ACF is 1 s. In summary, there is no fixed relation between the period identified by these techniques and the period of a cycle.

Figure 6.4 shows the "better behaved" traffic sent for a different *service 2*. The spectrogram shows two main frequencies at 0.2 Hz (and its integer multiples, or harmonics) and 0.5 Hz, visible as red horizontal lines. An increase in the noise level can be identified after 15:00. Note the increased energy across the whole frequency spectrum at this point (overall, the figure becomes less green and more orange). A closer inspection of the data reveals that this increase is due to small variations in the periodic burst interval. While before 15:00 the bursts have a period very closed to 5 s, after 15:00 the period presents increased variation in the range [4.8, 5.1] s. Note that this is an expected variation when considering the fact that small network delays are common. This high sensibility to small period variations constitutes the second major shortcoming of spectral analysis methods.



(a) Timeseries - service 2



(b) Spectrogram - service 2

Figure 6.4: Detecting anomalies in *service 2*

6.3.2 Automata

Based on the observation that the traffic exchanged between an HMI and a PLC consists of requests for the same values being sent periodically, Goldenberg and Wool [62] proposed to model Modbus traffic by means of DFA. In their approach, the automaton captures the order at which requests and their respective responses are normally exchanged and triggers alarms when an unexpected transition, i.e., an unexpected sequence of two messages, is observed. The proposed approach is able to automatically learn the automaton given a training set. The authors validate their approach using two datasets captured at a system used to monitor the campus power grid at Tel Aviv University.

In the proposed approach, each *channel* representing the communication

between an HMI and a PLC is described in a separate automaton. A *channel* is identified by tuple (Master IP address, Slave IP address), except in cases where the PLC serves as gateway chaining one or multiple PLCs. In this case, a channel is identified by the 3-tuple (Master IP address, Slave IP address, Unit Identifier)[1]. In the automaton, transitions are triggered by the observation of a new message (either request or response), and each state is labeled with the identifier of the last message received.

Formally, a DFA is defined as a five tuple $(S, \Sigma, \delta, s_0, F)$, where S is a finite set of states, Σ is a finite set of input symbols referred to as alphabet, $\delta : S \times \Sigma \rightarrow S$ is a transition function, $s_0 \in S$ is start state and $F \subseteq S$ is set of accept states.

In Figure 6.5, we show an example of the automaton used in their work. The state space is composed by the different requests and responses observed by the learning module. In the figure, each of the four states s_1 to s_4 are labeled either as Q_n or R_n, where Q denotes a request, R denotes a response, and n denotes the order at which the messages were observed, i.e., Q_1 is the first observed request.

The alphabet is composed by four fields in the Modbus header, specifically the 3-tuple `(function code, starting address, quantity of registers/coils)`. Requests are identified by this 3-tuple, and responses are matched to requests using the *transaction identifier* field.

Similarly to the states, transitions are labeled either as q_n or r_n. Four types of transitions are defined:

- *Normal* transitions represent the occurrence of the next expected symbol in the periodic pattern;
- *Retransmission* is the recurrence of a symbol;
- *Miss* is the occurrence of a symbol out of the expected order, and;
- *Unknown* is the occurrence of a previously unknown symbol.

The start state is defined as the first request observed and no accept states are defined, meaning the automaton continuously monitor a data stream.

Despite the low number of false alarms when testing the approach using 120 h of Modbus traffic, the approach presents the following two shortcomings.

First, even if a channel contains cycles with multiple periods, a single automaton will be used to model it. For instance, consider the case observed by

[1]For more information about Modbus features, such as PLC chaining, the reader is referred to Appendix A.

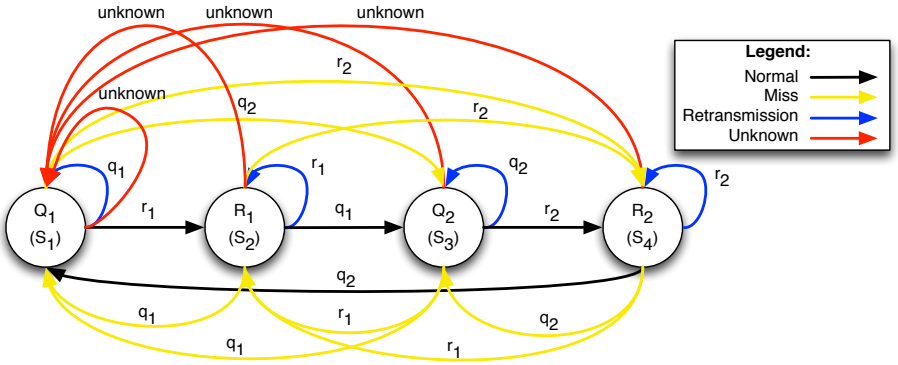


Figure 6.5: A 2-request Modbus automaton [62]

the authors for one of the monitored PLCs. Requests were issued for this PLC at three different rates: a low frequency pattern with 24 h period, a mid frequency pattern with 15 min period and a high frequency pattern, for which the authors do not present the exact period. Modeling the traffic to this PLC in a single DFA resulted in a very large chain of approximately 9.6 million states. The authors acknowledge that such automaton would be difficult to learn, as a large amount of "clean traffic" (i.e., traffic without the occurrence of anomalies) is necessary. In addition, the resulting automaton would be inaccurate, as the relative order at which messages are sent does not necessarily remain the same. When a different order is observed, either *miss* or *unknown* transitions will be taken.

To address this problem, the authors propose an extension that deals with multiple periods by chaining multiple automata. However, the number of automata has to be manually defined, therefore the learning is no longer completely automated. The authors describe a two-level approach, able to deal with the mid and high frequency patterns described above. In the reported experiment with the two-level approach, spurious alarms are still generated for the low frequency pattern.

A second limitation is that the model does not directly capture timing aspects, that is, the automata capture only the order at which message are exchanged, but not the time between them. This limitation can be exploited

to perform an attack. For instance, an attacker could rapidly inject a valid sequence of requests in an attempt to overload the PLC. No alarm would be raised in response to such attack since the sequence of messages would be identical to the one learned.

6.3.3 Data mining

The problem of periodicity detection in time series also has been addressed by the data mining community [97]. In this context, a time series is a sequence of equally spaced time intervals in the form D_i, where D is a symbol from a certain alphabet and i is a time index. Therefore, a time series S takes the form $S = D_1, D_2, D_3, \cdots, D_n$, which is commonly represented as a text string, e.g., `abcdabyz`.

Han and Yin [72] address the problem of mining partial periodic patterns in symbolic time series. In their paper, the authors propose a method that searches for *imperfect* and *partial* periodic patterns. In a *perfect* periodic pattern, the pattern reoccurs in *every* cycle, rather than in *most* cycles, as is the case of the *imperfect* counterpart. In a *full* periodic pattern, every symbol (approximately) contributes for the periodic behavior, while in *partial* periodic pattern, only a subset of the symbols contribute for the periodic behavior. In practice, partial means that patterns are allowed to contain one or more "don't care" symbols, which represent any symbol in the alphabet. For example, the partial pattern `ab**`, where `*` is the "don't care" symbol, is perfectly periodic in the sequence `abcdabdeabxbabab`.

Given a time series, a period and a confidence threshold (which quantifies how imperfect the pattern is allowed to be), the approach proposed by Han and Yin [72] searches for all periodic patterns present in the time series. The limitation of having the periods to be known in advanced is addressed in more recent works [19, 26].

Berberidis et al. [19] extend the approach by adding a pre-processing *filter step*, where the ACF is used to estimate a set of possible periods for each symbol in the alphabet. The algorithm proposed by Han and Yin [72] is then applied to each of these periods.

Cao et al. [26] proposes an alternative approach centered on a novel data structure called Abbreviated List Table (ALT). The ALT maintains occurrence counts for all symbols in the input sequence and its design allows it to be efficiently mined for partial periodic patterns using an algorithm proposed by the authors.

We argue that the data mining algorithms discussed are not directly applicable to the problem of learning periodicity in network traffic data. The typical approach used to transform network traffic into the time series format presented above is necessary to sample traffic at equally sized intervals. The first issue then is defining what the time series represent. One possible approach is to count the number of packet observed in an interval, as in the spectral analysis approach discussed in Section 6.3.1. However, this choice would lead to the same *semantic gap* problem discussed in that section.

Alternatively, one could label each interval with the message observed in it, as in the automata approach discussed in Section 6.2.2. Intervals without messages could be labeled with an extra "no message" symbol. However, this choice would lead to scalability issues. Since the time series should have a single symbol per interval, the sampling interval would need to be relatively small in order to avoid multiple packets falling in the same interval. As a consequence, the periodic patterns would likely contain a very large number of "no message" symbols. For instance, using this approach, a request being issued at 15 min, observed in a time series sampled at 1 ms, would generate a pattern of 900000 symbols.

We note, however, that even using 1 μs intervals cannot guarantee the requirement of one symbol per time interval. TCP is allowed to merge multiple application Protocol Data Units (PDUs) in a single segment, causing these PDUs (e.g., requests) to be observed in a single interval [138].

A third possibility is to ignore time information and model traffic as a sequence of messages $m_1, m_2, m_3, \cdots, m_n$. However, this approach would have the same limitation as [62] (Section 6.3.2). The lack of timing information can be exploited, for instance, by rapidly injecting a valid sequence of messages in attempt to overload a PLC.

More generally, the algorithms discussed in this section assume that the periodic patterns are "hidden" and need to be "mined". The problem we address in this chapter, however, assumes that most of the sequence is periodic, and we are ultimately interested finding changes in the periodic patterns and other non-periodic events.

6.4 PeriodAnalyser

Given the limitation of existing approaches, in this section we describe *PeriodAnalyzer*, a novel approach to learn periodic traffic patterns and detect anomalies. The starting point of our approach is the DFA method [62] discussed

in Section 6.3.2. As in that work, we model communication to and from PLCs as a series of requests, which are identified through deep packet inspection.

PeriodAnalyzer is composed by four modules: the Multiplexer, the Tokenizer, the Learner and the Monitor. Its basic architecture is shown in Figure 6.6. In this section, we describe an offline approach used to analyze our traffic datasets. We elaborate on the necessary changes to design an IDS based on *PeriodAnalyzer* in Section 6.6.2.

The Multiplexer, discussed in Section 6.4.1, receives the packets captured from the network, and discards all packets that are not relevant for the analysis. It then multiplexes the remaining packets based on an aggregation key composed by the IP addresses and transport port numbers of server and client. We refer to the group of packets with the same key as a *flow*.

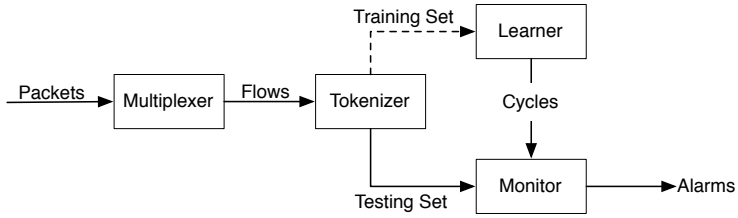


Figure 6.6: Our approach

The Tokenizer, discussed in Section 6.4.2, then processes each flow independently. The objective of this module is to reduce the amount of information that needs to be processed by subsequent modules, by keeping only the necessary information to identify the different requests and responses based on their contents. The basic idea here is that traffic exchanged by PLCs consists of requests for the same values (e.g., the pressure in a pump) repeated at fixed intervals and their respective responses. The Tokenizer identifies the different requests and keeps the information necessary to pair them to responses. In this chapter, we focus on two protocols used to communicate with PLCs in our datasets: Modbus and MMS. Therefore, the description of the Tokenizer is also tailored to these two protocols.

The Learner processes a *training set* with the goal of finding all *cycles*, i.e., requests that are sent with the same frequency. Note that some (possibly all) requests in a flow might not be sent in a periodic fashion, so the cycles learned might contain only a subset of the requests present in a flow. In addition,

note that although we do not include the responses in the cycle definition, it is still possible to detect anomalies caused by missing responses. The Learner is presented in more detail in Section 6.4.3.

We acknowledge that the SCADA system operators might know the cycle information or, alternatively, it could be extracted from configuration files [69]. Consequently, one could make use of this information instead of learning it from the network traffic, eliminating the need for the Learner module. However, in our experience with real-world utilities, this information is not readily available or might be incomplete.

Finally, in Section 6.4.4 we discuss the Monitor. Once the training time is over, all learned cycles are forwarded to the Monitor. The task of this module is then to verify whether the testing set presents anomalies in its periodic behavior. In particular, the Monitor generates alarms for the following events:

- **New Request:** generated when a request not previously processed by the Learner is observed.
- **Incomplete Cycle:** generated when the end of a cycle iteration is detected, but not all requests belonging to the cycle are observed.
- **Long Iteration:** generated when the time between the first request of two consecutive cycle instances is larger than the learned period for the cycle.
- **Repeated Request:** generated when the time between the recurrence of a request is less than the learned period for the cycle.
- **Unmatched Requests and Responses:** generated when the Monitor observes responses for which no requests were issued, and vice-versa.

6.4.1 Multiplexer

The objective of this module is to filter the SCADA protocol packets, discarding the remaining traffic, and to multiplex the packets into different *flows*.

The filtering task is performed by identifying all packets that contain a Modbus or MMS application header. Both protocols use TCP as transport layer, and our filter choice means we discard all packets that do not carry application data, such as the ones exchanged in the 3-way handshake and connection teardown, and TCP ACK packets.

In Section 6.2.1, we discussed that applications can either generate multiple connections periodically or use a single connections for all requests. As a consequence, we create flows based on two keys covering both cases.

We call cases in which a new connection is set up when requests need to be sent as *short-lived* flows. We used the following 4-tuple key for such flows:

(Server Address, IP Protocol, Server Port, Client Address)

We define *long-lived* flows for the cases in which a single connection carries all periodic requests. We add the client port information, thus using the following 5-tuple key:

(Server Address, IP Protocol, Server Port, Client Address, Client Port)

All elements of these aggregation keys can be directly mapped to fields in the TCP/IP header: server and client addresses are IP addresses, IP protocol is a field in the IP header and server and client port are the transport port field. Similarly to the Connection and Flow Creation module discussed of the flow whitelisting approach presented in Chapter 5, the aggregation keys used here require the identification of server and client. However, instead of using Algorithm 2 (see Chapter 5), we can use a much simpler method. Both Modbus and MMS use well-known TCP ports, 502 and 102 respectively; therefore we set the server side to whichever side uses these ports.

The distinction between short- and long-lived allows for a fine-grained view of the traffic. One of the advantages is that if some connections are later shown not to carry periodic data, such as in the case of data being retrieved manually or alarm conditions being reported by the PLCs, these can be separately considered, avoiding false alarms. In practice, we separate short and long flows based on their duration. In our datasets, short flows typically have their duration below a second.

The filtering task and the server side identification are the only parts that need to be updated in the Multiplexer to support new protocols.

6.4.2 Tokenizer

The goal of this module is to transform the packets from different protocols to a common format, which is later used by the Trainer and Monitor. Instead of processing the full-packet information, the Tokenizer identifies the different requests and keeps the information necessary to pair them to responses. As a

consequence, the Tokenizer is the only module that needs to be considerably adapted in order to support new protocols. The filter in the Multiplexer also needs to be updated, but we consider this to be a minor change.

For each received packet, the Tokenizer outputs a tuple formed by its time stamp and a *token*. In a request the token consists of two parts: a *message identifier*, used to identify the message and a *pair identifier*, used to pair request and response messages. In a response, the token consists only of the *pair identifier*. Remember that whereas we expect the same requests to be periodically repeated (e.g., *what is the pump pressure?*), the contents of its response are likely to change.

The token is based on application header fields, therefore its generation is protocol dependent. In this work, we consider two protocols used in SCADA environments: Modbus and MMS. We now described which fields are used by the Tokenizer to generate tokens.

We generate the *message identifier* for Modbus requests based on the following fields: (`length field`, `unit identifier`, `function code`, `data`), and we use the value present in the `transaction ID` field as *pair identifier* for both requests and responses.

As discussed in Section 6.2.2, we focus on seven MMS PDU types that we observed in periodic traffic exchanges: `confirmed request`, `confirmed response`, `initiate request`, `initiate response`, `unconfirmed request`, `conclude request` and `conclude response`. We generated the *tokens* based on the following fields.

- Confirmed requests are composed of four fields. The `invokeID` field is used as pair identifier. All remaining fields are used as message identifier: the `confirmedServiceRequest` field, which contains details about the request (e.g., a read request for a specific variable); and two optional fields.
- For unconfirmed requests all fields are used as message identifier: `unconfirmedService`, which is analogous to the `confirmedServiceRequest`, and one optional field. Note that confirmed requests do not trigger responses, so the pair identifier is not applicable.
- There must not be multiple pending initiate and conclude requests [89] between two end hosts, thus it is sufficient to use the string with the PDU type name (i.e., “initiate” and “conclude”) as message and pair identifier.

Finally, we note that the optional fields of confirmed and unconfirmed requests

are not observed in our datasets, therefore, not actually used in practice. Also, we note that adding new PDU types to the Tokenizer is a straightforward task.

Table 6.1 summarizes the information extracted from Modbus and MMS PDUs to generate the *message identifier* and *pair identifier* that form a token.

Protocol	PDU	Message Identifier	Pair Identifier
Modbus	request	`length field, unit identifier, function code, data`	`transaction ID`
	response	None	`transaction ID`
MMS	confirmed request	`confirmedServiceRequest` and optional fields	`invokeID`
	confirmed response	None	`invokeID`
	unconfirmed request	`unconfirmedService` and optional field	None
	initiate request	"initiate"	"initiate"
	initiate response	None	"initiate"
	conclude request	"conclude"	"conclude"
	conclude response	None	"conclude"

Table 6.1: Mapping Modbus and MMS PDUs to tokens

6.4.3 Learner

The objective of the Learner is to process each tokenized flow in a *training set* with duration T, and learn its cycles and non-periodic requests.

