A log mining approach for process monitoring in SCADA

Dina Hadžiosmanović · Damiano Bolzoni ·
Pieter H. Hartel

Published online: 21 April 2012

© The Author(s) 2012. This article is published with open access at Springerlink.com

Abstract SCADA (supervisory control and data acquisition) systems are used for controlling and monitoring industrial processes. We propose a methodology to systematically identify potential process-related threats in SCADA. Process-related threats take place when an attacker gains user access rights and performs actions, which look legitimate, but which are intended to disrupt the SCADA process. To detect such threats, we propose a semi-automated approach of log processing. We conduct experiments on a real-life water treatment facility. A preliminary case study suggests that our approach is effective in detecting anomalous events that might alter the regular process workflow.

Keywords ICS · SCADA · Security · SCADA log · Log analysis · Frequent pattern mining · Process related threat · HAZOP · PHEA · MELISSA

1 Introduction

SCADA systems can be found in critical infrastructures such as power plants and power grid systems, water, oil and gas distribution systems, building monitoring (e.g., airports, railway stations), production systems for food, cars, ships and other products.

Although failures in the security or safety of critical infrastructures could impact people and produce damages to industrial facilities, recent reports state that current critical

infrastructures are not sufficiently protected against cyber threats. For example, according to Rantala [26], around 2,700 organizations dealing with critical infrastructures in the U.S. detected 13 million cybercrime incidents, suffered \$288 million of monetary loss and experienced around 150,000 h of system downtime in 2005. Also, in a security study of 291 utility and energy companies in the U.S. [25], 67 % of the companies report that they are not using state of the art security technologies. Besides, 76 % of the companies report that they suffered one or more data breaches during the past 12 months.

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

Unfortunately, the stakeholders seldom enforce strong security policies. User credentials are often shared among users to ease day-to-day operations and are seldom updated, resulting in a lack of accountability. An example of such practice is the incident in Australia when a disgruntled (former) employee used valid credentials to cause a havoc [32].

Due to these reasons, SCADA facilities became more vulnerable to internal and external cyber attacks. Although companies reluctantly disclose incidents, there are several published cases where safety and security of SCADA were seriously endangered [27].

D. Hadžiosmanović (✉) · D. Bolzoni · P. H. Hartel
University of Twente, Enschede, The Netherlands
e-mail: dina.hadziosmanovic@utwente.nl

D. Bolzoni
e-mail: damiano.bolzoni@utwente.nl

P. H. Hartel
e-mail: pieter.hartel@utwente.nl

Like a “regular” computer system, a SCADA system is susceptible to threats exploiting software vulnerabilities (e.g., protocol implementation, OS vulnerabilities). However, a SCADA system is also prone to process-related threats. These threats take place when an attacker uses valid credentials and performs legitimate actions, which can disrupt the industrial process(es). Process-related threats also include situations when system users make an operational mistake, for example, when a user inputs a wrong value (e.g., a highly oversized value) for a given device parameter and causes the failure of the process. In general, process-related threat scenarios do not include any exploit of a software implementation vulnerability (e.g., protocol implementation).

Sometimes system- and process-related threats can be part of the same attack scenario. For example, an attacker can first subvert the access control mechanism to gain control over an engineering work station. This action would use a system-related threat (e.g., exploiting an OS vulnerability). Then, an attacker could use a valid SCADA control application to perform undesirable actions for the process (e.g., overload pipe system). This part of the attack is performed as a process-related threat scenario.

Traditional security countermeasures, such as intrusion detection systems, cannot detect, let alone mitigate, process-related threats. This is because typical intrusion detection systems look for patterns of the behaviour known to be malicious (e.g., known payload transfers, TCP header format) or look for anomalies in terms of statistical distributions (e.g., by statistically modelling the content of data packets). The anomalies generated by process-related threats are typically not reflected in communication patterns/data (e.g., injection of executable code to exploit a buffer overflow sent within network traffic data) and can only be detected by analysing data passed through the system at a higher semantic level. To understand the higher semantic level from network data, a protocol parser has to be used, such as in Bro [24]. Similarly, for host-based analyses, understanding the specific SCADA application is crucial.

Other approaches for monitoring SCADA behaviour include the usage of field measurements or centralized SCADA events as information resources. Field measurements represent raw values coming from field devices. Aggregated field measurements can provide information about the current status of the process. However, we argue that the field measurement values are too low level to extract user actions and to evaluate the semantics of the performed actions.

SCADA event logs provide a complete high-level view on the industrial process that is continuous over time and captures information about user activities, system changes in the field as well as system status updates [31].

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

This is a relevant and challenging problem to tackle. It is relevant because process-related threats affect the security and safety of critical infrastructures, which in turn could endanger human life. It is challenging because in the past the analysis of system logs has been applied to other security domains (e.g., in [15]) but failed to deliver convincing results.

We propose a semi-automated approach of log processing for the detection of undesirable events that relate to user actions.

We acknowledge that the success of a log mining approach depends on the context in which it is applied [23]. Therefore, we perform an extensive analysis of the problem context. In Fig. 1, we show the main steps of the our approach. We group the steps by two means of obtaining context information: (1) system analysis and (2) analysis by a focus group. The system analysis implies the inspection of available documentation and processing of logs. The focus group analysis implies sessions with the stakeholders where we obtain deeper insights about the SCADA process. Focus groups consist of process engineers who are aware of the semantic implications of specific actions, but typically cannot provide useful information for automatic extraction of log entries. This is due to the fact that engineers do not perform extensive analysis of system outputs and are not experts in data mining. On the other hand,

Fig. 1 Steps for mining SCADA logs

by performing the system analysis, we cannot infer semantic information that is implied in log entries, thus the stakeholders' knowledge is invaluable.

A sequence of actions in Fig. 1 represents the chronological order of the steps that we perform. In steps 1 and 2, we systematically identify process deviations caused by user activity. For this, we adapt two methodologies from the domain of hazard identification (PHEA and HAZOP [18]). We then use the stakeholders knowledge to identify which of the analysed deviations represent a legitimate threat to the process (step 3). In steps 4 and 5, we perform log transformation and generalization to extract a subset of log attributes suitable for log mining. Also, we discuss the requirements of the mining algorithm that is useful for our context. In steps 6 and 7, we include the stakeholders in the mining process. This implies leveraging the stakeholders' knowledge about the process to improve the semantics of the mined events. The stakeholders analyse the output of the mining process and verify anomalies. Finally, the anomalies are checked for the consistency with the threats identified in step 3. Also, this step is used to revise the list of potential threats and perhaps introduce new threats.

To support the proposed analysis, we build a tool that can perform the log processing in search for process-related threats. Our tool leverages a well-known data mining algorithm to enumerate (in)frequent patterns within a given set. Despite being quite simple and straightforward, our benchmarks show that the chosen algorithm is effective in detecting previously overlooked behavioural anomalies.

Contributions The main contributions of this paper are the following:

- we propose a new methodology to identify and analyse process-related threats caused by the activity of users,
- we propose an approach to detect process-related threats and build a tool to automate the analysis of SCADA logs, which can be used to monitor the industrial process,
- we perform experiments to validate our approach using data from a real facility.

