Radboud University Nijmegen

TenneT

Effective Host-based Intrusion Detection for Real-Time Industrial Control Systems

Pol Van Aubel

29-09-2013

Master's Thesis

Radboud University Nijmegen
Institute for Computing and Information Sciences
Graduation nr. 673

Supervisor
Dr. Jaap-Henk Hoepman
Radboud University Nijmegen
jhh@cs.ru.nl

Second reader
Prof. dr. Bart Jacobs
Radboud University Nijmegen
bart@cs.ru.nl

TenneT supervisor
Jos Weyers
TenneT TSO B.V.
jos.weyers@tennet.eu

Foreword

This Master's thesis is the product of six months of research and discussions with many people within TenneT TSO B.V., N.V. Nederlandse Gasunie, Alliander N.V., Locamation B.V., SecurityMatters B.V., and the Radboud University Nijmegen. I consider myself lucky to have received so much support and information from within the energy sector. Without it, this work would not have been possible.

There are several people I would like to thank in particular. First, my university supervisor, Jaap-Henk Hoepman, for his input and his advice on writing this thesis. My supervisor at TenneT, Jos Weyers, for the discussions on electrical grid security and for insights in when security is secure enough. Hopefully we will not need the epoxy kit. All my other colleagues at TenneT, who are too numerous to mention, for the discussions and insights into the electrical grid. Paul Bloemen and Ton Lenting at Gasunie, for the discussions we had on their SCADA and intrusion detection systems. Frank Baldinger and Saghar Khadem at Locamation, for the very open discussion on the substation management systems. Damiano Bolzoni, for several ideas on effective intrusion detection techniques. Ed Schouten, for the discussions on FreeBSD and Linux kernels and overhead. And finally, Bart Jacobs, for getting me in contact with TenneT in the first place.

Although this thesis was created in collaboration with TenneT TSO B.V. and other companies, the views expressed within are my own and do not necessarily represent the official views of these other parties.

Summary

The digital threat to industrial control systems is no longer underestimated. Especially in the vital sector of energy supply, a lot of effort is put into enhancing security. One of the ways in which the industrial control systems of the electrical grid can be secured is by adding a host-based intrusion detection system. However, since these industrial control systems are specialized real-time systems, this is not as simple as installing an existing solution. In this thesis, we propose a possible architecture for an effective host-based intrusion detection system for these industrial control systems.

The electrical grid is made up of several different types of real-time systems which control the day-to-day operation of the grid. This includes performing measurements, executing commands from operators, and ensuring grid safety in abnormal situations such as a short circuit. The category in which a real-time system belongs dictates what a host-based intrusion detection system can do. For example, on a system on which only limited actions are available to the system administrator (black-box or grey-box), a host-based intrusion detection system might only be able to collect aggregate memory usage information, whereas on a system where the administrator has full access and is free to observe system state (white-box), an intrusion detection system could keep track of system calls performed by each process.

The most important threats to the electrical grid are grid disruption, physical damage, loss of life, and (corporate) espionage. Attack vectors include installation of unsanctioned hardware, infection by viruses or other malicious software, and direct access to user accounts. We propose a solution which is able to detect abuse of most, if not all, of these vectors.

Intrusion detection systems can be categorized as signature-based and anomaly-based systems. The strengths of signature-based systems lie in detecting known attacks, whereas anomaly-based systems are able to detect new and targeted attacks. Both approaches have merits, and our solution proposes a combination of the two. Intrusion detection systems can further be classified based on whether they are network- or host-based, their detection time, and their deployment structure. There are stand-alone systems, distributed systems, mobile systems, and collaborative systems. Collaborative systems utilize multiple intrusion detection systems, possibly in a hierarchical structure, to provide a complete view of a network of targets.

We analyse the possible effects that adding an intrusion detection system may have on real-time systems. We first look at the influence extra utilization of system resources might have on the existing processes, especially in the case of exclusive access to shared resources. To model this, we propose an extension on an existing process algebra, ACSR, and demonstrate its use through a fictitious model of a

grid safety system. We also examine the overhead that system call tracing adds to a process, as well as the overhead that collection of other bookkeeping information might induce.

Finally, based on all this information, we propose a collaborative, hierarchical intrusion detection architecture in which the detection engines run on dedicated machines and the data is collected by collection processes on each host. This minimizes the impact of the system on the hosts being monitored. The detection engines we propose use both anomaly detection based on machine learning, statistical models, and hard limits on different metrics, as well as signature detection on system call payloads and files on the filesystem.

Unfortunately, many systems in the electrical grid are black- or grey-box, limiting the possibilities of testing such a system without vendor support. We propose several avenues of future research, including ways in which both detection accuracy as well as impact on real-time constraints can be tested. Vendor support is required for this. Finally, an interesting possibility of using the physical properties of the electrical grid as an anomaly detection system, e.g. the ways in which certain measurements are related, is described as a direction of research which should be explored.

Contents

1 Introduction

1.1 Critical Infrastructure

Modern society and its economy depend, in large part, on a number of vital products and services provided by government and commercial entities. These services are essential for the daily lives of many people, and are therefore called the critical infrastructures. The Dutch government has divided the critical infrastructures in 12 sectors providing these vital services, which encompass 31 vital products and services. Health care, transportation, public government, public order, food supply, and drinking water are all examples of these vital services. They, in turn, are in large part dependent on one vital sector: energy. The energy sector consists of electricity, natural gas, and oil[51].

The distribution grids used for these resources are controlled by industrial control systems (ICS), in particular supervisory control and data acquisition (SCADA) systems. These systems report the state of remote stations, such as gas processing plants or high-voltage stations, to a controller which can in turn send commands back to those remote stations in order to control them. E.g. a SCADA system in a high-voltage station reports that a line in the electricity distribution grid is under heavy load, after which a controller sends the command to start using another line. As a result, at the high-voltage station, another system commands the physical devices used to control the lines to switch on the selected line.

Continuity of the vital sectors is essential to prevent a breakdown of social order. For the electricity sector, the Dutch Transmission System Operator (TSO) TenneT manages the electricity transmission grid and is therefore concerned with protecting against grid disruptions. Electrical grids could be disrupted if e.g. the systems controlling them fail or are controlled by malicious parties.

1.2 Industrial Control Vulnerabilities

The discovery in 2010 of the Stuxnet worm and its later derivatives has been called many things, but above all it can be seen as a wake-up call. Since then, the rate at which vulnerabilities in industrial control systems, in particular SCADA systems, are published has increased significantly. Likewise, the way in which the industry creating and using these systems handles vulnerabilities has improved, with most vulnerabilities being fixed within 30 days[32].

Finding and fixing vulnerabilities in ICSs is just one aspect of improving their security. A fairly recent development is the creation of network intrusion detection systems (NIDS) specifically tailored to SCADA systems[13, 75, 47]. These systems monitor the network communication to and from the SCADA systems. They detect specific attacks by looking for known attack patterns (signature based detection), or look for deviations from normal communication (anomaly detection). When an

intrusion is suspected, the administrator of the system is notified and appropriate steps can be taken. Depending on their placement, however, these systems do not monitor all possible lines of communication to or from a SCADA system. On top of that, ports which are only used for maintenance, or not supposed to be used at all, are mostly left unmonitored.

A sophisticated attacker could try to exploit these unmonitored paths, e.g. by using social engineering techniques to trick technicians maintaining these systems to install malicious software or open new communication channels with the outside world. Another example is a technician who would rather do his maintenance remotely, and does so via an unauthorized uplink to the global internet (e.g. via the mobile phone network), exposing the system to a whole new range of targeted and untargeted attacks. ICSs used to be custom-built with specialized hardware for their intended purpose. However, there is a trend to use more commercial off-the-shelf technology. Therefore, these systems might become vulnerable to some of the same attacks that target more conventional IT systems.

1.3 Host-based Intrusion Detection

One of the most widely used ways, in conventional IT, to detect this kind of direct attack on a system is by using a host-based intrusion detection system (HIDS). There are many parallels between HIDSs and NIDSs, both in what they monitor as in how they detect intrusions. The main differences are that an HIDS can monitor more than network traffic, e.g. whether the executable files on a system are unmodified from a known installation base; and that an HIDS runs, at least partly, on the system it is monitoring[55].

Intrusion prevention systems (IPS) take the principle of intrusion detection one step further. These systems are designed to block behaviour which is deemed a suspected intrusion, instead of simply alerting an administrator. An anti-virus system is one example of an IPS, as it is designed to block the execution of malicious code, instead of simply detecting it.

When considered in the light of SCADA systems, and in particular the control systems used in the energy transmission grid, two problems with the host-based approach become apparent:

- A SCADA system must not cease to operate when a possible intrusion is detected, and
- SCADA systems can have hard real-time constraints.

To illustrate why the first problem is relevant, consider the track-record of virus-scanners. A single false positive which is quarantined or blocked by a virus scanner has the potential of crippling a certain piece of software, or even the entire operating system[16, 21, 45, 74]. Furthermore, even when an actual intrusion is

detected, a system must not simply cease operation. The decision to take a system offline, or to end an infected process, can influence the functioning of the grid. For systems which control the functioning and safety of the electrical grid, such an incident must be prevented. An IDS does not have this problem, but for an IPS this must be taken into consideration.

The second problem stems from the fact that SCADA systems perform roles which are bound by time-constraints, making them real-time systems. Furthermore, their soft- and hardware is often tailored to their operation. Adding an HIDS to a system may alter the timing characteristics of the system. In some cases this might be negligible and acceptable. However, in the case of systems which control safety measures such as power cut-off in the case of a short-circuit to ground, there are hard real-time constraints on the system which must be met, regardless of the presence of an HIDS.

1.4 Research Questions

Considering all this, we are interested in exploring the possibilities of host-based intrusion detection on the electrical grid, taking into consideration the real-time constraints of the systems in use. This leads us to the following research question:

> What is an effective design for a host-based intrusion detection scheme for real-time systems as used in an electrical grid?

Effective, in this case, means that the HIDS has a high detection rate with a low false positive and negative rate, without violating the constraints of the real-time system or requiring a significant hardware upgrade which could increase the cost of the systems to an unacceptable level.

To answer this question, we will explore the following subproblems:

1. *What kind of real-time systems exist within electrical transport operators and the gas distribution operators?*
2. *Which constraints exist for these real-time systems?*
3. *What kind of threats do these systems face in the context of a TSO?*
4. *Which intrusion detection approaches exist? What are their respective advantages and disadvantages?*
5. *What are the different architectural approaches to designing a HIDS?*
6. *What influence would these approaches have on the (real-time) constraints of the systems?*
7. *How can we model the influence an IDS will have on a real-time system?*

The remainder of this thesis is structured as follows: To answer the first two questions, we first describe the energy grid and its structure, define a categorization

of real-time systems, identify the systems within the energy grid which are real-time, and establish their real-time constraints and their category. We do this in sections 2 and 3.

To answer question 3, we describe the actors we view as potential threats to the vital energy sector, the main impacts attacks might have on the electrical grid, and several vectors that could be used by these actors to perform attacks, in section 4.

Questions 4 and 5 are covered by a description of different intrusion detection approaches in section 5.

We explore question 6 through an analysis of several different techniques of data collection on a host in section 6.

Finally, to answer question 7 we propose an extension on an existing process algebra and model one of TenneT's real-time systems, both without and with intrusion detection, in sections 6 and 7.

The main research question is answered in section 7, followed by a discussion, proposal for future work, and conclusion in sections 8 and 9.

1.5 Related Work

A large amount of research is being done in the field of intrusion detection. We touch upon much of this work in this thesis. However, the amount of research on intrusion detection in industrial control systems is more limited. [13] is some early work on network-based intrusion detection for SCADA networks. Similarly, in [75] the author proposes a network-based intrusion detection system for SCADA systems. Bolzoni presents the SilentDefense network intrusion detection system in [14], and recently Lin et al. have written a DNP3 SCADA protocol analyser for the Bro network intrusion detection system[47]. Host-based intrusion detection research for SCADA systems however, appears to be non-existent.

2 The Energy Grid

2.1 Electricity Distribution

The Dutch electrical ecosystem consists of four distinct parts:

- Electricity generation
- High-voltage transmission (110kV and up)
- High-voltage and low-voltage distribution (up to 110kV)
- Electricity delivery

Electricity generation is mainly handled by power plants, which are connected to the national transmission grid by the Transmission System Operator (TSO). High-voltage transmission using 110kV and higher is used to transport the energy to substations, where it is transformed to lower voltages and handed off to the distribution grids of the regional grid operators (RNB), who ultimately deliver it to consumers.

Since 1998 TenneT is the designated independent operator of the Dutch electrical transmission grid consisting of segments for 220kV and 380kV. With the introduction of the law on independent grid operation ("Wet Onafhankelijk Netbeheer"), the operation of most of the 110kV and all 150kV transmission grids was transferred to TenneT as well. Because of these changes to the Dutch situation, TenneT is the sole operator of the Dutch high-voltage electrical grid of 110kV and up since 2011. TenneT has also become the sole owner of most of these grids. Since this puts TenneT in a monopolist position, the Dutch government has introduced an overseeing body, the "ACM energiekamer", which analyses TenneT's performance and investment proposals, and determines the tariffs which TenneT must use.

Operation of the grid happens on two locations: the National Operation Centre ("Landelijk Bedrijfsvoeringscentrum" or "LBC") in Arnhem which handles the 380kV and 220kV grids, and the LBC in Ede which handles the 110kV and 150kV grids. Both locations are able to take over the other's responsibilities, should this be necessary.

TenneT has also acquired a large part of the German electrical grid, which is operated from locations in Germany. The Dutch grid has a number of connections with grids in Germany, Belgium, Great-Britain and Norway, with more connections planned and under construction, as can be seen in image 2. TenneT now manages over 20.000km of high voltage transmission lines and serves, through RNBs, 36 million end users.

An electrical line consists of three phase wires and a ground wire. Like most transmission grids, the Dutch grid uses a redundant system where two lines are

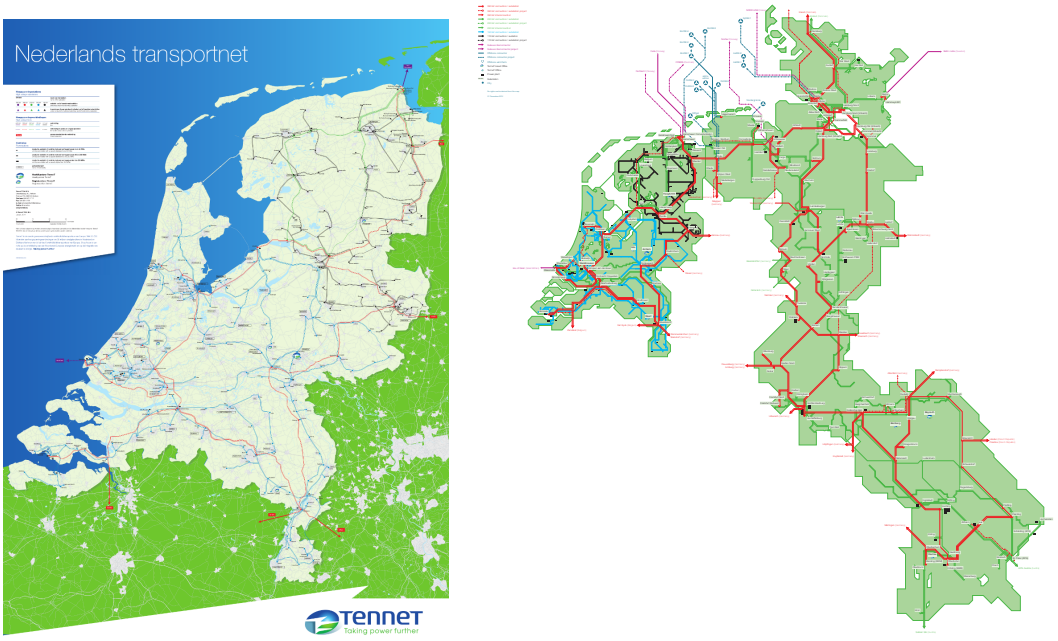
used on each segment (or "field") of the network, i.e. each connection between two HVSs consists of at least two lines. In the 110kV and 150kV grids the ground wire is often shared between the lines. This redundancy ensures that one side of a line can be switched off for maintenance or other reasons without interrupting transmission. This setup can be seen in image 1. The phase wires are recognizable by the glass isolators. The topmost outer wires, without isolators, are the ground wires.

Image 1: Dutch high voltage mast carrying one line on each side[1].

The Dutch grid is based on rings, as can be seen in image 2. A ring consists of multiple fields which are linked by HVSs. For example, the main 380kV ring runs from Maasbracht, via Eindhoven, Geertruidenberg, Krimpen aan de IJssel, Diemen, Lelystad, Ens, Zwolle, Hengelo, Doetinchem, Dodewaard, and Boxmeer, back to Maasbracht. This is another form of redundancy: if both lines of a field are severed, the other side of the ring can be used to reach the HVS at the other end.

A high voltage station has several tasks. First and foremost, it must link the fields of the transmission grid. The HVS has two or more "rails" to which the fields

[1] Source: www.hoogspanningsnet.com Licence: CC BY-NC-ND



(a) Dutch grid, 2011.

(b) Dutch and German grid, 2013.

Image 2: The electrical grid operated by TenneT in the past and present.

can be connected. A rail simply consists of three aluminium tubes, one for each phase, mounted on poles. Each phase terminates in the HVS at several pieces of equipment. Among these are high-voltage circuit breakers which must sever the connection if there is a fault in the line. The actual connection of the field to the rail happens by a physically detachable pantograph which sits between the field equipment and the aluminium tube. Image 3 shows these components.

Other equipment can be attached to the rails. E.g. many HVSs have connections to fields of different voltages, and by transforming between them they connect the different grids. For this purpose, transformers can connect to the rails. Most grids have multiple connections to others, so that there is no single point of failure. Hand-off to the RNBs also happens at these stations.

Grid safety systems are used in ensuring grid safety. This, in particular the way in which short-circuits are handled, is an important aspect of grid management. Short-circuits can be caused by a number of reasons, e.g. a line snapping and falling to the ground, or something hitting the line and creating a circuit to the ground. To handle this problem, every field is monitored by a pair of grid safety devices. These devices are stationed at opposite ends of the field at their respective HVSs.

(a) Line termination at circuit breakers.

(b) Rails and pantographs.

Image 3: Components of a high-voltage station[2].

