SEVENTH FRAMEWORK PROGRAMME

Theme SEC-2011.2.5-1 (Cyber attacks against critical infrastructures)

D6.5 Distributed Detection Framework

Contract No. FP7-SEC-285477-CRISALIS

Workpackage	WP6 - Network-driven detection
Editor	Rens van der Heijden, Frank Kargl
Version	1.0
Date of delivery	M40
Actual Date of Delivery	M40
Dissemination level	Public
Responsible	UULM
Data included from	UT

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°285477.

SEVENTH FRAMEWORK PROGRAMME
Theme SEC-2011.2.5-1 (Cyber attacks against critical infrastructures)

The CRISALIS Consortium consists of:

Siemens AG	Project coordinator	Germany
Alliander		Netherlands
Chalmers University		Sweden
ENEL Ingegneria e Innovazione		Italy
EURECOM		France
Security Matters BV		Netherlands
Universiteit Twente		Netherlands
University of Ulm		Germany

Contact information:
Michael Munzert
Siemens AG
Otto-Hahn-Ring 6
81739 Munich
Germany

e-mail: michael-munzert@siemens.com

Contents

Abstract

This deliverable discusses a framework for distributed misbehavior detection, capable of collecting and evaluating data communicated within a distributed control system (DCS). This data can be analyzed within the framework by individual mechanisms that detect attacks in different ways – either by specification or by anomaly-based means. The framework can also accept results from external detection mechanisms, such as a host-based intrusion detection system (HIDS). These components can be added and removed dynamically, and the framework merges the results produced by both types of mechanisms together using a reasoning componenent based on *subjective logic*. We evaluate the feasibility of this approach by implementing a proof of concept, which we apply to the UT Testbed described in Deliverable D3.1.

1 Introduction

In this deliverable we develop a framework to perform *misbehavior detection* in cyber-physical systems (CPS). Misbehavior detection refers to the detection of attacks that are related to the correctness of the data: a misbehavior detection framework can therefore answer the question *"Does this data correspond to the real situation?"*. Our framework will focus on the security aspects of this question, but the same question is also interesting from a safety perspective. To determine whether data corresponds to the real world, our framework will rely on a distributed set of detectors, which can be programs that are distributed throughout the network (*external detectors*), or results from detectors that operate on the data within the framework (*internal detectors*). An example of a external detector would be a HIDS, which triggers an alert when unauthorized activity is detected, while an example of an internal detector would be a prediction algorithm that compares a received voltage measurement to an expected voltage. The framework will take these results and merge them using a logic framework called *subjective logic* [5]. One of the most important reasons to undertake this effort is that misbehavior detection mechanisms are often designed to detect a very specific class of attack. Our framework allows the combination of individual detection results produced by individual detectors, regardless of where these detectors are executed (internal or external). The framework is implemented as a proof of concept, which is discussed in this deliverable: some initial tests show the advantages and disadvantages of our approach.

1.1 Cyber-physical Systems

Our framework is being developed for cyber-physical systems (CPSs), which is loosely defined as computer systems that interact with the real world. Within CRISALIS, several types of these systems are analyzed: DCS, industrial control system (ICS) and advanced metering infrastructure (AMI), but there are many other types of CPSs, such as vehicular networks. This section briefly discusses the common characteristics of these systems, which are necessary for the framework to work.

1.1.1 Industrial Control Systems

industrial control system (ICS) are a prime example of a (networked) CPS. In previous work [7], we have identified several security-relevant characteristics of CPSs, that lead to specific requirements compared to standard information and communication technology (ICT) systems. These characteristics are:

Ubiquitous access In many CPSs, there is either no clear boundary between insiders and outsiders, or this boundary contains so many devices that some of these may be subject to compromise.

Limited physical security As nodes in CPSs are often distributed in a potentially hostile environment, they may be subject to hijacking, analysis, and reprogramming by attackers, even if they are centrally managed.

Potential impact CPSs control critical infrastructures in which a large variety of attackers may have an interest. Due to the direct connection with human safety, CPSs can be a target for attackers with high financial resources.

Sensor values as security assets In typical network security scenarios, the primary security assets are the components in the network and the packets or user data transmitted in the network; these assets may be subject to attacks, and the goal in network security is to protect them from these attacks. On the contrary, in CPSs, the primary security assets are the sensor values contained within transmitted packets, as well as the actuators controlled based on this data. Consequently, the networking patterns in CPSs are application-dependent and built around the semantics of exchanged information, rather than the classical ISO/OSI stack of communication layers.