A fundamental part of our algorithm is identifying potential cycles, which we refer to as *candidates*. Remember that we do not assume that requests are sent in a fixed sequence, so we cannot rely on the order requests are sent to identify candidates. Instead, we make use of repeating requests to delimit candidates; once a repeated request is observed a new candidate is created. If two consecutive candidates do not have the same requests, we discard all but the last candidate. The search for candidates is over when we identify N identical candidates, where N is a manually set parameter.

For instance, consider a multi-threaded application sending four requests `abcd` periodically. The following sequence is observed by the Learner: `abcd cdba dcba abcd`. Using repeated symbols we are able to isolate the four iterations. The symbol `c` is repeated at the 5^{th} position, delimiting the first iteration. Similarly, `d` at the 9^{th} and `a` at the 13^{th} positions delimit respectively the second and third iterations.

Note also, that if we started to capture traffic at a later moment, say, at the first `d`, the Learner would eventually synchronize to the periodic activity. The first candidate would be `dc`, as `d` repeats at the 6^{th} position (considering the original sequence). Similarly, the second candidate would be `dba`, as `d` repeats at the 9^{th} position. After failing in these two attempts, the Learner would then correctly synchronize to the periodic behavior, by identifying the third and fourth candidates, respectively as `dcba` and `abcd`.

Using symbol repetition to delimit candidates also allows us to deal with non-periodic requests, as long as they do not happen too often. More precisely, the Learner can identify periodicity if N cycle iterations are observed without the occurrence of a non-periodic request.

Once we identify a group of N candidates, we verify if they are indeed periodic, as the grouping my have occurred by chance. For that, we calculate the duration of each candidate. The duration of a candidate is defined as the time between its first request and the first request in the following candidate. We then verify if the difference between the maximum and minimum duration of the tested candidates is within a threshold θ. If the candidates are periodic, but their durations present a high standard deviation, setting θ to a low value might cause the test to fail. In contrast, setting θ to high values might cause the test to accept non-periodic candidates. In our tests, we set θ to 1 s.

If the sequence we are observing contains multiple periods, or too many non-periodic messages, the algorithm will not be able to identify N identical candidates. To address this limitation, we propose a preprocessing step, where we group requests that are likely to belong to the same group. The requests are grouped according to their number of occurrence in the training set. The idea is that requests sent with the same frequency will occur roughly the same number of times in the training set. In case multiple cycles are present in a flow, they should have different numbers of occurrences and, thus, be separated at this step.

Note, however, that even messages in the same cycle might have a different number of occurrence, for instance, if the duration of the training set T is not a multiple of the cycle duration(s) present in the training set, or if requests are lost. We introduce a tolerance ϵ to deal with this issue. Requests with the same number of occurrences, more or less ϵ, belong to the same group. In our tests, we set ϵ to 1.

Algorithm 4 shows the pseudo-code for the Learner. The algorithm can be split into 3 steps:

1. **Group Requests.** Count the number of occurrences of each request, and form groups of requests with the same number of occurrences, considering the tolerance ϵ.

2. **Find Candidates.** Explore each group searching for cycle candidates. A candidate starts as an empty set. Requests are added to the candidate, until a request is repeated, triggering the creation of a new candidate. If N identical candidates are found, proceed to step 3. If it fails, iteratively test all subset combinations of requests in the group. Requests failing all subsets are reported as non-periodic. We stress that the order of the messages is irrelevant, e.g., a candidate `abc` is identical to `cab`.

3. **Test candidates.** Extract the minimum (dur_{min}) and maximum (dur_{max}) duration of all candidates, and verify if the difference is above

Input : A tokenized flow and parameters N, ϵ, θ
Output: A list of tuples (request set, $dur_{min}, dur_{max}, dur_{std}$)
`/* Step 1: group requests */`
count request occurrences ;
group requests with same counter $\pm\epsilon$;
for *each group* **do**
  for *each subset in group* **do**
    `/* Step 2: find candidates */`
    for *each request in subset* **do**
      candidates ← N repeating cycles;
      `/* Step 3: test candidates */`
      dur_{min}, dur_{max} ← minimum and maximum candidate duration;
      if $dur_{max} - dur_{min} < \theta$
        dur_{std} ← cycle duration standard deviation;
        store request set, $dur_{min}, dur_{max}, dur_{std}$;
        continue to next subset;

      else
        reset candidates;

    end
    store remaining subset requests as non-periodic
  end
end

Algorithm 4: The Learner algorithm

a threshold θ. If the test succeeds, return the requests in the candidate, dur_{min}, dur_{max} and the standard deviation of the duration dur_{std}. If it fails, go back to step 2 and find another candidate set.

The algorithm implicitly assumes that one request does not appear in multiple cycles. As the request sent with faster frequency makes the information retrieved by the one sent with lower frequency redundant, issuing the same requests with different frequencies is inefficient and should not happen in practice. Alternatively, one could re-write the polling application, removing the low frequency request without loss of information.

The time complexity of the algorithm is exponential with the number of messages of the largest group identified in the first step of the algorithm, as all subset combinations are tested. Suppose n is the size of the largest group. In the worst-case, where all requests are not periodic, the algorithm has to test $\binom{n}{n} + \binom{n}{n-1} + \binom{n}{n-2} + \cdots + \binom{n}{1} = 2^n - 1$ combinations. For instance, if the group consists of messages 10 messages, in the worst-case, 1023 combinations need to be tested.

Note, however, that we do not test all requests combinations blindly. In step 1, initial groups are formed by the grouping requests with roughly the same number of occurrences. Furthermore, non-periodic requests might be discarded when searching for candidates in step 2. Keep in mind that we assume that periodic request-response pairs represent the majority of the traffic, therefore, we do not consider the case where flows have a high number of non-periodic requests. As we will show in Section 6.6.3, in practice we always found the Learner to finish after a relatively small time in comparison with the training set size T.

Finally, as discussed in Section 6.2.1 small timing variations are expected, therefore, in order to avoid false-positives when monitoring a cycle, we relax the learned thresholds dur_{min} and dur_{max} based on the calculated dur_{std} and manually set parameters min_{thr} and max_{thr}. Each threshold is updated as follows:

$$\begin{aligned} dur_{min} &= dur_{min} - max(min_{thr}, 2 \times dur_{std}), \\ dur_{max} &= dur_{max} + max(max_{thr}, 2 \times dur_{std}). \end{aligned} \tag{6.1}$$

As we show in Section 6.5.3, setting min_{thr} and max_{thr} can reduce the number of alarms caused by (normal) timing variations.

6.4.4 Monitor

The goal of the Monitor is to watch the periodic flows uncovered by the Learner, and report anomalies in the periodic behavior. Each cycle within a flow is monitored independently.

The Monitor algorithm is depicted in Figure 6.7 as a flowchart. In the flowchart, diamonds represent decision points, squares represent actions and hexagons represent alarms. When a request is sent to the Monitor, it first checks whether it was learned. If not, a **New Request** alarm is generated. If yes, the request is forwarded to the corresponding cycle monitor.

Each cycle monitor has one state variable, termed *IDLE*, and one timer. *IDLE* is true if all cycle requests are already observed in the current iteration, and false otherwise. The timer is used to keep track of the duration of the current iteration, by comparing it to the minimum and maximum learned cycle durations, dur_{min} and dur_{max}, respectively. The timer is restarted every time a new iteration starts. Initially, *IDLE* is false, and the timer starts with the arrival of the first request.

When monitoring a cycle, we use a combination of the *IDLE* state and timer to define where iterations start. There are three possibilities:

- In the normal behavior, a new iteration starts when *IDLE* is true, and a request is received after the timer exceeds dur_{min}, but not dur_{max}.
- A **Long Iteration** alarm is generated when *IDLE* is true, but the next request only arrives after the timer exceeds dur_{max}.
- An **Incomplete Iteration** alarm is generated when *IDLE* is false, and a request is received after the timer exceeds dur_{max}.

If a request is received more then once in an iteration, a **Repeated Request** alarm is generated. Finally, we also keep track of **Unmatched Requests** and **Unmatched Responses**, i.e., a request for which no response is observed and vice-versa.

6.5 Evaluation

In this section, we the evaluation of *PeriodAnalyzer* feasibility. We first present the datasets used in the analysis in Section 6.5.1. The evaluation consists of two tests. In the first test, we verify whether our assumption regarding connections

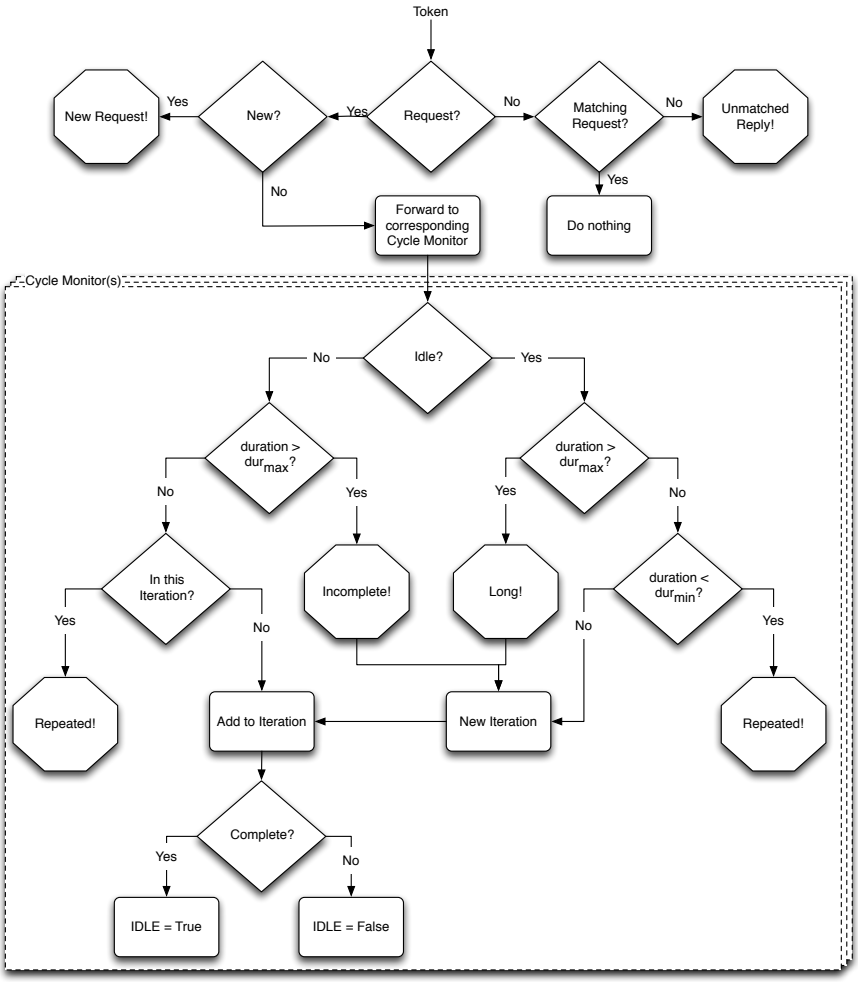


Figure 6.7: Monitor diagram

to field devices is valid, that is, if the flows present in our dataset consist only of a series of periodic requests and their responses. This is the topic of Section 6.5.2.

In the second test, discussed in Section 6.5.3, we monitor the learned cycles

for time-related anomalies. Our objective is to establish whether it is possible to learn reliable duration thresholds from the training set, and avoid false-positives caused by normal delays in network traffic.

6.5.1 Datasets

We start our discussion of the evaluation by presenting the used datasets. For our analysis, we use SCADA measurements collected at three locations, previously discussed in Chapter 2. One location uses Modbus and two use MMS.

Table 6.2 shows the number of flows encountered by the Multiplexer in each dataset.

Dataset	# short flows	# long flows
Modbus	0	20
MMS 1	2	20
MMS 2	11	14

Table 6.2: Short and long flows counters

We split each flow in the datasets in two parts: a training and a testing set. In our datasets, short flows typically have their duration below a second, while long flows last for approximately the whole dataset duration. Unless stated otherwise, the training set consists of the first 30 min of a flow, and the testing set is the remainder of the traffic. Considering that all datasets contain more than one day of data (up to 13 days), and that the periods we observed in the analysis presented in Chapter 3.3.2 for the SCADA datasets are in the order of a few seconds, this choice is a good compromise between a sufficient number of samples and the length of the datasets.

6.5.2 Validating the communication model

In this first analysis we evaluate whether our assumption that SCADA flows consist of a series of periodic requests, and their responses, is valid. The objective of this analysis is two-fold. First, we study the number of flows that can be modeled by our approach. In addition to describing the learned periodic patterns, we manually inspect the flows to verify if they agree with the SCADA protocol communication model described in Section 6.2.

We classify each periodic flow in our dataset according to three characteristics: single or multi cycle, single or multi request (per cycle), and single or multi connection (per flow). For a flow to be considered multi request, it is sufficient that one cycle contains more than one request.

Second, we test whether the learned cycles are representative of the flow behavior, or if new requests are observed in the testing set. We use the Monitor and study the number of *new request* events in our testing sets.

In this part of the analysis we are not interested in time-related anomalies, such as *long iteration*. Therefore, we do not aim at generating an accurate estimative of the cycle duration thresholds dur_{min} and dur_{max}, setting then initially to 0.1 s, and use only two candidates for learning ($N = 2$). Considering that the periods observed for PLC traffic in Chapter 3 are at most 21 s, we create a training set using the first 30 min of each dataset (i.e., $T = 30$ min), which should be sufficiently large to contain 2 full cycles without the occurrence of non-periodic requests. The impact of N, dur_{min} and dur_{max} on the number of generated events by the Monitor is studied in Section 6.5.3.

In the plots presented in this section, each point represents a request. The *x-axis* shows the time since the first displayed request, and the *y-axis* shows a unique request identifier. For ease of visualization, the first request shown in the figure is always a request that starts a cycle iteration. As a consequence, often the first displayed request is not the first request observed in the training set. Finally, responses are not shown in the figure.

Modbus dataset

In the Modbus dataset, 19 out of 20 flows are reported as periodic. We identify four different periodic patterns in this dataset.

Single cycle/single request/single connection: The first and simplest pattern is a single request issued every second. We exemplify this pattern in Figure 6.8. In this dataset, 15 flows are fully described by this pattern, i.e., no *new request* is observed in the testing set.

Single cycle/multi request/single connection: The second pattern is shown in Figure 6.9. Like in the first case, the period is 1 second. However, the number of requests is much larger, 20 in the example. Of the three flows that present this pattern, two have 20 requests in a cycle, and 22 in the remaining one.

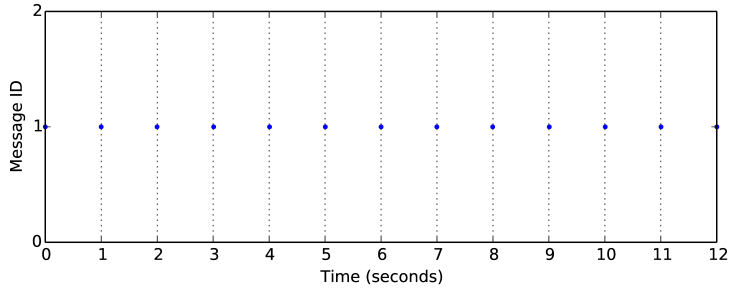


Figure 6.8: Single cycle, single request flow in the Modbus dataset

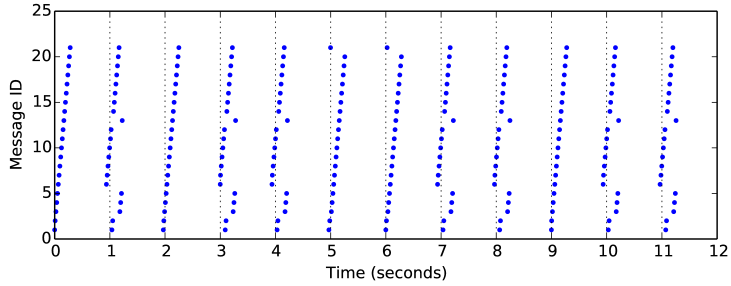


Figure 6.9: Single cycle, multiple requests flow in the Modbus dataset

Interestingly, the order between the requests is not fixed. In the figure, three different orders can be observed. For instance, see the iterations starting at seconds 0, 1 and 5. The lack of order could have been caused by a multi-threaded application, as anticipated in Section 6.2.1. Note that such periodic pattern cannot be correctly modeled by the approach proposed in [62], as it assumes that the order at which requests are issued is fixed.

One of these flows consists only of the learned requests, however this is not the case for the other two flows. In the testing set, one exhibits two new requests, generating a total of 44 alarms, while the remaining flow has three new requests, generating a total of 12 alarms. All requests are reading commands and their occurrence is restricted to small portions of the testing set, therefore we assume this non-periodic activity is related to manual data retrieval.

Clock-like pattern: Two flows display the third pattern shown in Figure 6.10. Considering the initially used training set of 30 minutes (Figure 6.10(a)), these flows are (correctly) classified as non-periodic. Each flow presents 30 different requests in the training set.

However, when we increase the training set size T, a periodic pattern arises. In Figure 6.10(b), we show approximately the first 25 hours of the same flow. It becomes clear that a group of 60 requests are exchanged hourly. Note also there is indication of requests being exchanged daily (requests with ID from 61 to 87).

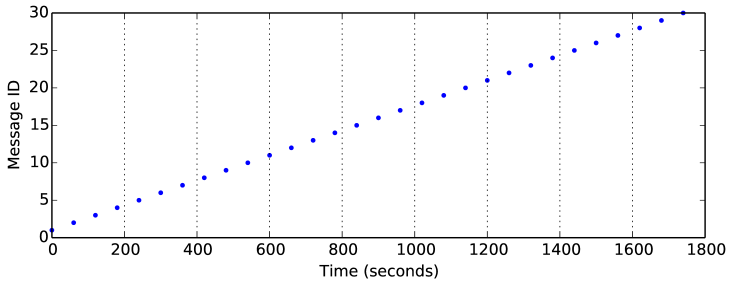
Upon closer inspection, we verify that all requests are for *write* operations. The hourly requests, are sequentially writing values from 0 to 59, while *most* of the daily requests are writing values from 0 to 24. In the iteration starting at hour 22, marked red in the figure, two additional requests are made. We speculate that these messages are a clock synchronization messages: minutes being repeated hourly and hours daily.