They are linked to the circuit breakers for their field, which they can trigger when they detect a fault. They perform two types of measurement.

The first type is based on measuring the resistance of the line, and is performed by each device autonomously. This is referred to as distance based monitoring, because it is intended to gauge the distance at which a fault has occurred. This is displayed in figure 4.

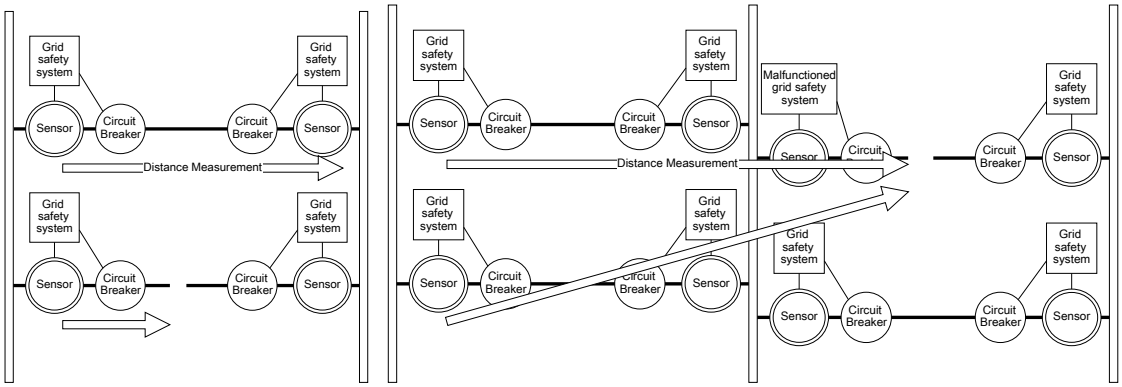
The second type is based on measuring the difference in phase vectors (phasors) at each end of the line. If the phasors differ more than a certain amount, this indicates a fault within the field. This is referred to as differential monitoring. However, in order to compute the difference, the measurements must be taken at the same time and transported from one grid safety device to the other. The internal telecommunication network, described in the next section, is used for this.

The grid safety systems currently in use are equipped with an I/O-port for direct management purposes, to which a maintenance engineer can connect his management laptop. The entire functioning of the system can be influenced through this port. Another option provided is reading the memory of the system, and, in particular, providing information about the fault in the grid and the response of the different grid components. This is valuable in finding the location and cause of the fault.

2.2 Telecommunication Network

The backbone of TenneT's operations is a private telecommunication network. The communication lines for this system vary. In the case of the 220kV and 380kV

[2] Source: www.hoogspanningsnet.com Licence: CC BY-NC-ND



(a) Direct distance fault detection. (b) Distance fault detection in surrounding lines.

Figure 4: Situations where distance fault detection triggers circuit breakers.

grids, a fibre-optic wire runs through one of the ground wires of each field. In the case of 110kV and 150kV, there are separate fibre-optic links linking the stations. For now, we will focus on the 380kV grid, since the other grids are based on the same principles.

Because the 380kV network is based on rings, the fibre-optic network is as well. The transport medium of the network is the fibre running through the ground wires. Like the ground wires, the fibre terminates at each HVS at the end of a field. The equipment there can either relay signals onto the next segment of the ring, or send them into the local systems. The ring is linked to Arnhem and Ede, where most of the signals originate or terminate. Should the fibre-optic be severed in one location, the network will build new circuits using the other side of the ring. To separate part of the system from centralized control, the line would need to be severed at at least two separate locations.

The backbone is based on circuit switching using Synchronous Digital Hierarchy (SDH) for its transport protocol. Plesiochronous Digital Hierarchy (PDH) systems are attached to the SDH systems to provide legacy support for this older protocol. The telecommunication backbone is used by several different, logically separate, networks. The Energy Management System (EMS) network runs on it, but there is also a telephony network and the communication for grid safety systems. A schematic view of the network structure can be seen in figure 5.

The telecommunication network is monitored and managed using a central management system, the Network Management System. Monitoring is done over the telecommunication network itself. For this purpose, (logically) separate management connections are used, which are not accessible to third parties such as RNBs. Configuration changes and software updates for the telecommunication systems can be pushed out through these connections.

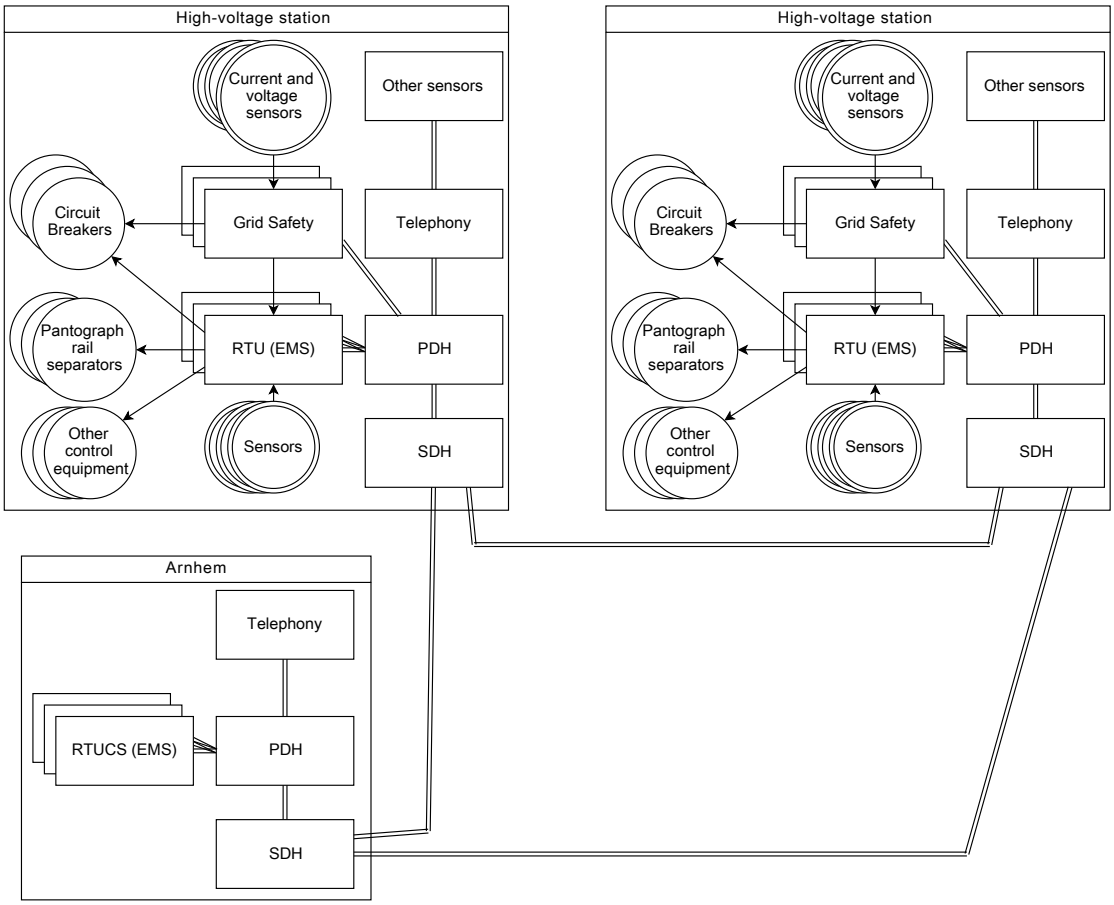


Figure 5: Schematic view of the network structure of TenneT.

2.3 Energy Management System Network

Day to day operation of the transport grid is mostly a manual process. Linking fields, turning transformers on and off, and shutting off lines for maintenance are all control actions done by humans. This happens at the LBCs, which have been built for this purpose, in Arnhem and Ede. In order to reach the systems in the HVS, a network of Energy Management Systems / Supervisory Control And Data Acquisition systems (EMS/SCADA) is used. This network uses the private telecommunication network described in the previous section as its transport layer.

At each HVS, a Remote Terminal Unit (RTU) is linked to both the sensors monitoring as well as the physical systems controlling the grid. In the central location in Arnhem, several SCADA supervisory systems, the RTU Control Servers (RTUCS), are used to communicate with the RTUs in the field. At the moment, the IEC 60870-5-101 transmission protocol is used for this communication. The RTUs gather sensory data and pass this on to the RTUCS. Decisions the human

controllers in the LBC make are heavily influenced by the data reported to them. Any control decisions made are then sent to the RTUs by the RTUCS, and carried out at the HVS. Should the EMS/SCADA network fail for whatever reason, the possibility to manually manage the systems at the HVS remains. However, this is meant only as a last resort, and the continuous operation of the EMS/SCADA network is one of the primary concerns within TenneT.

Many EMS/SCADA systems support the logging of Sequence of Events (SoE), an important feature when e.g. the cause of a systems failure must be found. Basically, measurements from systems for which SoE is enabled are tagged with the exact time of arrival at the RTU before being transmitted to the RTUCS.

Besides the communication for measurements and control, there is also protection equipment which should function independent from the EMS network. A prime example of these systems is the grid safety equipment. These systems communicate to the RTUs using the IEC 60870-5-103 protocol, and are bound to hard real-time constraints, as will be explained in section 3.2.

2.4 Other Sensors

There are several categories of systems deployed in HVSs which are not part of the EMS/SCADA network, the grid safety systems, or the telecommunication network.

Power Quality measurement devices are used to measure the state of the grid in all 380kV HVSs and then boil this down to a "quality" of power, which is used only to provide information to the overseeing body. The frequency of measurements is high, and for communication with the central recording system an analogue link with modems using TenneT's internal telephone system (separate from the EMS network, but running on the telecommunication network) is used. This technology is moving from analogue links to IP, however, so in the future they might become part of EMS/SCADA.

kWh measurement devices are used to keep track of the supply of power to the RNBs. Again, analogue links with modems using TenneT's internal telephone system is used for communication with the central recording system. The total amount of power provided can be deduced from these measurements, and therefore they should be considered confidential.

Specialized systems tasked with only measuring and recording data about malfunctions are placed in several HVSs. These complement the recording task of the grid safety systems. They have a very high frequency of measurements and record the data at the HVS itself, only to be read after an actual malfunction occurs. These measurements are then used to reconstruct a sequence of events. This

analysis and reconstruction is usually done in the office environment, so remote polling of this data is desirable. For this, again, an analogue link and modem using TenneT's internal telephone system is used. However, it is possible that these systems will become part of the EMS/SCADA network in the future.

2.5 The Natural Gas Grid

TenneT's counterpart in the natural gas distribution is Gasunie. After a process similar to the unbundling of the Dutch energy retail companies, Gasunie now operates as a distribution company. It manages, maintains and expands the gas grid, and offers associated services. Gasunie has acquired part of the German gas grid, for which it performs the same tasks. Since Gasunie is a monopolist too, the ACM energiekamer has been appointed overseeing body for Gasunie as well.

The gas grid managed by Gasunie is 15.500km in length, and includes 22 compressor stations, 19 blending stations, 93 metering and regulating stations, 14 export stations, 1300 gas delivery stations and a few specialized facilities. All these stations contain ICS/SCADA systems and are monitored. Some facilities are controlled from within the facility itself, others are monitored and controlled from a central location. To this end, Gasunie employs a virtual private network rented from a major telecommunication infrastructure provider.

The structure of the ICS/SCADA systems used by Gasunie is slightly different from that employed by TenneT. At the lowest level, the real-world objects are controlled and monitored by Programmable Logic Controllers (PLCs), which communicate amongst each other. The PLCs are controlled through ethernet connections with RTUs, which are in turn controlled by operators in the plant, or from a central location. Gasunie runs a rudimentary host-based anomaly detection systems on the RTUs, which we will explain in some detail in section 5.5.

3 Real-Time Systems

Real-time systems are a class of computing systems, hardware and software, where the time it takes for the system to compute a result is an important factor in the operation of the system. The IEEE Technical Committee on Real-Time Systems defines them as

> ... systems in which its temporal properties are essential for reliability and correctness; the example applications include embedded systems, control systems, monitoring systems, and multimedia systems. [3]

The Euromicro Technical Committee on Real-Time Systems uses the definition

> [Real-time] computing systems have to provide results which are not only correct, but also delivered in time. Instead of average behaviour as for standard computing, real-time systems have to allow for guarantees that the temporal requirements will be met. [4]

The second definition is rather more strict than the first, since it includes that real-time systems have temporal guarantees, not just requirements. For the purpose of our research, however, we prefer the definition that a real-time system is a system which is relied upon to provide a result within a certain time, regardless of whether the system guarantees that constraint will be met. The definition of a real-time system used in this thesis is therefore "a system which must provide correct results in time, as defined by its application".

Note that none of these definitions imply that a real-time system must be fast (high-performance), nor that it must simulate realistic passing of time (real-time simulation). The only requirement is that there exists some time-constraint before which the system should provide its result. This can be in the order of microseconds, milliseconds, but also seconds or even minutes.

In the remainder of this section we first describe the classifications of real-time systems we will use. Next, we list the systems we have identified within the electrical grid that can be classified as real-time systems, their real-time constraints, and their class.

3.1 Categories of Real-Time Systems

Real-time systems can be categorized as hard, firm, and soft, based on the effect that not meeting their real-time deadline would have on the service provided by the system[48].

[3] http://tcrts.org/about/
[4] http://www.ecrts.org/

– In hard real-time systems, missing a deadline is considered a complete failure of the system. Reasons for this requirement vary, but often they either stem from the fact that missing the deadline causes physical harm to humans, harm to the environment, or simply great financial loss. Examples of this kind of system are car airbag control systems, pacemakers, and some industrial process control systems.
– Soft real-time systems are systems on which missing a deadline is not a failure of the system, but the quality of service is diminished. The result such a system produces is less useful after missing the deadline, but still has some value, and the system can continue to operate. Compare this with a public transport service, where, if e.g. a train is late, quality of service is diminished but the people are still where they needed to go. Examples of such systems are network routers, reservation systems (where response time impacts both user experience and seat availability), and human computer input.
– Firm real-time systems fit between the previous categories. On these systems, missing a deadline means the task it was working on has failed, and the result of that task no longer has any value. However, the system can continue to operate and simply discards the late results. An example of a firm real-time task is a video player, which often simply discards frames when decoding is too slow.

For the purposes of this research we also categorize real-time systems based on the visibility of their internal workings. Since we are concerned with host-based intrusion detection, we need access to certain data of the host. White-box, grey-box and black-box real-time systems differ in the amount of data that can be accessed by a normal user of the system.

– White-box real-time systems are systems on which the user has full access and complete control of what is observed, and how it is observed. For example, a Linux system on which the user has root privileges, or a Windows system with an administrator account.
– Grey-box real-time systems are systems on which the user has partial access to information. Maybe because there is only a limited interface, e.g. a graphical interface served over a network connection with limited information, or maybe the user has an extended interface such as a login-shell but is limited in the actions he can perform.
– Black-box real-time systems only perform their designated task, and do not provide any means for the user of the system to access information not required for that task.

Finally, we consider the implementation structure an important aspect of real-time systems. There are *dedicated* real-time systems, where the system is a single program running on dedicated hardware, performing only its real-time task. There

are also *mixed* real-time systems. These consist of a supervisory program, often an operating system with multiprocess capabilities, memory management, etc., running the real-time processes as well as non-real-time tasks. The precise implementation does not matter for this categorization, however, we have seen systems based on Linux, on Windows, and on proprietary Unix-derived operating systems.

3.2 Real-Time Systems within TenneT

Grid Safety Systems were described in section 2.1. A failure of these systems to disconnect a field before the deadline can be life-threatening, and can cause damage to systems. This might cause large financial damage as well as disrupt the grid. Therefore, the grid safety systems are hard real-time systems. They function under two hard real-time constraints. The first is that under normal operation, circuit breakers must trip within 100ms after a short-circuit manifests. The second is that, in the event that the safety system of the field cannot satisfy the constraint of 100ms for whatever reason, the field with the short-circuit must be shut down within 300ms. This can be facilitated by e.g. triggering the circuit breakers of surrounding fields when the fault is detected by the outlying safety systems (as in figure 4).

For normal operation of the differential monitoring, where the systems communicate with one another, the 100ms breaks down into the following constraints:

- less than 1ms to perform the measurement,
- 6ms to transport the information to the other side,
- less than 1ms to determine the fault,
- 2ms to signal the circuit breaker it should trigger, and
- 10-50ms for the circuit breaker to trigger.

The final range is 40ms wide because not all circuit breakers trigger as fast as others, and they are constructed in such a way that they wait for an "ideal" moment to break the circuit. This still leaves 40ms leeway in the worst case chain of events. Distance monitoring has the same breakdown, except that transportation is left out entirely.

The system currently in use by TenneT for these purposes is a black-box system provided by Siemens. It does have an I/O port for configuration and management purposes, as well as the option to read the memory state. Because of these options, we assume this is a *mixed, black-box, hard real-time system.*

The RTUs employed by TenneT are quite powerful computers in their own right. Their most important task is relaying telemetry and commands. One type of RTU in use by TenneT is the Sprecher Automation Sprecon-E-C-series. These are a

combination of SCADA controller and PLCs in one device. The device is extensible, and all plugin modules are connected through an internal bus interface. This includes power supply, CPU and the I/O modules.

Processing sensor readings and relaying them to the RTUCS is a firm real-time process. The measurements are done every few seconds, and a measurement which is not processed and relayed before the next one is done has no value to the RTUCS, since it will be immediately superseded by the next one.

These systems can also record a sequence of events, as mentioned in section 2.3. The results of this recording are used in recovery and post-mortem analysis after problems in the grid, and are useless if their deadlines are missed. However, the system will not cease operating, so this is considered a firm real-time process. Timestamping of these events can be done with very high precision using GPS-synchronized clocks. The real-time constraint of this has not been clearly defined, but considering the high accuracy required, we assume that timestamping must be done within 1ms after the event takes place.

The RTU can be controlled locally or remotely, and it can make autonomous decisions through the use of limit values on its inputs. Autonomous decision-making can be a hard, firm or soft real-time process depending on its application. However, we have not seen this capability used on RTUs within TenneT, and thus have no information on this real-time aspect. Regarding remote or local control, the system is considered hard real-time, but in the order of minutes. When a command is given, it is preferable for it to be processed as soon as possible. If the system does not process the command within 20 seconds, the command is retried by the RTUCS. After this is done three times, the system is considered to have failed.