The rest of the paper is organized as follows. In Sect. 2, we describe a typical SCADA system. Section 3 describes the performed threat analysis (steps 1, 2 and 3 from Fig. 1). Section 4 describes our approach and step 4 in Fig. 1). In Sect. 5, we discuss the algorithm choice (step 5) and describe the architecture of our tool. Section 6 describes steps 6 and 7. Related work is presented in Sect. 7. Finally, we present our conclusions and future work directions in Sect. 8.

2 Preliminaries

In this section, we explain how a typical SCADA system works. A SCADA system consists of two main domains: the process field and a control room (Fig. 2).

Large systems may have more than one control room. The network infrastructure binds the two domains together. SCADA users control the industrial process from the control room and are provided with a real-time overview of the process field device parameters (data about tank loads, pump statuses, temperatures, etc).

Depending on the underlying process, SCADA systems differ from each other. For example, a power-related SCADA installation contains power switches and transformers while a water-related installation contains water pumps and valves. However, based on interviews with the stakeholders (who represent process engineers, operators and computer network experts from 4 different facilities), we believe that the computer systems controlling these processes behave in a similar way.

2.1 System architecture

Despite the fact that there are different vendors, the system architectures in various SCADA systems are similar and the terminology is interchangeable. Figure 3 shows a typical SCADA layered architecture. Layer 1 consists of physical field devices, PLCs (programmable logic controllers) and RTUs (remote terminal units). The PLCs and RTUs are responsible for controlling the industrial process, receiving signals from the field devices and sending notifications to upper layers. Layer 2 consists of SCADA servers responsible for processing data from Layer 1 and presenting process changes to Layer 3. Connectivity Servers aggregate events received by PLCs and RTUs and forward them to SCADA users in the control room. The Domain Controller in Layer 2 holds local DNS and authentication data for user access. The Aspect Server is responsible for implementing the logic required to automate the industrial process. For example, an Aspect Directory in the Aspect Server holds information about working ranges of the field devices, the device topology, user access rights, etc. Besides, the Aspect Server collects and stores data from the Connectivity Servers into audit and event logs. The various clients in Layer 3 represent SCADA users.

2.2 System users in process control

There are two kinds of SCADA users: engineers and operators. An engineer is responsible for managing object libraries and user interfaces, setting working ranges for devices, defining process setpoints, writing automation scripts, etc. An operator monitors the system status and reacts to events,

Fig. 2 SCADA system overview: control room and process field

Fig. 3 SCADA system layered architecture

such as alarms, so that the process runs correctly (Fig. 2). Typical operator actions, depending on the underlying industrial process, include commands such as: change switch status, increase temperature, open outlet, start pump. Although industrial processes in various domains differ in the details (or some user roles may be assigned to external parties such as vendors), the user interaction with a SCADA system is broadly similar. Our stakeholders acknowledged that an engineer is a more powerful system user than an operator (e.g., an engineer writes scripts that define process automation while operators usually only run the script). Also, operators perform actions that are predefined by engineers (e.g., an engineer defines pump speed range, while an operator works within the range only). This means that operator actions are security and safety constrained depending on the way the engineer implemented safety controls. By contrast, there is no mechanism that will ensure that engineer actions are safe for the process (e.g., an engineer can, by mistake, assign a capacity 10 times bigger than in reality to a tank, and thus shut

tank level alarms off). Although individual operator actions are legitimate and should be safety constrained, the stakeholders acknowledge that a sequence of operator actions can still produce damage to the process.

2.3 System logs

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

Generally, a user action leaves a trace in the log on two ways: (1) as a direct action (e.g., the exact user action of performing a reconfiguration) or (2) as a consequence (e.g., an consequence of a performed action or a sequence of actions-process script). The first type of trace implies a log entry

that captured the time, the location and the user name of the person who performed the action.

The second type of trace implies an indirect action consequence or system response. Although caused by a user action, this trace typically does not consist of user name who performed the initial action nor the location of the failure source. This is because the captured trace does not represent the source but the victim of the specific action that propagated [14, 22].

SCADA system already use logs during operation. In particular, engineers compile alarm triggers and thus highlight specific events during operation. For example, an alarm trigger is designed to go off when a specific field value reaches the threshold (e.g., tank level less than 100L). This is good because users can define known malicious behaviours to be alerted. However, such alarms cannot extract events that are unexpected compared to usual patterns of behaviour.

In the next section, we analyse potential threats that can occur in a SCADA environment.

3 Threat analysis

We classify possible threats against SCADA systems in two groups: system- and process-related. System-related threats typically exploit software/configuration vulnerabilities (e.g., a buffer overflow or a flaw in a communication protocol [4]). Such attacks are low level and typically occur at Layer 1 and Layer 2 of the SCADA architecture (Fig. 3).

On the other hand, process-related threats exploit weak process controls and imply that an attacker obtains (e.g., through social engineering) user access rights and issues legitimate SCADA commands to disrupt the industrial process. We analyse process-related threats during a focus group session with stakeholders. Based on two possible places where a SCADA user interacts with the process, we distinguish two types of process-related threats: (1) threats that leverage access controls on field devices and (2) threats that leverage vulnerabilities of the centralized SCADA control application. The first type of threat typically results in sending bad data to the SCADA state estimation, which can then produce errors in system state analysis [19]. The second type of threat includes scenarios of performing legitimate user actions (from the control application) that can have negative impact on the process production or devices. In this paper, we focus on the second type of process-related threats. These attacks are high level (occur via the process control application) and typically relate to user actions (Layer 3 on Fig. 3).

3.1 Identification of process-related threats

To identify process-related threats in SCADA, we analyse user activities in SCADA control software. We describe user

activities as actions that are (1) performed by a signed-on user or (2) performed on a known user workstation. We analyse the threats that leverage legitimate system commands performed by a legitimate user, or by an illegitimate user who has managed to obtain valid credentials. Our focus is on the threats that can be triggered by a single user action (i.e., we do not analyse sequences of actions). Based on interviews with the stakeholders, we distinguish two threat scenarios, namely (1) an attacker impersonates a system user or (2) a legitimate system user makes an operational mistake.

To identify process-related threats and the leveraged vulnerabilities, we analyse a real-life SCADA system controlling a water treatment facility located in the Netherlands.

3.1.1 Methodology

We combine two known methodologies to systematically identify process-related threats. Those are PHEA (predictive human error analysis) and HAZOP (hazard and operability study) [18]. PHEA takes a user-oriented approach to analyse human errors by building a task analysis tree. The task tree consists of possible user actions in the specific system. We use the PHEA tree to represent possible user actions in a SCADA system. Originally, PHEA analyses the tree using the human error classification (e.g., an action is taking too long, an action is performed incomplete [18]) to identify potential threats for every action in the tree. This part of the analysis is not suitable in our case. The reason for this is that our goal is to identify actions that produce malicious consequences on the process, whereas PHEA originally focuses on identifying possible causes of human errors. To identify possible misuses of user actions (on the PHEA tree), we use the HAZOP methodology.

HAZOP is the best documented methodology for addressing process safety problems [18]. The methodology requires that, for every specific context, a set of process keywords and a set of process guidewords is defined. The process deviations (e.g., temperature decrease) are built by combining chosen process keywords (e.g., temperature, pressure, power) and process guidewords (e.g., increase, decrease, go reverse). All deviations are then analysed to identify the severity, effects and possible mitigation strategies. As a result, some deviations are ruled out as noncritical while others are highlighted and presented as potential threats.