1.1.2 Use case description

Because our evaluations base on the UT testbed (see D3.1), we briefly describe the concrete use case for which this testbed is implemented. The testbed models a power distribution network and provides the control algorithm necessary to adapt energy produced by a control algorithm, implemented on a programmable logic controller (PLC), based on measurements of the current demand from the network. The demand is measured by another PLC, which measures the current voltage on the grid. The testbed is not concerned with an AMI, but it instead concentrates on changes in the grid itself.

The aim is for the grid to remain as stable as possible, and therefore detection mechanisms will be designed to detect attacks that influence that stability. This could either happen due to packet injection or other malicious activity causing the controlling PLC to behave incorrectly. In addition, measurements are displayed to an operator through a human-machine interface (HMI), which shows the raw measurement data transmitted to it by the measuring PLC. If the operator sees falsified values at the HMI, she may believe the control algorithm is malfunctioning and perform manual changes to the process, or conversely may not observe an ongoing attack. Therefore, we also want to validate the consistency of the measurements with the behavior of the controlling PLC, and the plausibility of the measurements themselves.

This process requires that we have access to the communication performed between PLCs. As the UT Testbed is set up to use S7 as its communication protocol, we need to be able to parse this protocol. For this purpose, we use Wireshark's S7 protocol dissector[1], which supports both S7 and the S7+ protocol used to communicate with the HMI. We also use code developed at UT to convert the raw measurements into actual voltage readings, which it computes as $V = \frac{x \cdot 10}{27648}$. All other proof of concept code is available by request to the authors.

1.1.3 Misbehavior Detection

Misbehavior detection may be regarded as a special case of intrusion detection, which detects attacks through the application of models that describe correct behavior. This is similar to specification-based intrusion detection, where detection occurs based on protocol semantics, a topic that was extensively studied as part of WP6 [2, 1] as well as by many other authors. The difference is that misbehavior detection is primarily concerned with application layer semantics and physical world behavior. Taking our use case description from the previous section as an example, a misbehavior detection mechanism detects inconsistencies and implausibilities in measurements, or inconsistencies between different sources of data. Our framework aims to merge results from misbehavior detection mechanisms and other intrusion detection mechanisms in order to resolve some of the challenges associated with misbehavior detection, such as the fact that it is often highly application-specific.

[1] Available at `http://sourceforge.net/projects/s7commwireshark/`

1.2 Logics to reason about data

In this section, we briefly describe a logic called *subjective logic*, which we use as a tool to reason about the output of different misbehavior detection mechanisms. Subjective logic was originally developed for distributed trust management, but it has found application in other areas, such as sensor data fusion. Because it works as an extension of several other logics, we discuss it in the context of those other logics.

1.2.1 Comparison with other logics

There are many other logic systems that can be used to fuse data from different sources into a single conclusion. This is exactly the use case we have; we have output from different detection mechanisms that we merge to reach an overall conclusion about the fact observed by those mechanisms. There are many ways to merge this evidence, including (weighted) majority voting, Bayesian inference and dempster-shafer theory (DST); subjective logic combines ideas from several of these, which we will therefore introduce first. A significant part of this section is based on earlier work by Raya et al. [6]. They use these different logics to combine reports from different data sources (in their case, other vehicles; in general, this could include sensors).

Majority voting Majority voting is a simple approach that takes the binary outputs (attack or no attack) of the different mechanisms and makes a decision by outputting the majority; either there is mechanisms claiming an attack, or not. A slight improvement of this approach is to average all the inputs, e.g., summing with values 1 for attack and 0 for no attack and dividing by the amount of inputs. This outputs an attack probability.

Weighted majority voting Taking the idea of majority voting one step further, weighted majority voting takes probabilistic outputs from mechanisms and computes the average of these results, as above. It also outputs an attack probability, but preserves more information from the outputs of the individual mechanisms.

Most Trusted Rather than relying on potentially unreliable measurements from different detectors, this approach simply selects the most reliable result. One problem with this approach is that it is necessary to estimate the reliability of the output of different detectors. If we assume all detectors have a similarly distributed probabilistic output, this approach will always take the detector that reports the highest attack risk; thus it will potentially lead to many false positives.