From the technical documentation of these systems it appears that they are grey-box. TenneT does not have complete control of the systems, but does have access to part of the internal state.

Strictly speaking, the RTU is therefore a mixed, grey-box, hard real-time system. However, since the only hard real-time requirement is in the order of minutes, and normal processing of commands normally takes only several seconds, there is a very large margin for a host-based intrusion detection system to work in. Therefore, we will treat the RTUs as *mixed, grey-box, firm real-time systems.*

The RTUCS has to process the information of the RTUs in real-time as well. Following the same reasoning as for the RTUs themselves, the RTUCS is considered a *mixed, grey-box, firm real-time system.*

Telecommunication Systems providing the transportation of measurement data for grid safety systems have a hard real-time constraint of 6ms transportation time, as mentioned earlier in this section. The infrastructure currently in use for this is a combination of plesiochronous digital hierarchy (PDH) and synchronous digital hierarchy (SDH) systems, as mentioned in section 2.2. These systems

are not designed as real-time systems in the sense that they guarantee a trip time of at most 6ms. However, network systems are in principle soft real-time because they have to process and forward data at the same rate as they receive it. Otherwise, they have to drop data and service deteriorates. The configuration used within TenneT is designed and tested to satisfy the hard real-time constraint.

The SDH and PDH systems in use are provided by Alcatel-Lucent. These systems have two layers: the *management and configuration layer*, which is used to manage and configure the system, and compiles this configuration into a set of simple rules for the *forwarding layer*, which processes the network traffic based on these rules. These layers are separate, so that the activities on the management layer do not impede the functioning and throughput of the forwarding layer. The real-time operation of these systems, therefore, lies in the forwarding layer. The user is not entirely free in what he can do on the management layer, however, so this is still a grey-box system.

The SDH and PDH systems carrying grid safety information are therefore *mixed, grey-box, hard real-time systems*. The systems which do not carry information for grid safety systems do carry other information, such as measurements and commands, between RTUs and RTUCSs. Since these are both firm real-time systems, these SDH and PDH systems are also firm real-time, making them *mixed, grey-box, firm real-time system*, giving us two classes of telecommunication systems.

In the future, TenneT will likely be switching to packet-switched based network protocols. Although this means that the equipment used will change, it is not expected that the real-time classification of these systems will change. Furthermore, it is expected that the latency for these new systems is still able to satisfy the 6ms constraint.

Power Quality measurement devices, as described in section 2.4, have a high frequency of measurements (hundreds of measurements per second), so their real-time deadline is in the order of a few milliseconds. As with some other measuring systems, if a measurement is not processed before its real-time deadline passes it is superseded by the next measurement. They are *dedicated, black-box, firm real-time systems*.

kWh measurement devices, also described in section 2.4, are soft real-time. Measurements which are not processed in time still have value, as the aggregate will still be correct. These are *dedicated, black-box, soft real-time systems*.

The specialized systems tasked with measuring and recording data about malfunctions, the final category in section 2.4, also have a very high frequency of

measurements. Since they are concerned with sequence of events reconstruction in the case of grid failure they have a real-time constraint of under a millisecond, and they are *dedicated, black-box, firm real-time systems*.

4 IT Threats to the Electrical Infrastructure

The most visible part of the electrical infrastructure is physical: there are power masts throughout the country, on locations accessible to the general public. Physical attack on these masts or the lines is a potential threat to the grid. The system is designed to be quite redundant, however, so it would take a lot of effort to effectively bring down (part of) the power grid by attacking the masts.

A less visible part of the electrical grid is digital. The grid is run using IT, and is therefore potentially vulnerable to attacks on that front. In recent years, we have seen an explosive increase in the number of attacks on SCADA systems connected to the internet. With Stuxnet, we have seen how even an industrial control system without links to the internet can fall victim to a virus attack. The EMS/SCADA network is another network which cannot be reached from the internet, but could nonetheless fall victim to such an attack.

In this section we will first examine some different actors who could threaten the EMS/SCADA network. Next, we will examine some threats recognized by TenneT. Finally, we will list several vectors which could be utilized by attackers.

4.1 Actors

Actors are the parties who pose potential threats to the electrical infrastructure. An actor does not have to be the attacker: consider the case where a nation state pays a criminal organization to destabilize the banking sector of another nation state. The actors involved are a nation state and professional criminals, but the attacks are performed by the criminals, who may in turn attempt to influence internal actors[52, 60].

State actors are all actors which are part of, or facilitated by, the government of a nation state[52, 60]. State actors are interested in helping their nation state, e.g. by improving its international diplomatic or military position, or by intimidating activists opposing the government. For this purpose, they develop specialized tools and train experts in digital "warfare". The main threats from state actors to an electrical grid are espionage (to gather military or economically relevant information) and sabotage (e.g. to disrupt the grid as part of a conflict between nation states). If a conflict with another nation state arises, one of the best ways to cripple part of the enemy state is by depriving it of electricity, clean water, and natural gas. However, even in times without open conflict, these systems may become targets for the purpose of destabilizing other nations. Only recently, a hacking group believed to be part of the Chinese army was caught taking over a water supply honeypot, attempting to tamper with the systems[77, 71, 7]. It is also believed by

some that the same hacking group is actively attacking the electrical grid of the United States[31, 7].

Since they are facilitated by governments, these actors have a considerable amount of money to spend on their activities. This implies they can invest heavily both in technology and in people with advanced skills, and are therefore considered to be capable of the most advanced attacks[52].

Terrorists are individuals, not necessarily affiliated to a nation state, who use acts of terror in an attempt to reach their goals, such as influencing the political process or spreading fear among a population. Considering the potential effects of a major power outage, and the high visibility of an attack on the electrical grid, this could be an attractive target. The actual direct damage of an attack would not need to be large, as long as it creates fear among the public and elicits a repressive response by the government. Both the skills as well as the willingness to use digital attacks appears to be on the rise amongst terrorist groups[50], and we should therefore consider the threat they pose to the electrical grid. Their funds are dependent on whether they have wealthy backers, and therefore may vary from virtually none to quite extensive. At the moment, however, the actual capabilities of terrorist groups in this context are considered limited by intelligence agencies, and they are therefore considered only a limited threat[52].

Professional criminals are, contrary to most actors, mainly interested in money. They use advanced digital means to attack banks, companies, and people, in order to make money. The energy market is quite large, and criminals might be interested in the communication going on in the separated networks of the electrical grid to enable them to commit fraud. Furthermore, they often rent out their services to other parties interested in performing digital attacks but without the capability to do so. They could also try to use blackmail, threatening to cause major blackouts unless they are paid. They can have considerable funding, especially when the expected gains from their criminal activities are large[52].

Hacktivists are hackers who attack out of ideological motives. There are many different groups of hacktivists, with as many different ideologies. Notable examples are the loose collective Anonymous[5] and the related group LulzSec. Their successes have demonstrated that at least some people within these groups are skilled enough to perform complicated attacks. However, there is also a large number of people within these groups with a lower skill level, who nevertheless play a large role in

[5] It should be noted that Anonymous is not a single group, nor does it have a single ideology. It is a fluid collective of people who use the name, and at any one point in time there is a number of subgroups working towards different goals.

e.g. distributed denial of service attacks using tools such as the Low Orbit Ion Cannon[52]. These people are known as script-kiddies in the hacking community, and they exist in other groups of actors as well – many professional cybercriminals are script-kiddies.

In the 1970s and 1980s a heated debate about nuclear power took place in the Netherlands. In the 1980s this lead to some activist groups actively attacking the power grid. These attacks required few materials and little technical knowledge. A publication from 1985 also mentions that the redundant structure of the grid should be taken into account for these attacks, that safety measures should be taken, and how best to publicize the attack[53]. Considering the similarity to current hacktivist groups, it is not inconceivable that a new generation of energy activists might use hacks of the electrical grid as a way to e.g. put pressure on the government or the energy industry to close nuclear power stations, or to increase the market share of renewable energy. In fact, since the number of people willing to participate in anti-nuclear protests has shrunk to only a fraction of what it was in the 1980s, they might even perceive this as one of the only ways in which they can make their voices heard, since it requires only limited manpower to have a large impact. Their funding, however, is rather limited, and their threat is considered average[52].

Commercial parties on the energy market may be interested in information which improves their competitive position. Often, this is information relating to financial data, e.g. bids made by other players in the market. In this way they are similar to computer criminals. Market manipulation and fraud, especially in negotiations over import and export pricing, could be facilitated with detailed grid information[52].

Internal actors are individuals working for the organization being targeted, or who have worked for that organization in the past. They have detailed knowledge of the internals of the organization, and can use this to cause significant damage. The threat from internal actors is twofold. First, there are unintentional threats. E.g. internal actors can misconfigure systems, or they can be targeted by other actors and be used as an attack vector through phishing. Intentional threats come from e.g. disgruntled employees who want to hurt the organization, people who disregard rules because they want remote management capabilities (e.g. by installing a WiFi access point or a GPRS link using the public mobile phone network) or employees who "just want to do their job" and in doing so actively circumvent security measures[52].

Other actors are people who do not fit into one of the other categories. Cyber vandals are one of these groups. They are people who perform attacks, often just

"to show they can". They are not motivated by financial gain, nor an ideological goal. They vary in skill level, and are as a group sometimes capable of performing sophisticated attacks. They should therefore not be discounted as a threat: if they find a way to get into the grid, they might do so without needing further justification[6][52].

4.2 Possible Impacts

Malicious activities by the aforementioned actors can have many impacts. We describe the most serious ones in this section.

Grid disruption is a major concern, since the electricity supply is a critical resource. Grid disruption can take many forms, from a total blackout to small, localized outages. The length of a disruption is, after scale, another important aspect. A German committee[62] analysed the possible consequences of a prolonged and wide-ranging power outage. They conclude that it has the potential to become a national disaster, severely disrupting transport, supply of clean water, wastewater disposal, food supply, healthcare, the economy, and even the prison system. Furthermore, increased duration of the power outage may have unforeseen effects on the general population; although not enough research has been conducted on the behaviour of groups and individuals in disaster situations, there may be an increased risk of anti-social behaviour, even rioting, due to the stress and breakdown of public order[62].

Grid disruptions may be caused by physical damage to the grid, but also by tampering with the grid control systems or grid safety systems. If the grid control systems are tampered with, this may also require manual intervention at the location which can no longer be controlled, which is a slow and relatively expensive process.

Physical damage to the grid is another concern. It needs to be repaired, causing a financial loss to TenneT, even if it does not cause a grid disruption. One of the ways in which physical damage could be caused is if the grid safety systems stop functioning, in which case a failure within the grid might damage the equipment normally protected by an immediate shutdown. Another way this may be caused is by tampering with the grid control systems.

Loss of life is another effect which may be caused by non-functioning safety systems. The risk of this happening must (for obvious reasons) be minimized.

[6] The distinction between groups of cybervandals and hacktivists is often a blurry one. In the case of Anonymous, for example, many people within a subgroup often have different motives. Some are indeed true hacktivists, working towards a certain ideological goal. Others, however, simply see these activities as a way to cause some havoc, i.e. vandalism.

(Corporate) espionage is the final threat. An actor could try to gain access to internal information to improve its own position in the market. Another threat of espionage is reconnaissance by an actor to facilitate in further attacks. Actors may even try to tamper with grid operation to influence the market price for electricity.

4.3 Vectors

We have determined several possible vectors which actors might use to attack the EMS/SCADA network. Many of these were discussed internally within TenneT, and some were also discussed with other experts within the industry.

Direct access to the EMS/SCADA network from the internet is an attack vector which is often used in real-world cases of SCADA compromise. In the case of TenneT, however, this vector is well understood and the EMS/SCADA network is implemented as an isolated network which is not publicly reachable. There remains a concern with actors manipulating the infrastructure on which the EMS/SCADA network runs directly; however, the cables are not easily reached, nor easily tampered with. Nevertheless, communication on the network should be monitored for possible ongoing attacks.

Manipulation of internal actors to grant access to the EMS/SCADA network, or to perform an attack directly. The threat internal actors pose is well-known and effort is put into educating people to perform their work in a secure manner. However, a determined internal actor is very likely to be able to perform malicious actions successfully, which is why the work of employees on the EMS/SCADA network should be scrutinized.

Installation of unauthorized hardware in the network could e.g. provide a direct connection to the internet by installing a GPRS uplink, or a machine performing autonomous attacks. This could even take the form of hardware which is thought to be legitimate, but in fact carries malicious functionality such as a built-in backdoor. The next two vectors are related to this.

Use of USB sticks is a significant attack vector, as they are a potential carrier of viruses and other malicious software. Stuxnet used USB mass storage devices combined with vulnerabilities in Windows to infect machines inside a separated network. Even if USB mass storage devices are forbidden, ingenious hackers could use other techniques to infect machines through peripheral devices, like combining a human interface device such as a mouse with a USB stick and hub internally[58].

The following vectors assume that an actor has somehow gained access to the EMS/SCADA network.

Viruses and other malicious software placed on existing machines are in a position to manipulate measurements, commands, and other communication passing through these systems. The most likely scenario is that these viruses work autonomously. From here, the viruses can also attempt to propagate in the network.

Access to user-accounts on the machines of the EMS/SCADA network is another vector which provides attackers or autonomous malicious software with the capability to manipulate measurements, commands, and other communication. For example, a virus on one system could attempt a "legitimate" login on another system to manipulate it. Access to accounts can be obtained through e.g. password bruteforcing, the use of a default administrative password, or obtaining a password through other means such as phishing; or, when the actor is internal, legitimate access to the acounts for malicious purposes.

Collateral damage happens when e.g. a virus, not intended for or targeted at the EMS/SCADA network, nevertheless infects machines in the network and affects their behaviour. This can happen when commercial off-the-shelf (COTS) systems are used for EMS/SCADA functions. These COTS systems share characteristics and vulnerabilities with personal home computers, and therefore a virus targeted at e.g. internet banking could infect them. Even though this virus would never actually perform its intended function, it could still affect the functioning of the network, hamper grid operation, and cause grid disruption.

5 Intrusion Detection Technology

In this section we explain various classifications based on the different aspects of intrusion detection technology, list advantages and disadvantages of these aspects, consider the effectiveness of the anomaly-based and signature-based approaches, and look at a case study focused on Gasunie.

5.1 Detection Approaches

The first classification of IDSs is based on their detection method. IDSs gather data from their monitoring targets, which they then analyse to detect intrusions. The two different detection methods are signature-based, which models malicious behaviour and detects that, and anomaly-based, which models normal behaviour and detects deviations from it. In both cases, the challenge of effective intrusion detection is to build a model that is both complete (the system detects all intrusions) and accurate (the system does not detect false positives).

It should be noted that actually proving an IDS complete would mean proving that it can detect every intrusion, known and unknown, in the target it is monitoring. Such a system is trivial: simply build a system that marks everything as an intrusion. However, such a system would certainly not be accurate. Testing of IDSs is done against a dataset with known attacks. By their very definition, we cannot prove an IDS can detect truly unknown attacks. We do not believe it is possible to build a system with a 100% detection rate without false positives, so the challenge becomes building a system which has an acceptable rate of false negatives (intrusions that go undetected) vs. false positives.

Signature-based intrusion detection is built on known attacks having recognizable patterns. A detection engine based on signatures will analyse the incoming data looking for these known patterns, which are provided to the system as a set of rules. For instance, an attack which is recognizable by creation of a file called "attack156.sh" can be detected by an IDS with a rule to raise an alert if it encounters that exact filename. It can also use a pattern, e.g. if the attack file varies the three numbers the rule would look for a file matching the expression "attack[0-9]{3}.sh". New signatures can be added, but a purely signature-based system cannot detect unknown attacks – unless they happen to have the same signature as another known attack – thereby making them less useful in situations where most attacks are newly crafted, as would be the case in a highly specialized environment like the energy grid.

Advantages of a signature-based system are

- that they are usually computationally cheap,
- very effective at detecting known attacks for which complete signatures exist,

– are able to identify the vulnerability being exploited, since the signatures are built with specific intrusions in mind, and
– can have very few false positives.

Regarding the last point it should be noted that if the signatures contain generic patterns (such as wildcard matching of strings), which is often done to enhance completeness, they are more susceptible to matching on legitimate behaviour[55].

Disadvantages are, however, that they

– are unable to detect unknown attacks,
– that signatures are often provided without full understanding of an attack and therefore ineffective at detecting it, and
– that they are fairly easily bypassed even if the rules have good coverage of attacks. They can, by their very nature, never be complete[30, 59, 65, 49].

Anomaly-based intrusion detection attempts to identify deviations from normal behaviour. For this purpose, anomaly-based systems have a model of normal behaviour and will assess the data they receive according to this model. If there is a certain deviation from the model an alert is raised. A model of normal behaviour is either learnt by the system or specified by a human operator. There are currently three classes of anomaly-based detection methods[37, 24]:

– statistical detection methods,
– data mining and machine learning methods, and
– specification based methods.

Systems based on the statistical methods use a training phase to build a profile of normal behaviour of monitored objects. Then, after deployment, they compile the data sources into a statistical profile of the current activities, and compare this with the trained profile. An alert is raised if there is a significant discrepancy between the two. However, it is still up to the designer of the system to determine what activities to monitor.

In [22] Denning proposes IDES, which uses such a statistical model, and monitors users and their interaction, through actions, with objects. For instance, login and session activity is a category in which the actions are “login” and “logout”, and the objects are the user’s location (workstation, network host, etc.). Some profiles that can be used in this category are login frequency, elapsed time per session, CPU usage per session, number of failed login attempts, etc. Similarly, categories for command and program execution and for file access activity are proposed. All these profiles use a manual selection of activities, and either a model where a sample is compared with a fixed limit, or a mean and standard deviation model. E.g. the number of failed login attempts triggers an alert when a user goes past a fixed limit determined by the operator based on past experience, whereas

the CPU usage per session profile uses the mean and standard deviation model which is learnt by the system from existing CPU usage data. This technique is computationally cheap, but generates a large number of false positives, both for legitimate behaviour not captured in the training set, and whenever the behaviour of a user changes.