To confirm our assumption, we redefine the training set time T as the first 25 h of the dataset for these two flows, and require a single candidate for training ($N = 1$). Although the pattern has a 24 h period, the Learner algorithm requires one additional hour to detect a repeated request, which delimits the end of a candidate.

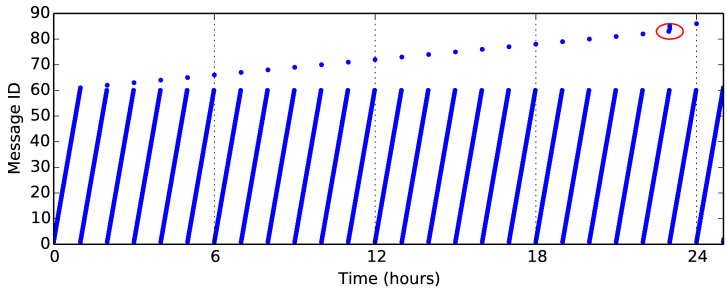
Indeed the hourly and daily patterns are repeated throughout the testing set, but 69 new request alarms are also generated. Some alarms are caused by previously unclear weekly pattern: requests writing values from 0 to 7. For instance, in Figure 6.10(c) it is possible to observe that the extra requests marked in red in Figure 6.10(b) are repeated after a week. In addition, some new requests (marked in green) repeat after 9 days, suggesting an even longer cycle. Finally, 10 new requests are observed at the iteration starting at day 13, which are not periodic in this dataset.

The clock-like pattern can be seen as special case of the multi cycle/multi request/single connection pattern.

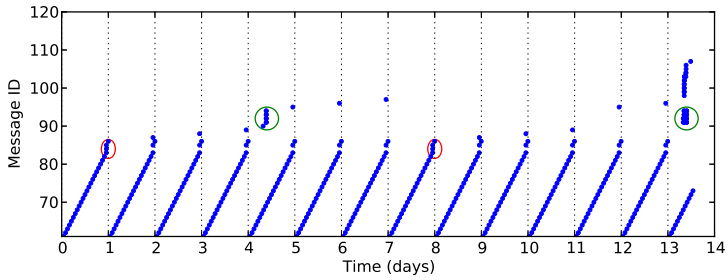
Non-periodic pattern: The last observed pattern consists of write requests, as in the clock-like pattern. However, even extending the training set for periods longer than a day, not clear periodicity appears. We show the first 25 h of this flow in Figure 6.11.



(a) Original training set



(b) Extended training set



(c) Training set plus testing set

Figure 6.10: Clock synchronization flow in the Modbus dataset

MMS 1 dataset

In the MMS 1 dataset, 9 out of 23 flows are reported as periodic. All periodic flows are fully described by our model, i.e., no new requests are observed in the

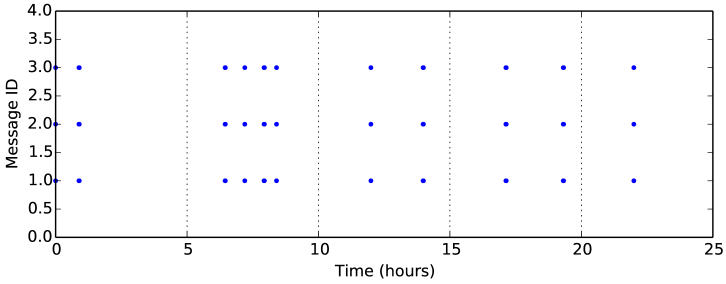


Figure 6.11: Non-periodic flow in the Modbus dataset

testing set. We observe four different patterns in the dataset. We start our discussion with Figure 6.12, which shows two similar periodic patterns.

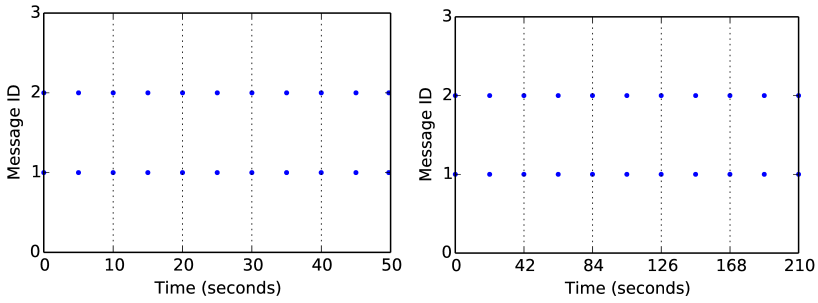
Single cycle/multi request/single connection: Figure 6.12(a) represents a flow where two read messages are sent every 5 s. Two flows present this behavior.

Single cycle/multi request/multi connection: Figure 6.12(b) represents a flow that consists of a series of periodic connections made every 21 s. Each connection consists of two requests, one `initiate` request followed by a `confirmed service` request, and their responses. Interestingly, no conclude request PDUs are observed in these connections. Five flows present this behavior.

Single cycle/single request/single connection: The two remaining periodic flows have a single cycle with a single message over a single connection, as previously depicted in Figure 6.8.

Non-periodic patterns: Five flows are reported as periodic by the Learner, however, the Monitor reports over a million new request events for each of these flows. One of these flows is shown in Figure 6.13. The learned cycle is shown in Figure 6.13(a): three different cycles at 0.5 s, 10 s and 30 s.

However, after approximately 30 min new requests start being observed. In Figure 6.13(b), we show the first day of this flow. A remarkable aspect of these flows is the long periods of "well-behaved periodicity", for instance, between



(a) Single connection with periodic requests

(b) Periodic connections

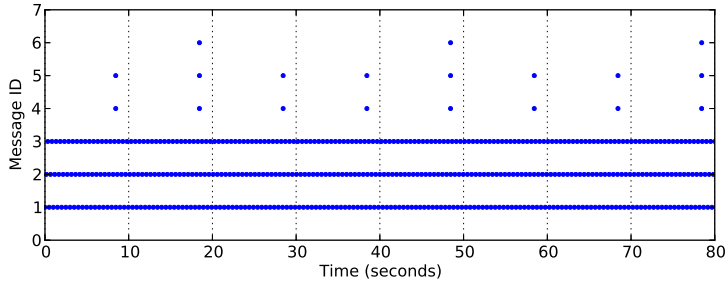
Figure 6.12: MMS flow cycles examples in the MMS 1 dataset

hours 10 and 15. Such periods are common in the remainder of the trace, and applying the Learner (and later the Monitor) to these periods reveals perfectly periodic patterns.

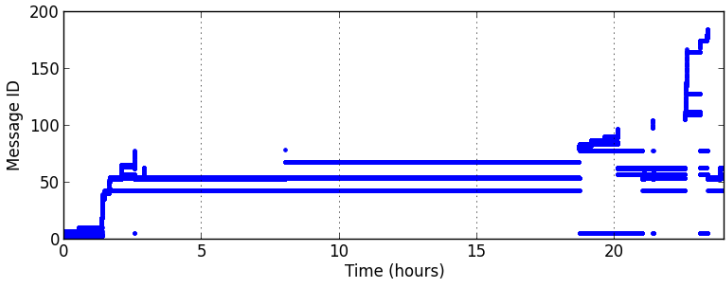
When investigating this pattern in more detail, we observed that the learned requests are `confirmed requests` issuing reading commands to an `unconstrained address`. The new requests are also reading commands, but to a slightly modified address. For instance, in some cases the newly requested address is the previously requested address with some additional bytes.

We speculate that these new requests simply replace the old ones without breaking the periodic pattern. Unfortunately, the meaning of this address is not defined by the standard, and its interpretation is implementation specific. Therefore, changing the Tokenizer to verify this assumption is not a trivial task, which is left as future work.

Of the nine remaining flows, four are single connection flows issuing only `unconfirmed requests` without any apparent pattern and five are single connection flows consisting of several different `confirmed request`, including `get program invocation attributes`, `take control` and `initiate upload segment`. Given that the client-side of these flows is a configuration database and the sessions are reasonably small (in the order of minutes) we speculate these flows are human-generated.



(a) Seeming periodic flow



(b) New requests

Figure 6.13: Flow cycles examples from MMS 1 dataset

MMS 2 dataset

In the MMS 2 dataset, 16 out of 20 flows are reported as periodic. As in the previous section, we start with the periodic patterns.

Single cycle/multi request/multi connection: Two flows consist of periodically made connections, identical to the pattern encountered in the MMS 1 dataset (Figure 6.12(b)).

Single cycle/single request/single connection: Four flows consist of a single request sent every second, a pattern previously encountered in the Modbus dataset (Figure 6.8).

Multi cycle/multi request/single connection: The last periodic pattern observed in this dataset is shown in Figure 6.14. In the figure, two cycles can be observed: request 1 to 4 are sent every second, while request 5 is sent every 5 s. Nine flows present this pattern, however the cycles vary in number of requests and period.

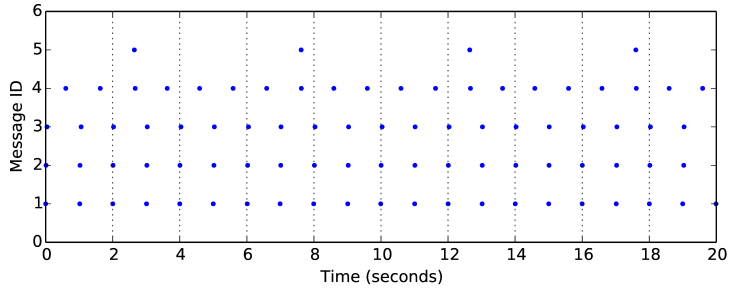


Figure 6.14: Multiple period, multiple request flow in the MMS 2 dataset

Unexpected periodic pattern: One flow, shown in Figure 6.15, presents an unexpected periodic pattern. Using the three cycles for training ($N = 3$), the Learner detects that request 1 has a 0.5 s period, and request 2 is not periodic. However, when manually inspecting this flow, we observe that every 10^{th} request 1 is missing. Request 2 is reported as not periodic, although it can be seen as a group of 2 requests being repeated every 2.5 s.

Our Learner algorithm fails to learn these patterns because they violate the basic assumption that the same request cannot be repeated multiple times within a cycle. As both requests observed in this flow are read commands, it is not clear why such unusual pattern is used to retrieve data.

Non-periodic pattern: The four remaining non-periodic flows present patterns previously observed in dataset MMS 1. Two flows are single connections consisting of `unconfirmed requests` and two flows are initially periodic but present a massive number of *new requests*, similarly to the one shown in Figure 6.13.

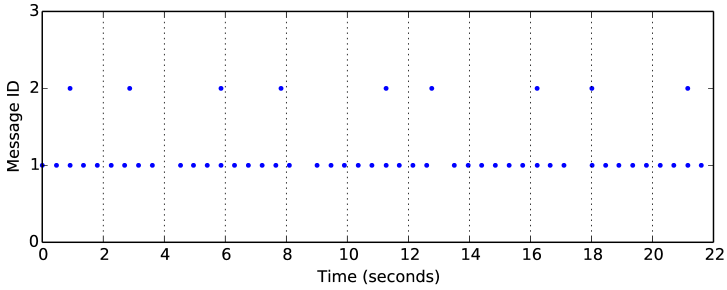


Figure 6.15: Non-periodic flow cycle example in MMS 2 dataset

Summary

In Table 6.3, we show a summary of our first test. Although we defined 8 classes based on the number of cycles, requests and connections, only 4 are observed in our datasets. Between the periodic flows the most common classes are the single cycle, single request and single connection, with 21 occurrences, followed by the multi cycle, multi request and single connection, with eleven occurrences.

Cycle	**Request**	**Connection**	**Modbus**	**MMS 1**	**MMS 2**
Single	Single	Single	14	2	4
Single	Multi	Single	3	2	0
Single	Multi	Multi	0	5	2
Multi	Multi	Single	2	0	9
Non-periodic			1	14	5
Total			20	23	20
#Periodic flows w/ new requests			4	0	0

Table 6.3: Number of flows per type

Most periodic flows observed in our datasets fit the assumption that the traffic consists of a series of periodic requests and their responses. In the Modbus dataset, five flows present *new requests*: two are likely to be caused by manual activity and two are clock synchronization flows, which could not be fully learned. A single flow is reported as non-periodic.

In the combined MMS datasets, 25 out of 43 flows are periodic. The high

number of non-periodic flows could be expected given the more advanced controls supported by MMS, in comparison to Modbus. This difference reflects the scenarios at which these protocols were designed to be used. Modbus is a typical SCADA protocol, supporting only simple supervisory commands, while MMS is a typical DCS protocol supporting more advanced commands, which enable a more tight integration with closed control loop used in the process. Still, over half of the flows consist only of a series of periodic requests.

In the MMS data we observed two patterns that are not captured by our approach, but might still be modeled as periodic traffic. The first of these patterns violates our assumptions that the same request is not repeated within one cycle. In the second case, although requests seem to be exchanged at periodic intervals, their content changes over time. However, if these changes could be predicted, these flows might still be modeled as periodic data.

6.5.3 The impact of N

In the section we evaluate whether the cycle duration thresholds (dur_{min} and dur_{max}) can be learned reliably. We first investigate the impact of the number of candidates N in the number of alarms generated by the Monitor. We expect that the higher N, the better is the estimate for the duration of a cycle. However, large values of N also increase the chances of learning some anomalous pattern as normal behavior.

The analysis is performed as follows. For a given value of N, we plot the number of alarms a in the *x-axis* and in the *y-axis*, the number of flows that generate a or less alarms. This plot is similar to an empirical Cumulative Distribution Function (cdf), however we use the absolute number of alarms, instead of the cumulative probability in the *y-axis*. We vary N from 2 to 30 in increments of 2. We typically see no improvements for $N > 20$. For ease of visualization, we show only results from 4 to 24 in increments of 4.

In this analysis we consider only alarms that are time-based. More precisely, the total number of alarms reported in this section is the sum of the following alarms: *long iteration*, *incomplete iteration* and *repeated requests*. The remaining alarms are invariant with respect to N.

Modbus dataset

Figure 6.16 shows the results for the Modbus dataset. The lines for every value of N overlap, i.e., the number of time related anomalies is the same regardless the value of N.

Most flows generate no alarm, or a single one. Only three flows present more than 10 alarms, which are caused by an "extra" iteration that does not fit the periodic behavior. *Repeated request* alarms are generated for every request sent in this "extra" iteration. In addition, these requests cause the Monitor to become out of synchronization with the periodic pattern, which, in turn, causes the Monitor to generate a few (false) alarms until it synchronizes to the periodic pattern again. We speculate that this "extra" iteration is triggered by a manual update. In a real-world deployment, these alarms could be suppressed or aggregated in a single alarm. As we will discuss in Section 6.6.2, from a security perspective, we are only interested in a high number of *repeated request* alarms as that would be an indicative of a DoS attack.

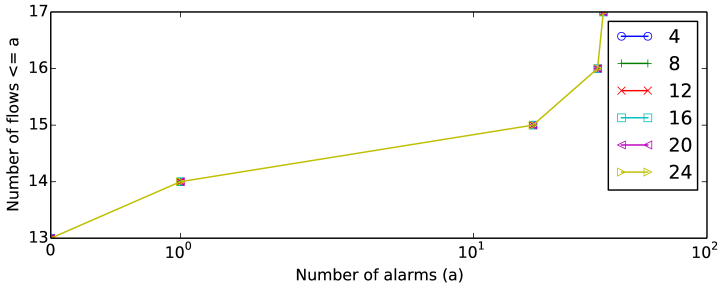


Figure 6.16: Effect of N in the Modbus dataset

We do not consider the flows with a clock-like pattern (see the Modbus results in Section 6.5.2) in this graph. Even when considering a single candidate for learning ($N = 1$), the only observed alarms are the ones previously discussed in Section 6.5.2.

MMS 1 dataset

N has a large impact on the number alarms in the MMS 1 dataset, as it can be observed in Figure 6.17. For $N = 4$ and $N = 8$ (lines overlap), five flows present more than 10^4 alarms. The number of alarms is reduces as N increases, until $N = 20$. At this point, most flows present 10 or less alarms. No improvements are observed using $N > 20$ (note that the lines representing $N = 20$ and $N = 24$ overlap).

Two of the flows still present more than 20 alarms with $N = 20$. In both

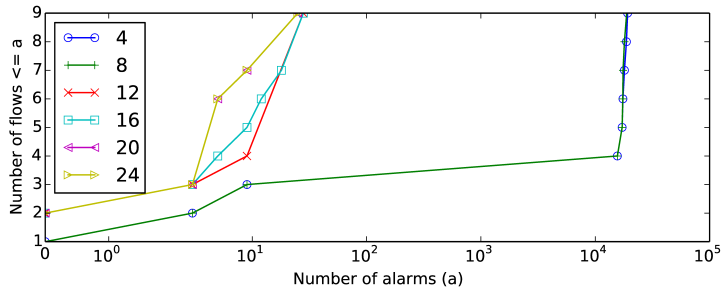


Figure 6.17: Effect of N in the MMS 1 dataset

flows, most alarms follow the *repeat request to long iteration* pattern exemplified in Figure 6.18. In this example, a pair of requests are sent every 21 s. However, the second iteration has a slightly lower duration, that is, the request starting the third iteration is observed 20.4 s after the request starting the second iteration. As a consequence, the Monitor fails to start a new iteration when observing the 5^{th} request. Instead the 5^{th} and 6^{th} requests (marked in red) trigger *repeat requests* alarms. The Monitor starts a new iteration upon the arrival of the 7^{th} request (also marked in red), however, a *long iteration* alarm is triggered as the measured duration of the iteration is 41.4 s. By manually adjusting min_{thr} and max_{thr} to 1 s, it is possible to eliminate these alarms.