Bayesian Inference This approach uses Bayes' Theorem to determine the probability of an event being true by computing its probability given a set of observations. For example, one might determine how likely it is that an attack occurs given all the observations, assuming all probabilities of an attack given these individual observations are known, and the probability of an attack is known.

Dempster-Shafer Theory Dempster-Shafer Theory is an early approach to separating *confidence* and *correctness* in probability theory. The idea is that we can model the distribution of an event (how likely is it that there is an attack?) from the confidence in our measurements (how reliable are my measurements that indicate this attack probability?). The theoretical underpinnings of this approach are beyond the scope of this deliverable; however, an important application of this theory is in (distributed) trust management, where it is used to predict the reliability of other participants, and this is what Raya et al. have evaluated.

1.2.2 Subjective Logic

This section briefly introduces subjective logic; for the mathematical details, as well as information about current research and extensions of the logic, refer to [5, 4].

The purpose of subjective logic is to represent both uncertainty and probability of occurrence for an event in a single data structure, called an opinion. The simplest type of opinion consists of four elements; belief, disbelief, uncertainty and base rate. An opinion is always issued by a subject and given on an object (sometimes called statement). The base rate represents the probability that a statement is true, in the absence of any additional information. Any additional information that a subject has is represented by her belief and disbelief about the statement, respectively indicating how likely and how unlikely the statement is. Finally, the uncertainty represents the lack of knowledge about the statement. For example, consider the statement "it will rain tomorrow": the base rate could represent the probability that it rains at any given time, while belief depends on how much it rained today, and disbelief on how long it did not rain. Uncertainty could then represent how often the weather changes. Clearly each of these parameters are related, which is why the subjective logic opinion represents these three values as a single data structure, such that the sum of these values is 1. We can also easily calculate a prediction about an event by computing $e = b + a \cdot u$, where b is belief, a is the base rate and u is uncertainty; then the result e is the probability of the event, modified for uncertainty.

Subjective logic opinions can be merged in several ways. First of all, there is *trust transitivity*: if subject A has an opinion of subject B, and subject B has an opinion

about statement C, we can compute a transitive opinion by A on C. If A has additional information, she can merge multiple opinions through *fusion*, to compute a final opinion on C. There are several ways to perform this fusion, which depend on whether or not the evidence that the opinions are based on are independent (in the sense of probability theory). For this work, we assume all variables we have are independent; future work may improve on our work by adding dependencies to our world model (see 3.1).

2 Motivation

This chapter describes the attacks on our use case in detail, discusses the potential impact of these attacks, and describes how misbehavior detection and a distributed detection framework can be elements that improve the detection rate. Most of this is parallel to existing detection approaches, an idea that we will also support in this chapter. Finally, we describe the attacker model in more general terms, and the attack scenario that will be used in the validation of this work.

2.1 Misbehavior

Before discussing the attacker model in detail, this section provides a motivating example for misbehavior detection, when compared to other intrusion detection systems.

An attacker gains access to the network of an industrial plant and can transmit falsified messages in this network. Because the messages between the PLC and the engineering workstation (EWS) are not properly protected, the attacker can spoof communication between them: this can be protected against by applying standard cryptographic techniques. However, when the attacker manages to compromise a PLC (see e.g., Stuxnet), he can transmit arbitrary messages in the name of the PLC, and these will be properly signed. This is an example of a semantic attack on the application layer. However, when we employ misbehavior detection mechanisms, we can potentially detect these attacks by looking at the message contents. If the attacker manipulates sensor values, we can detect an irregular or unexpected pattern in these values. Although this could lead to alerts for other irregular or anomalous sensor readings, this is not problematic: such readings imply a real issue with the plant, and thus it would be good to issue a warning.

2.2 Attacker Model

The type of attacks we are trying to detect are similar in scope to Stuxnet, meaning we assume an attacker with a large pool of resources and some intrinsic interest in the success of an attack (e.g., military or economic interest in damaging specific components within an ICS). The fact that this attacker has a lot of resources means that they will likely possess some kind of 0-day exploit that cannot be detected using signature-based

intrusion detection or virus scanners. In addition, this means that the attacker will be able to determine which hardware is deployed, and that the attacker can at least partially replicate this setup.