Denning also proposes a multivariate model, based on correlations between two or more metrics. This is where we enter the realm of data mining. When using a data mining method, the determination of what activities should be monitored is done by first feeding all training data into a data mining method such as associate rule discovery[44]. A statistical model can then be built based on the selected features. The advantage of this technique is that it provides a more reliable model of anomalies. However, it is computationally expensive, both in the learning phase and in the detection phase[44], and it still has a high false positive rate[24]. A related problem is the high cost of updating the model, since, if the system uses a single monolithic model based on an expensive data mining technique, all updates to the model require complete recomputation. [44] proposes to use adaptive learning[26] to mitigate this problem. When the model needs to be updated, a light-weight classifier is built for only the new data. When the original model detects an anomaly, it is sent to the new classifier for classification. This approach is significantly faster than recomputing the new model, however, it has the drawback that, as more classifiers are used, detection slows down.

Continuing on from data mining, machine learning techniques can then be used to build a detection engine. The line between data mining and machine learning intrusion detection techniques is somewhat blurred. A distinction we can make is that "pure" data mining techniques use the rulesets gained from e.g. association rule discovery directly, whereas machine learning techniques build a learning system based upon the selected features[29]. Examples are the use of neural networks [81], support vector machines[40], genetic algorithms[72, 46], artificial immune-systems[38, 8] and hidden Markov models (HMMs)[36, 37, 24, 39, 9].

All these techniques are computationally expensive to train. Furthermore, genetic algorithms and artificial immune systems are expensive to use. Most of these techniques have an unpredictable but high false positive rate, and the neural networks and support vector machines also tend to lack an explanation of why a detection decision was made[73, 29]. The exception to this appears to be hidden Markov models in specific situations: they have a very high detection rate and low false positive rate, whilst being computationally cheap during the detection phase.

A Markov process is a process where its current state is enough to make accurate predictions about its future behaviour, which is, intuitively, a good approximation of computer systems: any behaviour in the future can only depend on behaviour in the past if it is in some way expressed in the current state of the

system. A Markov model is a representation of such a process in which the state is visible. Most of the time, however, the internal state of a process is not directly observable, but the output of the process is. In that case, a hidden Markov model can be used to model the system. Based on observed outputs, it learns about a discrete number of states the process can be in, probability of transition from one state to another, and probability of emission of a certain output when in a given state. It can then be used to perform predictions of the system's behaviour, and if the observed behaviour deviates from the prediction, an alert can be raised.

Finally, specification based anomaly IDSs build a model of the monitored target based on its specification, e.g. a protocol. A protocol anomaly detection engine can be a simple state machine capturing all legal states and transitions of the protocol, which would then produce an alert if a transition outside of the legal state machine is attempted. However, since not all attacks need to "step outside of the protocol", such a detection engine cannot achieve a 100% detection rate[24, 12]. Another option is building a model of all the states, transitions, and parameters that are actually in use by the monitored system, and raise an alert if there is a certain number of deviations from those parameters. This would also detect attacks which use legal transitions in the protocol[70].

Combining some of these anomaly detection techniques may lead to better results. For instance, combining specification based and statistical model techniques could be done by first building a complete model of a protocol, then training a statistical model for which transitions are used and how often[70].

In general, advantages of anomaly detection techniques in contrast with techniques based on signatures are

- an ability to detect new and unknown attacks,
- a higher detection rate of attacks, and
- better resilience to being bypassed.

Disadvantages, however, are

- low detection efficiency of current solutions, where a large number of false positives reduces the usefulness of the system significantly,
- high computational cost, which is a problem when running on relatively weak hardware, and when the load of data to analyse becomes very high,
- an inability to identify the vulnerability being exploited, and
- high cost of adapting the system to changing conditions.

Hybrid approaches combine signature-based and anomaly-based detection engine. A basic application of this is when a signature-based engine could perform a first pass over the input data, detecting any known attacks, after which the anomaly-based engine can scan for anomalous behaviour. More advanced applications, where multiple anomaly-detection and signature-detection techniques are

applied and a weighted analysis of their alerts is performed, can also reduce the number of false positives of an intrusion detection system[10, 33, 24, 37].

5.2 Monitoring Target

A second classification for intrusion detection systems is based on which target they monitor. IDSs can be divided into network-based, host-based, and application-based systems.

Network-based intrusion detection systems (NIDS) monitor the network traffic between hosts. They use the network and its protocols as the source of their information. They analyse the network traffic, for which they can look at different aspects: bandwidth used, which hosts are communicating, what information individual packets carry, etc. They can then use e.g. signature-based detection on the protocol payload data, or anomaly-based detection on the traffic flow patterns[57, 29, 61, 68].

Both the Bro Network Security Monitor[61, 1] and the Snort Intrusion Prevention and Detection System[68, 5] are widely deployed, network-based IDSs.

Snort is a free, open source system first published in 1998. It provides network intrusion detection as well as prevention capabilities. At its core it uses signature-based detection, and provides mechanisms for pre-processors to expand the core functionality. This can be used for e.g. anomaly detection. The public basic rule set for the signature-based detection is well-maintained. In [63] the pre-processor-mechanism is used to generate security event logs from network traffic between SCADA devices, and in [56] intrusion detection signatures are proposed for the MODBUS protocol.

The Bro Network Security Monitor is a modular system which first breaks down network traffic into events, which it then passes up to a higher level which is designed to run user-defined scripts on the events. This makes Bro a very adaptable system suited for both signature-based as well as anomaly-based detection in the same instance. The scripts can be written to be application-aware; indeed, the first publication of Bro already used a partial specification-based detection script for FTP[61]. It has also been adapted to SCADA environments in [47], however, only for the DNP3 protocol, which is not used by TenneT. The approach could conceivably be applied to other SCADA protocols, however.

A relatively new NIDS is SilentDefense, produced by Security Matters. It is tailored for ICS/SCADA networks[14, 15]. This NIDS is anomaly-based, is able to analyse many SCADA protocols and to report on anomalous behaviour[4].

Network-based intrusion detection systems have several advantages over host-based systems:

– They see network traffic from multiple hosts at a central point. Therefore, they are able to monitor a large number of hosts at once.
– Because they see network traffic from multiple hosts, they are better equipped to detect attacks involving more than one host.
– They do not require adaptations of the hosts being monitored.
– They run on dedicated machines for their purpose, and therefore do not require any processing power from the hosts being monitored.

But there are also disadvantages:

– They are blind to the contents of encrypted traffic.
– If, when the network is under heavy load, they cannot keep up with the traffic, they have to drop traffic, and can fail to detect intrusions when this happens.
– They can only see network traffic that passes their collectors. If an attack happens through another channel, such as an "illegal" internet uplink, or a subnetwork that is routed without passing a scanner, it will not be detected.
– Likewise, they cannot see attacks which are confined to a single host. E.g. a host which is infected through the use of a USB-stick with a virus which only tampers with that host, will not raise an alert.

Host-based intrusion detection systems (HIDS) monitor computer systems, and use the state and behaviour of such systems to detect intrusions. An HIDS might monitor total CPU time used, system calls performed, memory allocations, users logging in, and so on. A study of the state of the art in IDS performed in 1990[54] already distinguishes different levels of functionality of HIDSs. The simplest systems only analyse the adherence to security policies, and are not considered IDSs. Some also verify if crucial system files have been unmodified and attempt to detect viruses. Next, there are systems which reduce the audit trail of a system to a meaningful and manageable log. This can involve, among other things, singling out audit records for specific events and disposing of useless data. The intrusion detection system IDES as described by Denning is the first clear example of a HIDS also using performance data of the system[54, 76, 22].

More recently, however, a large amount of research has focused on the use of system call tracing as a mechanism for host-based intrusion detection[23, 27, 66, 36, 39, 55]. In [69] a method of analysing library calls, not system calls, is used to detect malicious javascript in PDF documents. And in [79, 19, 80] approaches with more granularity, monitoring at the CPU control flow level, are suggested. The disadvantage of these last approaches is that they require changes to hardware architectures. It would take a long time before these new hardware architectures are ready to be deployed, so this is not suited for our purposes.

Some examples of HIDSs deployed in the real world are OSSEC and Verisys. Verisys is a system which takes the basic premise of the simplest intrusion detection

systems, to check the integrity of system files, and tries to take it as far as it will go. It uses a multitude of file properties, such as name, creation time, size, but also access control list and sha256 hashes to detect any change to defined sets of files. The claim is that "if files are altered in any way, Verisys will detect it."[6]

OSSEC is a free, open source system which combines several techniques: security policy monitoring, integrity checking of system files, audit trail analysis, rootkit detection, and the capability to automatically respond to intrusions with more than just an alert. The system combines anomaly-based and signature-based detection. It scans for common attack patterns, but also for deviations from normal behaviour defined in rules for the system. Most of its live detection capability stems from a powerful log analysis engine used to analyse the audit logs as they come in in real-time[3].

The advantages of HIDSs are:

- They have access to a broad selection of detailed information on the host, and are not limited to what can be learned from network traffic, eliminating three disadvantages of NIDSs.
- They can potentially see network traffic before it is encrypted and after it is decrypted, thus they are not blind to attacks using encrypted channels.

And the disadvantages are:

- Unless they are designed to report their data to a central system, or each other, they cannot correlate data from multiple hosts to detect attacks involving more than one host. Most modern HIDSs have some way of reporting their alerts to a central management system.
- They require adaptations of the host being monitored, at least in the form of additional software.
- They use resources of the host being monitored.
- They can add overhead to the running of other processes, depending on the type of monitoring being done.

Application-based intrusion detection systems are a specialization of host-based ID. They monitor events from specific applications on a host. They isolate data sources from these applications before performing analysis and detection, but can in theory see the same sources that an HIDS can. One proposed way to perform this detection is to insert probes into a program, such that the program reports its state to an outside process monitoring its behaviour[11].

The advantages and disadvantages are the same as those for HIDSs, except that application-based IDSs are guaranteed to be application-aware and therefore might be more accurate (advantage), and that application-based IDSs potentially add even more overhead to the running of an application (disadvantage).

5.3 Detection Time

Another aspect of intrusion detection systems is whether they are able to run in real-time or periodically, i.e. if they can detect intrusions as they happen, or only after the fact. Most modern HIDSs are designed to run real-time, however, the file integrity scan of OSSEC, or complete system scans of anti-virus packages, are examples of IDSs which still run periodically.

5.4 Deployment Structure

We can also distinguish several different forms of deployment of IDSs[25]. These are based on how a single IDS functions, but also on how several IDSs might collaborate[24].

Stand-alone IDSs run independently on each node. They are limited to the information they can gather from their own node, and report their alerts locally. They are useful for systems which are directly monitored by an operator, e.g. personal computers.

Distributed IDSs monitor multiple nodes at the same time. They run a collection agent on each node, which communicates the data gathered to a central detection system.

As an example, the aforementioned OSSEC can run on a single host, but also provides the ability to monitor multiple hosts simultaneously through the use of collection agents on the hosts and a single server program running on a central machine performing the analysis.

Mobile IDSs use mobile agents – programs which can transmit themselves and their state to other machines in the network and resume execution – with different tasks to perform their analysis. They are most useful when targeted at virtualized environments and cloud computing.

Collaborative IDSs (CIDS) expand upon the previous concepts by running several different IDSs which compose the entire intrusion detection system. The key point is that different IDSs use different rules and algorithms, which are able to complement each other in detecting intrusions. They come in three forms:

- Centralized CIDSs have a single collaboration unit to which multiple IDSs report as detection units. The collaboration unit is responsible for analysing the alerts generated by the individual detection units. It is, however, a single point of failure in the intrusion detection system and limited in processing capacity.

- Hierarchical CIDSs introduce a hierarchy to the centralized model. The network of IDS nodes is divided into several groups. At the lowest layer in each group, IDSs function as detection nodes. They report their alerts to a collaboration unit equipped for both correlation and detection, which in turn reports to a unit in a higher level for analysis. The collaboration units are still sensitive points in this hierarchy[24]. This approach scales better than a centralized CIDS, but is still limited because of the small number of nodes in the upper hierarchy. Furthermore, since each level of correlation loses some information, the upper nodes are limited in their detection abilities.
- Fully distributed CIDSs attempt to address the problems still present in hierarchical CIDSs. They share and process all information from each IDS node in a truly distributed fashion[24]. The information sharing is done using e.g. peer-to-peer or multicast protocols. This fixes the scalability issue, however, several issues remain. First, to speed up detection time and to limit the amount of information that needs to be transferred, decisions are often made without waiting for alert information from all other nodes, so the detection accuracy is potentially less than a centralized or hierarchical system. Second, the alert information used in a fully distributed system is limited in scope, since it needs to be efficiently shared with all the participating nodes, introducing a new scalability problem while addressing the old one. Finally, a method to effectively balance the load between participating nodes is required. If a naïve method is chosen, participating nodes might flood a single node off the network simply by reporting back all information to that node.

5.5 Case Study: N.V. Nederlandse Gasunie

Recall the structure of the gas grid control network described in 2.5. The RTUs in use in the gas grid are *mixed, white-box, firm real-time systems*. They run on a version of Windows and are fully controllable. The PLCs used to control the physical objects in the real-world are *dedicated, grey-box, hard real-time systems*. If they miss their real-time deadlines, an entire gas plant might shut down in a fail-safe mode.

Gasunie has decided that it wants to monitor its SCADA network for anomalies (all anomalies, not just intrusions). For this purpose they have built, among other things, a host-based, statistical IDS which monitors the RTUs. These RTUs are overdimensioned for their tasks, and because of this, Gasunie is able to leverage the Windows Management Instrumentation (WMI)[28] infrastructure to report certain metrics to a central monitoring unit. This reporting is done over the same network as used by the command signals. The metrics used are based on internal research by Gasunie. There are fourteen metrics in total, the most important of which are the CPU usage, running processes, total and per-process memory usage,

network utilization, and disk utilization. The central monitoring unit, as its name implies, monitors the data received from the WMI processes on the RTUs. When a significant deviation is detected it raises an alert for an operator to investigate. This is quite similar to the proposed operation of IDES[22].

This system is host-based – it reads the metrics directly from the host – and runs on a real-time system. It is also anomaly-based, since it looks for deviations from the normal operation of the units being monitored.

The main reasons for the ease of implementation are

- the use of WMI, a well-documented and tested framework for data collection, and
- the overdimensioned nature of the system.

Users of WMI are warned not to let the data collection interfere with the performance of their system. The metrics selected by Gasunie have been chosen in part because the performance impact of their collection is negligible. The RTUs still meet their real-time constraints, and Gasunie is satisfied by the accuracy of the system. This proves that on a white-box system, at least, it is possible to run a statistical anomaly detection system, for the specific case of a collection engine running on the host reporting to a detection engine on another machine.

6 Influence of IDS on Real-Time Constraints

In this section we look at some possible ways a host-based IDS can negatively influence real-time systems. For this purpose we do not consider effectiveness of the IDS, only the cost of running it. Furthermore, we split the IDS into two distinct parts: the data collector and the data analyser. As the name implies, the data collector is concerned with collecting data and sending it to the data analyser, which in turn is responsible for analysing the data and generating alerts.

6.1 System Resource Influence on Runtime Behaviour

The runtime behaviour of a real-time system is influenced by three things: input data, process synchronization and system resources. Input data, such as sensor readings, user commands, or messages from other processes, are what ultimately determine what the program should do. Process synchronization is the first influence on how fast the process performs its actions, since a program waiting for another process cannot continue execution. How a process uses system resources is the second influence on process speed. Therefore, system resource usage, and in particular the *availability* of system resources are an important factor determining whether a real-time system meets its real-time constraints[42].

Examples of system resources are CPU time, system memory, network bandwidth, and shared states. The way the former three can affect program behaviour is obvious: if no CPU time is available, a program cannot execute at all, if no memory is available, a program cannot store any state information and has to cope with this situation, e.g. by using a garbage collector, and if no network bandwidth is available a program cannot send information onto the network. However, the notion of shared states requires some explanation.

On a multiprocess system a kernel is responsible for process management, memory management, interrupt behaviour, networking, filesystem access, user input processing, and other things. For the purposes of this discussion we will look at the behaviour of two operating system kernels on Unix-derived systems: the Linux kernel and the FreeBSD kernel. We will only distinguish between these when their behaviour is significantly different.

Apart from the kernel being responsible for dividing network bandwidth, system memory, and CPU time, it also maintains an internal state for each process running. This state consists of, among other things, the process priority, list of privileges, list of file descriptors (structures describing files on the filesystem the process has requested access to), count of CPU time used, and other bookkeeping data. We call these the kernel data structures of a process. It should be noted that the process itself has no direct access to these structures. They are written by the kernel when needed, e.g. when a process issues a system call to open a file

the file descriptor list is updated, and when a process uses CPU time the kernel updates the counter to reflect that. They are read, also by the kernel, whenever information from the bookkeeping is needed, e.g. when a process attempts to open a file the privileges section is checked to see whether the process should be allowed access to that file. However, they can also be read when another process requests information from them, e.g. when a process requests the CPU time another process has spent executing (the programs "ps" and "top" are two examples of such processes). Thus, although the state is kept in the kernel, we can view it as shared state between processes, accessed through the kernel.

Sometimes an action requires that the kernel locks some kernel data structures from being read. When a process performs a system call requesting access to a locked data structure, the call will block until the structure is unlocked, thus preventing the process from continuing. A lot of effort goes into keeping the kernel itself deadlock-free, ensuring that a system call will always, eventually, return. Most of the time, a locked data structure is only locked for a very short time, and only when it is written, thus causing only very slight delays. This can of course still cause a problem in very high-speed hard real-time systems, but that is not our main concern. Our concern is with large data structures that are locked while they are *read*, because an intrusion detection collector will only read data from other processes, and never explicitly cause writes.

A large shared data structure, such as a kernel data structure, or shared memory segment, that must be locked while it is read can cause significant delays in the process to which the data structure belongs, depending on how long the reading takes. Consider, for example, the case where a data collector monitors the files a real-time process is using: this requires reading the file descriptor table. In both the Linux kernel and the FreeBSD kernel, the file descriptor table is a simple C-style array, i.e. a single block of memory that must be completely copied and expanded when more space is required. This means that, in order to read from it, it must be locked, because otherwise the table might be moved to another section of memory whilst being read, resulting in reading invalid memory. If this happens often, the file descriptor table is locked a significant amount of time, which heightens the chance that the real-time process itself will block when attempting an action accessing the table, and causing it to break its real-time constraints.

To summarize, any time the data collector uses system resources, there is a potential to influence the real-time behaviour of the system. However, whereas in the case of CPU time, system memory, and network bandwidth these influences are obvious, there can be ways in which a data collector interferes with the real-time system which may not be immediately apparent. It is important to be mindful of these possibilities when designing our data collector.