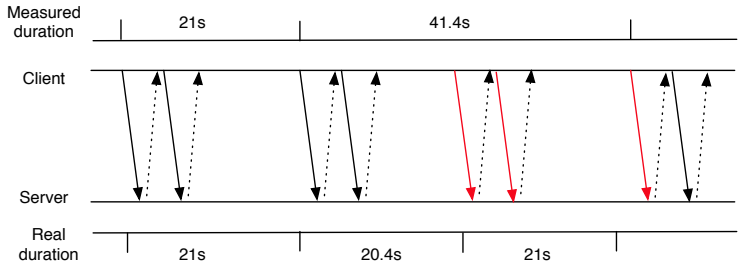


Figure 6.18: Repeat request to long iteration alarm pattern

MMS 2 dataset

Like in MMS 1, the number of candidates have a clear effect on the number of generated alarms. Figure 6.19 shows that increasing N continuously reduces the number of alarms, until $N = 20$ and no improvements are observed using $N > 20$ (note that the lines representing $N = 20$ and $N = 24$ again overlap).

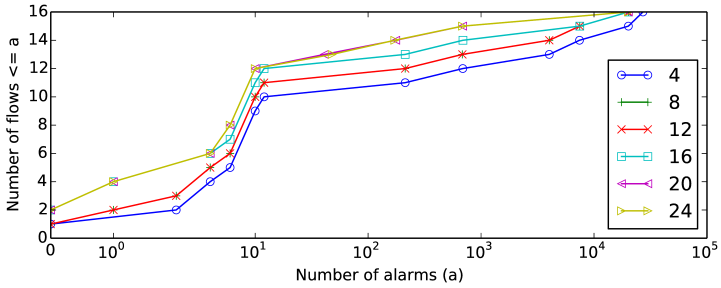


Figure 6.19: Effect of N in the MMS 2 dataset

Even with $N = 20$, four flows present a very high number of alarms, up to 10^4. The *repeat request to long iteration* pattern is very common in these flows. These alarms are caused by very large variations in the iteration duration; the average duration is approximately 1 s, however, durations of 0.5 s are also common in these flows. Given these large variations, it would be necessary to manually set min_{thr} to at least 0.7 s to completely remove these alarms.

Together these flows contain hundreds of TCP retransmissions, and the largest number of *unmatched request* and *unmatched responses* in our datasets. This suggests that the high number of alarms is caused by a bad link quality.

6.6 Discussion

In this section we discuss practical aspects of *PeriodAnalyzer*. In Section 6.6.1, we discuss how real-world attacks can be detected using our approach. In the sequence, in Section 6.6.2, present how *PeriodAnalyzer* can be adapted to perform real-time detection. Finally, in Section 6.6.3, we propose some optimizations.

6.6.1 Dealing with real-world attack scenarios

In this section, we describe attacks from some of the different categories proposed in [74] and discuss how they can be detected with *PeriodAnalyzer*. Our examples are taken from an open access list of real-world SCADA attacks signatures used in the Quickdraw Intrusion Detection System[2]. The list includes signatures to protect Modbus TCP[3] and DNP3[4], two well-know standards for SCADA communication.

Information Gathering attacks may precede other attacks and are an attempt by the attacker to gather as much knowledge as possible of the target system. A typical way to acquire this information is through *scans*, like the *Modbus TCP, Function Code Scan*. In Modbus, *function codes* represent different services, such as reading and writing single or multiple registers. Requesting a non-implemented or disabled function code should trigger an error message, allowing the attacker to map which services are available.

In general, attacks of this type require a large number of parameters (addresses, ports, function codes, etc.) to be tested. Therefore, they are likely to make use of requests not observed during the learning phase, triggering *new request* alarms. Even if it is possible to craft an attack using the learned periodic requests, the scan would still be identified as *repeated requests*.

Denial of Service attacks prevent a legitimate user to access a service or reduce its performance. For example, the *DNP3 - Unsolicited Response Storm* attempts to overload a DNP3 server by sending a number of unsolicited response packets, normally used to report alarms. Again, if the attack uses new requests, they will be immediately identified. The attacker could attempt to repeat the messages from a cycle rapidly, but that would also generate a large number of *repeated requests*.

Network attacks manipulate the network protocols. For instance, the *Modbus TCP - Clear Counters and Diagnostic Registers* attack uses a single packet with a specific code function to clear counters and diagnostics registers in a SCADA server, in an attempt to avoid detection. The *Modbus TCP - Slave Device Busy Exception Code Delay* attack consists of answering every request

[2] `http://www.digitalbond.com/tools/quickdraw/`
[3] `http://www.modbus.org/`
[4] `http://www.dnp.org/`

with the "device is busy" message preventing a response timeout. Such attacks rely on the use of uncommon requests, which would readily be identified as *new requests*.

Buffer Overflow attacks try to gain control over a process or crash it by overflowing its buffer. For example, the *Modbus TCP - Illegal Packet Size, Possible DoS* attack sends a single packet with an illegal packet size, exploiting a bug in the implementation of the protocol stack. This class of attacks relies on forging invalid packets, which are by definition not commonly used, thus also generating *new request* alarms.

In summary, a powerful line of defense created by our approach is the protection against data injection: only requests commonly used during normal behavior might be used to perform attacks. Furthermore, if attacks can be created with such requests, they still need to be sent in a way that does not considerably differs from normal periodic behavior, otherwise they will be detected.

It is important to stress that our goal is the detection of anomalies, i.e., deviations from the normal periodic behavior of the traffic. Such deviations are not necessarily malicious. For example, a manual access to a PLC for testing purposes would seem as an anomalous behavior and trigger alarms.

6.6.2 PeriodAnalyser as a real-time IDS

We identify three aspects that need to be changed in order to transform the offline approach proposed in *PeriodAnalyzer* into a real-time IDS.

First, *PeriodAnalyzer* does not consider the dynamic nature of network flows. In our offline analysis we consider that flows are unrelated, but that is not the case in practice. For instance, two (long) flows in the MMS 1 dataset are interrupted at a certain point. After a few minutes, we observe two new connections, between the same end hosts, issuing the same requests as the previous connections. In our analysis we considered these four distinct flows, however, they are clearly related and a real-time IDS should be able to cope with this situation.

To deal with this situation, we propose the following changes to the Monitor. When initializing the Monitor, all learned flows, short- or long-lived, are associated with the same *4-tuple key* (see Section 6.4.2). The Monitor then matches incoming flows, raising alarms if non-learned flows are observed. Associating the periodic requests of long-lived flow with the *4-tuple key* allows the Monitor to cope with the interruption problem described above, as both the interrupted and new flows have the same key information.

Second, to better deal with non-periodic requests, we propose adding them to a whitelist, possibly after approval from a network administrator. After this change, requests observed by the Monitor would fall under one of three categories: whitelisted requests, periodic requests and new requests.

The last aspect is that not all variations in the periodic behavior are relevant from a security perspective. While detecting *new request* alarms are a powerful protection against data injection attacks, the time related alarms might only indicate performance issues, not security threats. For instance, most of the alarms observed in MMS 1 are cause by large variations in the duration of an iteration, which are possibly caused by normal network delays.

Considering only security aspects, *repeated requests* alarms become an indicator for DoS attempts. While a small number of repeated requests do not pose a security threat, a large number indicates a possible attempt to overload the receiver, by rapidly repeating requests from a cycle. *Long iteration* alarms indicate the senders has become inactive. However, SCADA applications are designed to deal with delays [8], therefore accepting delays larger than those learned from the data might be acceptable, or, even desirable. Finally, while a low number of *unmatched requests* is not a serious threat, a high number indicates that the receiver has become inactive.

6.6.3 Optimizing the learner

The algorithm used by the Learner to identify the cycles has a high run-time complexity. In the worst-case scenario it has to test $2^n - 1$ combinations, where n is the number of requests with approximately the same occurrence in the training set (see Section 6.4.3).

In our datasets, this high complexity did not result in any practical problems. We performed the tests with a standard desktop computer with a 2.66 GHz Intel Core 2 Duo processor and 4 GB of RAM. When analyzing the flows in our training sets, corresponding to 30 min of traffic, the Learner finishes after a few milliseconds for most cases, and takes only approximately 100 ms in the worst-case. These results are orders of magnitude higher than simply reading the same flows from the disk (always in the order of a few microseconds), but hardly a practical problem.

Finally, *PeriodAnalyzer* does not assume ordered requests and idle periods between cycles. However, the candidate test operation can be optimized in the case these assumptions hold. If the requests are always sent in a specific order, the algorithm can immediately discard not-ordered candidates. If we can assume a period of inactivity between cycles, i.e., the algorithm could use such

"idle" periods to synchronize to the start of a cycle, by identifying the longest inter-request time in a candidate.

6.7 Conclusions and Future Work

In this chapter we propose *PeriodAnalyzer*, an approach to learn and detect anomalies in periodic network flows, common in SCADA environments. We evaluate *PeriodAnalyzer* using three real-world traffic traces, covering two protocols: Modbus and MMS. Our results show that most flows of SCADA protocols such as Modbus can be accurately modeled by our approach. Although more variation is encountered in DCS protocols, such as MMS, still a large number of flows consist only of periodic requests, which can also be accurately modeled by *PeriodAnalyzer*.

Our novel approach to learn periodic patterns in network traffic addresses the limitations of existing approaches. We avoid the "semantic gap" of spectral analysis by identifying which messages generate the periodic behavior. Also, we directly capture timing information, making it possible to identify and monitor the frequency at which messages are exchanged. Finally, our approach is able to *automatically* learn multiple periods.

Our main contribution is an anomaly detection method that provides two layers of protection:

1. **Data Injection:** The periodic patterns in SCADA protocol traffic are generated by a polling mechanism that causes the commands sent towards devices in the field network to be highly repetitive. These *requests* are the source of the periodic behavior. Therefore, by learning which requests are normally sent, we can effectively detect data injection attempts, and issue alarms. In other words, we create a whitelist for the possible requests in a flow.

2. **DoS:** The attacker might still be able to overload the PLC by sending more (whitelisted) requests than the PLC is able to process, performing a DoS attack. To detect this type of attack, we learn how often requests are sent, and then issue an alarm if this number deviates too far from the learned behavior. This approach also allow us to identify whether a specific request (or a set of requests) is not sent after a long period of time, for instance, in case the process responsible for issuing the request(s) becomes inactive.

Further research efforts are required to enhance our offline approach in order to create a real-time IDS. We also plan to improve the Learner and Monitor, by testing if certain assumptions can be made about the periodic request cycles, for instance checking if requests are sent in a fixed order. Also, the approach should be extended to deal with other SCADA protocols, such as DPN3 [93] and IEC 60870-5-104 [81].

Finally, we plan to investigate the use of *PeriodAnalyzer* for the generation of synthetic traffic traces. By gathering more information regarding the cycle, such as the distributions of inter-request time and cycle iteration duration, we should be able to produce realistic traffic patterns.

CHAPTER 7

Conclusions and Future Work

7.1 Summary

Supervisory Control and Data Acquisition (SCADA) networks are commonly deployed to aid the operation of large critical infrastructures, including some that are considered to be essential for our society, such as water treatment and power generation facilities. In the past, these networks were completely isolated and relied on special-purpose hardware and software. However, under pressure to increase productivity and cut costs these networks are becoming increasingly interconnected to corporate networks, and even the Internet, thus bringing new security threats. As incidents become more common, the vulnerability of SCADA networks has received the attention of industry, government and academia.

In this thesis, we addressed the problem of intrusion detection of SCADA networks. Despite the increasing attention on the topic, our review of the literature revealed that publications on SCADA networks generally do not rely on empirical data as obtained from real-world measurement. In our view, however, measurements must play an essential role in validating results in network research, and can sometimes lead to surprising insights.

To the best of our knowledge, this thesis provides the first extensive analysis of real-world SCADA network traffic traces. We monitored and analyzed traffic captured at SCADA networks used in the utility domain: two water treatment facilities, one gas utility and one (mixed) electricity and gas utility. We addressed two research questions:

> **RQ 1:** *Does SCADA network traffic differ from the traditional IT network traffic? If yes, what are these differences?*

We showed that SCADA network traffic is relatively "well-behaved" in comparison to traditional IT network traffic. In Chapter 2, we showed that traditional

traffic models are ill suited to describe SCADA traffic. For instance, the diurnal patterns of activity commonly observed in traditional IT networks are not present in SCADA networks. The main reason for this difference is that while in most of traditional IT networks traffic is triggered by human behavior, in SCADA networks, most traffic is generated automatically, for example, by the polling mechanism used to retrieve data from devices in the field network, such as Remote Terminal Units (RTUs) and Programmable Logic Controllers (PLCs).

In addition to the lack of diurnal patterns, we showed that, differently from traditional IT networks, SCADA traffic does not present self-similar behavior and that neither heavy tail nor lognormal distribution provide a good fit for the distribution of connection sizes.

In Chapter 3, we provided empirical evidence for two commonly held assumptions regarding SCADA traffic. We confirmed that SCADA networks present a *stable connection matrix*, that is, we showed that, when comparing to its traditional IT counterpart, the connection matrix of SCADA networks presents considerably less and smaller changes over time. We also confirmed the presence of *traffic periodicity* in SCADA networks. Periodic patterns are particularly common in connections involving PLCs, due to the polling mechanism used to retrieve data from the field.

In summary, we showed that SCADA traffic is considerably different from traditional IT traffic. These differences characteristics, combined with the security requirements discussed in Chapter 4, are the motivation for specialized anomaly detection solutions for the SCADA domain. Which brings us to the second research question:

> **RQ 2:** *How to exploit SCADA traffic characteristics to perform Anomaly Detection?*

To address this research question, we proposed anomaly detection methods that are able to exploit the SCADA traffic characteristics studied in Chapter 3: the stable connection matrix and traffic periodicity.

To exploit the stable connection matrix, we revisited a well-known intrusion detection technique: whitelisting. In Chapter 5, we proposed a *flow whitelisting* approach that resembles an access control list at the network level. We showed that our approach is viable, as flow whitelists have a manageable size, considering the number of hosts in the network.

In Chapter 6, we proposed *PeriodAnalyzer*, a novel approach that exploits the *traffic periodicity* to detect traffic anomalies. We observed that SCADA

protocol connections consist of a sequence of periodic requests and replies. Our approach uses deep packet inspection to automatically identify the different requests belonging to each connection and the frequency at which they are issued. Once such normal behavior is learned, *PeriodAnalyzer* can be used to detect data injection and Denial of Service attacks.

7.2 Main Findings and Implications

In this section we discuss the main findings of this thesis and their implications.

7.2.1 SCADA network traffic show simpler patterns than traditional IT traffic

The investigation of differences between SCADA and traditional IT network traffic, proposed in *Research Question 1*, revealed that the former present simpler patterns than the latter. Our results showed, for example, that SCADA traffic displays periodic traffic patters, does not present self-similar behavior, and its connection matrix is considerably more stable than that of traditional IT traffic.

One of the main causes of the high rate of false alarms in anomaly detection methods developed for traditional IT networks is the enormous variability of network traffic [133]. Given that SCADA traffic present simpler traffic patterns, our results indicate that anomaly detection methods are particularly promising for SCADA networks.

7.2.2 Despite the high regularities, changes in SCADA network traffic do occur

Given the stability of the connection matrix and the displayed traffic periodicity, one could expect that SCADA traffic is easily predictable. However, our results indicate that this is not the case.

In Chapter 3, we showed that, small changes in the connection matrix are common. Also not all traffic is periodic. For instance, when modeling traffic periodicity in Chapter 6, we observe that some connections between SCADA servers and PLCs are not periodic. Such irregularities need to be incorporated in the models that describe SCADA operation. For instance, an intrusion detection system that generates alarms for each newly observed connection, as proposed in [144], would generate a considerable number of false alarms. The

finding of these irregularities support our earlier statement that measurements are essential to network research.

7.2.3 Whitelisting is a promising approach for detecting intrusions in SCADA networks

Whitelisting consists of explicitly enumerating which entities may access which resources. Any access attempt not described in the whitelist should be denied. Despite the high level of protection that can be achieved if correctly configured, a common problem with this approach is that maintaining whitelists is burdensome for the user, due to the large number of changes [54]. For example, constructing a list of all legitimate connections in a traditional IT network will not be a practical solution. There are simply too many possible legitimate connections, thus the size of such a list would be unmanageable. However, in Chapter 5, we showed that this approach is feasible in SCADA networks.

Nonetheless, as noted before, changes in the connection matrix do occur in SCADA networks. Therefore, in Chapter 5, we discussed practical issues when deploying a flow-level whitelist in a SCADA network. We identified two main sources of instability in the connection matrix: manual access and dynamic port allocation services. As blocking legitimate connections can cause severe disturbances in SCADA networks, we suggested only generating alarms until the network administrators are convinced that all configuration mistakes in the whitelist are solved.

We also noted that the traffic model created by *PeriodAnalyzer* in Chapter 6 can be considered as a form of whitelisting. The model generated by *PeriodAnalyzer* determines which messages can be exchanged between the SCADA server and a PLC, i.e, a whitelist where the entities are the SCADA servers, and the resources are the messages these servers are allowed to send to PLCs.

7.2.4 Periodicity of SCADA traffic can be exploited to detect anomalies

The periodicity of SCADA protocol traffic suggests that SCADA traffic is predictable. However, in Chapter 6, we argued that existing methods are not suitable for learning periodic patterns observed in SCADA traffic. We therefore proposed a new approach, called *PeriodAnalyzer*, which is capable of automatically learning communication patterns generated by SCADA protocols.

The proposed approach provides two layers of protection. First, *PeriodAnalyzer* provides protection against *data injection* by generating a whitelist of regularly sent requests. However, an attack could still be performed with whitelisted requests. The attacker can perform an *Denial of Service* attack by issuing requests faster than the target can process them. By learning the frequency at which requests are normally exchanged, the proposed approach provides a second layer of protection against this type of attacks.

7.3 Future Directions

An intrinsic characteristic of SCADA systems is their close relationship with a physical process (e.g., water treatment or electricity generation). An interesting research topic not investigated in this thesis is to model the physical process controlled by the SCADA system. Traffic exchanged in SCADA networks can have an immediate impact in the physical systems, so a model that combines network traffic and physical process information might provide a fruitful source of information for anomaly detection.