We implement the combined methodology in two steps. First, we build a task analysis tree (as in PHEA). Our tree consists of possible user tasks within a SCADA customized Aspect Directory. Figure 4 represents a part of an anonymized and simplified AD of the stakeholder company.

Second, we perform the HAZOP step. We compile a specific set of keywords and guidewords for our application context. The leaves of the PHEA tree represent HAZOP keywords (such as pumps, tanks). In the analysed facility, there

Fig. 4 Part of the AD in a working SCADA facility for water treatment

are 36 different keywords. The keywords are compiled (and generalized) from the AD tree of object paths. We describe the tree generalization process in Sect. 5.3. By performing this step, we increase the level of abstraction in the analysis. For example, instead of analysing deviations for each specific device (e.g., *PUMP1*, *PUMP2*, *PUMP3*), we analyse deviations for a group of same devices (*pumps*). The SCADA control application allows a user to perform only three distinct operations: add, modify and delete. We use these operations as HAZOP guidewords. Following the concept of the HAZOP methodology, we build process deviations for all possible combinations of given keywords and guidewords. Table 1 shows examples how deviations are generated. The first column of the table consists of all chosen keywords (leaves of the PHEA task analysis tree from Fig. 4). The second column consists of three guidewords. Each keyword from the first column is combined with all three guidewords from the second column to build a deviation in the third column. The character of the specific deviation depends on the position of the keyword in the tree. Typically, one keyword can be found on several branches of the AD tree (e.g., in Fig. 4 keyword *tanks*). A deviation built on the keyword *tanks* in the *Plant1/Functional mode/Street2/Devices/Operational parameters/Groups of devices/tanks* branch of the

tree implies actions on operational parameters of the device such as capacity, desirable tank level, etc. Such devised deviations are the following: a user modifies the capacity of tank and a user modifies the value of the desirable tank level. Deviations of the same keyword from a different branch of the tree imply different types of actions. For example, keyword *tank* in the *Plant1/Control module/Production/Cleaning/Alarm settings/Groups of devices/Tanks* branch of the tree implies operations on alarms that relate to tanks (e.g., add alarm, modify alarm, delete alarm).

The enumeration of possible actions of each keyword in various branches of the tree represents specific, application-dependant knowledge. We believe that such extensive description is not interesting for this paper.

3.1.2 Identified threats

The HAZOP step of the methodology in our case comprises 108 combinations of keywords and guidewords. However, depending on the context (i.e., the specific software control implementation), some combinations do not apply (e.g., operational parameters cannot be deleted, an item on the access list can only be added or deleted but not modified).

Table 1 Identified system deviations caused by user activity

Keyword	Guideword	Deviation/potential threat
Valve (topology) ^a	Add	A user adds a valve to the group
	Modify	A user modifies the name of a valve
	Delete	A user deletes a valve from the group
Tanks (operational parameters) ^b	Add	No
	Modify	A user modifies the capacity of tank (e.g., the capacity of tank is increased by double)
	Delete	No
Pumps (access settings) ^c	Add	A user adds action type to the allowed actions (e.g., a user adds “inserting setpoint” and/or “changing pump status” to the list of allowed operator actions on a pump)
	Modify	No
	Delete	A user deletes action type from allowed actions (e.g., a user deletes “inserting setpoint” and/or “changing pump status” from operator actions on a pump)

^a Plant1/Control module/Topology/Groups of devices/Valves^b Plant1/Functional mode/Street2/Devices/Operational parameters/Groups of devices/Tanks^c Plant1/Control module/Production/Cleaning/Access settings/Groups of devices/Pumps

This way, together with the stakeholders, we decrease the total number of deviations to 72. Finally, during a focus group session with the stakeholders, we analysed 35 deviations. For all compiled deviations, we perform a detailed analysis to define the cause, effects and mitigation recommendations. Table 2 shows the results of the detailed analysis of three deviations.

After analysing causes and effects, the stakeholders labelled 18 deviations as potential threats. We distinguish two types of threats: scripting errors and misconfiguration. Both types of threats typically originate from the activity of engineers. The threats exploiting a scripting error imply writing (and loading) faulty process automation scripts or leveraging scripts already developed by system engineers. A misconfiguration implies forcing the settings of unsafe configurations. Examples of the identified threats are presented in Table 3.

After the performed analysis, together with the stakeholders, we discuss possible methods for mitigating the identified threats. In particular, we analyse if the current SCADA application can apply to identified threat recommendations (Recommendations in Table 2). To this end, we compile a list of vulnerabilities in the system that relate to the identified threats:

- no process safety checks are in place during the manual system mode;
- an input is not validated before executing an engineer command;
- weak password policy;
- no detail operator auditing;
- limited separation between production and administration (no principle of least privilege: e.g., at the same time, an engineer has access rights to both process configuration and user-account administration).

Table 2 Understanding mitigation strategies

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

Table 3 Identified process-related threats and threat examples in the system context

Type of threat	Threat	Example
Scripting error	An engineer loads a script that causes errors in the system automation	An engineer inserts a wrong ratio of chemical components
Misconfiguration /functional	An engineer modifies a device parameter	Change capacity of a tank to prevent the alarming system from going off
Misconfiguration /functional	An engineer modifies auditing policy	Turn off all auditing
Misconfiguration /control	An engineer modifies the range of allowed actions for a specific device	A pump cannot be any longer stopped
Misconfiguration /control	An engineer modifies the system topology	Some devices become invisible, and thus inaccessible

Due to the apparent mismatch between the current system features and the devised recommendations, we acknowledge that the current SCADA application lacks controls to detect the identified threats.

We discuss two ways of mitigating identified threats: (1) by upgrading the proprietary SCADA software to support additional functionalities (such as an additional input check in manual mode or by introducing safety checks for engineer actions before execution) and (2) employing an independent tool to analyse data resources from SCADA and detect malicious behaviour.

We choose to build a tool to analyse and identify undesirable behaviours.

4 Mitigation approach

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

We argue that, due to the size and complexity, manual inspection of SCADA logs is infeasible. Automated filtering of interesting events may provide some results. In Sect. 2.3, we describe that user actions may leave a trace in two ways: as a direct user action and as a consequence. The threats identified in Sect. 3.1.2 directly relate to user activity. Such user actions could be detected by implementing the following filters:

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

However, the extraction of action consequences is difficult. This is because the prediction of potential consequences of a performed action and the propagation of such consequences is not straightforward as it implies an in-depth analysis of process dependencies.

We argue that detecting direct user actions solely might not reveal the undesirable character of the user action. For example, a user might write an erroneous (or malicious) script that produces postponed faults in the system. The act of writing a script is not unusual (and is typically scheduled), thus the log trace of the action is legitimate. Therefore, a rule-based approach would either: (1) raise an alert and be cleared by the operators as legitimate or (2) not raise an alert at all. However, the script might produce indirect consequences (or faults) that are undesirable for the process work. Due to the fact that the enumeration of all possible faults that a script might trigger is typically infeasible, such fault would not be detected.