6.2 Modelling Runtime Behaviour

We have not seen research dealing with the problem of access to shared state in the area of real-time systems, but there are methods to reason about real-time behaviour and system resources. Formal methods are often used for this purpose. Certain specification methods allow formal verification of systems, which is a wise practice in real-time and critical systems which, above all, need to be reliable and correct. Real-time system specifications are often implementation-independent, specifying the system in a temporal logic such as linear-time temporal logic[64], computational tree logic[20], or graphical interval logic[67]. Therefore, these specifications are not concerned with actual system resources, and the logic offers no way to model them apart from the notion of time[42]. Similarly, models of implementations of real-time systems often use a timed process calculus like duration calculus[35, 18] or ACSR[17, 43], or time-extensions on other process calculi like timed spi-calculus[34] or CCS+time[78]. Some process calculi do not use time, but do have a notion of priority. We would like a process calculus that takes into account both the notions of system resources and time in a manner which allows us to model the aforementioned behaviour of processes with shared system resources.

Only two process calculi, ACSR and SCRP, explicitly take into account system resources. Both these methods have a drawback making them unfit to use directly to model shared system resources in a real-time setting:

- SCRP does a good job of modelling shared system resources, and has the ability to model things like parallel access to resources, mutual exclusion, and handover of resources. However, it lacks a notion of time.
- ACSR, on the other hand, has a good model of time, interrupts, and process synchronization, but assumes that all resources are exclusive, i.e. resources can only be used by a single process at a time. This is a good way to model different processes sharing a CPU, however, it fails to capture the shared nature of e.g. kernel data structures which are only exclusive for some computations.

Extending SCRP for modelling real-time systems with these shared resources would require adding a time domain, whereas extending ACSR only requires us to extend its notion of resources to allow shared access. In the next section we propose this extension for ACSR with limited formalization.

ACNSR, or Algebra of Communicating Non-exclusive Shared Resources with Dense Time and Priorities, will be our proposed extension to ACSR. To understand this section, the source paper for ACSR, [17], must be used as a reference to the rules of the language.

To extend ACSR[17] to non-exclusive resources, we first add a finite set of concurrently usable resources, denoted by $\mathcal{S}$. A timed action is now defined by

a pair (t, A, B), where t is a time value, A is now an action from the domain $\mathcal{P}((\mathcal{R} \bigcup \mathcal{S}) \times \mathbb{N})$ and B is an action from the domain $\mathcal{P}(\mathcal{S} \times \mathbb{N})$. Resources in A and B can only be listed once per timed action. As a consequence, resources from $\mathcal{S}$ in A cannot be listed in B. Intuitively, A is the set of exclusively locked resources for a timed action, and B is the set of shareable resources.

Now, a timed action $X = (t, A, B)$ can only proceed if all resources from both sets A and B are available. However, a resource is only unavailable if it is listed in a concurrently executing A-section of another action.

As an example, the action $(t, \emptyset, \{r, 5\})$ has a non-exclusive use of the resource $r \in \mathcal{S}$ at a priority 5 for t units of time. If it were to execute concurrently with $(t, \{r, 3\}, \emptyset)$, its execution would pause for t units of time while the second action has an exclusive use of r, *regardless* of priorities. In this model, therefore, exclusive use of a resource always preempts non-exclusive use. It is up to the process specification to deal with this preemption.

To capture these alterations, we make some changes to the syntax and transition system of ACSR. The grammar describing the syntax is changed to

$$P ::= NIL | (t, A, B) : P | (a, n).P | P + P | P \parallel P |$$
$$P \triangle_t P | P \rhd P | [P]_I | P \backslash F | recX.P | X$$

The change to the meaning of the syntax is self-explanatory. The changes to the structured transition system are as follows:

ActI $\dfrac{}{(a,n).P \xrightarrow{(a,n)} P}$	
ActT1 $\dfrac{}{(t,A,B) : P \xrightarrow{(t,A,B)} P}$	$(t > 0)$
ActT2 $\dfrac{}{(t,A,B) : P \xrightarrow{(t',A,B)} (t - t', A, B) : P}$	$(0 < t' < t)$

The only change is the inclusion of B in $ActT1$ and $ActT2$.

The *Choice* operators are unchanged, since they will use our updated $ActT1$ and $ActT2$ rules.

The most important change is in the $ParT$ operator, since this one establishes the conditions in which two processes may execute in parallel.

$$\text{ParT}\ \frac{P \xrightarrow{(t,A_1,B_1)} P' \quad Q \xrightarrow{(t,A_2,B_2)} Q'}{P \parallel Q \xrightarrow{(t,A_1\cup A_2,B_1\cup B_2)} P' \parallel Q'}\ \rho(A_1)\cap\rho(A_2) = \rho(A_1)\cap\rho(B_2) = \rho(A_2)\cap\rho(B_1) = \emptyset$$

The main change here is in the condition:

$$\rho(A_1)\cap\rho(A_2) = \rho(A_1)\cap\rho(B_2) = \rho(A_2)\cap\rho(B_1) = \emptyset$$

This ensures that there is no overlap between locked and unlocked resources. If any resource in use by both processes appears anywhere in the A-sections, they cannot continue in parallel. Unless one of the processes provides another way to continue from this situation, without using that resource, the processes will deadlock.

The other *Par* rules remain unchanged, since we are not changing the behaviour of events.

The only *Timeout* rule we need to change is *TimeoutCT*:

$$\text{TimeoutCT}\ \frac{P \xrightarrow{(t',A,B)} P'}{P \triangle_t Q \xrightarrow{(t',A,B)} P' \triangle_{t-t'} Q}\ 0 < t' \leq t$$

We do not need to change any *Except* or *Res* rules. *CloseT*, however, needs adaptation:

$$\text{CloseT}\ \frac{P \xrightarrow{(t,A_1,B_1)} P'}{[P]_I \xrightarrow{(t,A_1\cup A_2\cup B_2,B_1-B_2)} [P']_I}\ A_2 = \{(r,0)\,|r \in I - \rho(A_1)\}, B_2 = B_1 \cap I$$

Apart from adding resource pairs for unutilized resources, this also moves resources from the shared set B to the exclusive set A if they occur in I. Recursion can stay the same.

We also need to slightly adapt the way priorities are handled. Case 2 and 3 stay the same, however, we need to adapt case 1. β preempts α if both $\alpha = (t, A_1, B_1)$

and $\beta = (t', A_2, B_2)$ are timed actions in $\mathcal{D}_R$, with

$$\begin{aligned} &(\rho(A_2) \subseteq \rho(A_1 \cup B_1)) \\ \bigwedge &(\forall r \in \rho(A_1).\pi_r(A_1) \leq \pi_r(A_2)) \\ \bigwedge &(\exists r \in \rho(A_2).\pi_r(A_1) < \pi_r(A_2)) \end{aligned}$$

This is only a slight change, which states that β also preempts α in the case that it tries to acquire an exclusive resource which is used by α as a non-exclusive resource, *regardless* of its priority, if β also uses all of α's exclusive resources at at least the same priority levels. In other words, locking always supersedes sharing.

We do not prove correctness of this extension of ACSR in this thesis. However, we do use it to model a small part of a fictitious version of TenneT's grid safety systems.

An example model using our expanded version of ACSR can help us check some timing properties of the grid safety systems. Recall the real-time constraints for the grid safety systems using distance measurement:

- less than 1ms to perform the measurement,
- less than 1ms to determine the fault,
- 2ms to signal the circuit breaker it should trigger, and
- 10-50ms for the circuit breaker to trigger.

We choose distance measurement so that we do not have to include a repeating process sending measurements. For simplicity of the model, we ignore all resources except a shared kernel structure *shared* which must be accessed when the safety system triggers, and a sensor m on which the measurement is performed. We ignore the CPU setup of the system because modelling the interleaving behaviour of processes is a complex and, in this case, unnecessary addition. The only reason why a process would not be able to continue is the unavailability of either m or *shared*. 1 time unit is defined to be 10 microseconds.

G is the grid safety measurement process. It performs a measurement every 100 microseconds, so every 10 time units. The measurement itself takes 10 microseconds, so 1 time unit. Recall from [17] that $\delta(P)$ is shorthand for

$$((\infty, \emptyset) : NIL) \rhd P$$

which means "idle until P is started".

$$G = (1, \{m, 1\}, \emptyset) : (((\overline{fault}, 5).(alert, 5).G) + ((9, \emptyset, \emptyset) : G))$$

Intuitively, G performs a measurement, using the sensor m exclusively. Then, if a fault is detected, signalled by synchronization on $fault$, it will start the contingency process H by synchronizing on $alert$. If no fault is detected, it will idle for 9 time units (90 microseconds), after which G restarts.

H is the contingency process:

$$\begin{aligned} H =& \delta((\overline{alert}, 5).H') \rhd H'' \\ H' =& \delta((100, \emptyset, \{shared, 5\}) : (trigger, 5).\delta((\overline{alert}, 5).H')) \\ H'' =& H' \rhd H'' \end{aligned}$$

This is a rather complicated construction. H is used to begin the process in the correct state. It waits idle until an alert is raised. Then, it starts H'. The exception mechanism already present in H is never used when idling in δ.

H' idles until $shared$ can be accessed. Then it uses $shared$ for 100 time units, signals the circuit breaker through the event $trigger$ (not modelled further), and then idles again until another alert is raised.

If the access of $shared$ in H' is at any point preempted by another process, the exception clause is used. H'' is used to reintroduce a copy of H'. This effectively restarts H' from the beginning. It also reintroduces the exception clause, in order to catch any future preemptions.

F is a process we use to simulate the introduction of a fault in the grid.

$$F = (20, \emptyset, \emptyset) : \delta((fault, 1).\emptyset)$$

It starts idling, and after 20 time units it will attempt to synchronize with G through $fault$, after which it will once again idle.

We will use the notation $=_t$ and $\Rightarrow_t$ to denote that t time units have passed since the start of computation at every transition or rewrite.

The complete model is

$$\begin{aligned} S =& _0(F \parallel G \parallel H)\backslash\{fault, alert\} \\ \\ =& _0(((20, \emptyset, \emptyset) : \delta((fault, 1).\emptyset)) \\ & \quad \parallel ((1, \{m, 1\}, \emptyset) : (((\overline{fault}, 5).(alert, 5).G) + ((9, \emptyset, \emptyset) : G))) \\ & \quad \parallel H)\backslash\{fault, alert\} \end{aligned}$$

H, at this point, is still idling. So we reduce this to

$$\begin{aligned} \Rightarrow_1 & (((19, \emptyset, \emptyset) : \delta((fault, 1).\emptyset)) \\ & \quad \parallel (((\overline{fault}, 5).(alert, 5).G) + ((9, \emptyset, \emptyset) : G)) \\ & \quad \parallel H)\backslash\{fault, alert\} \end{aligned}$$

This cannot be reduced by $ActI$ since $fault$ is restricted. So we get

$$\Rightarrow_1(((19,\emptyset,\emptyset) : \delta((fault,1).\emptyset)) \parallel ((9,\emptyset,\emptyset) : G) \parallel H)\backslash\{fault, alert\}$$

$$\Rightarrow_{10}(((10,\emptyset,\emptyset) : \delta((fault,1).\emptyset)) \parallel G \parallel H)\backslash\{fault, alert\}$$

By the same steps, this can be reduced to

$$\Rightarrow_{20}(\delta((fault,1).\emptyset) \parallel G \parallel H)\backslash\{fault, alert\}$$

$$\Rightarrow_{21}(\delta((fault,1).\emptyset) \parallel (((\overline{fault},5).(alert,5).G) + ((9,\emptyset,\emptyset) : G)) \parallel H)\backslash\{fault, alert\}$$

The actual fault was introduced after 20 time units. Note that 1 time unit has elapsed since F reached the state in which it is waiting to signal $fault$, simulating a possible small delay between manifestation and detection of the fault. So it has been 10 microseconds since the fault was introduced. The model can now, according to priorities and restrictions, be reduced to

$$\Rightarrow_{21}((\emptyset) \parallel ((alert,5).G) \parallel H)\backslash\{fault, alert\}$$

The fault has been detected. F is now an idle process, so we can ignore it in the further analysis.

$$=_{21}(((alert,5).G) \parallel (\delta((\overline{alert},5).H') \rhd H''))\backslash\{fault, alert\}$$

$$\Rightarrow_{21}(G \parallel (H' \rhd H''))\backslash\{fault, alert\}$$

$$=_{21}(((1,\{m,1\},\emptyset) : (((\overline{fault},5).(alert,5).G) + ((9,\emptyset,\emptyset) : G)))$$
$$\parallel (\delta((100,\emptyset,\{shared,5\}) : (trigger,5).\delta((\overline{alert},5).H')) \rhd H'')\backslash\{fault, alert\}$$

So now the contingency process H' has started, and G is running its normal scans. Since we have seen that G, when not faced with a fault, loops every 10 time units, the instruction in H' takes a multiple of 10 time units (100), and since there is no process capable of interrupting H' in this case, we are going to skip to the end of computation for the current instruction of H':

$$\Rightarrow_{120}(((1,\emptyset,\emptyset) : G)$$
$$\parallel ((1,\emptyset,\{shared,5\}) : (trigger,5).\delta((\overline{alert},5).H') \rhd H''))\backslash\{fault, alert\}$$

The next step will cause the bottom process to send the *trigger* signal. Note that, since the introduction of the fault, only 100 time units have passed.

$$\Rightarrow_{121}(G \parallel ((trigger, 5).\delta((\overline{alert}, 5).H') \rhd H''))\backslash\{fault, alert\}$$

$$\Rightarrow_{121}(G \parallel (\delta((\overline{alert}, 5).H') \rhd H''))\backslash\{fault, alert\}$$

$$=_{121}(G \parallel H)\backslash\{fault, alert\}$$

101 time units have passed since the introduction of the fault at 20 time units. It was detected within 10 microseconds, and 100 time units later, exactly 1 millisecond, the signal was sent to trip the circuit breaker. This fictional system, therefore, meets the real-time demands for the grid safety systems. We will now analyse the possible consequences of adding an IDS sensor.

Adding an intrusion detection sensor which collects some information every few milliseconds to this model requires the addition of a process. Let us suppose that it needs access to the shared segment which is also used by the contingency program H', *shared*. For the sake of simplicity, we ignore other collection activities.

We model the collection engine with a process C:

$$C = \delta((100, \emptyset, \{shared, 1\}) : \delta((100, \{report\}, \emptyset) : C))$$

This process idles until it can access the *shared* resource in a non-exclusive way, then idles until it can access the *report* resource exclusively to report its findings to the detection engine, then repeats. The other processes remain unchanged:

$$\begin{aligned}
F =& (20, \emptyset, \emptyset) : \delta((fault, 1).\emptyset) \\
G =& (1, \{m, 1\}, \emptyset) : (((\overline{fault}, 5).(alert, 5).G) + ((9, \emptyset, \emptyset) : G)) \\
H =& \delta((\overline{alert}, 5).H') \rhd H'' \\
H' =& \delta((100, \emptyset, \{shared, 5\}) : (trigger, 5).\delta((\overline{alert}, 5).H')) \\
H'' =& H' \rhd H''
\end{aligned}$$

The complete model is now

$$S =_0 (F \parallel G \parallel H \parallel C)\backslash\{fault, alert\}$$

$$\begin{aligned}
=_0 (&((20, \emptyset, \emptyset) : \delta((fault, 1).\emptyset)) \\
&\parallel ((1, \{m, 1\}, \emptyset) : (((\overline{fault}, 5).(alert, 5).G) + ((9, \emptyset, \emptyset) : G))) \\
&\parallel H \\
&\parallel \delta((100, \emptyset, \{shared, 1\}) : \delta((100, \{report\}, \emptyset) : C))))\backslash\{fault, alert\}
\end{aligned}$$

Since none of the processes running need exclusive access to *shared*, we fast-forward to the introduction of the fault in the grid:

$$\begin{aligned}
&\Rightarrow_{20}(\delta((fault,1).\emptyset) \parallel G \parallel H \\
&\qquad \parallel ((80,\emptyset,\{shared,1\}) : \delta((100,\{report\},\emptyset) : C)))\backslash\{fault,alert\}
\end{aligned}$$

$$\begin{aligned}
&\Rightarrow_{21}(\delta((fault,1).\emptyset) \\
&\qquad \parallel (((\overline{fault},5).(alert,5).G) + ((9,\emptyset,\emptyset) : G)) \\
&\qquad \parallel H \\
&\qquad \parallel ((79,\emptyset,\{shared,1\}) : \delta((100,\{report\},\emptyset) : C)))\backslash\{fault,alert\}
\end{aligned}$$

There is no real change from the earlier sequence of events yet. We process the instantaneous events to get

$$\begin{aligned}
&\Rightarrow_{21}(G \parallel (H' \rhd H'') \\
&\qquad \parallel ((79,\emptyset,\{shared,1\}) : \delta((100,\{report\},\emptyset) : C)))\backslash\{fault,alert\}
\end{aligned}$$

$$\begin{aligned}
&=_{21}(((1,\{m,1\},\emptyset) : (((\overline{fault},5).(alert,5).G) + ((9,\emptyset,\emptyset) : G))) \\
&\qquad \parallel (\delta((100,\emptyset,\{shared,5\}) : (trigger,5).\delta((\overline{alert},5).H')) \rhd H'') \\
&\qquad \parallel ((79,\emptyset,\{shared,1\}) : \delta((100,\{report\},\emptyset) : C)))\backslash\{fault,alert\}
\end{aligned}$$

The contingency process H' has started. The first timed instruction in H' after idling requires access to *shared*. Since both H' and C only require non-exclusive access, both processes can continue. We skip ahead 79 time units, since that is the remaining amount of computation in the current action of C.

$$\begin{aligned}
&\Rightarrow_{100}(((1,\emptyset,\emptyset) : G) \\
&\qquad \parallel (((21,\emptyset,\{shared,5\}) : (trigger,5).\delta((\overline{alert},5).H')) \rhd H'') \\
&\qquad \parallel ((100,\{report\},\emptyset) : C))\backslash\{fault,alert\}
\end{aligned}$$

report is free for use, so C can continue:

$$\begin{aligned}
&\Rightarrow_{120}(((1,\emptyset,\emptyset):G)\\
&\qquad \parallel (((1,\emptyset,\{shared,5\}):(trigger,5).\delta((\overline{alert},5).H'))\rhd H'')\\
&\qquad \parallel ((80,\{report\},\emptyset):C))\backslash\{fault,alert\}
\end{aligned}$$

$$\begin{aligned}
&\Rightarrow_{121}(G \parallel (((trigger,5).\delta((\overline{alert},5).H'))\rhd H'')\\
&\qquad \parallel ((79,\{report\},\emptyset):C))\backslash\{fault,alert\}
\end{aligned}$$

$$\Rightarrow_{121}(G \parallel (\delta((\overline{alert},5).H')\rhd H'') \parallel ((79,\{report\},\emptyset):C))\backslash\{fault,alert\}$$

$$=_{121}(G \parallel H \parallel ((79,\{report\},\emptyset):C))\backslash\{fault,alert\}$$

As before, 101 time units have passed since the introduction of the fault. It was detected within 10 microseconds, and exactly 1 millisecond later the signal was sent to trip the circuit breaker, so the real-time constraints are still met. From now on, the process H will once again idle, and G loops every 10 time units.

$$\Rightarrow_{200}(((1,\emptyset,\emptyset):G) \parallel H \parallel C)\backslash\{fault,alert\}$$

After this, C will repeat its collection cycle, G will continue its measurement cycle, and H will remain idle. This model does not take into account the computation time required to run C, otherwise we would likely be able to see some timing differences. However, adding computation requires us to model an exclusive CPU and idling possibilities for almost every action, which we think is not required to illustrate the most important point: what happens if C, instead of shared access, requires exclusive access to *shared*?