Understanding the relationships between SCADA traffic and the physical process is fundamental in assessing the impact that attacks have on the physical process. For instance, in Chapter 6 we propose to generate alarms when requests are not observed after a certain period. However, making a precise estimation of the maximum period that the system can operate without these messages requires knowledge regarding the physical process. Therefore, besides supporting anomaly detection, examining the relationships between the SCADA system and the physical process can aid the development of realistic risk assessments. A first step in this direction is proposed by Cárdenas et al. [31]. In this work, the authors propose an anomaly detection mechanism that is capable of predicting the physical process response to control inputs received via the network.

In addition to protecting the SCADA system, it is important to address the security of industrial systems as a whole. An interesting challenge in this area is to understand the interdependencies between the various sectors that deploy SCADA technologies. An evident example is the dependency that various systems have in the power grid. Without electricity gas pipelines, refineries, irrigation, telecommunications, banking and other systems will halt [137]. Dependencies can also be also found within a single domain. For instance, failures in one power substation can have a cascading effect, bringing the whole power grid down [124]. The dependencies between the various components of waste

water a treatment facility have been studied in [61]. In this work, Hybrid Petri nets with a single general one-shot transition have been used to analyze the survivability of the system under a number of scenarios, such as heavy rainfall and under the failure of components (which could be cause by an attack). [61] is the first work capable of predicting the impact of failures in such a system in a quantitative way. Such analysis can provide useful insight in which security issues should be addressed with priority.

The anomaly detection approaches described in this thesis could also benefit from further research. For instance, the flow-whitelisting approach discussed in Chapter 5 could be extended to include time. For instance, traffic related to human activity might unlikely outside business hours. Also, *PeriodAnalyzer* has been tested using only two SCADA protocols, Modbus and MMS. Future research should include validating the concepts proposed in this thesis to other protocols, such as DPN3 [93] and IEC 60870-5-104 [81].

APPENDIX A

SCADA Protocols

A.1 Modbus

Initially designed by Modicon in the late 70's, Modbus [109] is a serial communication protocol used to connect industrial devices. It later became an open standard, managed by the Modbus Organization[1]. It's simplicity contributed to it's widespread adoption becoming particularly predominant in the gas and oil sectors [77].

Modbus is a master/slave protocol, that is, only one of the communicating hosts, termed master, can initiate message exchanges. For each request made by the master, the slave answers with either a normal response or an exception, indicating some error has occurred. A slave never sends a message, unless requested by its master.

More recently, Modbus has been extended to support the TCP/IP stack (Modbus TCP). Note that, even in its TCP/IP flavor, Modbus still is a master/slave protocol. In this case, the slave is termed "server". The server continuously listen for incoming TCP connections on port 502. The master is termed "client", which actively opens TCP connections. In a given TCP connection the master and slave roles are fixed, however they can be reverse in other connection between the same hosts.

The Modbus TCP header is shown in Figure A.1. It consists in the following fields:

- `transaction identifier` is used to match replies to queries.
- `protocol identifier` is always set to 0 for Modbus messages, and other values are reserved for future extensions.

[1] `http://www.modbus.org/`

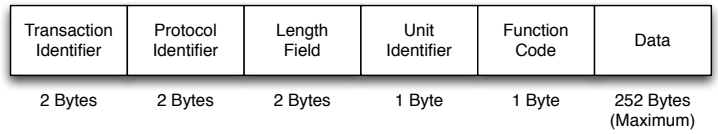


Figure A.1: Modbus protocol data unit

- `length field` is the size in bytes of all fields to its right, namely `unit identifier`, `function code` and `data`.
- `unit identifier` is typically used when for communicating with multiple serial (i.e., not TCP/IP compatible) slaves via a TCP/IP gateway. Each, so-called, "chained" slave can be addressed via its unique `unit identifier`. In case this feature is not used, the field should be set to 0.
- `function code` identifies different operations, such as read and write. The standard defines three types of function codes: (i) public function codes have their meaning defined in the standard, (ii) user-defined function codes can be used by vendors for non-standard operations, and (iii) reserved function codes are kept for legacy operations.
- `data` carries the content of the message and its specific format and size are dependent on the `function code`. For instance, a request to read a *coil* (`function code` 1) code implies a data field composed by a `starting address` field and a `quantity of coils` field, each of 2 bytes. The response for the same function code contains a `byte count` field of 1 byte and a `coil status` containing 1 byte for requested coil.

Most public function codes relate to read and write operations on the different data types defined in the standard, and to diagnostic functions. The only exception is function code 43, which relates to encapsulating other protocols over Modbus.

Modbus defines only four basic data types (Table A.1): *discrete inputs*, *coils*, *input registers* and *holding registers*. The two "input" types are read-only, while the remaining two can also be written by the client. The two "register" types are 16 bits long, while the remaining two are a single bit.

Each data type is addressed separately by a 16-bit field, allowing 65536 entries for each data type. Different function codes operate on different data

name	access	length
discrete input	read-only	1 bit
coil	read-write	1 bit
input register	read-only	16 bits
holding register	read-write	16 bits

Table A.1: Modbus data types

types. For instance, function code 1 reads coils and function code 2 read discrete inputs. Read operations are defined over a contiguous sequence of values (up to 2000 for 1-bit types or 125 for 16-bit types). To read a value, the client determines the starting address and the quantity of values that should be read. Writing multiple values is also possible, but in this case, different function types are used. For instance, function code 5 writes a single coil and function code 15 writes multiple coils. Note that the address mapping is completely vendor dependent.

Finally, we note that Modbus does not provide any security feature. Messages are exchanged in "plain text" and no authentication is performed; requests from any source will be accepted by a Modbus server.

A.2 MMS

The Manufacturing Message Specification (MMS) protocol [89] was initially developed by General Motors, as part of its Manufacturing Automating Protocol (MAP) project. ISO Technical Comitee (TC) 187 adopted the MMS specification as part of its efforts in creating a non-industry specific communication protocol, which resulted in the standard ISO/IEC 9506. This standard consists of six documents:

- *Part 1: Services*
- *Part 2: Protocols*
- *Part 3: Companion standard for robots*
- *Part 4: Companion standard for numeric control*
- *Part 5: Companion standard for programmable logic controllers*

- *Part 6: Companion standard for process control*

Part 1: Services describes the Virtual Manufacturing Device (VMD) and the messages (or services) that can be exchanged between hosts. *Part 2: Protocols* describes the communication rules, i.e., the order at which messages are exchanged and their encoding, and the interfaces with the other protocol layers. The companion standards (Parts 3 to 6) define domain specific information.

One of the central concepts of MMS is the VMD model, depicted in Figure A.2. The VMD determines how a client (e.g., a SCADA server) can interact with a server (e.g., a PLC). The VMD can be seen as an abstract description of a real industrial device, which is described by means of *objects and services*. The standard defines 15 objects, such as variables, variable type definitions, programs, files and journals (historical data); and over 80 services, including creating and deleting of objects, reading and writing variables, uploading and downloading files, and starting and stopping programs.

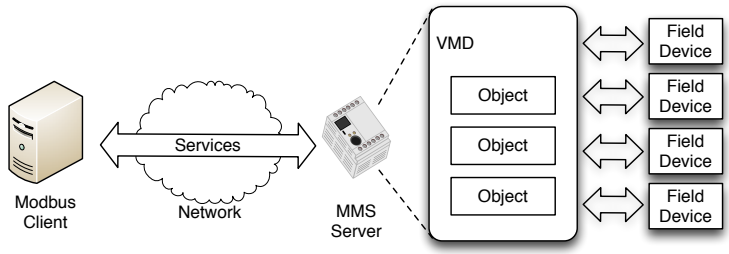


Figure A.2: MMS Communication Model

To perform these services, MMS defines fourteen basic PDUs, which can be grouped as follows:

- *Confirmed*: are messages used to start confirmed services, such as reading and writing variables and starting programs. A confirmed request implies either a response or an error as response.
- *Unconfirmed*: are services that do not require a response.
- *Reject*: are used in response to invalid messages. For instance, in the case a message exceeds the agreed maximum PDU size.
- *Cancel*: are used to abort previously sent messages.

- *Initiate*: are used to establish a MMS session between to hosts. During the initiate message exchange, the hosts can negotiate the parameters used in the connection according to their capabilities.
- *Conclude*: are used to tear-down a previously established MMS session.

With exception of *unconfirmed* and *reject*, each of these groups contain three types of PDUs: request, response and error. Like Modbus, request messages can be replied with either a response or an error message. However, differently from Modbus, MMS is not a master/slave protocol, as both sides are allowed to initiate a message exchange.

MMS uses the Abstract Syntax Notation One (ASN.1) [88] for defining its objects and services. It defines 13 basic data types, including boolean, integer and two types of floating point data. To access an object, MMS distinguishes between *named* and *unnamed* objects. The standard define three types of *named* objects:

- **VMD specific:** These object names are unique, they are always the same regardless of the client accessing then.
- **Domain specific:** A domain is an association of objects (e.g., variables, programs, files, etc...), and a VMD might contain multiple domains. A domain specific variable have a unique name within a domain, but the same name might represent another object in a different domain.
- **Application association specific:** A connection between a client and a server is termed a application association. Application association specific objects are only valid within such a connection.

Unnamed objects are accessed by an address. The specification defines three types of address:

- **Numeric:** represented by an integer (`unsigned32`).
- **Symbolic:** represented by a character string (`VisibleString`).
- **Unconstrained:** represented by a ("untyped") byte string (`OCTET STRING`).

We note that the distinction between *named* and *unnamed* refers to how an object is accessed, and not to the object itself. For example, a measurement value (e.g., the pressure of a pump) can have a numeric address (e.g., 125) and

a domain specific name (e.g., "Tank2$Pump3$Presure"). A basic difference between these types is that *named* objects can be manipulated, i.e., defined and deleted, while the address used by a *unnamed* object is fixed.

MMS defines access methods for objects, but these offer little security, if any. The access method is managed by a special object called *access control list*. When a named object is defined, it should contain an access method field. If this field contains the value *public* it is accessible, and if it contains any other value, it is not accessible. However, no mechanism is in place to prevent the creation of a new named object, which replicates all attributes of a non-public object, but changes the access method to public. Furthermore, unnamed objects always have their access method set to public. Finally, the implementation of the access control list object is optional, and it does not seem to be commonly implemented [134].

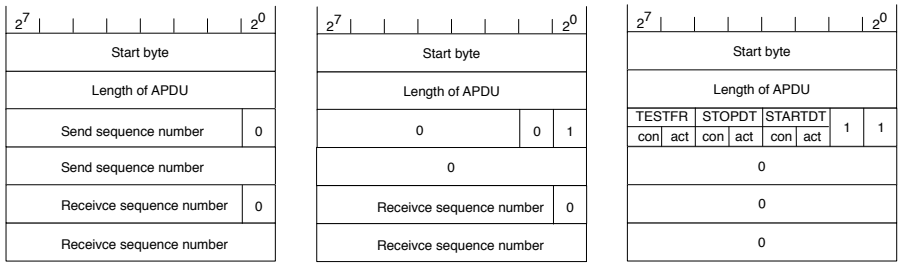
Besides the access method, MMS does not have any security service. The standard suggest that if security is a concern, it should be implemented by other layers of the protocol stack.

A.3 IEC 60870-5

The IEC standard series 60870, elaborated by TC 57, specifies SCADA systems for the electrical engineering and power systems domain. Within TC 57, working group 3 have designed the 60870-5 series [79], called "Telecontrol equipment and systems", which originally consisted of 6 documents:

- 60870-5-1: Transmission Frame Formats
- 60870-5-2: Data Link Transmission Services
- 60870-5-3: General Structure of Application Data
- 60870-5-4: Definition and Coding of Information Elements
- 60870-5-5: Basic Application Functions
- 60870-5-6: Guidelines for conformance testing for the IEC 60870-5 companion standards

These standards provide definitions for both link and application layers of the OSI model. The companion standard IEC 60870-5-101, describes the usage of the 60870-5 series for communications with an RTU over a serial link connection, and the companion standard IEC 60870-5-104, extends the application



(a) I-Format (b) S-Format (c) U-Format

Figure A.3: The three APCI formats

layer described in IEC 60870-5-101 allowing communications over a TCP/IP connection.

The PDU defined in IEC 60870-5-104 can be divided in two parts: the Application Protocol Control Information (APCI) and the Application Service Data Unit (ASDU). An APCI have three possible formats, shown in Figure A.3. The types are distinguished by the first and second bits at the third byte (termed `control octet 1`) of the APCI.

- **Information (I-format):** Contains an ASDU, which is used to send application data, and sequence numbers for both sent and received data, which are used for confirmed transmission. I-format frames are identified by "0" in the first bit of the `control octet 1`.
- **Supervisory (S-format):** Contains only a sequence number for the received data. Used for acknowledging the receipt of I-format frames. S-format frames are identified by the sequence "10" in the first two bits of the `control octet 1`.
- **Ununberd (U-format):** Are used for 3 types of connection management functions. *STARDT* is used to initiate a data transmission and *STOPDT* to end it. *TESTFR* is used to check if the communicating hosts is responding. The *act* bit represents a request and the *con* a (positive) response. U-format frames are identified by the sequence "11" in the first two bits of the `control octet 1`.

Application data is sent according to the ASDU format defined in IEC 60870-5-101 (Figure A.4). It consists of the following fields:

- `Type identification:` The type of the *information object* sent. All information objects sent in the same ASDU (up to 127) have the same format. Of the 256 possible values, 127 are reserved for "private" use, 59 are defined in the standard and the remainder is left for future extensions.
- `Variable structure qualifier:` Contains information regarding the number of *information objects* sent in the ASDU and if they refer to a single *information element* or a sequence of *information elements*.
- `Cause of transmission:` Contains information indicating the data acquisition method in use. E.g.: cyclic/periodic, spontaneous and requested.
- `Originator address:` Used in case a host is used as "concentrator" relaying messages from other hosts. In this case, the originator address is used to identify the original sender. The field is set to 0 otherwise.
- `Common address of ASDU:` Contains the "station address" of the sender. The field can also be set to 0, meaning the address is not used, or to 65535, meaning the *broadcast* address.
- `Information object address:` Identify the address of the requested object.
- `Set of information elements:` Contains the actual data. Its size depends on the *type identification*. For instance, the `type identification 1` has a single byte, `type identification 2` has 4 bytes, and the `type identification 30` has 8 bytes. All these *type identification* request a single-point information (1 byte), but using different time tags.

A time tag is a timestamp used to guarantee that the information object sent is up-to-date. The original IEC 60870-5 series defines 3-byte time time tags, encoding hour, minute and second information. IEC 60870-5-101 introduces more precise, 7-byte time tags, encoding information from years to milliseconds.

An interesting aspect of IEC 60870-5 series is the different data acquisition methods, identified by the `cause of transmission` field. Besides the general request/response method used by Modbus and MMS, which in IEC 60870-5 is termed polling, two other methods for acquiring general measurement data are defined: *cyclic data transmission* and *acquisition of events*. In the former, the RTUs are configured in a way that the measurement data is sent periodically, without the need of requests by the SCADA server. The latter, defines that certain data (termed events) should be sent if some condition is met; again, without

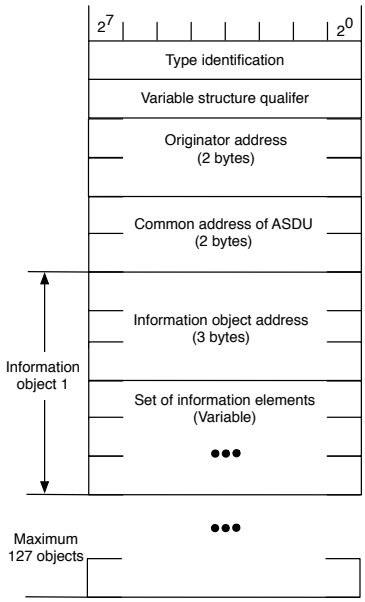


Figure A.4: ASDU format

the need of requests by the SCADA server. Special data acquisition methods are also defined for clock synchronization, integrated totals (i.e., counter-like data), file transfer and others.

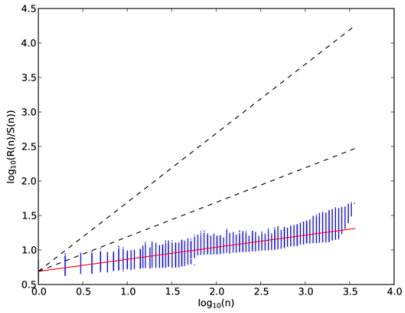
As it is the case of Modbus and MMS protocols, no security mechanisms are defined in the IEC 60870-5 standards.

APPENDIX B

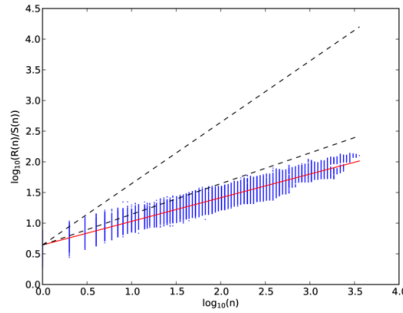
Additional Results

B.1 Applicability of Traditional Traffic Models

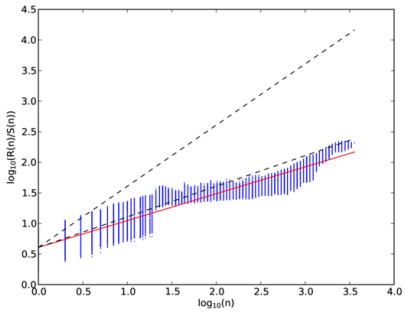
In this section, we present the results for the visual self-similarity tests omitted from Chapter 2.4.2. Figures B.1 and B.2 show the omitted R/S statistic plots, Figures B.3 and B.4 show the omitted variance-time plots, and Figure B.5 and B.6 show the omitted periodograms. The odd-numbered figures show the results for the *pkts/s* time series, while the even-numbered figures show the results for the *bytes/s* time series.