Thus, we believe that SCADA logs should also be processed in a heuristic approach. In contrast to the rule-based approaches, this approach implies that models of normal and anomalous behaviour are derived automatically.

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

The basic idea of our approach is that a frequent behaviour, over an extended period of time, is likely to be normal because event messages that reflect normal system activity are usually frequent [3,20,35].

Similar to the authors in [8], we found a large fraction of events that always appear with the same number of daily occurrences (e.g., timer-triggered event). Thus, a rare event, in a semi-automated and stable environment as SCADA, is likely to be anomalous. For example, an engineer operating from a machine that is usually inactive outside the working hours is considered suspicious.

Fig. 5 Log translation:
a mapping log entries into
 itemsets and items, **b** mapping
 itemsets into patterns

Our analysis consists of pattern mining on SCADA logs. To do that, we translate SCADA log entries into patterns. Figure 5 depicts the relation between a log entry, an itemset, an item and a pattern. Each unique log entry, with several attributes, represents a single itemset (Fig. 5a). A unique value of an attribute in the log entry represents one item.

A support count is the number of log entries that contain the given itemset. Formally, if the support count of an itemset I exceeds a predefined minimum support count threshold, then I is a pattern [12]. In Fig. 5a, log entries 2 and 3 are the same, thus the corresponding pattern has the support count 2 (Fig. 5b).

We use an algorithm for mining frequent patterns to identify the most and the least frequent (expected to be anomalous) patterns of system behaviours. We describe such algorithm in Sect. 5.2.1.

4.1 Input data for analysis

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

- time (*Timestamp*),
- type of action (*Type of action*, *Aspect of action*),
- action details (*Message description*, *Start value*, *End value*),
- user (*Username*, *User full name*),
- location (*Object path*, *Source*, *SCADA node*)

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

4.1.1 Attribute subset selection

Common approaches for attribute selection exploit class labels to estimate information gain of specific attribute (e.g., decision tree induction [12]). Unfortunately, our data set does not consist of class labels (i.e., labels for normal and undesirable behaviour), thus we cannot perform the “traditional” attribute evaluation. However, some approaches may evaluate attributes independently. For example, principal component analysis (PCA) [12] searches for k n -dimensional orthogonal vectors that can be used to represent the data. The original data is thus projected into a much smaller space and represented through principal components. The principal components are sorted in the order of decreasing “significance”. Finally, the dimensionality reduction is performed by discarding weaker components, thus those with low variance. By performing the PCA on our data, we discard two low variable attributes (*Start Value*, *End Value*) since they only had one value in the whole data set. Also, we identify two redundant attributes (*Username* and *User full name*). Thus, we discard one of them.

Table 4 Example of a simplified SCADA log

W. shift	Aspect of action	Type of action	Object path	User account	SCADA node
2	–	Process simple event	–	–	CS01
2	Layout change	Configuration change	Plant1 /./ layout	Engineer1	EN01
1	–	Operator action	Plant1 /./ tanks	Operator2	OP03

As expected, the attribute *Timestamp* showed the highest variance. We aggregate this attribute in three working shifts. We describe the details of this aggregation in Sect. 5.3.

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

Due to the fact that the highly variable attributes can produce overfitting [17,29], we try to lower the number of distinct values in the most variable attributes (in our context, the ones over 150 distinct values- *Object path*, *Source*, *Message description*). The attribute *Object path* represents structured text. In Sect. 5.3, we describe the details of generalizing the values of this attribute.

The attribute *Source* represents an ID of the field or network device and consists of around 350 different values. This attribute is highly variable, but does not contribute to the data mining process due to fact that it uniquely identifies a device. For example, a credit card number almost uniquely identifies a customer and thus does not represent a useful attribute to generalize customers behaviour thus to be used in the data mining. Thus, we omit this attribute from the analysis. Similar to this, authors in [17,23] perform de-parameterization of data by replacing IP addresses, memory locations and digits by tokens.

The attribute *Message description* represents unstructured text and consists of 280 values. We perform an in-depth analysis of values to determine means of aggregation. We conclude that a portion of values represents redundant data to other attributes (e.g., information in *Message description*: “Action A on source B is acknowledged by C” is repeated already in the same entry by the attributes *Type of action*: A, *Source*: B, *User*: C). The rest of messages are presented in an inconsistent way and provide information which, at this moment, we cannot parse and aggregate in a meaningful way. An alternative approach would be unsupervised clustering of messages, such as in [37]. Such clustering, however, does not guarantee semantic similarity of messages. We believe that the remaining attributes can compensate the information loss from this attribute. On the other hand, we are sure that such highly variable attribute does not contribute to the data generalization. Thus, we do not consider this attribute during the analysis.

Our final set consists of 6 nominal attributes: *Working shift*, *Aspect of action*, *Type of action*, *Object path*, *User account* and *SCADA node*. Some attributes are not applicable for all entries. As a result, every entry uses between

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

Some events in the log are more severe than others. The severity of a SCADA event depends on the combination of attribute values. Thus, a correct evaluation of specific attribute values can help to detect events that are undesirable for the normal process flow. For example, the value *Audit-EventAcknowledge* of the attribute *Type of action* is semantically less important than the value *AspectDirectory*. This is because the first value implies an action where an operator acknowledged an alarm while the latter value implies that a new action was performed on the main configuration directory. Leveraging the stakeholders’ knowledge about the process and the semantics of nominal attribute values can help to distinguish critical and noncritical events in the complete log. In Sect. 6.2.1, we describe how we use this knowledge to improve our detection results.

4.1.2 Data set validation

Our stakeholders argue that, at the time of logging, there were no known security incidents. We investigate the ways of validating this claim. We argue that due to size and high dimensionality of the log, manual inspection is infeasible. Thus, a (semi)automated approach is required. Typically, common log analysis tools imply the usage of predefined rulesets, which filter events out of logs. For example, in [28], various rulesets for analysing logs, such as syslog and ssh log, are

Fig. 6 MELISSA architecture

maintained. Unfortunately, such ruleset for analysing SCADA system logs does not exist. Thus, we cannot perform a reliable log analysis to establish the ground truth.

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

We acknowledge that, lacking the notion of the ground truth, we cannot perform an extensive discussion about false negatives. We are aware of this shortcoming in our approach. Nevertheless, the primary goal of our approach is to help operators uncover security-related events from real data, which would be overlooked otherwise.

5 Architecture

MELISSA consists of two interacting components: the data preparator (DP) and the pattern engine (PE). Figure 6 depicts MELISSA and its internal components.

5.1 Data preparator

We perform data aggregation (e.g., variance reduction) and transformation (e.g., value coding) on the data set to get a suitable data format for pattern mining. We describe performed operation in Sect. 5.3.

5.2 Pattern engine

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

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

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

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

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

There are various strategies to extract a useful subset of itemsets from the complete set. For our context, our stakeholders agreed that no subset of attributes carries enough semantics to distinguish between anomalous and normal events. For example, it is not sufficient to describe an event with only two attributes (e.g., itemset attributes $\{Type\ of\ action, User\ account\}$; itemset instance $\{Operator\ action, Operator\ 2\}$). Therefore, we set a requirement that the algorithm should deliver output patterns which consist of as many attributes as possible (take the superset itemset that satisfies the minimum threshold). This type of mining is in data mining terminology referred as mining maximal patterns [12]. Formally, an itemset X is a maximal frequent itemset in set S if X is frequent and there exists no super-itemset Y such that $X \subset Y$ and Y is frequent in S [12]. In our context, a maximal itemset is one log entry consisting of all applicable log attributes.