To illustrate this, we now model C as

$$C = \delta((100,\{shared,1\},\emptyset):\delta((100,\{report\},\emptyset):C))$$

Until H starts running, nothing changes, except that C is using *shared* exclusively. Then, when the fault has been detected and H has been signalled:

$$\begin{aligned}
&\Rightarrow_{21}(G \parallel (H'\rhd H'')\\
&\qquad \parallel ((79,\{shared,1\},\emptyset):\delta((100,\{report\},\emptyset):C)))\backslash\{fault,alert\}
\end{aligned}$$

$$\begin{aligned}
&=_{21}(((1,\{m,1\},\emptyset):(((\overline{fault},5).(alert,5).G)+((9,\emptyset,\emptyset):G)))\\
&\qquad \parallel (\delta((100,\emptyset,\{shared,5\}):(trigger,5).\delta((\overline{alert},5).H'))\rhd H'')\\
&\qquad \parallel ((79,\{shared,1\},\emptyset):\delta((100,\{report\},\emptyset):C)))\backslash\{fault,alert\}
\end{aligned}$$

We can see that a transition in which both H' and C continue with non-idle actions is not possible. C has priority by the preemption rules, so the only transition from this situation is with H' idling in δ. H' remains idle for 79 time units:

$$\begin{aligned}\Rightarrow_{100}(((1,\emptyset,\emptyset) : G) \\ \parallel (\delta((100,\emptyset,\{shared,5\}) : (trigger,5).\delta((\overline{alert},5).H')) \rhd H'') \\ \parallel \delta((100,\{report\},\emptyset) : C))\backslash\{fault,alert\}\end{aligned}$$

Now, finally, *shared* can be used in a transition in which both H' and C are not idling:

$$\Rightarrow_{200}(((1,\emptyset,\emptyset) : G) \parallel (((trigger,5).\delta((\overline{alert},5).H')) \rhd H'') \parallel C)\backslash\{fault,alert\}$$

$$\Rightarrow_{200}(((1,\emptyset,\emptyset) : G) \parallel H \parallel C)\backslash\{fault,alert\}$$

So this time, from fault detection to signalling took 179 time units, or 1.79ms. Still within the constraint of 2ms, but only barely so. However, there is a way in which arguably the same processes break their real-time constraint, which can be reached by simply changing the time at which the fault is introduced: after 120 time units instead of 20 time units. We redefine F as

$$F = (120,\emptyset,\emptyset) : \delta((fault,1).\emptyset)$$

The process proceeds normally for the first 100 time units: G loops every 10 time units, H idles, C completes the first timed action which uses *shared* exclusively, and F completes the first part of its idle state. The processes now look like this:

$$\begin{aligned}\Rightarrow_{100}(((20,\emptyset,\emptyset) : \delta((fault,1).\emptyset)) \parallel G \parallel H \\ \parallel \delta((100,\{report\},\emptyset) : C))\backslash\{fault,alert\}\end{aligned}$$

This is very similar to their previous initial state, however, it should be noted that C is no longer accessing the resource *shared*. We step ahead 21 time units to the detection and signalling of the fault:

$$\Rightarrow_{121}(G \parallel (H' \rhd H'') \parallel ((79,\{report\},\emptyset) : C))\backslash\{fault,alert\}$$

which looks again very similar to the earlier state. Notice, however, that the fault was introduced at time unit 120, not 20. *shared* is now available to a transition

in which H' uses it non-exclusively, so we get, after another 79 time units:

$$\begin{aligned}
&\Rightarrow_{200}(((1,\emptyset,\emptyset):G)\\
&\qquad \parallel (((21,\emptyset,\{shared,5\}):(trigger,5).\delta((\overline{alert},5).H')) \rhd H'')\\
&\qquad \parallel C)\backslash\{fault,alert\}
\end{aligned}$$

$$\begin{aligned}
&=_{200}(((1,\emptyset,\emptyset):G)\\
&\qquad \parallel (((21,\emptyset,\{shared,5\}):(trigger,5).\delta((\overline{alert},5).H')) \rhd H'')\\
&\qquad \parallel \delta((100,\{shared,1\},\emptyset):\delta((100,\{report\},\emptyset):C)))\backslash\{fault,alert\}
\end{aligned}$$

By the parallelization rules, *shared* is now not available for a transition in which both H' and C continue in the timed action using *shared*. Furthermore, by the preemption rules, the timed action of C preempts the one H' is currently in, so the only transition from this state is the one where H' uses the exception rule:

$$\begin{aligned}
&\Rightarrow_{200}(((1,\emptyset,\emptyset):G)\\
&\qquad \parallel H''\\
&\qquad \parallel \delta((100,\{shared,1\},\emptyset):\delta((100,\{report\},\emptyset):C)))\backslash\{fault,alert\}
\end{aligned}$$

$$\begin{aligned}
&=_{200}(((1,\emptyset,\emptyset):G)\\
&\qquad \parallel (H' \rhd H'')\\
&\qquad \parallel \delta((100,\{shared,1\},\emptyset):\delta((100,\{report\},\emptyset):C)))\backslash\{fault,alert\}
\end{aligned}$$

Now, H' will remain idle until *shared* becomes available again, this takes 100 time units. Then, it will be able to continue for another 100 time units, and then

trigger the circuit breaker:

$$\Rightarrow_{300}(((1,\emptyset,\emptyset):G)$$
$$\quad \parallel (\delta((100,\emptyset,\{shared,5\}):(trigger,5).\delta((\overline{alert},5).H')) \rhd H'')$$
$$\quad \parallel (\delta((100,\{report\},\emptyset):C)))\backslash\{fault,alert\}$$

$$\Rightarrow_{400}(((1,\emptyset,\emptyset):G)$$
$$\quad \parallel (((trigger,5).\delta((\overline{alert},5).H')) \rhd H'')$$
$$\quad \parallel C)\backslash\{fault,alert\}$$

$$\Rightarrow_{400}(((1,\emptyset,\emptyset):G)$$
$$\quad \parallel (\delta((\overline{alert},5).H') \rhd H'')$$
$$\quad \parallel C))\backslash\{fault,alert\}$$

$$=_{400}(((1,\emptyset,\emptyset):G) \parallel H \parallel C)\backslash\{fault,alert\}$$

Although this looks identical to the previous final state, the time from fault detection to signalling was this time $79 + 100 + 100 = 400 - 121 = 279$ time units, which corresponds to 2.79 milliseconds, so this has broken the real-time constraint. We see that not only the fact that shared data is being accessed, but also the nuances of access timing, can play a role in breaking the time constraints of a real-time process. We think our proposed enhancement to ACSR is another step towards being able to accurately model these interactions, and so gain a better understanding of the timing behaviour of real-time systems on mixed multi-process systems.

6.3 The Cost of System Call Tracing

Of all the host-based intrusion detection methods considered for our solution, system call tracing is likely the most expensive one in terms of real-time process slowdown. In both [80] and [79], Zhang et al. claim that systems using system-call tracing suffer from "huge performance degradation even when operating at system call level". We have run several tests on some simple Linux programs using the strace 4.8 package[7], and have indeed found that in our worst cases the program slows down by a factor of 5. Obviously, this depends on the number of system calls being used. Our tests, however, as well as the initial work on system call tracing in [27], use strace, which is in turn based on ptrace. ptrace is a method of

[7] http://sourceforge.net/projects/strace/

system call tracing which first appeared in Version 7 of AT&T Unix. Every time a process being traced by another process using ptrace receives a signal, that process stops execution and waits until the tracing process permits its to continue. The context switching overhead required by this mechanism is largely responsible for the slowdown we see in straced processes[41].

There is also a ptrace-based method for library call tracing, another possibility for host-based intrusion detection, called ltrace. However, the performance degradation for ltrace is similar to, if not worse than, the degradation for strace.

The ptrace method, however, is not the only way to do system call tracing. In fact, it is arguably the least suitable way if all we are interested in is a report of the system calls performed and their arguments. Since the ptrace method halts the process being traced until the tracing process can allow it to continue, the tracing process would always need to satisfy the same real-time guarantees as the tracee.

In [41] the authors present an alternative method of system call tracing: kernel-based tracing. Instead of relying on user-space programs communicating through signals, the tracing system is built directly into the operating system kernel. This leads to significantly lower overhead (25 microseconds per system call for the ptrace method, 0.16 microseconds per system call for the kernel-based method)[8].

We therefore think that a system call based approach does not necessarily cause a noticeable slowdown, as long as the operating system kernel has support for this purpose built in.

6.4 Bookkeeping Collection

An HIDS monitoring bookkeeping, like the amount of CPU used, which processes are running, the memory used by each process, etc., must collect these statistics somehow. For Windows there is the Windows Management Instrumentation. For Unix-derived systems there are several methods, of which the two main methods are currently the /proc virtual filesystem (VFS) and the sysctl interface.

The /proc virtual filesystem exposes a filesystem-like interface to the kernel bookkeeping structures. In listing 1 a listing of /proc on a typical Linux system can be seen. The numbered folders contain the bookkeeping for each running process, the numbers are the process IDs. The named files and folders contain information not associated with a single process, like /proc/meminfo, which, when read, provides a detailed report on the memory state of the system (46 separate measurements), or /proc/net/, which contains details of the network configuration and state.

[8] Unfortunately the system as proposed in the 2007 paper [41] was never accepted as such in the mainstream Linux kernel. However, the concept of kernel-based probing has been merged in 2012[9]and could still be used to implement kernel-based tracing.

[9] See https://lwn.net/Articles/499190/ for details.

Listing 1: Output of 'ls /proc' on a typical Linux system

```
# ls /proc
1       asound        execdomains  locks         stat
10      buddyinfo     fb           meminfo       swaps
102     bus           filesystems  misc          sys
103     cgroups       fs           modules       sysrq-trigger
1057    cmdline       interrupts   mounts        sysvipc
10741   config.gz     iomem        mtrr          timer_list
1094    consoles      ioports      net           timer_stats
10949   cpuinfo       irq          pagetypeinfo  tty
10960   crypto        kallsyms     partitions    uptime
10966   devices       kcore        sched_debug   version
11      device-tree   key-users    schedstat     vmallocinfo
11005   diskstats     kmsg         scsi          vmstat
1101    dma           kpagecount   self          zoneinfo
1102    dri           kpageflags   slabinfo
acpi    driver        loadavg      softirqs
```

The nature of the /proc VFS is such that when it is not observed, it does not use system resources. However, when read, the kernel collects the requested data and provides it as though it is a file. This causes overhead in a few different ways:

- First, reading a /proc file requires a system call to open the file. This creates and returns a file descriptor for the file.
- Second, the data requested is collected by the kernel. The overhead this causes depends on the nature of the data. Some items can be quickly read from kernel data structures, other items require locking and larger reads.
- Third, the data is converted to a text representation.
- Fourth, the data is represented as a file. This requires the process requesting it to issue several more system calls to read.
- Finally, since the data is presented as text, i.e. strings, it must be parsed by a text parser and some post-processing might be required to convert it to the desired format, e.g. integers.

The 'top' system monitor utility accesses the /proc filesystem to collect its data. When we run 'top' in a resource intensive mode, refreshing every 100ms instead of the usual 2s, and on a high priority, top uses 100 times more CPU resources than it normally does. This illustrates that intensive reading from /proc can potentially cause a quite heavy system load.

The sysctl interface is a newer system call interface which provides a direct access mechanism to the information that used to be provided by the /proc VFS. On BSD systems, this eliminates most of the overhead caused by the /proc filesystem: only a single system call is needed, the data is returned as a result of that system

call, and the data can be returned in its original type, so no conversion is required. The internal data collection, however, is unchanged.

On Linux systems, however, the sysctl interface is currently merely a wrapper around the /proc VFS, so provides no performance benefits over using the /proc VFS directly.

These two mechanisms show that, although small, reading the bookkeeping information required for some forms of host-based intrusion detection does incur a slight overhead, which must be kept in mind when designing and testing an HIDS. Collection of this information should be limited so that the total overhead incurred does not impede functioning of the system.

7 A Host-Based Intrusion Detection System for Real-Time Systems

In this section we will explain our proposed structure for a host-based intrusion detection system for real-time systems. First we will list our requirements. Second, we will discuss some different deployment structures, and select the deployment structure most likely to be effective in the situation of the electrical grid. Third, we will describe what data sources and detection methods to use in the case of different types of real-time systems. Finally, we will discuss some challenges rising from our design.

7.1 Requirements

The primary requirement for our intrusion detection architecture is that it does not interfere with the real-time running of the monitored system. This implies several constraints. We have seen different types of real-time systems. Several of these systems might be powerful enough to run an IDS detection engine themselves. Other systems, however, can only perform the collection on the host and need the detection to run on another system.

We want intrusion detection to happen in real time, i.e. as the intrusion happens. If, for whatever reason, this is not possible, we want the detection to happen as close in time to the intrusion as possible. For instance, if the system batch-processes a listing of all files on the filesystem in seconds, and it takes several minutes to gather this list, then the next collection round should start as soon as the previous one is completed.

Except when you run the entire host as a specially instrumented virtualized system, there will always be a collection engine running on the host. Even when using a virtualized system, the collection engine ultimately runs on the same hardware as the host. Therefore, the collection engine is always constrained in how much processing time and memory it can use, based on the amount the real-time processes on the host need. Ideally, a collection engine has a low memory footprint and low collection overhead, while collecting several different data streams.

On-host detection engine
If the detection engine of the system is on the host being monitored, it cannot be very computationally intensive. Furthermore, it will be constrained in memory usage, either because there is a limited amount of memory available, or because a large memory footprint might force the system to unintentionally swap out vital parts of the real-time process. However, in some cases, the cost of hardware is not a big issue to either vendor or client, in which case using more powerful processing units and more memory is an option to enable the use of a “heavier” detection

engine. This is the case with the more specialized applications of real-time systems such as high-voltage substations.

The collection engine is not constrained in how much data it can collect other than the processing and memory constraints already mentioned, as long as the detection engine can keep up with the data flow.

Off-host detection engine
In the case of an off-host detection engine, the processing and memory constraints are not an issue any more. Advantages of an off-host detection engine are that the hardware can be tailored to the detection engine being used, there is no need to expand the capabilities of the hosts being monitored, and multiple hosts can report to the same detection engine, making it a distributed system. The main downside is that the detection engine becomes a single point of failure in the distributed IDS. However, we deem this an acceptable solution, considering that the alternative is not having an IDS at all for many systems.

The collection engine in such a situation is, however, not only constrained in just processing power and memory, but also in the amount of data that can be collected. Since the data must be sent over some kind of network link to the system running the detection engine, the collection rate of all attached systems combined cannot be more than that network link can handle. Furthermore, it is required to protect this data somehow, since tampering with the collected data might falsely trigger alarms, opening a new denial of service attack vector. We do not propose to encrypt the data, however, computing signatures might be a good way to mitigate tampering attacks. Of course, this computation costs additional resources and should be taken into account when deciding whether to protect data on each system. It might be acceptable to have some systems at the bottom of the hierarchy report unsigned data via a direct link to a detection engine. The tamper-proofing is then provided by the physical security of the direct link, instead of signatures over the data.

The fact that the detection engine does not run on the same host might free some processing power and memory for the collection engine.

7.2 Deployment Structure

An on-host detection engine is undesirable for a number of reasons. First, most detection engines are simply too computationally expensive to run on the same host as a real-time process. Unless the host is overdimensioned enough, most likely one of two things will happen: the detection engine will be unable to keep up with incoming data when the system is under load, or the detection engine will starve the real-time process of resources and cause it to break real-time constraints. The

former will cause it to miss attacks, the latter is unacceptable from a real-time point of view.

Second, if the system is being attacked, the attackers might first target the IDS itself. An off-host detection engine might be situated such that it is less vulnerable to direct attack. For instance, it could be situated in the network behind a firewall configured to only allow direct communication with the systems being monitored.

Third, an on-host detection engine requires that interaction with the real-time host takes place in order to change detection rules or parameters. Some of these actions might be computationally expensive, which, if done at the wrong time, directly interferes with the real-time constraints of the system. Furthermore, we want to minimize user interaction with these systems, to minimize the chances of mistakes being made, and because it adds a new way into the system which might be used as an attack vector. A collection engine might still need updates, e.g. when more metrics are selected for analysis, but this should be much less frequently required than rule updates for detection engines.

The resource usage of a collection engine is also more predictable than a detection engine, since it depends simply on the characteristics of its data collection methods used, and amount of data collected and sent to the detection engine. Therefore, we prefer a distributed system with an off-host detection engine.

A distributed system has been shown to work at Gasunie in section 5.5, and minimizes the problem of IDS interference with the host system to the data collector. Thus, we are free to pick the best detection engine for our purposes. Furthermore, a single detection engine can be used for multiple host systems, decreasing cost and potentially increasing detection accuracy because of the increase of available data to analyse.