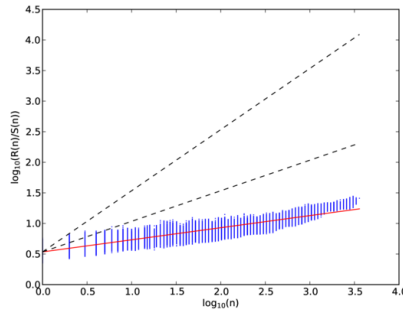
(a) SCADA1



(b) SCADA2-control

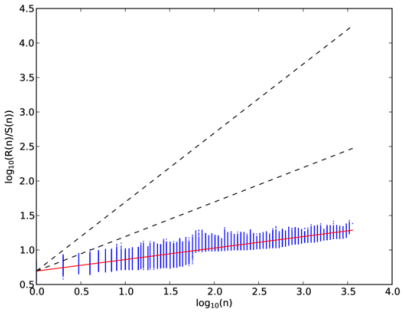


(c) SCADA2-field

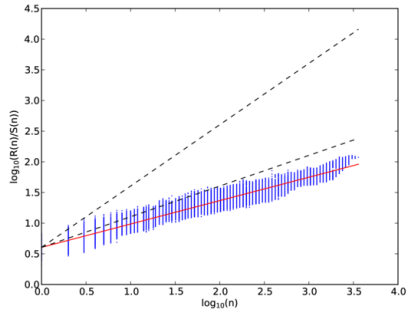


(d) SCADA3

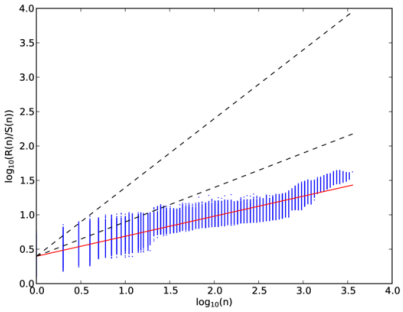
Figure B.1: R/S statistic diagrams for packet time series



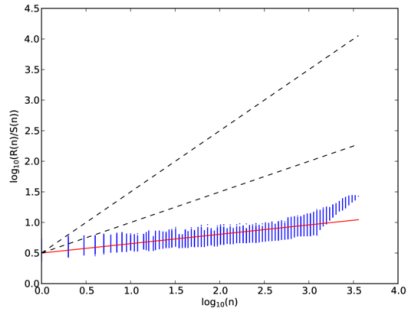
(a) SCADA1 trace)



(b) SCADA2-control

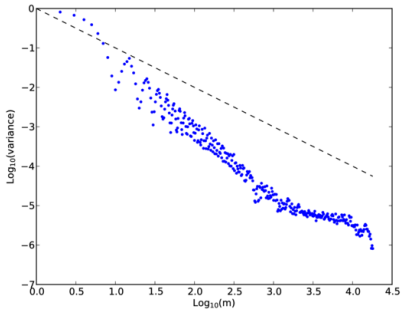


(c) SCADA2-field

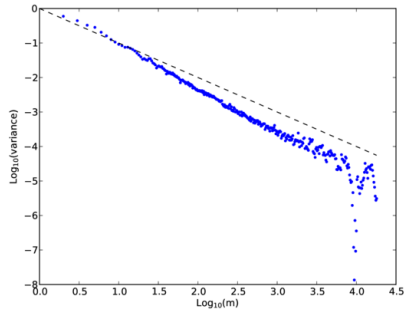


(d) SCADA3

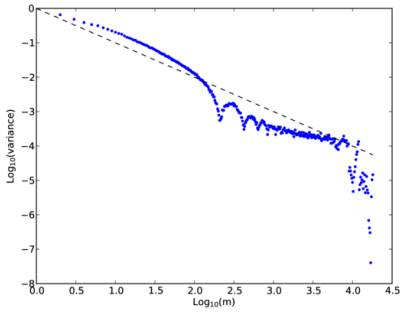
Figure B.2: R/S statistic diagrams for bytes time series



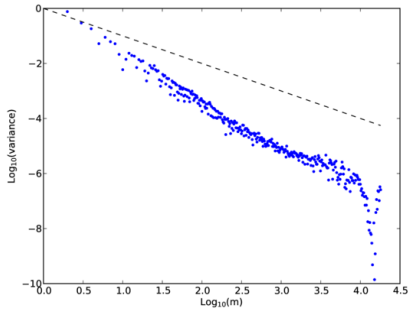
(a) SCADA1



(b) SCADA2-control

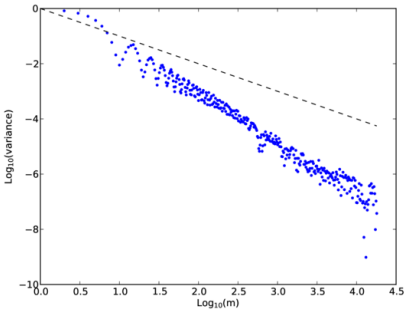


(c) SCADA2-field

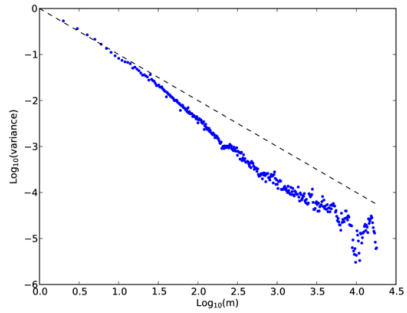


(d) SCADA3

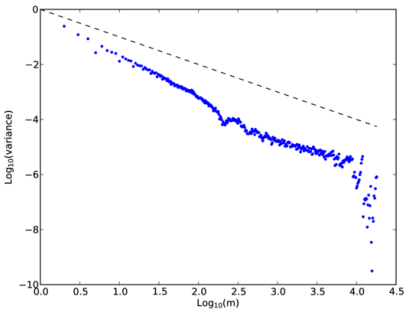
Figure B.3: Variance-time plots for packets time series



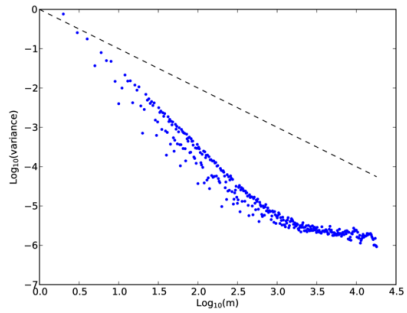
(a) SCADA1



(b) SCADA2-control

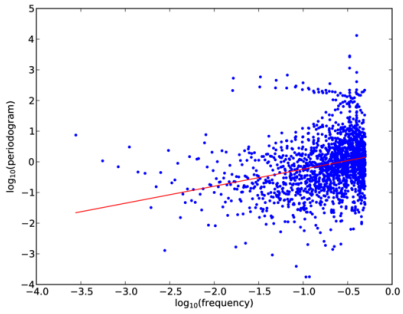


(c) SCADA2-field

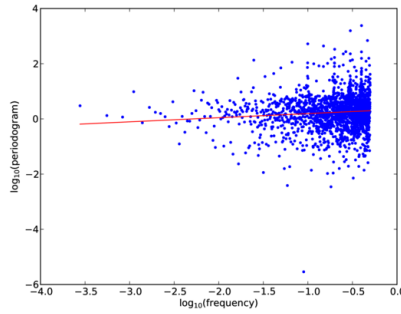


(d) SCADA3

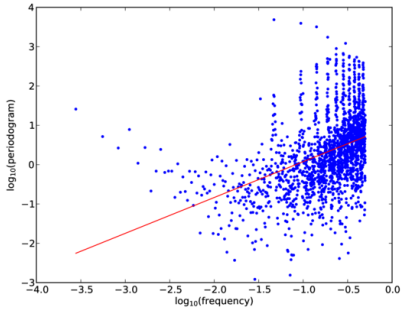
Figure B.4: Variance-time plots for bytes time series



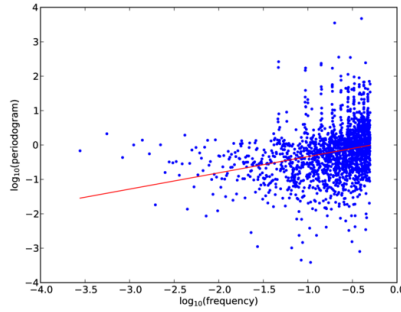
(a) SCADA1



(b) SCADA2-control

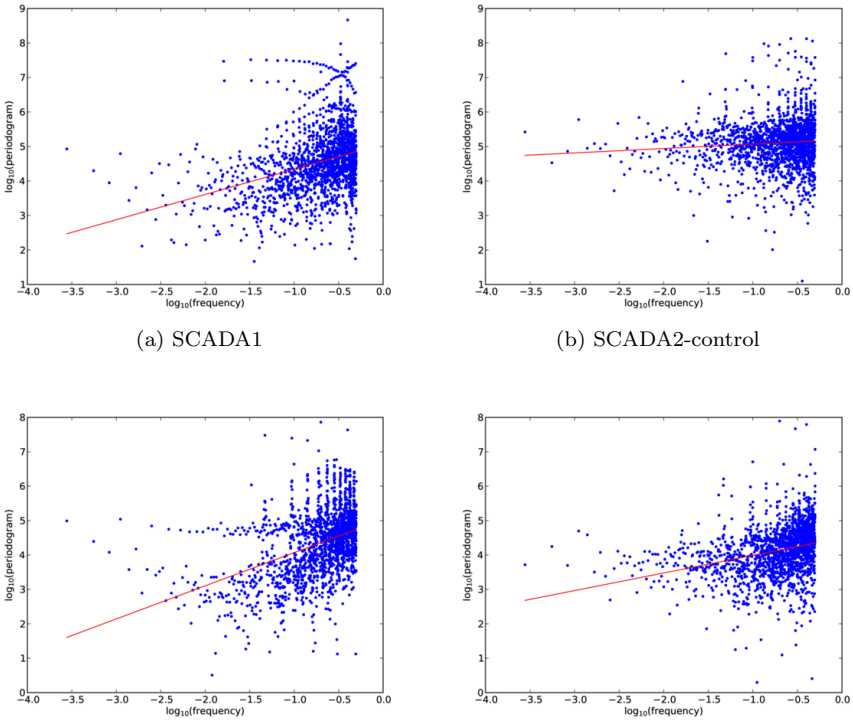


(c) SCADA2-field



(d) SCADA3

Figure B.5: Periodograms for packets time series



(a) SCADA1

(b) SCADA2-control

(c) SCADA2-field

(d) SCADA3

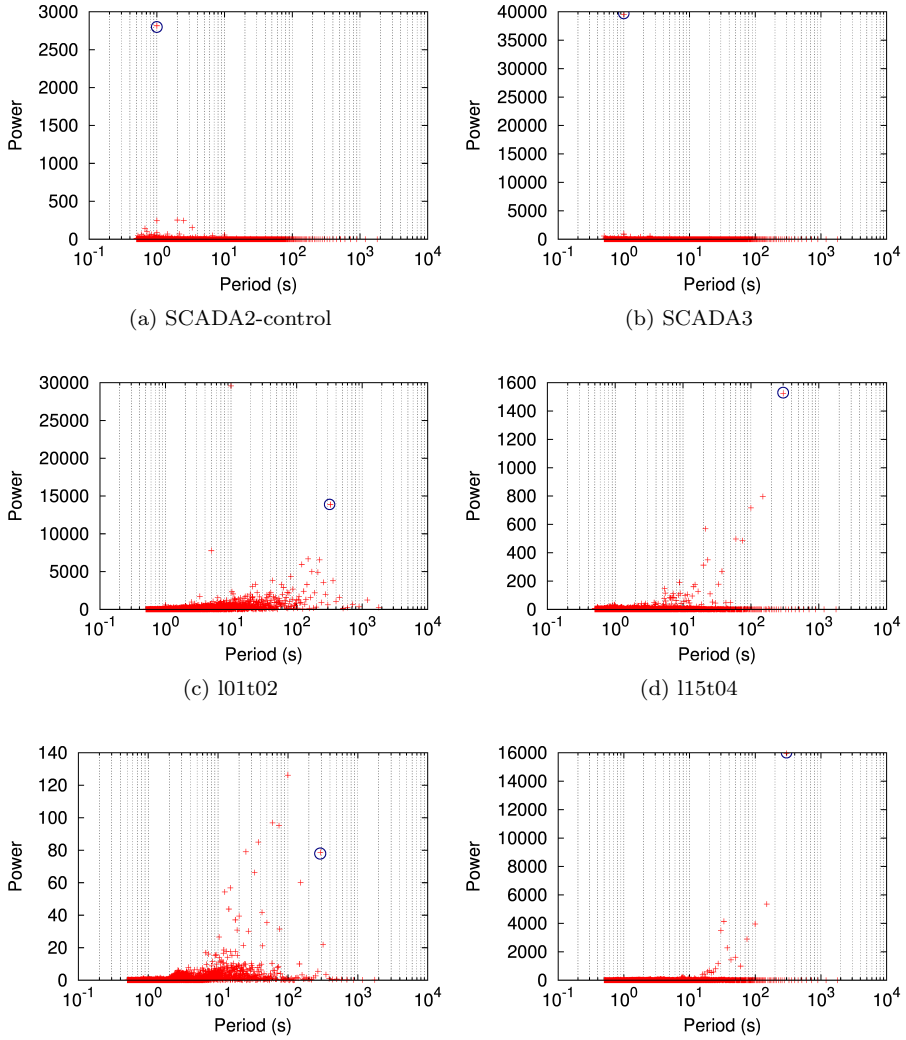
Figure B.6: Periodograms for bytes time series

B.2 SCADA Traffic Characterization

In this section, we present the periodograms omitted from the spectral analysis, presented in Chapter 3.3.2. Figure B.7 shows the results for the omitted datasets.

In Figure B.8, we show periodograms constructed using different parameters. The first row shows the periodogram from all traffic in *l05t01* in both linear (Figure B.8(a)) and logarithmic scale (Figure B.8(b)). Note that the logarithmic scale reveals a possible periodicity at 60 s, not visible at the linear scale. The second row shows a subset of 24 h from dataset *l11t01* sampled every second (Figure B.8(d)) in addition two the 1 h subset sampled using 10 ms bins (Figure B.8(c)). Note that the 300 s periodicity becomes clearer when using a larger, 24 h sample. These results further illustrate the difficulties in identifying traffic periodicities using spectral analysis methods, previously discussed in Chapter 6.

In Figure B.9 and B.10 we show the results omitted in Chapter 3.4. The former shows the current IP connections time series discussed in Chapter 3.4.1, while the later shows the number of new IP connections discussed in Chapter 3.4.3.