Scalability For the cases when the same plant setup is running for years, we might want to run the tool continuously and receive events as they occur. Thus, the tool needs to scale well when processing logs that may consist of years of

plant work. The tool can then leverage the knowledge of past behaviours to update the top patterns and detect anomalies.

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

For mining a k -size itemset, an algorithm that uses candidate generation may need up to 2^k scans of the data set. By contrast, an algorithm that does not use candidate generation typically requires only two scans of the data set to mine itemsets of arbitrary size. These algorithms are based on a recursive tree structure and are referred in the literature as the FP-growth methods. During the data preparation, we already scan the whole data set several times. We expect our log size to grow up to several million entries (e.g., around 2,500,000 entries correspond to the stakeholder's annual system logs). Also, benchmark results on a frequent itemset mining implementation (FIMI) workshop [10] show that the FP-growth methods scale better for most data sets. Therefore, we choose to use an FP-growth algorithm to comply with the scalability and speed requirements.

There are various implementations of the FP-growth method [7, 11]. These algorithms implement different structures to improve algorithm performances (e.g., in [11] authors use array structures, in [7] authors use a bitmap compression schema). We acknowledge one general problem of the FP-growth methods. These algorithms may scale bad with respect to memory consumption for small values of minimum support count (i.e., the threshold for the frequency of total occurrences). This is because a small value for the minimum support count, depending on the data set character, may produce a large number of unique itemsets that each need a separate tree branch. This results in a complex FP-tree building and mining. However, with respect to our context (a limited number of items to mine: number of users, system nodes and a low number of different operations), we believe that there will not be a significant growth in the total number of items in our logs. Thus, we expect the memory consumption to remain in ranges of our initial experiments when scaling up to millions of entries.

Selection of interesting events based on absolute support count To distinguish between interesting and uninteresting itemsets, algorithms use the concept of “cut off” parameter. For example, some algorithms use an absolute minimum support count (e.g., consider an itemset frequent if it appears at least 5 times in the data set) while others use a relative minimum support (e.g., consider an itemset frequent if it appears in at least 10 % of total data set entries). Some algorithms use top k ranking of patterns (e.g., consider frequent if an itemset is in top 5 ranked patterns and satisfies absolute minimum support). In our context, the output produced by

the algorithm is then inspected by security operators. This implies that the number of extracted patterns directly influences usability. Thus, we believe that an absolute support count with ranking suits our context better than a relative support count. We determine the final “cut off” parameter with stakeholders (discussed in Sect. 6.1).

An algorithm that meets most of our requirements is the FP-growth algorithm by Grahne et al. [11].

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

5.2.1 FP-growth algorithm

The first data set scan in an FP-growth method finds and counts all individual items in the data set (Fig. 7a). The items found are inserted into the header table in decreasing order of their count (Fig. 7b). In the second scan, data set entries are read and inserted in the FP-tree as branches, where items represent tree nodes. If an itemset shares some of the items with a branch previously inserted in the tree, then this part of tree will be shared between entries. Every tree node holds a count, which represents the number of entries where the item occurs (with considering preceding items).

After the second data set scan, all entries are inserted in the FP-tree. The header table holds links to tree nodes for each item. For every item in the header table, a conditional pattern base and a conditional tree are built. The conditional pattern base represents a list of tree paths that a given item (e.g., item F) appears in. This represents a new data set restricted for item F (Fig. 7c). The main algorithm is now repeated on the restricted data set. As a result, a new tree of paths is built (Fig. 7d). The branches of the tree that satisfy minimum support count represent frequent patterns (Fig. 7e).

5.3 Implementation

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

Also, we aggregate the attribute *Object path*. This attribute is in the format of structured text and represents a hierarchical tree of locations in the plant (both functional and geographical). This tree represents a textual representation of the AD task tree in SCADA control application. In practice, the values of this attribute represent the “address path” of a device where the event has taken place or, in case of a configuration

Fig. 7 FP-growth algorithm:
a full data set, **b** building
 FP-tree from the data set,
c extracted itemsets that precede
 item *F*, **d** building recursive tree
 from the conditional base,
e extracted itemsets that satisfy
 minimum support count

change, the system path of the change. The last substring of the path represents the name of a device (e.g., plant1/functional mode/street1/pumps/pump3). To aggregate values, we take the substring of the tree path with up to and excluding the leaves of the tree. This way we semantically group together devices, which are on the same location (or configuration paths) and thus aggregate the original attribute from 170 down to 36 nominal values.

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

In the Pattern Engine, we use an algorithm for mining maximal frequent patterns proposed in [11].

6 Benchmarks

To evaluate the effectiveness of our approach, we collected a data set of logs generated by the SCADA system of the stakeholders, which processes waste, surface and drinking water. The 101,025 log entries were collected during a 14 day period, and each log entry consists of at most 12 attributes. The logs were captured with the default audit set-up of the SCADA system that collects events continuously through time.

We use the subset of 6 log attributes that consist of 69 unique values (i.e., items). Since we aim at identifying the least frequent patterns, our minimal support count is 1. This means that each unique event which occurred at least once represents a pattern.

6.1 Testing MELISSA

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

- we were provided with only 2 weeks of system logs;
- we cannot claim that these 2 weeks represent a complete set of behaviours that occur in the facility through a year;
- water-related systems are considered as slow processes (the consequences of actions are delayed—e.g., it takes several hours to overload a tank even while pumping at maximum speed), thus we can afford to run the analysis with a delay.

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

6.2 Preliminary results

We first summarize the results of daily inspections.

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

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

We now describe the context of the pattern that was labelled as anomalous. Figure 8 represents a projection of the pattern analysis from this day. The table consists of 8 columns. The first column represents the pattern support count. The remaining columns represent the attributes used in

the analysis. The wavy horizontal line represents the border between interesting and uninteresting patterns as decided by the stakeholders (maximum 20 events per day). On the right-hand side of the table, the stakeholders labelled each pattern as either normal or anomalous (e.g., A1). For the anomalous pattern, circles imply why the pattern is unusual.

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

6.2.1 Introducing process knowledge

During the preliminary analysis of results, we note a shortcoming of our approach. Currently, we assume that all events equally impact the process. In reality, this is not true. When using the threshold of 20 events per day, our stakeholders acknowledged that in several cases, some uncritical events

Fig. 8 MELISSA testing, day 4 (before introducing semantic knowledge)

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

Fig. 9 MELISSA testing, day 4
(after introducing process
knowledge)

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

A1
 A2
 A3

were within the threshold while some severe (and suspicious) events were omitted due to the restriction on the number of selected events per day. Therefore, we decide to include the process knowledge to our algorithm and thus improve the quality of results. We do this by implementing a loose ordering on the algorithm output. The order is based on the process severity of specific events. By applying the new order, we perform a fine-grained tuning of results so that the semantically more severe patterns appear within the usability threshold while less severe patterns (although with a low frequency of occurrence) tend to appear lower on the output list (and thus appear to be less interesting). The loose order is defined by evaluating the semantical meaning of values of one log attribute.