However, based on existing research and anecdotal evidence, we believe that combining several different types of IDSs, on different kinds of systems within the grid, will lead to the best possible detection coverage and accuracy. Therefore, we propose using a

collaborative, hierarchical system with a similar structure to the electrical grid itself. Each type of system operating in the grid, such as the EMS network or the telecommunication backbone, should have its own IDS, as depicted in figure 6.

Each high-voltage station should have an IDS for the RTUs that service a single field. Each RTU runs one or several collection processes that transmit the data to the field's detection engine. These detection engines should then report their alerts to a correlation unit for that HVS, which in turn reports to a central unit in the grid control centre.

Another section of IDSs should be dedicated to the grid safety systems. Again, each grid safety sensor should run a collection process that transmits data of system

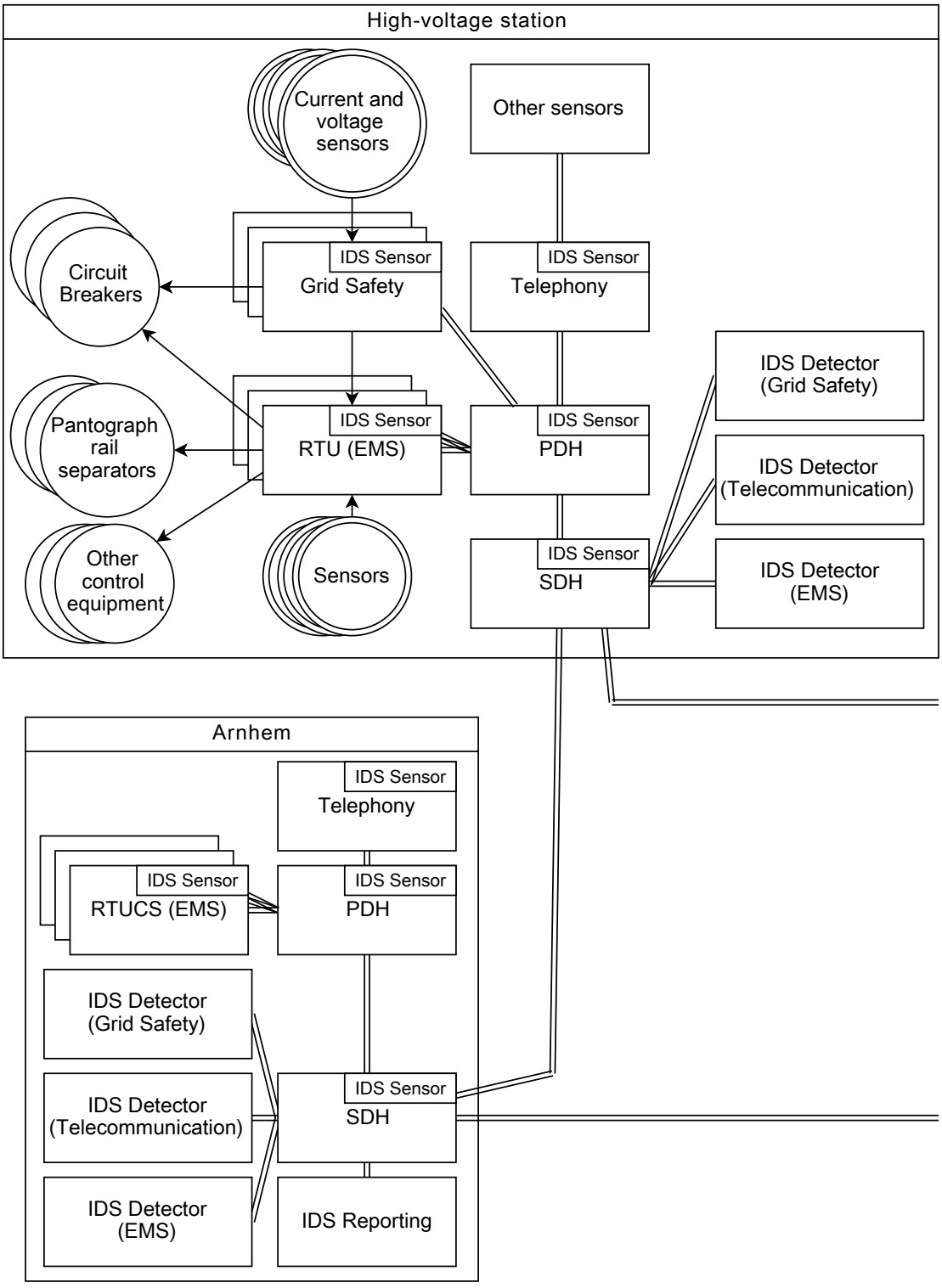


Figure 6: Schematic view of the network structure of TenneT with proposed IDS.

activity to a detection engine particular to the field, which can report to the same correlation unit as the RTUs.

Similarly, the infrastructure on which the EMS network runs should be monitored for intrusions as well. Each communication substation should run collection processes on the networking hardware. These processes would not, however, be monitoring any network traffic, but only local activity on the systems themselves.

Finally, we think that this infrastructure should be extended with network-based intrusion detection systems designed for SCADA networks and protocols, like the SilentDefense NIDS[14] or an approach based on an adaptation of Bro[47].

Based on the literature and the discussions with personnel working in this field, we think this deployment model would give a very complete view of the activity in the entire EMS and supporting systems, while still using a limited number of detection engines. However, we have not defined a set of strict metrics by which to score the options available to us. This is a possible avenue of future research.

7.3 Data Sources and Detection Models

We have discussed the state of the art in host-based intrusion detection in section 5. From the advantages and disadvantages of each technique, we have determined what, currently, would be our ideal detection model. It is a hybrid model, using both anomaly-based and signature-based detection, with the following techniques:

- Anomaly-based
  - system call tracing with hidden Markov models
  - system resource utilization with statistical learned models
  - system configuration change tracking with fixed limits
- Signature-based
  - presence of certain files on the filesystem
  - system call payload signatures

Whether all these techniques can be used on a system depends on the type of system, the resources available for collection, as well as the internal visibility. We motivate these choices below.

Anomaly-based detection is very well suited to a relatively static environment such as the grid EMS. The systems have very little interaction with humans, so there is less noise to deal with. The static and purposeful nature of the environment should yield high-quality models capturing most normal behaviour if trained over a dataset of a few weeks. Finally, behavioural changes are rare, so the high cost of retraining models is mitigated.

System call tracing is a well-studied method of modelling process behaviour and has good detection performance. This performance appears to be even better

when used with hidden Markov models, so this is our first detection approach. It should be noted that this approach is not suited to an entire host, rather, each process on the host should be monitored separately. During execution of each process, the system calls performed are collected and sent to the detection engine. The detection engine determines the probability that a certain system call, or sequence of system calls, is encountered in the current state of the HMM, and if the probability is sufficiently low enough an alert can be raised. To improve detection accuracy, a combination with a system call database as in [37] can be considered.

A statistical view of system resource utilization, as proposed by Denning[22], is expected to be a good measure for systems with low user interaction. We propose to use some of the same metrics Gasunie uses: CPU time, system memory, network bandwidth, disk I/O, and which files are being used. Regarding the last metric, however, we urge caution, for reasons explained in section 6: if a large number of files is in use, even collecting this information might have unintended consequences for the real-time process.

Discussions with personnel at TenneT have yielded the view that system configuration changes, especially in the case of the communication network, can be flagged as an anomaly by default, since they should happen few and far between. Configuration changes include configuration files on-disk, but also a system's hardware configuration. E.g. plugging in a USB mass storage device changes the hardware configuration. This, of course, also detects all legitimate configuration changes as an anomaly, however, the proposed mitigation is to report all intended configuration changes in advance to the IDS monitor.

Signature-based detection has proven itself valuable in mitigating standard attacks. It is also, computationally, fairly cheap. Therefore we consider a signature-based detection engine to be an important part in a complete solution, as a first line of defence against known attacks. There are at least two ways in which signature-based detection can be used on hosts.

First, some attacks leave distinct files or files with recognizable content on the filesystem. Scanning for these files is a good way of detecting these attacks, a tried and tested method used by virus scanners.

Second, some attacks use system calls with recognizable arguments. Scanning not only the system call sequences, but also their payloads for known patterns, can detect these attacks.

The collection of information for these detection systems can happen in several ways. The following list describes the possibilities for the different visibility of real-time systems.

- Black-box systems
  These systems cannot provide us with any useful information for our current design. Companies should work with the vendors of these systems to adapt them to be used in an intrusion detection system, by making them grey-box.
- Grey-box systems
  These systems are partially observable. The data that can be collected depends on the interface to the system. If e.g. the only possible access is through a web-interface where aggregate statistics are viewable, data collection is limited to these statistics and how often the system is able to process a web request. "Abusing" such an interface for this purpose might also put a heavy load on the system in terms of CPU and memory. In this case, the collection processes should run on the same system as the detection engine.
  We also consider a system on which the user has system access (e.g. is able to login to a shell through SSH) yet is restricted in what he can do on the system, a grey-box system. The user will be able to read some metrics directly, but will not have access to things like system call traces or file descriptor tables.
  Especially in these cases, companies should work with vendors of the system to expand data collection capabilities. Since the basic mechanisms are already in place, adding these capabilities should not be a great hurdle.
- White-box systems
  These systems are the systems where the user has full access, and are therefore fully observable.
  - Precisely dimensioned systems
    These systems are dimensioned such that, under heavy load, the real-time processes running on it will make their real-time deadlines without margin. Therefore, on this kind of system, any collection process must be restricted to the absolute lowest priority and use no mechanisms which might block another process from continuing execution. This means that both expensive implementations of system call tracing (such as ptrace-based methods) and reading file descriptor tables are not an option. However, measurements of (aggregate) CPU time, system memory, and network bandwidth are still possible. They should, however, be extensively tested. Tracking configuration changes should also still be possible. Scanning the filesystem for known attack signatures should only be done periodically, if at all.
  - Overdimensioned systems
    These systems have a fairly large safety margin for their real-time processes to meet their deadlines. A collection process on such a system is probably able to use all collection techniques such as system call tracing, measuring CPU time, memory used, network bandwidth and disk I/O, configuration changes, and monitoring of filesystem changes. Of course, before deployment such a collection system must still be tested.

Sending all this information to the detection engine takes both CPU time and network bandwidth. This, too, should be taken into account when building such a collection engine; tests are required to determine the bandwidth required for a certain metric. If a real-time system depends on network communication for its operation, as is the case for a part of the grid safety systems, data collection traffic should be given lower priority and the network link should be utilized only to a safe maximum, leaving enough room for the real-time system to communicate.

7.4 Challenges

There are several challenges which must be overcome to implement this system architecture:

- Both detection and collection software for these systems does not yet exist. Building working collection systems will often require vendor cooperation. Considering the long chain of development and quality control, it may be several years before many of the systems deployed in the EMS grid have data collection capabilities. Detection software is, fortunately, a somewhat easier case, as in our design the detection engine does not run on the hosts being monitored. Therefore, it could be developed in-house or contracted to a software development firm, and tested with the data from white-box systems for which we can perform our own collection and instrumentation.
- There exists no analysis of the effectiveness of this design in the context of EMS networks. Our design is based on deduction from existing research on different applications of these techniques. Although we are fairly confident that this is an effective design, we advise that such a system first be deployed on a testbed environment to evaluate its performance, both in the sense of detection capabilities as well as influence on the overall system performance of the EMS network.
- When deployed, this system will consist of a significant number of sensors and detection engines. Care should be taken to catalogue and maintain all these assets.
- We do not consider further processing of alerts of this system in the current design, mostly because the design itself has not been proven to work in this context. It might even be the case that such a system generates a low number of alerts and has a high detection accuracy, and further processing and correlation is not required to enhance detection accuracy.

8 Discussion and Future Work

8.1 Proposed Intrusion Detection Architecture

Security of Industrial Control Systems

We focus on the real-time systems as used in the Dutch energy distribution grid. However, we think the principles are applicable to industrial control systems in general: position as few components as possible on the host itself, preferably limited to a collection engine, and utilize the distributed nature forced by this choice to improve detection accuracy. Even the simplest version of an anomaly detection system, utilizing only a few aggregate metrics such as CPU time and memory utilization, has the potential to improve awareness of the operation of a network of industrial control systems.

However, we have not defined precise models or techniques by which to evaluate different architectures and metrics. We have some suggestions for this. Quantifying the precise requirements for industrial control systems is an important task that still needs to be done. This could be combined with prototyping our proposed architecture, as discussed below. Furthermore, determining detection accuracy requirements, especially the acceptable false positive rates, is important to decide whether a proposed system performs acceptably. Specification models of the systems involved would be useful to perform predictive calculations of timing behaviour or dead-lock potential when a sensor is added.

We think that adding intrusion detection to industrial control networks is an important step towards improving the state of security in this vulnerable sector. Host-based intrusion detection, however, is one of the last lines of defence, and should be part of a layered solution beginning at the edge of the network. We do not think, however, that host-based intrusion *prevention* systems are a viable option for ICSs, because of the impact that a single false positive would have: power outages, factories shutting down, important systems that simply refuse to do their job because they deem it suspect. Intrusion prevention has its place in these networks, but for now, it is at the edge, not at the host.

It is unfortunate that many of the systems within the electrical grid are black- or grey-box. We cannot test our proposed architecture on these systems without vendor support, and TenneT will not be able to roll out a host-based IDS on many of these systems because of these limitations. Companies should work with vendors to adapt these systems for use in intrusion detection systems. Furthermore, because of the relatively long production life of many industrial control systems, it will likely be years, if not decades, before we will start seeing broad adoption of host-based intrusion detection capabilities on these systems. It requires support from vendors, awareness, and most importantly, further research into this field. In that light, this thesis serves merely as a stepping stone to future research, some of which is outlined below.

Further Research on Detection Accuracy and Real-Time Constraints
We have based our proposal on interpretations of existing research applied to the situation of TenneT. No prototype has been built, so we have not been able to test our assumptions. However, we are confident in the accuracy of our analysis of the ways our proposed system might impact the real-time systems it runs on, and have provided a basis for future research to build on. Several prototypes of this system should be built and tested before it is considered for real-world deployment. These prototypes would focus separately on the two important aspects of an IDS in this setting: influence on real-time constraints and detection accuracy.

A problem with testing detection accuracy is that this is mostly done using existing datasets with known normal and anomalous samples, and that these datasets do not originate from a low-interaction setting such as the EMS/SCADA systems used here. We postulate that anomaly detection on low-interaction industrial control systems is more accurate than on high-interaction, "normal" computer systems. We base this belief on the static nature of these systems, and the assumption that there is less "noise" to deal with. To examine this, however, research on the proposed techniques on low-interaction settings is an important next step. Any effort to test detection accuracy on these systems will either need to obtain a dataset from these systems, or simulate such systems. Obtaining a dataset from these systems can only be done after testing the impact of the collection on the systems.

To test the influence of collection on real-time systems, a testbed of the same systems deployed in the EMS/SCADA network should be used. Different collection techniques could be tested on these systems, and this will lead to an accurate picture of the impact each technique has on the system. Another option is modelling the systems, adding the collection process to the model, and checking it for real-time compliance. This will most likely lead to quite complex models, unlike the ones used as demonstration in this thesis, and unless the system builders themselves supply the models they are also likely to be incomplete. Furthermore, the modelling effort should, in the end, also be followed by a test implementation.

This test implementation can also be used to obtain the datasets required to test the detection accuracy of different detection techniques. At first, the testbed will provide a set of normal traces to train the detection engines. After enough of these traces have been captured, anomalies can be introduced to provide the anomalous traces to test the detection engines with. After the collection processes have been proven not to break real-time constraints, it may be possible to deploy them on live systems. This will provide real-world datasets to test the detection systems with.

If such traces cannot be obtained, detection accuracy of these techniques may also be tested by using another low-interaction system. A simulation system could

even be built for this specific purpose. However, testing with real systems is preferred.

Possible Implementation – Case Study: Locamation B.V.
Locamation B.V.[2] is a Dutch company that produces complete high-voltage substation monitoring and control solutions in use by several RNBs. This places them in a position where they have a white-box view of all the systems involved in substation monitoring and control: the centralized control unit (CCU, equivalent in function to an RTU), the attached interface modules (sensors and control systems), the communication module for interfacing with the CCU from outside, and backup systems.

The CCU is based on a proprietary, Unix-derived real-time operating system called ARTOS. Application modules run on top of ARTOS, such as a real-time database, the IP networking stack, a remote user interface, and more. Development on this system started in the 1980s, and the hardware it runs on nowadays is very overdimensioned. There are already several layers of security in this system, but there is no intrusion detection system in place. However, it does have extensive logging capabilities and provides very detailed logs to users of the system. Some examples of metrics that can be logged are delay times of communication between the CCU and attached sensor and control modules, system loads, commands received and given, and many more. The logs are uploaded over a network link to an external system, similar to how a collection engine for an intrusion detection system would work.

Even without white-box access, these logs are already a very valuable source of information for an anomaly detection system. With white-box access, these logging capabilities could be extended to cover e.g. system call traces and other information we want to collect. Since the system is so overdimensioned, adding a dedicated application module to collect this information should be possible. This would effectively turn the existing logging system into a collection engine for an intrusion detection system. The presence of such an extensive logging engine on the CCU is an indication that it is possible to add intrusion collection capabilities to RTUs, provided they are mixed systems and have enough network bandwidth available.

The interface modules are of particular interest. They are FPGA-based, precisely dimensioned, dedicated, hard real-time systems. Adding collection capabilities to such a system requires alterations and additions to the real-time process itself, and thus to the hardware description for the FPGA. Since they are precisely dimensioned, it would also require bigger FPGAs. However, if we assume that the cost of this hardware is acceptable, an FPGA could be programmed to report not only sensor readings, or process commands, but also report the internal state of the process to the CCU. This could then be included in the logs provided to

the user. This would require extensive testing, however, because the frequency of measurements is 28kHz. This means one measurement every 35.7 microseconds is sent to the CCU. If not just the measurement, but also a report of internal state is sent at this frequency, it might prove to be a too heavy load on communication with the CCU, and on processing by the CCU. In this case, it would be interesting to explore possible ways to minimize the amount of information sent. For example, it might be the case that the internal state of the FPGA is very static and can be represented by a simple finite state machine. Then the collection engine could be extended with specification-based detection logic, only sending information to the CCU when a deviation from the normal path in the finite state machine is detected. We do not know if this could be achieved without breaking the real-time constraints on these systems, or how much additional area this would require. This could be a potentially interesting topic for future study.