This creates a very powerful attacker that is difficult to defend against using only perimeter security. Although our attacker is very powerful, we note that our attacker model excludes an attacker that can either compromise the security architecture itself[1]. Our aim is to work towards an attacker model that can replicate parts of the security architecture, including models learned by anomaly- and knowledge-based intrusion detection systems (IDSs) and detectors used by the framework (obtained, for example, by reading scientific publications)[2]. The idea is that attack from such an attacker is either undetectable, because all data is perfectly consistent, or the attack has no significant impact (e.g., increasing all voltage measurements by the minimum possible value, which will be within tolerance levels). In other words, we assume that an attack will inevitably trigger multiple effects, that can be detected separately but attributed to the same attack. However, showing whether or not this is fundamentally true remains future work.

Finally, we consider denial of service attacks to be out of scope, as their detection requires orthogonal mechanisms that cannot rely on our framework, depending on the type of denial of service attack we are aiming to detect. However, it is technically possible to add a detector to our framework that monitors for a particular type of message, such as a keep-alive message.

2.3 Attack Scenario

Due to the problems with the testbed (see section 4.1), it was not possible to implement the original attack scenario, which was based on a generalization of Stuxnet, and consisted of the following capabilities:

1. Infect engineering workstation (EWS)
2. Deploy infected/modified PLC code
3. Send incorrect commands
4. Hide the effects by grid PLC responses

[1] Here, security architecture refers to the framework itself and the communication with external detection components. We consider attacks on these to be a separate research area.

[2] Basically this means that the attacker has some way to check whether his attack works on a replicated security system, and can thus test her attack repeatedly, as often done by malware developers.

a) ... by injecting sensor values on network level

b) ... by injecting/replaying to the HMI

Notice that each of these steps can conceivably be detected by some type of IDS; however, to avoid building a detector for each and every possible way to achieve these steps, we need a generic solution. For example, infections on a workstation may be discoverable by a HIDS, but this system will not have a false negative rate of zero (without having 100% false positives). Therefore, we must assume that the attacker can take this first step and determine how we can detect other actions an attacker might take. Building a detection mechanism for these has the same implications. The purpose of our framework is to combine the output of these different detectors and thus reduce the false positive rate without significant impact on the false negative rate. This is possible because the attacker is restrained by his goals: the attack is only considered successful when it has a significant impact on the system (see also the discussion of the attacker model in the previous section).

Due to the testbed issues, we were only able to implement attacks and detection for 1., 3. and 4b., rather than all of the points discussed above. The details of this implementation will be discussed in Section 4.3.

3 Architecture

This chapter describes the architecture of our framework. It is based on earlier work [7, 8, 3] and extends that work from vehicular networks to ICSs. For the latter to work, the framework needs to be extended with *external components*, which transfer sensor data from outside sources to the framework. These external components are semi-trusted; we assume they are not compromised, but they may send false data (for example, because they observe false data on the network).

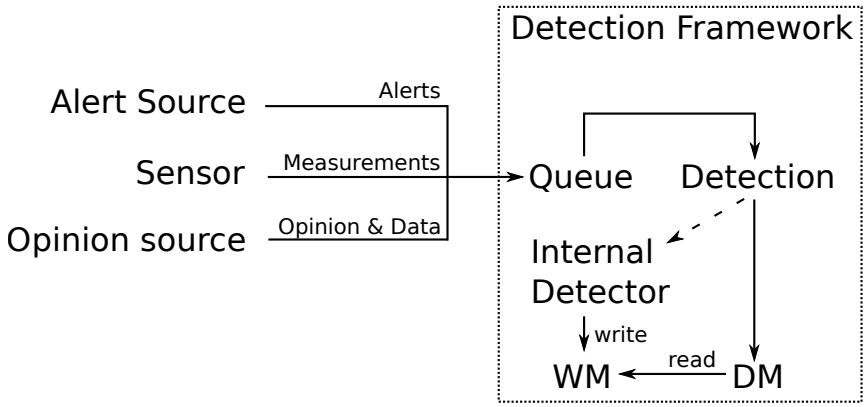


Figure 3.1: This figure shows the architecture of our framework: there are three types of data coming in from the outside, which are processed sequentially through a detection step, which asynchronously executes internal detectors. The data is stored in the world model, and then the decision module (DM) is triggered to determine whether an alert should be raised. The world model (WM) is further detailed in Figure 3.2.