To choose the suitable attribute, we perform a semi-automated preselection of attributes. As we mentioned earlier, not all attributes are used in all entries. Thus, we assume that only attributes used in the same entries as user actions (i.e., performed on user workstations) are semantically important for detecting threats identified in Sect. 3.1.

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

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

Here, the order of the values implies how severe that specific action for the process is. For example, an action that includes *AspectDirectory* is more severe than *Audit-Event Acknowledge* as explained in Sect. 4.1). We run several

experiments to generate different PE outputs using the selected attributes and the compiled lists. For each list, we add weights to the attribute values. For example, we add a negative weight to severe actions (to increase the chances that the action is closer to the top of the pattern list) and a positive weight for noncritical actions (to decrease the chances that the action is close to the top of the pattern list). We then submit the results to the stakeholders. The stakeholders selected the attribute *Type of action* as the one whose ranking performed the most useful results within the extracted patterns. Thus, we perform the final fine-grained re-ordering based on the severity weights of this attribute.

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

After providing the tuned results, the stakeholder labelled two more patterns as anomalous. These patterns also appear on day 4. Figure 9 shows the new ordering after introducing the process knowledge.

We now explain the context of the anomalous patterns. Anomalous patterns A2 and A3 occur twice. Node CS01 represents the primary Connectivity Server. *Network message* item typically reports problems in the network communications. *Operation* item reports system responses to a user action (such as input expression error messages, condition-triggered procedures). The stakeholders evaluate patterns A2 and A3 as anomalous because these patterns reflect network and operational errors on the main connection backbone node (CS01). After a thorough internal inspection, the stakeholders found out that all events from these two patterns occurred in the same minute of day 4. User Engineer 1 was logged in on CS01 during the time these errors happened. The stakeholders assume that Engineer 1 inserted a value which triggered an overflow in a device cache, which in turn generated an error report from the system. We verify that this is the only case, over 2 weeks of operations, that error

messages were triggered on Connectivity Servers. The stakeholders classify these patterns as misconfiguration threats where the user triggered cache overflows by inserting unexpected values. We note that this kind of error (e.g., an error reporting the input value is out of range) could be an indication of a masquerade attack. For example, an attacker with valid credentials would possibly be unaware of the working ranges for specific devices. Thus, he might insert a value that would trigger a cache overflow which would be logged.

In conclusion, by applying our tool, the stakeholders detected and acknowledged three unexpected events. All detected events relate to an undesirable engineer operation on the system. These events could not be detected by applying filters for various user actions (as discussed in Sect. 4). In fact, the entries were only indirectly related to user operation (and thus represent the consequence of an action, as defined in Sect. 4). In our context, the actual action of activating a script that generated anomalous event *A1* occurred before of the log capture time. Similarly, anomalous events *A2* and *A3* represent propagated errors on devices caused by a legitimate user action (which was logged and initially cleared as normal by the stakeholders).

In the next section, we discuss the usability of the tool.

6.2.2 Usability

Table 5 summarizes the output of the performed log analysis through different phases. To inspect system behaviour in a currently running SCADA system, the users would have to look at individual events (a few thousand per day). By transferring the level of analysis to patterns, instead of individual events, we help stakeholders in aggregating log information. To discard a large number of uninteresting patterns, we perform frequency pattern analysis. With the suggested “cut off” threshold, our stakeholders receive for inspection 131 unique patterns in 14 days. The number of patterns per day varies, but

on average it is less than 10. Finally, after context analysis of suspicious patterns (i.e., an additional round of analyses on suspicious patterns), we estimate that the user had to inspect in average 11 patterns per day.

6.2.3 System performance

Testing has been performed on a machine with an Intel Core 2 CPU at 2.4 GHz and 2 GB of memory. Table 6 shows runtime results of testing. The table consists of three columns. The first column shows the results of our testing on SCADA system logs. The second column shows benchmark results of the pattern mining algorithm by Grahne et al. [11] on the “Accidents” data set [6]. We use these results to estimate the runtime of the expected size of system logs over a year (shown in the third column). The complexity of the preprocessing is $O(n)$. Scalability of the used mining frequent patterns algorithm (in PE) is discussed in [11]. To estimate MELISSA’s performances on an annual SCADA log, we consider benchmarks of the pattern mining algorithm of [11] on the “Accidents” data set. We argue that this data set is more complex than the data set we use, due to the higher number of attributes. Thus, we take the results from [11] as our worst case.

To summarize, we estimate that our tool would preprocess and mine patterns in size of approximately 1 year of work in the stakeholder facility in around 22 min.

6.2.4 Enhancing effectiveness and usability

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

1. derive an automated method to identify patterns that describe normal SCADA behaviour,

Table 5 MELISSA results: size of input logs, size of tool output, number of first and second stakeholders’ inspection

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

Table 6 System performance results and estimation of processing annual logs

Data set information	SCADA log	“Accidents”	Estimated annual SCADA logs
Number of instances	101,025	1,000,000	2,500,000
Total number of items	69	500	70
Avg size of itemsets	6	45	6
Data preparator (s)	22.7	Does not apply	1,080
Pattern mining (s)	0.97	100	200
Total MELISSA processing time (s)	23.6	Does not apply	1,280

Fig. 10 Day 6 pattern analysis

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

2. build a self-calibrating threshold to distinguish between regular and unexpected patterns.

The first direction implies that we can determine which patterns occur with the same (or similar) frequency over a longer period of time. By knowing this, we can build a profile of normal behaviour in the SCADA system over time [35]. For example, we can determine a set of patterns that are regular in their presence and frequency. If a pattern suddenly changes his “regularity”, this can imply that a mimicry attack is taking place. On the other hand, if a regular pattern becomes less frequent, this can imply that a device is malfunctioning or has been reconfigured. Similarly, an operator can use the results of the rare pattern mining to compile rules for alerting similar events. This way the usual alarming system could be improves. By inferring models of normal and anomalous behaviour, we can compile rules and thus turn our tool into an online mode.

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

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

Indeed, we discovered that logs do present certain regularities. For each day in the log, there is a “gap”, which divides patterns with low and high frequency of occurrence. For example, Fig. 10 shows an ordered list of pattern frequencies for day 6 of the analysed log. The first column in the table presents an ordered list of frequency support values. At the top of the list, there are patterns with low frequency of occurrences. These patterns are then followed by patterns with a significantly higher frequency of occurrence. Interestingly, there is a “gap” in values between patterns with low and high frequency. We call the value that differentiates these groups of patterns as the “natural threshold”. Together with the stakeholders, we investigate the character of patterns on the list. The stakeholders agreed that bellow the “natural threshold” there are no events which can be interesting for security purposes. We argue that these patterns represent automated system (re)actions and periodical updates, which are time-triggered and potentially inter-dependant (one event triggers another one(s)). For example, some patterns always occur with the same frequency over days. After analysing 2 weeks of log, we found out that the pattern with the type of action *Services* typically occurs 420times per one working shift (Fig. 10). For this, we suspect that the system is configured so that devices are sending time-triggered messages signalling the online status.

In [30], authors show that observed failures in logs tend to be described in many log entries that occur consecutively forming repetitive patterns. We verified that the high-frequent patterns that we observed are not such burst of events and are spread through the whole day.