(a) SCADA2-control

(b) SCADA3

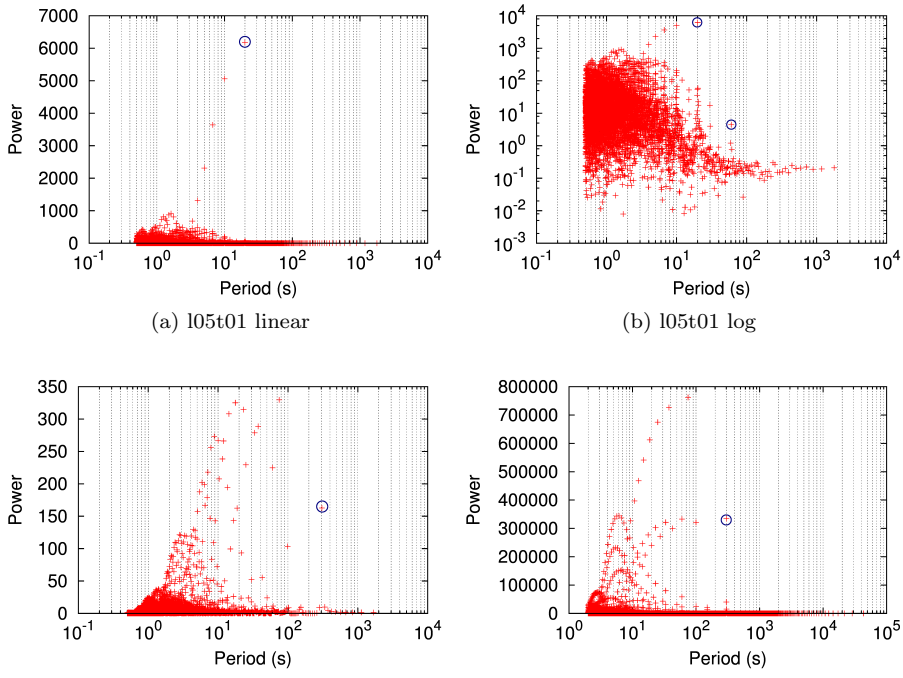
(c) l01t02

(d) l15t04

(e) l17t01

(f) l18t01

Figure B.7: Additional periodograms



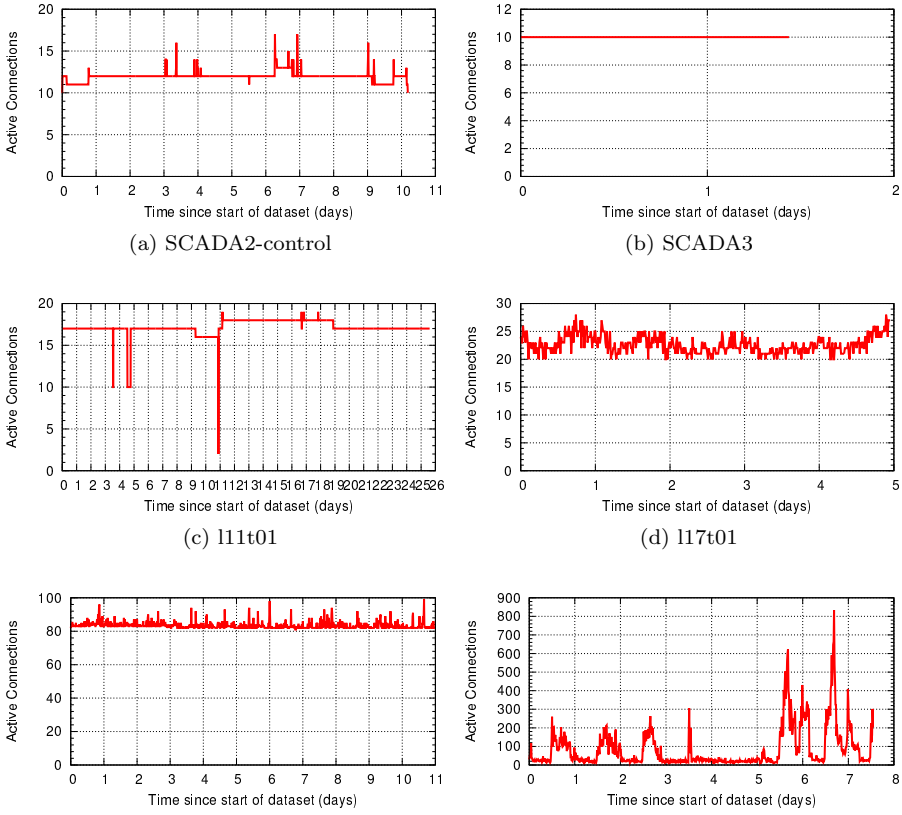
(a) l05t01 linear

(b) l05t01 log

(c) l11t01 1 h

(d) l11t01 24 h

Figure B.8: The effect of the periodogram parameters



(a) SCADA2-control

(b) SCADA3

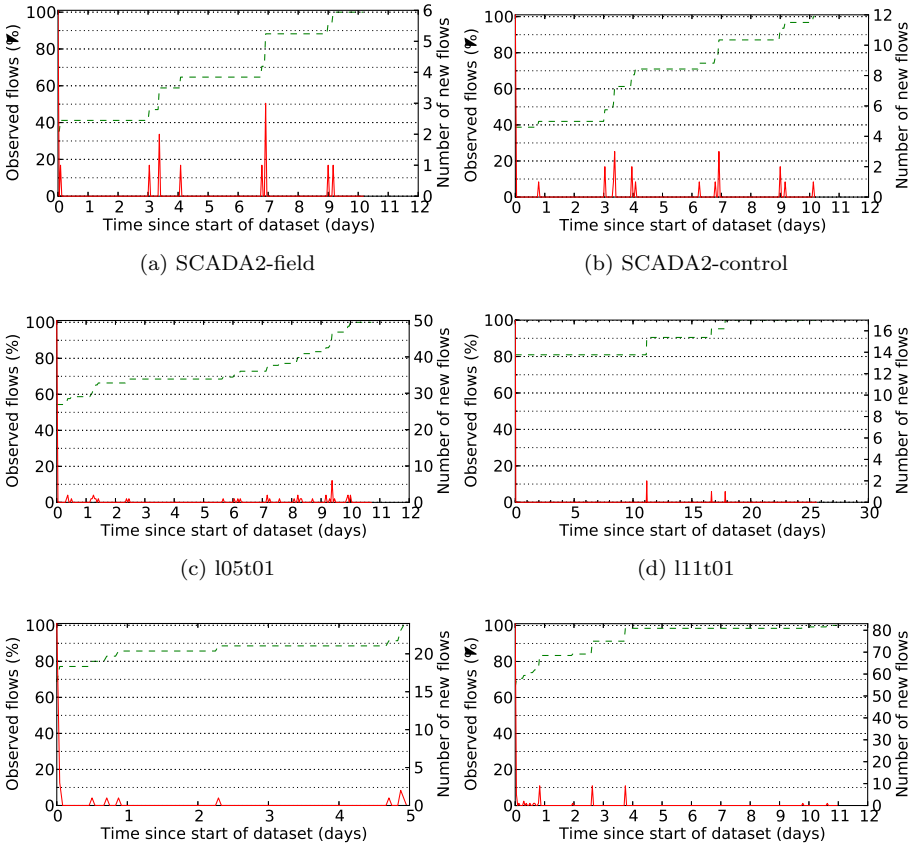
(c) l11t01

(d) l17t01

(e) l18t01

(f) IT

Figure B.9: Current IP connections



(a) SCADA2-field

(b) SCADA2-control

(c) l05t01

(d) l11t01

(e) l17t01

(f) l18t01

Figure B.10: Number of new IP connections over time

Bibliography

[1] C. L. Abad and R. I. Bonilla. An Analysis on the Schemes for Detecting and Preventing ARP Cache Poisoning Attacks. In *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*, pages 60–60. IEEE, 2007.

[2] C. Alcaraz, G. Fernandez, and F. Carvajal. Security Aspects of SCADA and DCS Environments. In Springer-Verlag, editor, *Critical Infrastructure Protection VI*, pages 120–149, Berlin, Heidelberg, 2012.

[3] S. Amin, X. Litrico, S. Sastry, and A. M. Bayen. Cyber Security of Water SCADA Systems - Part I: Analysis and Experimentation of Stealthy Deception Attacks. *IEEE Transactions on Control Systems Technology*, 21(5):1963–1970, Sept. 2013.

[4] O. Argon, Y. Shavitt, and U. Weinsberg. Inferring the Periodicity in Large-Scale Internet Measurements. In *INFOCOM, 2013 Proceedings IEEE*, pages 1–9. IEEE, 2013.

[5] S. Axelsson. Intrusion Detection Systems: A Survey and Taxonomy. Technical report, Chalmers University of Technology, 2000.

[6] E. Babeshko, V. Kharchenko, and A. Gorbenko. Applying F(I)MEA-Technique for SCADA-Based Industrial Control Systems Dependability Assessment and Ensuring. In *2008 Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX*, pages 309–315. IEEE, 2008.

[7] F. Baiardi, C. Telmon, and D. Sgandurra. Hierarchical, Model-Based Risk Management of Critical Infrastructures. *Reliability Engineering & System Safety*, 94 (9):1403–1415, Sept. 2009.

[8] D. Bailey and E. Wright. *Practical SCADA for Industry*. Newnes, 2003.

[9] M. Bailey, E. Cooke, F. Jahanian, D. Watson, and J. Nazario. The Blaster Worm: Then and Now. *IEEE Security and Privacy Magazine*, 3(4):26–31, July 2005.

[10] S. Baker, N. Filipiak, and K. Timlin. In the Dark: Crucial Industries Confront Cyberattacks. Technical report, McAfee, 2011.

[11] R. R. R. Barbosa and A. Pras. Intrusion Detection in SCADA Networks. In *Proceedings of the Mechanisms for autonomous management of networks and services, and 4th international conference on Autonomous infrastructure, management and security*, AIMS'10, pages 163–166, Berlin, Heidelberg, 2010. Springer-Verlag.

[12] R. R. R. Barbosa, R. Sadre, A. Pras, and R. van de Meent. Simpleweb/University of Twente Traffic Traces Data Repository. Technical report, Centre for Telematics and Information Technology, University of Twente, Apr. 2010.

[13] R. R. R. Barbosa, R. Sadre, and A. Pras. Towards Periodicity Based Anomaly Detection in SCADA Networks. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pages 1–4. IEEE, Sept. 2012.

[14] N. Basher, A. Mahanti, A. Mahanti, C. Williamson, and M. Arlitt. A Comparative Analysis of Web and Peer-to-Peer Traffic. In *Proceeding of the 17th international conference on World Wide Web - WWW '08*, page 287, New York, New York, USA, 2008. ACM Press.

[15] Z. Basnight, J. Butts, J. Lopez, and T. Dube. Firmware Modification Attacks on Programmable Logic Controllers. *International Journal of Critical Infrastructure Protection*, 6(2):76–84, June 2013.

[16] C. Beaumont. Stuxnet Virus: Worm 'Could be Aimed at High-Profile Iranian Targets'. `http://www.telegraph.co.uk/technology/news/8021102/Stuxnet-virus-worm-could-be-aimed-at-high-profile-Iranian-targets.html`, 2010. accessed: 2013-12-12.

[17] W. D. Beckner. NRC Information Notice 2003-14: Potential Vulnerability of Plant Computer Network to Worm Infection. Technical report, United States Nuclear Regulatory Commission, Washington DC, 2003.

[18] J. Beran. *Statistics for Long-Memory Processes*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1994.

[19] C. Berberidis, I. P. Vlahavas, W. G. Aref, M. J. Atallah, and A. K. Elmagarmid. On the Discovery of Weak Periodicities in Large Time Series. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '02, pages 51–61, London, UK, UK, 2002. Springer-Verlag.

[20] J. Bigham, D. Gamez, and N. Lu. Safeguarding SCADA Systems with Anomaly Detection. In *Second International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2003*, pages 171–182, 2003.

[21] I. F. Bo Lang. A Multipolicy Authorization Framework for Grid Security. In *Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*, pages 269–272. IEEE, 2006.

[22] S. A. Boyer. *SCADA: Supervisory Control and Data Acquisition.* ISA: The Instrumentation, Systems, and Automation Society, 4th edition, 2009.

[23] G. Broek, S. Hoeve, G. M. Moura, and A. Pras. SNMP Trace Analysis: Results of Extra Traces. Technical report, University of Twente, 2009.

[24] A. Broido and E. Nemeth. Spectroscopy of Private DNS Update Sources. In *Proceedings the Third IEEE Workshop on Internet Applications. WIAPP 2003*, pages 19–29. IEEE Comput. Soc, 2003.

[25] E. Byres, D. Leversage, and N. Kube. Security Incidents and Trends in SCADA and Process Industries. *The Industrial Ethernet Book*, 39(May):12—-20, 2007.

[26] H. Cao, D. Cheung, and N. Mamoulis. Discovering Partial Periodic Patterns in Discrete Data Sequences. In H. Dai, R. Srikant, and C. Zhang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 3056 of *Lecture Notes in Computer Science*, pages 653–658. Springer Berlin Heidelberg, 2004.

[27] Y. Cao, W. Han, and Y. Le. Anti-Phishing Based on Automated Individual White-List. In *Proceedings of the 4th ACM workshop on Digital identity management - DIM '08*, DIM '08, page 51, New York, New York, USA, 2008. ACM Press.

[28] A. Carcano and I. Fovino. State-Based Network Intrusion Detection Systems for SCADA Protocols: A Proof of Concept. In *4th International Workshop on Critical Information Infrastructures Security (CRITIS'09)*, pages 138–150, 2009.

[29] A. Carcano, A. Coletta, M. Guglielmi, M. Masera, I. Nai Fovino, and A. Trombetta. A Multidimensional Critical State Analysis for Detecting Intrusions in SCADA Systems. *IEEE Transactions on Industrial Informatics*, 7(2):179–186, May 2011.

[30] A. Cárdenas, S. Amin, S. Sastry, and A. A. C. Research Challenges for the Security of Control Systems. In *Proceedings of 3rd USENIX workshop on Hot Topics in Security (HotSec)*, 2008.

[31] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry. Attacks Against Process Control Systems: Risk Assessment, Detection, and Response. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security - ASIACCS '11*, page 355, New York, New York, USA, 2011. ACM Press.

[32] J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC 1157: Simple Network Management Protocol (SNMP), 1990.

[33] R. Chandia, J. Gonzalez, and T. Kilpatrick. Security Strategies for SCADA Networks. In *Critical Infrastructure Protection III*, volume 253, pages 117–131, 2007.

[34] M. Cheminod, L. Durante, and A. Valenzano. Review of Security Issues in Industrial Networks. *IEEE Transactions on Industrial Informatics*, 9(1):277–293, Feb. 2013.

[35] E. Y. Chen and M. Itoh. A Whitelist Approach to Protect SIP Servers from Flooding Attacks. In *2010 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR 2010)*, pages 1–6. IEEE, June 2010.

[36] S. Cherry. How Stuxnet is Rewriting the Cyberterrorism Playbook. `http://spectrum.ieee.org/podcast/telecom/security/how-stuxnet-is-rewriting-the-cyberterrorism-playbook`, 2010. accessed: 2013-12-12.

[37] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes. Using Model-Based Intrusion Detection for SCADA Networks. In *Proceedings of the SCADA Security Scientific Symposium*, volume 46, pages 1—-12. Citeseer, 2007.

[38] K. Cho, K. Fukuda, H. Esaki, and A. Kato. The Impact and Implications of the Growth in Residential User-to-User Traffic. *ACM SIGCOMM Computer Communication Review*, 36(4):207, Aug. 2006.

[39] D. Choi, S. Member, H. Kim, D. Won, S. Kim, and A. Supervisory. Advanced Key-Management Architecture for Secure SCADA Communications. *IEEE Transactions on Power Delivery*, 24(3):1154–1163, July 2009.

[40] Cisco Systems. IOS Flexible NetFlow Overview. `http://www.cisco.com/en/US/docs/ios/fnetflow/configuration/guide/fnetflow_overview.html`, 2006. Accessed: 2013-12-12.

[41] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), Jan. 2008. URL `http://www.ietf.org/rfc/rfc5101.txt`.

[42] G. Clarke, D. Reynders, and E. Wright. *Practical Modern SCADA protocols*. Newnes, 2004.

[43] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of computation*, 19(90):297–301, 1965.

[44] M. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 5(6): 835–846, 1997.

[45] S. D'Antonio, F. Oliviero, and R. Setola. High-Speed Intrusion Detection in Support of Critical Infrastructure Protection. In *First International Workshop on Critical Information Infrastructures Security (CRITIS'06)*, pages 222–234, 2006.

[46] H. Debar, M. Dacier, and A. Wespi. Towards a Taxonomy of Intrusion-Detection Systems. *Computer Networks*, 31(8):805–822, Apr. 1999.

[47] H. Debar, M. Dacier, and A. Wespi. A Revised Taxonomy for Intrusion-Detection Systems. *Annales des télécommunications*, 55(7-8):361–378, 2000.

[48] D. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, Feb. 1987.

[49] M. Di Santo, A. Vaccaro, D. Villacci, and E. Zimeo. A Distributed Architecture for Online Power Systems Security Analysis. *IEEE Transactions on Industrial Electronics*, 51(6):1238–1248, Dec. 2004.

[50] A. B. Downey. Lognormal and Pareto Distributions in the Internet. *Computer Communications*, 28(7):790–801, May 2005.

[51] P. Düssel, C. Gehl, and P. Laskov. Cyber-Critical Infrastructure Protection Using Real-Time Payload-Based Anomaly Detection. In *4th International Workshop on Critical Information Infrastructures Security (CRITIS'09)*, pages 85–97, 2010.

[52] S. East, J. Butts, M. Papa, and S. Shenoi. A Taxonomy of Attacks on the DNP3 Protocol. In *Critical Infrastructure Protection III*, pages 67–81, 2009.

[53] J. Eisenhauer, P. Donnelly, M. Ellis, and M. O' Brien. Roadmap to Secure Control Systems in the Energy Sector. Technical report, U.S. Department of Energy, 2006.

[54] D. Erickson, M. Casado, and N. McKeown. The Effectiveness of Whitelisting: a User-Study. In *The Fifth Conference on Email and Anti-Spam - CEAS '08*, page 10, Mountain View, California, USA, Aug. 2008.

[55] N. Falliere, L. O. Murchu, and E. Chien. W32. Stuxnet Dossier. Technical report, Symantic Security Response, 2011.

[56] A. Feldmann, A. C. Gilbert, W. Willinger, and T. G. Kurtz. The Changing Nature of Network Traffic. *ACM SIGCOMM Computer Communication Review*, 28(2):5–29, Apr. 1998.

[57] J. Fildes. Stuxnet Worm 'Targeted High-Value Iranian Assets'. `http://www.bbc.co.uk/news/technology-11388018`, 2010. accessed: 2013-12-12.

[58] S. Floyd and V. Paxson. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, 2001.

[59] I. N. Fovino, A. Carcano, T. D. L. Murel, A. Trombetta, and M. Masera. Modbus/DNP3 State-Based Intrusion Detection System. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 729–736. IEEE, 2010.

[60] I. n. Garitano, R. Uribeetxeberria, and U. Zurutuza. A Review of SCADA Anomaly Detection. In *Soft Computing Models in Industrial and Environmental Applications, 6th International Conference SOCO*, pages 357–366. Springer Berlin Heidelberg, 2011.

[61] H. Ghasemieh, A. Remke, and B. R. Haverkort. Analysis of a Sewage Treatment Facility Using Hybrid Petri Nets. In *7th International ICST Conference on Performance Evaluation Methodologies and Tools, Torino, Italy, December 10-12, 2013*, VALUETOOLS, pages 1–10, 2013.

[62] N. Goldenberg and A. Wool. Accurate Modeling of Modbus/TCP for Intrusion Detection in SCADA Systems. *International Journal of Critical Infrastructure Protection*, 6(2):63–75, June 2013.

[63] W.-b. Gong, Y. Liu, V. Misra, and D. Towsley. Self-Similarity and Long Range Dependence on the Internet: a Second Look at the Evidence, Origins and Implications. *Computer Networks*, 48(3):377–399, June 2005.

[64] J. Gonzalez and M. Papa. Passive Scanning in Modbus Networks. In *Critical Infrastructure Protection*, volume 253, pages 175–187, 2007.

[65] D. Goodin. Two US Power Plants Infected with Malware Spread via USB Drive. `http://arstechnica.com/security/2013/01/two-us-power-plants-infected-with-malware-spread-via-usb-drive/`, 2013.

[66] I. Grondman. Identifying Short-Term Periodicities in Internet Traffic. Master's thesis, University of Twente, 2006.

[67] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, pages 1–18, San Diego, California, USA, 2008.

[68] R. A. Guth and D. Machalaba. Computer Viruses Disrupt Railroad and Air Traffic. `http://online.wsj.com/news/articles/SB106140797740336000`, 2003. Accessed: 2013-12-07.

[69] H. Hadeli, R. Schierholz, M. Braendle, and C. Tuduce. Leveraging Determinism in Industrial Control Systems for Advanced Anomaly Detection and Reliable Security Configuration. In *2009 IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–8. IEEE, Sept. 2009.

[70] D. Hadiosmanovic, D. Bolzoni, P. Hartel, and S. Etalle. MELISSA: Towards Automated Detection of Undesirable User Actions in Critical Infrastructures. In *2011 Seventh European Conference on Computer Network Defense*, pages 41–48. IEEE, Sept. 2011.

[71] 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, Apr. 2012.

[72] J. Han and Y. Yin. Efficient Mining of Partial Periodic Patterns in Time Series Database. In *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, pages 106–115. IEEE, 1999.

[73] D. Hancock. Virus Disrupts Train Signals. `http://www.cbsnews.com/news/virus-disrupts-train-signals/`, 2003. Accessed: 2013-12-07.

[74] S. Hansman and R. Hunt. A Taxonomy of Network and Computer Attacks. *Computers & Security*, 24(1):31–43, Feb. 2005.