In Figure 3.1, the essential components of the architecture are shown. There are three types of (external) sources of events; alert sources, opinion sources and sensors. These provide different types of input for the framework, which is synchronously processed from a queue. For each item, internal detectors are asynchronously executed; these write to the world model, followed by a decision from the decision module. The world model is shown in more detail in 3.2; it represents information and detectors as vertices,

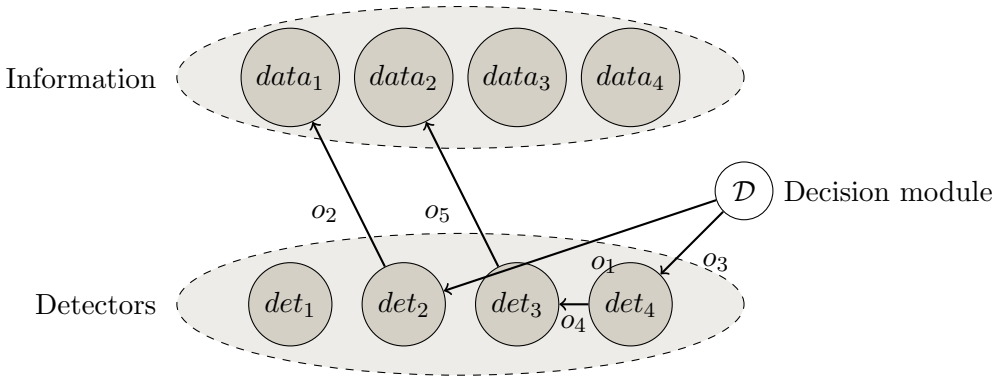


Figure 3.2: This figure shows the graph representation of our world model. It is based on [3], but adopted for ICSs; note that detectors det_i may be internal or external.

with (subjective logic) opinions forming edges between them. The decision module in this graph shows how a decision can be made; by computing all paths to a piece of information and then transitively applying the appropriate subjective logic operators. In this work, the edges from the decision module to detectors are simply $(1, 0, 0)$; in future work, we intend to change these values to scale the opinion based on detector performance.

3.1 Components

The essential components of our architecture are the representation of data items, the different types of detectors and the fusion of the evidence provided by these detectors. Data items can contain binary statements, alerts or measurements, and they can be thought of as observations made by sensors. They can be in a raw format or processed by some algorithm (for example, if the measurement frequency is too high to transmit, several measurements may be aggregated). Detectors evaluate data items and can be internal or external: external detectors transmit data items to the framework, possibly processed and possibly annotated with Opinions, while internal detectors process data items that are received from other detectors. Conceptually nothing prohibits the implementation of internal detectors that verify the inner workings of other detectors, although some performance limitations exist (e.g., when two detectors create conflicting results about each others' reliability, creating cycles). The detection results are triplets of the detector that produced the result, the opinion it produced, and the data item on

which the detection was performed. These results are then processed by the decision module, which merges the opinions using the operations provided by subjective logic, and creates an alert when a threshold θ is crossed.

Although opinions are merged by the decision module, the individual opinions created by the detectors remain stored in the framework as evidence, which can be provided to the user when an alert is triggered. This can help the user to determine the nature and source of the alert. Some data items can change over time: for example, a sensor produces sensor values periodically, and all these measurements measure the same thing in the physical system. Thus, these measurements are stored in the same data item; for performance reasons, older measurements may be discarded or aggregated. Another good reason to store these measurements in the same data item is that they can often be used to evaluate the correctness of new measurements. In some cases it may also be possible to evaluate sensor performance, which could also be an internal detector.

Although our discussion so far has only classified detectors as internal or external, there are also other potentially useful classifications, such as based on the type of data they can assess (node- or data-centric). A data-centric detector evaluates the data items available to it and determines the correctness thereof. Most of the examples we have discussed in this deliverable are data-centric: they analyze measurements and issue an opinion on their correctness. Many of these detectors already exist in the scientific literature, and the framework is designed in such a way that these detectors can be included as a data-centric detector. Another type of detector is the node-centric detector, which looks at the behavior of certain nodes and determines how reliable they are. This is especially useful when external detectors are deployed in unprotected or insecure environments, or where these are operated by external entities. This is not typical for ICSs, but it is a likely situation for many other CPSs, which is why we include them in our discussion.

As briefly discussed in the section on subjective logic, there are some limitations as to which opinions can be merged. In particular, it only makes sense to merge opinions that concern the same, or somehow related data items in the framework, because the purpose of the merge process is to determine what the overall opinion on that particular data item is. The decision module will, for example, merge two different opinions from two different detectors that have evaluated the same data item, containing sensor values from the same source, to determine whether the most recent sensor values are reliable. The decision module can then use the threshold θ to determine whether or not the sensor values are sufficiently reliable, and can trigger an alert to the user if these are in a dangerous state. The user can then use the results from the individual detectors to determine what is going on, or send someone to the sensor to check and possibly replace it.