Because of this, we believe that the analysis of the log over a longer period of time can provide interesting insights in the content of the logs. For example, we could extract

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

Fig. 11 Frequency of pattern occurrences over 1 week of SCADA log

patterns that are present on every day, and occur with the same (or similar) frequency of occurrence. These observations would define a profile of regular SCADA behaviour.

According to the stakeholders, the patterns above the “natural threshold” represent potentially interesting patterns for the inspection. These patterns are “incidental”. They consist of regular (but unfrequent) user actions and potentially suspicious events. The threshold value varies for different days in the log. Figure 11 shows how the “gap” between patterns of low and high frequency changes over the second week in the log. For some days in the log (e.g., day 13 and 14), it

seems easy to determine the threshold between “incidental” and regular patterns. However, for other days (e.g., day 11), it is hard to decide where the threshold is. Thus, we argue that the threshold value should be determined dynamically.

After inspecting the summarized SCADA log, we argue that there are regularities which are suitable for building a self-calibrating pattern output threshold. Also, we believe that the log contains a number of patterns that describe normal plant work. Unfortunately, at this stage, we cannot confirm our intuitions in a mathematically sound way. This is due to the fact that we only have 2 weeks of plant logs, which is a short time in process life or a stable system, such as SCADA facility.

Nevertheless, we believe that these observations should be further investigated. Also, we believe that the evidences found in the logs further corroborate the paradigm that SCADA facilities are stable and repetitive environments [34].

As a final observation, we acknowledge the difficulty of mining a log, which is provided in an unstructured manner. We remark that the SCADA logs are structured better than some other types of logs, such as telecommunication system [29], console logs [37]. However, in Sect. 4.1, we decide to omit one attribute (*Message description*) from the analysis due to the redundant, unstructured and highly variable character. Although we believe that we did not loose (significantly) on the data quality by doing so, we acknowledge a concern that such solution in general may represent a trade-off. A solution for this would be a log with structured information. Such log could easily be parsed into various (and consistent) attributes, become computer-readable and thus decrease the uncertainty about inferring important information.

6.2.5 Limitations of the approach

We now describe the limitations of our approach.

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

Secondly, an important limitation of our approach is the possibility for an attacker to evade the detection by repeating the same command a number of times. To overcome this, we propose to enlarge the “knowledge window” and so learn what are normal patterns of behaviour over a longer period

of time, as described in Sect. 6.2.4. Since our current log capture is limited, we could not have implemented this yet. This also applies to the limitation of the currently manually set output threshold.

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

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

6.2.6 Approach generalization

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

First, we acknowledge that the deviations identified during the threat analysis are compiled from the tailored Aspect Directory of a specific SCADA application. Thus, the set of undesirable user actions is not universal. This is because, due to the nature of plant process, a set of undesirable actions in one environment might only be a subset of all undesirable actions in another environment. However, after the discussion with the stakeholder from four different facilities (two water treatment, one gas distribution and one power distribution company), we believe that the proposed approach is transferable.

In Sect. 3.1.2, we identify several application and practice vulnerabilities which illustrate that the identified process-related threats can endanger the control system. A security report by the U.S. Department of Homeland Security (DHS) [9] performed on 18 different control systems identified similar vulnerabilities as we did. This confirms that the undesirable actions identified in Sect. 3 pose legitimate threats to various facilities across the domain, thus our knowledge is transferable.

Second, we believe that the log processing cannot be generalized. This is because log content and log format may differ in applications of various vendors. However, under the condition that the parsed log is either (1) provided by the vendor or (2) inferred during the mining process, and does

represent a continuous SCADA monitoring log, we hypothesize that our approach is applicable. We base this on the knowledge that SCADA typically represents a "chatty" systems whose main task is process monitoring (and thus it continuously communicates with its components). Therefore, we expect the logs to be continuous in other environments also.

We plan to test our hypotheses presented here in the future work.

7 Related work

Traditional methodologies for addressing safety problems in process control systems (e.g., FMEA, FTA, HAZOP [18]) do not consider security threats. By introducing a special set of guidewords, Winther et al. [36] show how HAZOP can be extended to identify security threats. Srivantakul et al. [33] combine HAZOP study with UML use case diagrams to identify potential misuse scenarios in computer systems. We take a similar approach to combine PHEA study with HAZOP and analyse user (engineer) behaviour in a SCADA environment.

To detect anomalous behaviour in SCADA systems, authors use approaches based on inspecting network traffic [2], validating protocol specifications [4] and analysing data readings [19]. Process-related attacks typically cannot be detected by observing network traffic or protocol specifications in the system. We argue that to detect such attacks one needs to analyse data passing through the system [2,5] and include a semantic understanding of user actions. Bigham et al. [5] use periodical snapshots of power load readings in a power grid system to detect if a specific load snapshot significantly varies from expected proportions. This approach is efficient because it reflects the situation in the process in a case of an attack. However, data readings (such as power loads) give a low-level view on the process and do not provide user traceability data.

Authors in [30] discuss the difficulties in processing logs with unstructured format. In [17], authors present an approach for failure prediction in an enterprise telephony system. Authors propose to use context knowledge for efficient process visualization and failure prediction.

Several researches explore pattern mining of various logs for security purposes (e.g., alarm logs in [15,20], system calls in [16], event logs in [13]). These authors use pattern mining on burst of alarms to build episode rules. However, pattern mining can sometimes produce irrelevant and redundant patterns, as shown in [15]. We use pattern mining algorithms to extract the most and the least frequent event patterns from SCADA log.

In [21], authors propose to combine various log resources in a process control environment to detect intrusions. The detection is operator-assisted. To the best of our knowledge, only Balducelli et al. [2] analyse SCADA logs to detect

unusual behaviour. There, the authors use case-base reasoning to find sequences of events that do not match sequences of normal behaviour (from the database of known cases). The authors analyse sequences of log events that originate from a simulated testbed environment. In contrast, we analyse individual logs from a real SCADA facility.

8 Conclusion and future work

We analyse process-related threats that occur in the computer systems used in critical infrastructures. Such threats take place when an attacker manages to gain valid user credentials and performs actions to alter/disrupt a targeted industrial process, or when a legitimate user makes an operational mistake and causes a process failure.

Currently, no control (e.g., monitoring tools) is available to mitigate process-related threats. To detect process-related threats, logs could be analysed. These logs hold critical information for incident identification, such as user activities and process status. However, system logs are rarely processed due to (1) the large number of entries generated daily by systems and (2) a general lack of the security skills and resources (time).

We propose an analysis tool that extracts non-frequent patterns, which are expected to be the result of an anomalous events such a undesirable user actions. We benchmarked the tool with real logs from a water treatment facility. Although no real security incident occurred in the log we took into account, at least five events were labelled by the stakeholders as anomalous. We believe that SCADA logs represent an interesting data resource which gives a new perspective on SCADA behaviour. We argue that the analysis of SCADA log represents a complement to the traditional security mitigation strategies.