[75] J.-H. Hoepman and B. Jacobs. Increased Security Through Open Source. *Communications of the ACM*, 50(1):79–83, Jan. 2007.

[76] M. Hoeve. Detecting Intrusions in Encrypted Control Traffic. In *Proceedings of the first ACM workshop on Smart energy grid security - SEGS '13*, pages 23–28, New York, New York, USA, 2013. ACM Press.

[77] P. Huitsing, R. Chandia, M. Papa, and S. Shenoi. Attack Taxonomies for the Modbus Protocols. *International Journal of Critical Infrastructure Protection*, 1(C):37–44, Dec. 2008.

[78] ICS CERT. Monthly Monitor October Dececember, 2012.

[79] IEC. 60870-5, Telecontrol Equipment and Systems, 1988.

[80] IEC. 62210, Power System Control and Associated Communications - Data and Communication Security, 2003.

[81] IEC. 60870-5-104, Transmission protocols - Network Access for IEC 60870-5-101 Using Standard Transport Profiles, 2006.

[82] IEC. 62443, Industrial Communication Networks - Network and System Security - Part 1-1: Terminology, Concepts and Models, 2011.

[83] IEC. 62351, Power Systems Management and Associated Information Exchange, 2013.

[84] IEEE. Guide for Electric Power Substation Physical and Electronic Security, 2000.

[85] IEEE. 1711, Trial-Use Standard for a Cryptographic Protocol for Cyber Security of Substation Serial Links, 2011.

[86] IEEE. P1686, Draft Standard for Intelligent Electronic Devices (IEDs) Cyber Security Capabilities, 2013.

[87] V. M. Igure, S. A. Laughter, and R. D. Williams. Security Issues in SCADA Networks. *Computers & Security*, 25(7):498–506, Oct. 2006.

[88] ISO/IEC. 8824, Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), 1987.

[89] ISO/IEC. 9506, Manufacturing Message Specification, 1990.

[90] ISO/IEC. 15408-1, Evaluation Criteria for IT Security - Part 1: Introduction and General Model, 2009.

[91] ISO/IEC. 27000, Information Technology - Security Techniques, 2009.

[92] S. Karnouskos and A. W. Colombo. Architecting the Next Generation of Service-Based SCADA/DCS System of Systems. In *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, pages 359–364. IEEE, Nov. 2011.

[93] C. Ken. A DNP3 Protocol Primer. Technical report, DNP Users Group, 2000.

[94] A. Khelil, D. Germanus, and N. Suri. Protection of SCADA Communication Channels. In J. Lopez, R. Setola, and S. D. Wolthusen, editors, *Critical Infrastructure Protection VI*, pages 177–196. Springer, 2012.

[95] T. H. Kobayashi, A. B. Batista, A. M. Brito, and P. S. M. Pires. Using a Packet Manipulation Tool for Security Analysis of Industrial Network Protocols. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 744–747, 2007.

[96] R. Krutz. *Securing SCADA Systems*. Wiley, 2005.

[97] S. Laxman and P. S. Sastry. A Survey of Temporal Data Mining. *Sadhana*, 31 (2):173–198, Apr. 2006.

[98] W. E. Leland, W. Willinger, M. S. Taqqu, and D. V. Wilson. On the Self-Similar Nature of Ethernet Traffic. *ACM SIGCOMM Computer Communication Review*, 25(1):202–213, Jan. 1995.

[99] O. Linda, T. Vollmer, and M. Manic. Neural Network Based Intrusion Detection System for Critical Infrastructures. In *2009 international joint conference on neural networks*, pages 1827–1834. ieee, June 2009.

[100] E. Lloyd and D. Warren. The Historically Adjusted Range and the Historically Rescaled Adjusted Range. *Stochastic Hydrology and Hydraulics*, 2(3):175–188, 1988.

[101] P. Loiseau, P. Gonçalves, G. Dewaele, P. Borgnat, P. Abry, and P. V.-B. Primet. Investigating Self-Similarity and Heavy-Tailed Distributions on a Large-Scale Experimental Facility. *IEEE/ACM Transactions on Networking*, 18(4):1261–1274, Aug. 2010.

[102] E. Luiijf. SCADA Security Good Practices for the Drinking Water Sector. Technical report, TNO, 2008.

[103] T. F. Lunt. A Survey of Intrusion Detection Techniques. *Computers & Security*, 12(4):405–418, June 1993.

[104] R. T. Marsh, J. R. Powers, and M. E. Adams. Critical Foundations: Protecting America's Infrastructures. Report of the president's commission on critical infrastructure protection. Technical report, President's Commission on Critical Infrastructure Protection, 1997.

[105] T. McEvoy and S. Wolthusen. Detecting Sensor Signal Manipulations in Nonlinear Chemical Processes. In *Critical Infrastructure Protection IV*, pages 81–94, 2010.

[106] M. Meier and T. Holz. Intrusion Detection Systems List and Bibliography. `https://www-rnks.informatik.tu-cottbus.de/en/node/209`, 2001.

[107] Microsoft. Service Overview and Network Port Requirements for Windows.

[108] J. Mikel. The 2003 Northeast Blackout–Five Years Later: Scientific American, 2008.

[109] Modbus Organization. Modbus Application Protocol Specification V1.1b3, Aril 2012.

[110] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. *IEEE Security & Privacy Magazine*, 1(4):33–39, July 2003.

[111] B. Mukherjee, L. Heberlein, and K. Levitt. Network Intrusion Detection. *IEEE Network*, 8(3):26–41, May 1994.

[112] North American Electric Reliability Corporation. Guidance for Enforcement of CIP Standards. Technical report, North American Electric Reliability Corporation, 2008.

[113] Norwegian Oil and Gas association. 104 - Recommended Guidelines for Information Security Baseline Requirements for Process Control, Safety and Support ICT Systems. Technical report, Norwegian Oil and Gas Association, 2009.

[114] C. Nuzman, I. Saniee, W. Sweldens, and A. Weiss. A Compound Model for TCP Connection Arrivals for LAN and WAN Applications. *Computer Networks*, 40 (3):319–337, Oct. 2002.

[115] T. Oetiker. MRTG - The Multi Router Traffic Grapher. In *Proceedings of the 12th USENIX conference on System administration*, pages 141–148, Berkeley, CA, USA, 1998. USENIX Association.

[116] P. Oman and M. Phillips. Intrusion Detection and Event Monitoring in SCADA Networks. In *Critical Infrastructure Protection*, volume 253, pages 161–173, 2007.

[117] K. Park and W. Willinger. *Self-Similar Network Traffic and Performance Evaluation*. Wiley Online Library, 2000.

[118] V. Paxson. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, Dec. 1999.

[119] V. Paxson and S. Floyd. Wide Area Traffic: the Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[120] S. Peerlkamp and M. Nieuwenhuis. Process Control Network Security. Technical Report 1689428, Vrije Universiteit Amsterdam, 2010.

[121] L. Pi and P. Sitbon. Cryptographic Key Management for SCADA Systems-Issues and Perspectives. In *2008 International Conference on Information Security and Assurance (isa 2008)*, pages 156–161. IEEE, Apr. 2008.

[122] U. K. Premaratne, J. Samarabandu, T. S. Sidhu, R. Beresh, and J.-C. Tan. An Intrusion Detection System for IEC61850 Automated Substations. *IEEE Transactions on Power Delivery*, 25(4):2376–2383, Oct. 2010.

[123] P. a. S. Ralston, J. H. Graham, and J. L. Hieb. Cyber Security Risk Assessment for SCADA and DCS Networks. *ISA transactions*, 46(4):583–94, Oct. 2007.

[124] S. Rinaldi, J. Peerenboom, and T. Kelly. Identifying, Understanding, and Analyzing Critical Infrastructure Interdependencies. *IEEE Control Systems Magazine*, 21(6):11–25, 2001.

[125] J. Rrushi and R. Campbell. Detecting Cyber Attacks on Nuclear Power Plants. In *Critical Infrastructure Protection II*, volume 290, pages 41–54, 2009.

[126] J. Rrushi and K. Kang. Detecting Anomalies in Process Control Networks. In *Critical Infrastructure Protection III*, pages 151–165, 2009.

[127] R. Sadre and B. R. Haverkort. Changes in the Web from 2000 to 2007. In *Managing Large-Scale Service Deployment. Proceedings of the 19th IFIP/IEEE International Workshop on Distributed Systems (DSOM 2008)*, volume 5273 of *LNCS*, pages 136–148. Springer, 2008.

[128] D. E. Sanger. Obama Order Sped Up Wave of Cyberattacks Against Iran. `http://www.nytimes.com/2012/06/01/world/middleeast/obama-ordered-wave-of-cyberattacks-against-iran.html`, 2012. Accessed: 2013-12-11.

[129] J. Schönwälder, A. Pras, M. Harvan, J. Schippers, and R. van de Meent. SNMP Traffic Analysis: Approaches, Tools, and First Results. In *10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 323–332. IEEE, May 2007.

[130] G. Serazzi and S. Zanero. Computer Virus Propagation Models. In M. Calzarossa and E. Gelenbe, editors, *Performance Tools and Applications to Networked Systems*, volume 2965 of *Lecture Notes in Computer Science*, pages 26–50. Springer Berlin Heidelberg, 2004.

[131] J. Slay and M. Miller. Lessons Learned from the Maroochy Water Breach. In *Critical Infrastructure Protection II*, volume 253, page 73. Springer, 2008.

[132] R. Sommer and A. Feldmann. NetFlow: Information Loss or Win? In *Proceedings of the second ACM SIGCOMM Workshop on Internet measurment - IMW '02*, page 173, New York, New York, USA, 2002. ACM Press.

[133] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316. IEEE, 2010.

[134] J. T. Sørensen. Security in Industrial Networks. Technical report, Norwegian University of Science and Technology, 2007.

[135] A. Sperotto. *Flow-Based Intrusion Detection*. PhD thesis, University of Twente, Enschede, The Netherlands, Oct. 2010.

[136] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An Overview of IP Flow-Based Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010.

[137] K. A. Stouffer, J. A. Falco, K. A. Scarfone, and K. Kent. SP 800-82: Guide to Industrial Control Systems (ICS) Security. Technical report, NIST, Gaithersburg, MD, United States, 2013.

[138] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2002.

[139] A. S. Tanenbaum and M. Van Steen. *Distributed Systems*, volume 2. Prentice Hall, 2002.

[140] C.-W. Ten, G. Manimaran, and C.-C. Liu. Cybersecurity for Critical Infrastructures: Attack and Defense Modeling. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 40(4):853–865, July 2010.

[141] H. Tipton. *Information Security Management Handbook*. CRC Press, Inc., Boca Raton, FL, USA, 5 edition, 2003.

[142] B. Trammell and E. Boschi. Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103 (Proposed Standard), Jan. 2008. URL `http://www.ietf.org/rfc/rfc5103.txt`.

[143] P. P. Tsang and S. W. Smith. YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems. In *Proceedings of The IFIP TC11 23rd International Information Security Conference*, volume 278, pages 445–459, Milano, Italy, 2008. Springer US.

[144] A. Valdes and S. Cheung. Intrusion Monitoring in Process Control Systems. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–7. IEEE, 2009.

[145] A. Valdes and S. Cheung. Communication Pattern Anomaly Detection in Process Control Systems. In *2009 IEEE Conference on Technologies for Homeland Security*, pages 22–29. IEEE, IEEE, May 2009.

[146] J. van den Broek. Periodicity of SNMP Traffic, 2007.

[147] A. Vázquez, R. Pastor-Satorras, and A. Vespignani. Large-Scale Topological and Dynamical Properties of the Internet. *Physical Review E*, 65(6):066130, June 2002.

[148] M. Vlachos, P. S. Yu, V. Castelli, and C. Meek. Structural Periodic Measures for Time-Series Data. *Data Mining and Knowledge Discovery*, 12(1):1–28, Feb. 2006.

[149] Y. Wang. sSCADA: Securing SCADA Infrastructure Communications. *International Journal of Communication Networks and Distributed Systems*, 6(1):59, 2011.

[150] A. K. Wright, J. A. Kinast, and J. McCarty. Low-Latency Cryptographic Protection for SCADA Communications. In *Second International Conference on Applied Cryptography and Network Security (ACNS)*, pages 263–277, Yellow Mountain, China, 2004.

[151] K. Xiao, N. Chen, S. Ren, L. Shen, X. Sun, K. Kwiat, and M. Macalik. A Workflow-Based Non-intrusive Approach for Enhancing the Survivability of Critical Infrastructures in Cyber Environment. In *Third International Workshop on Software Engineering for Secure Systems (SESS'07: ICSE Workshops 2007)*, pages 4–4. IEEE, May 2007.

[152] D. Yang, A. Usynin, and J. Hines. Anomaly-Based Intrusion Detection for SCADA Systems. In *5th Intl. Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC&HMIT 05)*, pages 12–16, 2005.

[153] M. Yoon. Using Whitelisting to Mitigate DDoS Attacks on Critical Internet Sites. *IEEE Communications Magazine*, 48(7):110–115, July 2010.

[154] J. Zhang and A. Moore. Traffic Trace Artifacts due to Monitoring Via Port Mirroring. In *2007 Workshop on End-to-End Monitoring Techniques and Services*, pages 1–8. IEEE, May 2007.

[155] G. Ziemba, D. Reed, and P. Traina. Security Considerations for IP Fragment Filtering. RFC 1858 (Proposed Standard), Oct. 1995. URL `http://www.ietf.org/rfc/rfc1858.txt`.

[156] C. C. Zou, W. Gong, and D. Towsley. Worm Propagation Modeling and Analysis Under Dynamic Quarantine Defense. In *Proceedings of the 2003 ACM workshop on Rapid Malcode - WORM'03*, page 51, New York, New York, USA, 2003. ACM Press.

Acronyms

ACF AutoCorrelation Function.

BITW Bump-In-The-Wire.

COTS Commercial Off-The-Shelf.
CPS Cyber-Physical System.

DCS Distributed Control System.
DFA Deterministic Finite Automaton.
DFT Discrete Fourier Transform.
DHS Department of Homeland Security.
DoS Denial of Service.
DPI Deep Packet Inspection.

FFT Fast Fourier Transform.

HMI Humam-Machine Interface.

IACS Industrial Automation Control Network.
ICS Industrial Control System.
IDS Intrusion Detection System.
IEC International Electrotechnical Commission.
IT Information Technology.

LAN Local Area Network.
LRD Long Range Dependent.

MMS Manufacturing Message Specification.
MTU Master Terminal Unit.

NCS Networked Control System.

NERC North American Electric Reliability Coorporation.

PCN Process Control Network.

PCS Process Control System.

PDU Protocol Data Unit.

PLC Programmable Logic Controller.

RTU Remote Terminal Unit.

SCADA Supervisory Control And Data Acquisition.

SNMP Simple Network Management Protocol.

STFT Short-Term Fourier Transform.

WAN Wide Area Network.

About the author

Rafael Ramos Regis Barbosa was born in Vila Velha, Espírito Santo, Brazil on November 19th, 1983. In 2006, he received his Bachelor of Science (B.Sc.) degree in Computer Science at the Federal University of Espírito Santo (UFES). After graduating, he was accepted with a full scholarship for a master program in Computer Science at the State University of Campinas (UNICAMP), where he enrolled in 2007. However, just 6 months after starting the program, he decide to move to the Netherlands, where he had been awarded a scholarship to pursue a master degree in Telematics at the University of Twente. After achieving his Master of Science (M.Sc.) degree, he enrolled as a Ph.D. candidate at the Design and Analysis of Communications Systems (DACS) group at the University of Twente. In 2013, he is started working as a researcher at the European Network for Cyber Security (ENCS) in The Hague.

A list of his publications in reverse chronological order:

- R. R. R. Barbosa, R. Sadre, and A. Pras. *Flow Whitelisting in SCADA Networks*. International Journal of Critical Infrastructure Protection 6(3-4):150–158, Dec. 2013.
- R. R. R. Barbosa, R. Sadre, and A. Pras. *Towards Periodicity Based Anomaly Detection in SCADA Networks*. In Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation, pages 1–4. IEEE, Sept. 2012.
- R. R. R. Barbosa, R. Sadre, and A. Pras. *A First Look into SCADA Network Traffic*. In 2012 IEEE Network Operations and Management Symposium volume 17, pages 518–521. Springer, IEEE, Apr. 2012.
- R. R. R. Barbosa, R. Sadre, and A. Pras. *Difficulties in Modeling SCADA Traffic: A Comparative Analysis*. Passive and Active Measurement: 13th International Conference, Pam 2012, Vienna, Austria, March 12-14, 2012.
- R. S. Schwartz, R. R. R. Barbosa N. Meratnia, G. Heijenk, and H. Scholten. *A Directional Data Dissemination Protocol for Vehicular Environments*. Computer Communications 34(17):2057–2071, Nov. 2011.

- I. Drago, R. R. R. Barbosa, R. Sadre, A. Pras, and J. Schönwälder. *Report of the Second Workshop on the Usage of NetFlow/IPFIX in Network Management.* Journal of Network and Systems Management 19(2):298–304, 2011.
- A. Pras, A. Sperotto, G. C. Moura, I. Drago, R. R. R. Barbosa, R. Sadre, R. de O. Schmidt, and R. Hofstede. *Attacks by "Anonymous" WikiLeaks Proponents not Anonymous.* Technical Report TR-CTIT-10-41, CTIT, University of Twente 2010.
- R. R. R. Barbosa, R. Sadre, A. Pras, and R. Meent. *Simpleweb/University of Twente Traffic Traces Data Repository.* Technical Report TR-CTIT-10-19, CTIT, Univeristity of Twente, Apr. 2010.
- R. S. Schwartz, R. R. R. Barbosa, N. Meratnia, G. Heijenk, and H. Scholten. *A Simple and Robust Dissemination Protocol for VANETs.* In 2010 European Wireless Conference, pages 214–222. IEEE, 2010.
- R. R. R. Barbosa and A. Pras. *Intrusion Detection in SCADA Networks.* In Proceedings of the 4th International Conference on Autonomous Infrastructure, Management and Security, pages 163–166. Springer, 2010.