3.2 Detection System

The triplets in which the detection results are stored, consisting of detector, opinion and data item, are a simplistic representation of a graph. This graph contains the individual data items and detectors as vertices, while the opinions are edge annotations. In addition, the decision module can be added to the graph, which will then have edges to the detectors. This allows us to model the decision process as finding paths to different data items, merging these paths through the subjective logic operations, until there is only an opinion from the decision module to the data item. This opinion represents the validity of this data items' content, which can easily be reduced to a single value and compared with a threshold.

Although this process provides a fairly intuitive and clear representation of the detection process, the system described so far also has some open issues:

- Opinions on data items are always on the most recent value
- Representing conflicting or correlated data items
- Cycles in the graph need to be resolved
- Synchronized access to the graph

In the remainder of this section, we discuss different possible resolutions for these issues.

3.2.1 Time

If a triplet of detector, opinion and data item contains measurements from a particular sensor, these will update periodically. When the data item is updated, and then the decision module determines the correctness of the data item, the opinion may not yet have been updated. There are a number of solutions to this problem. A simple solution would be to add timestamps to the Opinion, such that at every time t, the decision module can find the most recent measurement in the data item that was known at this time. However, this means that the detection process may take significantly longer, and the opinions on older sensor values are still lost. Another approach is splitting the data item into multiple data items: one for each measurement. This allows for a much more detailed picture of the sensors' behavior over time, and the detectors' opinions of this behavior, but it will have a prohibitive cost in terms of processing time, because the graph will grow really quickly.

3.2.2 Conflicting or Correlated Data

Although it is quite simple to represent data that is conflicting, simply by adding the data items that contain this data to the graph, it still poses a potential source of problems in the decision module. The reason that it is important to know that data is correlated here is that subjective logic provides different operations to merge this data depending on whether the underlying probability distributions are correlated or not. Although it is possible to assume that data is not correlated, when in fact it is unknown whether they are correlated, this may affect detection rates. Correlation is also relevant when working with the detectors: often, a detector will use data from multiple sources exactly for this reason. For example, a detector may compare measurements from three different sensors measuring the same event at the same time, and determine what the correct measurement is through averaging or majority voting. Such a detector relies on a correlation between the data items are processed; in this case, the individual data items can actually receive opinions based on their deviation from the result. However, when only two sensors are used, it is generally impossible to say which of the two is correct, without additional information. Our framework aims to provide this additional information through other detectors, but this requires that we give the conflicting data an opinion from the detector that spots the conflict.

There are several possible solutions to this issue: one way is to give both data items the same opinion, and rely completely on subjective logic to take care of determining which of the two is correct. This greatly simplifies the process of adapting individual detectors to work within our framework, but it may not produce the best detection results, as some of the information available is not represented in the graph. Another approach is to allow the detector to update its produced opinions based on results produced by other detectors. This allows the detector to respond to new information – for example, another detector determines that one of the two data item is almost certainly false –, but this creates the potential for cascading effects caused by false positives of a single detector. It also means that the order in which detectors are executed matters for the results. For this reason, we have chosen to take the initial approach.

A further alternative worth discussing is to somehow extend the graph to contain this information. There are two options: adding edges and adding vertices. Adding an edge to represent conflicts or correlations is the most intuitive way: an edge between two data item vertices represents a relation between these vertices, and it can have a correlation coefficient that provides the exact correlation. This is a useful way to represent the information, but it is difficult to apply subjective logic with this correlation directly, which means the decision module becomes more complex. Additionally, this approach will create a highly connected graph, which makes the evaluation of a single

data item dependent on many others, increasing the computational complexity of the decision module as well. Another alternative is to add correlation vertices, which have a statement like "data item A and data item B are mutually exclusive", with opinions from it to the two other data items. This integrates quite well with subjective logic; however, it leaves open the challenge of what these data items should be, and the only way to deal with new information is to update these opinions, bringing us back to the earlier discussion of opinion modification.