As future work, we aim at expanding our tool to address anomalous sequences of actions, rather than single events/operations.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) In: Proceedings of the 20th International Conference on VLDB, pp. 487–499. Morgan Kaufmann (1994)
2. Balducelli, C., Lavalle, L., Vicoli, G.: Novelty detection and management to safeguard information-intensive critical infrastructures. *Int. J. Emerg. Manag.* **4**(1), 88–103 (2007)
3. Begnum, K., Burgess, M.: Principle components and importance ranking of distributed anomalies. *Mach. Learn.* **58**, 217–230 (2005)
4. Bellettini, C., Rrushi, J.: Vulnerability analysis of SCADA protocol binaries through detection of memory access taintedness. In: John Hill, L.T.C. (ed.) Proceedings of 8th IEEE SMC Information Assurance Workshop, pp. 341–348. IEEE Press (2007)
5. Bigham J., Gamez D., Lu, N.: Safeguarding SCADA systems with anomaly detection. In: Proceedings of 2nd International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security, LNCS 2776, pp. 171–182. Springer (2003)
6. Brijs, T., Geurts, K., Wets, G., Vanhoof, K.: Profiling high frequency accident locations using association rules. In: Proceedings of 82nd Annual Transportation Research Board, Washington DC (USA), pp. 123–130. Transportation Research Board (2003)
7. Burdick, D., Calimlim, M., Flannick, J., Gehrke, J., Yiu, T.: MAFIA: A maximal frequent itemset algorithm. *IEEE Trans. Knowl. Data Eng.* **17**, 1490–1504 (2005)
8. Burns, L., Hellerstein, J.L., Ma, S., Perng, C.S., Rabenhorst, D.A., Taylor, D.J.: Towards discovery of event correlation rules. In: Proceedings of IEEE/IFIP International Symposium on Integrated Network Management, pp. 345–359 (2001)
9. Control Systems Security Program. Common cybersecurity vulnerabilities in industrial control systems. U.S. Department of Homeland Security (2011)
10. Goethals, B., Zaki, M. (eds.): FIMI '03, Frequent itemset mining implementations, Florida, USA, vol. 90 of CEUR Workshop Proceedings (2003)
11. Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using FP-Trees. *IEEE Trans. Knowl. Data Eng.* **17**, 1347–1362 (2005)
12. Han, J., Kamber, M.: Data mining concepts and techniques, 2 pap edn. Morgan Kaufmann, San Francisco, CA (2006)
13. Hellerstein, J.L., Ma, S., Perng, C.-S.: Discovering actionable patterns in event data. *IBM Syst. J.* **41**, 475–493 (2002)
14. Hieb, J., Graham, J., Guan, J.: An ontology for identifying cyber intrusion induced faults in process control systems. In: Palmer, C., Shenoi, S. (eds.) Critical Infrastructure Protection III, vol. 311 of IFIP Advances in Information and Communication Technology, pp. 125–138. Springer, Boston (2009)
15. Julisch, K., Dacier, M.: Mining intrusion detection alarms for actionable knowledge. In: Proceedings of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '02, pp. 366–375. ACM, New York, NY, USA (2002)
16. Lee, W., Stolfo, S.: Data mining approaches for intrusion detection. In: Proceedings of 7th Conference on USENIX Security Symposium—vol. 7, pp. 6–6. Berkeley, CA, USA, USENIX Association (1998)
17. Lim, N., Singh, N., Yajnik, S.: A log mining approach to failure analysis of enterprise telephony systems. In: Proceedings of the IEEE International Conference on Dependable Systems and Networks with FTCS and DCC, pp. 398–403. (2008)
18. Lees, F.P.: Less' Loss Prevention in the Process Industries. 3rd edn. Butterworth-Heinemann, Guildford (2005)
19. Liu, Y., Ning, P., Reiter, M.: False data injection attacks against state estimation in electric power grids. In: Proceedings of 16th ACM Conference on Computer and Communications Security, CCS '09, pp. 21–32. ACM, New York, NY, USA (2009)
20. Manganaris, S., Christensen, M., Zerkle, D., Hermiz, K.: A data mining analysis of RTID alarms. *Comput. Netw.* **34**, 571–577 (2000)
21. Naedele, M., Biderbost, O.: Human-assisted intrusion detection for process control systems. Accepted for 2nd Int. Conf. on Applied Cryptography and Network Security (ACNS) (2004)
22. Narayanan, N.H., Viswanadham, N.: A methodology for knowledge acquisition and reasoning in failure analysis of systems. *IEEE Trans. Syst. Man Cybern.* **17**(2), 274–288 (1987)

23. Oliner, A., Stearley, J.: What supercomputers say: a study of five system logs. In: Proceedings of 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 575–584. (2007)
24. Paxson, V.: Bro: a system for detecting network intruders in real-time. *Comput. Netw.* **31**, 2435–2463 (1999)
25. Ponemon Institute.: State of it security: Study of utilities and energy companies, 2011. <http://q1labs.com/resource-center/white-papers.aspx>
26. Rantala, R.: Cybercrime against businesses. Technical report, U.S. Dept. of Justice, Office of Justice Programs, Bureau of Justice Statistics, Washington, DC (2004)
27. Rege-Patwardhan, A.: Cybercrimes against critical infrastructures: a study of online criminal organization and techniques. *Crim. Justice Stud.* **22**(3), 261–271 (2009)
28. Rouillard, J.: Real-time log file analysis using the simple event correlator (sec). In: Proceedings of 18th USENIX conference on System administration, pp. 133–150. USENIX Association, Berkeley, CA, USA (2004)
29. Salfner, F., Tschirpke, S.: Error log processing for accurate failure prediction. In: Proceedings of 1st USENIX conference on Analysis of system logs, WASL'08, pp. 4–4. USENIX Association, Berkeley, CA, USA (2008)
30. Salfner, F., Tschirpke, S., Malek, M.: Comprehensive logfiles for autonomic systems. In: Proceedings of 18th International Symposium on Parallel and Distributed Processing, p. 211 (2004)
31. Shaw, W.T.: Cybersecurity for SCADA Systems. PennWell Corp. Tulsa (2006)
32. Slay, J., Miller, M.: Lessons learned from the maroochy water breach. In: Goetz, E., Sheno, S. (eds.) *Critical Infrastructure Protection*, vol. 253 of IFIP International Federation for Information Processing, pp. 73–82. Springer, Boston (2007)
33. Srivatanakul, T., Clark, J., Polack, F.: Effective security requirements analysis: Hazop and use cases. In: *Information Security: 7th International Conference, LNCS 3225*, pp. 416–427. Springer (2004)
34. Stouffer, K., Falco, J., Scarfone, K.: Guide to Industrial Control Systems (ICS) Security, NIST Special Publication 800-82. National Institute of Standards and Technology (2011)
35. Vaarandi, R.: Tools and techniques for event log analysis. PhD thesis, Tallinn University of Technology (2005)
36. Winther, R., Johnsen, O., Gran, B.: Security assessments of safety critical systems using hazops. In: *SAFECOMP '01: Proceedings of 20th International Conference on Computer Safety, Reliability and Security, LNCS 2187*, pp. 14–24. Springer, London, UK (2001)
37. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.: Mining console logs for large-scale system problem detection. In: *Proceedings of 3rd Conference on Tackling Computer Systems Problems with Machine Learning Techniques, SysML'08*, pp. 4–4. USENIX Association, Berkeley, CA, USA (2008)