Another point of interest is the idea that physical properties of the grid being monitored could be very valuable to an intrusion detection system. For example, random noise in current and voltage could indicate that a system is functioning normally. If the noise is no longer random, or disappears, this would be a good indication that measurements are being tampered with. The interface modules send all data to the CCU, and collection of this information could therefore be done by the (overdimensioned) CCU. It does not require additions to the interface modules, or the communication capacity between interface modules and CCU.

8.2 Physical Properties of the Electrical Grid

The idea of using the physical properties of the electrical grid as an anomaly detection mechanism was suggested separately by employees of both TenneT and Locamation. However, it turns out that this is not a new idea. In [13], Bigham et al. propose an anomaly detection method based on invariant induction aimed at the data flows as seen in electrical networks. They expect this technique to be highly effective because, in an electrical network, many measurements are related based on physical aspects. For example, the current measurements from two sides of a field are obviously strongly related, and deviation from this relation, e.g. when only one side measures no current, indicates an anomaly. This can be caused by many things, such as a short circuit, an intruder tampering with measurements, or a broken sensor, but it is an anomaly regardless of cause. Bigham et al. go on to perform experiments with promising results. However, after this research, we have been unable to find further applications of this technique for intrusion detection.

However, in the EMS/SCADA network, a state estimator is used in support of the actual measurements. The task of this state estimator is to derive, from the existing measurements, the expected state of the network. This is a way to deal with corrupted measurements, and is based on the same principles, i.e. the

relations between measurements and systems based on physical aspects. It would be an interesting avenue of research to see if this state estimation could be used to find anomalies in the grid, e.g. by calculating if all measurements are physically possible.

8.3 ACNSR

Our proposed extension of ACSR, ACNSR, is designed to deal with non-exclusive shared resources. We use it for a fictitious model as a demonstration of how modelling the addition of a intrusion detection data collector could be done, and how such an addition could affect the runtime properties of a real-time system. However, we have not proven its correctness, and we think it could be further expanded to enhance its usefulness. The current proposal still cannot deal with shared but limited system resources, such as network bandwidth. We have also not found a clean way to deal with preemption in the case where a process is preempted and allowed to idle without losing its current computation. Both are additions which would improve the usefulness of a process algebra for real-time systems.

9 Conclusion

We have identified and categorized several real-time IT systems within the electrical grid, in particular those of the EMS/SCADA network as used by TenneT TSO B.V. We have listed the actors with possible motives to interfere with the functioning of the electrical grid, the main threats that cause TenneT concern regarding intrusions in the grid, and several vectors which could be utilized by malicious parties. We have provided an overview of intrusion detection categorizations and methods in use, and seen some of their advantages and disadvantages. Based on these, we have analysed some possible negative effects these techniques might have when applied to real-time systems.

During this analysis, we have determined that there exists no suitable method to model the behaviour of multi-process real-time systems utilizing shared, lockable resources, and have proposed an extension to the existing process algebra ACSR to remedy this situation. We have demonstrated the use of our extension by modelling a fictional version of one of the real-time systems in use by TenneT, both without and with a host-based intrusion detection system added.

Bringing all this together, we have proposed an architecture for a collaborative, hierarchical intrusion detection system for an EMS/SCADA network as utilized by TenneT, using a number of different distributed IDSs utilizing both anomaly-based and signature-based detection techniques. The selection of these techniques is done based on which methods we expect to be most accurate and practical, and how limited their impact on the real-time constraints of the monitored systems would be. In this architecture we have recognized that the freedom to observe the internal state of the system is of utmost importance, and have suggested that vendors should be contacted to increase the number of systems which can be monitored by a host-based system. We have also recommended that our host-based solution is paired with one of the network-based solutions on the market, for added detection accuracy.

Finally, we have discussed the impact this proposed IDS would have on ICS systems, some avenues of possible future research, and a possible form of implementation on an HVS control system.

Glossary

EMS Energy Management System.

HIDS Host(-based) Intrusion Detection System.
HVS High Voltage Station.

ICS Industrial Control System.
IDS Intrusion Detection System.
IPS Intrusion Prevention System.

LBC Landelijk Bedrijfsvoeringscentrum (National Operation Centre).

NIDS Network(-based) Intrusion Detection System.
NMS Network Management System.

PDH Plesiosynchronous Digital Hierarchy.
PLC Programmable Logic Controller.

RNB Regionale Netbeheerder (Regional Grid Operator).
RTU Remote Terminal Unit.
RTUCS Remote Terminal Unit Control Server.

SCADA Supervisory Control And Data Acquisition.
SDH Synchronous Digital Hierarchy.
SoE Sequence of Events.

TSO Transmission System Operator.

References

[1] The bro network security monitor. http://www.bro.org/.

[2] Locamation b.v. http://www.locamation.nl/.

[3] Open source security. http://www.ossec.net/.

[4] Silentdefense ics. http://www.secmatters.com/products-ics.

[5] Snort network intrusion detection and prevention system. http://www.snort.org/.

[6] Verisys file integrity monitoring system. http://www.ionx.co.uk/products/verisys.

[7] *APT1: Exposing One of China's Cyber Espionage Units*. Mandiant Corporation, 2013.

[8] U. Aickelin, J. Greensmith, and J. Twycross. Immune system approaches to intrusion detection – a review. In G. Nicosia, V. Cutello, P. Bentley, and J. Timmis, editors, *Artificial Immune Systems*, volume 3239 of *Lecture Notes in Computer Science*, pages 316–329. Springer Berlin Heidelberg, 2004.

[9] D. Ariu, R. Tronci, and G. Giacinto. Hmmpayl: An intrusion detection system based on hidden markov models. *Computers & Security*, 30(4):221 – 241, 2011.

[10] M. A. Aydın, A. H. Zaim, and K. G. Ceylan. A hybrid intrusion detection system design for computer network security. *Computers & Electrical Engineering*, 35(3):517 – 526, 2009.

[11] G. K. Baah, A. Gray, and M. J. Harrold. On-line anomaly detection of deployed software: a statistical machine learning approach. In *Proceedings of the 3rd international workshop on Software quality assurance*, SOQUA '06, pages 70–77, New York, NY, USA, 2006. ACM.

[12] R. Berthier and W. Sanders. Specification-based intrusion detection for advanced metering infrastructures. In *Dependable Computing (PRDC), 2011 IEEE 17th Pacific Rim International Symposium on*, pages 184–193, 2011.

[13] J. Bigham, D. Gamez, and N. Lu. Safeguarding scada systems with anomaly detection. In V. Gorodetsky, L. Popyack, and V. Skormin, editors, *Computer Network Security*, volume 2776 of *Lecture Notes in Computer Science*, pages 171–182. Springer Berlin Heidelberg, 2003.

[14] D. Bolzoni. *Revisiting anomaly-based network intrusion detection systems*. PhD thesis, Enschede, June 2009.

[15] D. Bolzoni, S. Etalle, and P. Hartel. Poseidon: a 2-tier anomaly-based network intrusion detection system. In *Information Assurance, 2006. IWIA 2006. Fourth IEEE International Workshop on*, pages 10 pp.–156, 2006.

[16] G. Bruneau. Mcafee dat 5958 update issues. https://isc.sans.edu/diary/McAfee+DAT+5958+Update+Issues/8656, 2010.

[17] P. Brémond-Grégoire, I. Lee, and R. Gerber. Acsr: An algebra of communicating shared resources with dense time and priorities. In E. Best, editor, *CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin Heidelberg, 1993.

[18] Z. Chaochen, M. R. Hansen, A. P. Ravn, and H. Rischel. Duration specifications for shared processors. In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, pages 21–32. Springer Berlin Heidelberg, 1991.

[19] A. Chaudhari and J. Abraham. Stream cipher hash based execution monitoring (schem) framework for intrusion detection on embedded processors. In *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*, pages 162–167, 2012.

[20] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, Apr. 1986.

[21] L. Constantin. Mse false positive detection forces google to update chrome. http://www.theinquirer.net/inquirer/news/2113892/mse-false-positive-detection-forces-google-update-chrome, 2011.

[22] D. Denning. An intrusion-detection model. *Software Engineering, IEEE Transactions on*, SE-13(2):222–232, 1987.

[23] P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: algorithms, analysis and implications. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 110–119, 1996.

[24] H. T. Elshoush and I. M. Osman. Alert correlation in collaborative intelligent intrusion detection systems — a survey. *Applied Soft Computing*, 11(7):4349 – 4365, 2011. Soft Computing for Information System Security.

[25] J. M. Estevez-Tapiador, P. Garcia-Teodoro, and J. E. Diaz-Verdejo. Anomaly detection methods in wired networks: a survey and taxonomy. *Computer Communications*, 27(16):1569 – 1584, 2004.

[26] W. Fan. Cost-sensitive, scalable and adaptive learning using ensemble-based methods. PhD thesis, Columbia University, Feb 2001.

[27] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 120–128, 1996.

[28] M. Friedman. *Microsoft Windows Server 2003 performance guide.* Microsoft Windows Server 2003 resource kit. Microsoft Press,, Redmond, Wash., 2005.

[29] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1–2):18 – 28, 2009.

[30] S. Gold. Advanced evasion techniques. *Network Security*, 2011(1):16 – 19, 2011.

[31] S. Gorman. Electricity grid in u.s. penetrated by spies. `http://online.wsj.com/article/SB123914805204099085.html`.

[32] G. Gritsai, A. Timorin, Y. Goltsev, R. Ilin, S. Gordeychik, and A. Karpin. Scada safety in numbers. Technical report, 2012.

[33] J. Gómez, C. Gil, N. Padilla, R. Baños, and C. Jiménez. Design of a snort-based hybrid intrusion detection system. In S. Omatu, M. Rocha, J. Bravo, F. Fernández, E. Corchado, A. Bustillo, and J. Corchado, editors, *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, volume 5518 of *Lecture Notes in Computer Science*, pages 515–522. Springer Berlin Heidelberg, 2009.

[34] C. Haack and A. Jeffrey. Timed spi-calculus with types for secrecy and authenticity. In M. Abadi and L. Alfaro, editors, *CONCUR 2005 – Concurrency Theory*, volume 3653 of *Lecture Notes in Computer Science*, pages 202–216. Springer Berlin Heidelberg, 2005.

[35] M. Hansen and Z. Chaochen. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9(3):283–330, 1997.

[36] X. Hoang and J. Hu. An efficient hidden markov model training scheme for anomaly intrusion detection of server applications based on system calls. In *Networks, 2004. (ICON 2004). Proceedings. 12th IEEE International Conference on*, volume 2, pages 470–474 vol.2, 2004.

[37] X. D. Hoang, J. Hu, and P. Bertok. A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference. *Journal of Network and Computer Applications*, 32(6):1219 – 1228, 2009.

[38] S. A. Hofmeyr and S. Forrest. An immunological model of distributed detection and its application to computer security. *The University of New Mexico*, 1999.

[39] J. Hu, X. Yu, D. Qiu, and H.-H. Chen. A simple and efficient hidden markov model scheme for host-based anomaly intrusion detection. *Network, IEEE*, 23(1):42–47, 2009.

[40] W. Hu, Y. Liao, and V. R. Vemuri. Robust support vector machines for anomaly detection in computer security. In *ICMLA*, pages 168–174, 2003.

[41] J. Keniston, A. Mavinakayanahalli, P. Panchamukhi, and V. Prasad. Ptrace, utrace, uprobes: Lightweight, dynamic tracing of user apps. In *Proceedings of the 2007 Linux Symposium*, pages 215–224, 2007.

[42] S. Konrad and B. H. C. Cheng. Real-time specification patterns. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 372–381, New York, NY, USA, 2005. ACM.

[43] I. Lee, P. Bremond-Gregoire, and R. Gerber. A process algebraic approach to the specification and analysis of resource-bound real-time systems. *Proceedings of the IEEE*, 82(1):158–171, 1994.

[44] W. Lee, S. Stolfo, P. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, and J. Zhang. Real time data mining-based intrusion detection. In *DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings*, volume 1, pages 89–100 vol.1, 2001.

[45] J. Leyden. Horror avg update ballsup bricks windows 7. `http://www.theregister.co.uk/2010/12/02/avg_auto_immune_update/`, 2010.

[46] W. Li. Using genetic algorithm for network intrusion detection. *Proceedings of the United States Department of Energy Cyber Security Group*, pages 1–8, 2004.

[47] H. Lin, A. Slagell, C. Di Martino, Z. Kalbarczyk, and R. K. Iyer. Adapting bro into scada: building a specification-based intrusion detection system for the dnp3 protocol. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, CSIIRW '13, pages 5:1–5:4, New York, NY, USA, 2013. ACM.

[48] R. Mall. *Real-Time Systems: Theory and Practice*. Pearson Education India, 2007.

[49] Alert Logic Security Intelligence. Ids / ips signature bypassing (snort). `http://www.alertlogic.com/idsips-signature-bypassing-snort/`, Sep 2012.

[50] Algemene Inlichtingen en Veiligheids Dienst. Jaarverslag 2012. `http://www.aivd.nl/publish/pages/2520/jaarverslag_2012.pdf`, 2013.

[51] Nationaal Coördinator Terrorismebestrijding en Veiligheid. Bescherming vitale infrastructuur. `http://www.nctv.nl/onderwerpen/nv/voorkomen-voorbereiden/bescherming-vitale-infrastructuur/`.

[52] Nationaal Cyber Security Centrum. Cyber security beeld nederland 2013. `http://www.nctv.nl/Images/ncscscbn-3nl-pp-03_tcm126-504698.pdf`, June 2013.

[53] Verzetsfront Willie Wortel en de lampjes. Sabotage in je vrije tijd. `http://zwarte.squat.net/files/Zwarte35.pdf`, 1985.

[54] N. McAuliffe, D. Wolcott, L. Schaefer, N. Kelem, B. Hubbard, and T. Haley. Is your computer being misused? a survey of current intrusion detection system technology. In *Computer Security Applications Conference, 1990., Proceedings of the Sixth Annual*, pages 260–272, 1990.

[55] B. Morin and L. Mé. Intrusion detection and virology: an analysis of differences, similarities and complementariness. *Journal in Computer Virology*, 3(1):39–49, 2007.

[56] T. H. Morris, B. A. Jones, R. B. Vaughn, and Y. S. Dandass. Deterministic intrusion detection rules for modbus protocols. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 1773–1781, 2013.

[57] B. Mukherjee, L. Heberlein, and K. Levitt. Network intrusion detection. *Network, IEEE*, 8(3):26–41, 1994.

[58] Netragard. Netragard's hacker interface device. `http://pentest.netragard.com/2011/06/24/netragards-hacker-interface-device-hid/`, June 2011.

[59] J.-T. Oh, S.-K. Park, J.-S. Jang, and Y.-H. Jeon. Detection of ddos and ids evasion attacks in a high-speed networks environment. *International Journal of Computer Science and Network Security*, 7(6):124–131, 2007.

[60] P. Oman, E. Schweitzer, and D. Frincke. Concerns about intrusions into remotely accessible substation controllers and scada systems. In *Proceedings of the Twenty-Seventh Annual Western Protective Relay Conference*, volume 160, 2000.

[61] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23–24):2435 – 2463, 1999.

[62] T. Petermann, H. Bradke, A. Lüllmann, M. Poetzsch, and U. Riehm. *What Happens During a Blackout - Consequences of a Prolonged and Wide-ranging Power Outage*. Technology Assessment studies series, no. 4. Büro für Technikfolgen-Abschätzung beim Deutschen Bundestag (TAB), Berlin, 2011.

[63] D. Peterson. Quickdraw: Generating security log events for legacy scada and control system devices. In *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, pages 227–229, 2009.

[64] Z. M. A. Pnueli. The temporal logic of reactive and concurrent systems.

[65] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, 1998.

[66] Y. Qiao, X. W. Xin, Y. Bin, and S. Ge. Anomaly intrusion detection method based on hmm. *Electronics Letters*, 38(13):663–664, 2002.

[67] Y. Ramakrishna, P. Melliar-Smith, L. Moser, L. Dillon, and G. Kutty. Interval logics and their decision procedures. *Theoretical Computer Science*, 166;170(1–2):1 – 47;1 – 46, 1996.

[68] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

[69] C. Schade. Fcscan: A new lightweight and effective approach for detecting malicious content in electronic documents. 2013.

[70] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 265–274, New York, NY, USA, 2002. ACM.

[71] T. Simonite. Chinese hacking team caught taking over decoy water plant. `http://www.technologyreview.com/news/517786/chinese-hacking-team-caught-taking-over-decoy-water-plant/`, August 2013.

[72] C. Sinclair, L. Pierce, and S. Matzner. An application of machine learning to network intrusion detection. In *Computer Security Applications Conference, 1999. (ACSAC '99) Proceedings. 15th Annual*, pages 371–377, 1999.

[73] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 305–316, 2010.

[74] A. Tan. Flawed symantec update cripples chinese pcs. `http://news.cnet.com/Flawed-Symantec-update-cripples-Chinese-PCs/2100-1002_3-6186271.html`, 2007.

[75] J. Verba and M. Milvich. Idaho national laboratory supervisory control and data acquisition intrusion detection system (scada ids). In *Technologies for Homeland Security, 2008 IEEE Conference on*, pages 469–473, 2008.

[76] S. Vogl and C. Eckert. Using hardware performance events for instruction-level monitoring on the x86 architecture. In *Proceedings of the European Workshop on System Security, New York, NY, USA*, 2012.

[77] K. Wilhoit. The scada that didn't cry wolf. `http://www.blackhat.com/us-13/briefings.html#Wilhoit`, August 2013.

[78] W. Yi. Ccs + time = an interleaving model for real time systems. In J. Albert, B. Monien, and M. Artalejo, editors, *Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*, pages 217–228. Springer Berlin Heidelberg, 1991.

[79] T. Zhang, X. Zhuang, S. Pande, and W. Lee. Hardware supported anomaly detection: down to the control flow level. Technical report, 3 2004.

[80] T. Zhang, X. Zhuang, S. Pande, and W. Lee. Anomalous path detection with hardware support. In *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, CASES '05, pages 43–54, New York, NY, USA, 2005. ACM.

[81] Z. Zhang, J. Li, C. Manikopoulos, J. Jorgenson, and J. Ucles. Hide: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification. In *Proc. IEEE Workshop on Information Assurance and Security*, pages 85–90, 2001.