3.2.3 Cycles

Cycles in the graph can be created either by node-centric detectors, as discussed above, or due to the introduction of additional types of edges in the graph, such as those discussed as a potential solution for the representation of conflicting data. They are symptoms of a general problem with trust: when two sources each claim that the other source is bad, it is necessary to decide which of these to trust more. This provides some kind of resolution of the cycles in the graph, but could lead to some problems. For example, if one of the detectors has an opinion based on new data, while the other has an opinion based on outdated data, then discounting the newer opinion leads to higher detection latency, while discounting the older opinion makes the system more vulnerable to false positives. Determining the right approach is likely dependent on the particular detectors and use case, and therefore out of scope of this deliverable; instead, we will work with the result from the most reliable source, even when the two sources are almost equally reliable.

3.2.4 Synchronized Access

To ensure that the framework produces deterministic results, access to the graph must by synchronized, and detectors should all receive new data items as they are received by the framework. When the graph is modified and read in parallel, some detectors may use newer information than others, which produces opinions that may conflict for this reason only. However, synchronized execution of all detectors is not an option, due to the time required to execute for some detectors. However, we can exploit some parallelism if we require all detectors to output their results independently, before writing these results into the graph in a synchronous way. Many performance tweaks remain possible in this approach; breaking off detectors that take too long could be an option, or even applying an expensive detector only to some of the data items, where results are uncertain. These and other performance optimizations are not part of the scope of this deliverable, as the purpose here is to develop a proof of concept that shows it is worth combining detectors

on the first place.

3.3 Practical Challenges

In addition to the conceptual issues described above, there are some practical open issues that need to be addressed. In particular, it is necessary to decide how subjective logic values are assigned by the detectors, and what their semantics are – does a positive opinion (high belief) say that the data item is likely to be correct, or that it is likely to be an attack? For our work this decision is quite arbitrary, and we will consider high belief to represent the *correctness* of the data, but it requires that we be careful with regards to the data items. In particular, the decision module needs to be aware whether the correctness of a data item means that an alert should be issued or not. For example, if we have a data item with measurements that show a temperature that is much too high, the decision module should issue an alert. An alternative to this approach is that we assign opinions based on how much the received data items deviate from the expected behavior; however, this requires that we also consider safety in our framework. Our prototype will rely on the latter implementation, but further research may be needed to determine whether this is the best approach.

A second important challenge is that we need to consider the fact that existing external detectors will not be able to communicate opinions. As opinions allow the representation of certainty and correctness, receiving only a probability or a warning means that some information is lost. In general, existing detectors were likely not designed to work with subjective logic, but our implementation should support the integration of such mechanisms to reach a good performance. One of the goals of the prototype is to evaluate how to do this correctly.

4 Proof of Concept

This section describes the implementation of a proof of concept that demonstrates the feasibility of our framework.

4.1 UT Testbed

The testbed in Twente is a small proof of concept of a SCADA system, whose main purpose is to demonstrate to the general public the possible impact of SCADA attacks, and therefore the relevance of the work done in CRISALIS. The testbed consists of a measuring PLC, a PLC that controls the power plant, an HMI, a switch and a SCADA workstation, and it is described in detail in CRISALIS deliverable D3.1. The control algorithm used in the testbed is relatively simple and sometimes causes spikes without changes in the load on the grid. Originally the testbed worked such that the measuring PLC transfers its measurements over the network to the controlling PLC, which then changes the output of the power plant using this data.

4.1.1 Problems

Unfortunately, the PLCs are no longer operating as discussed in D3.1, presumably due to a soft- or hardware fault in one of the PLCs. For this reason, there is no observable PLC to PLC communication available in the testbed network. For presumably similar reasons, it is also not possible to observe directly how the control algorithm is behaving (i.e., whether it is in- or decreasing power plant production). This issue was detected relatively late in the development process, because we initially relied on samples recorded from the testbed. Despite our best efforts to find the issue, it was no longer possible to repair the testbed, and we have adapted our experiments to compensate for this issue.

4.2 Framework implementation

The framework itself is implemented in Java 7, and includes the components described in the previous chapter. The overall workflow is shown in Figure 4.1: for simplicity, we use a simple queue in which packets are stored; these packets are then polled from the queue

Figure 4.1: Overall workflow of the framework. Marked in red are the parts that are explained in more detail.

and processed in parallel by every detector. The parallel execution poses some limits on our framework, which we discuss in more detail in the evaluation. The detectors produce results, which are stored in the world model, represented as a list of detection triplets. After this parallel execution, the decision module will decide whether or not to raise an alert, after every detector has finished (although future implementations may be more flexible here). A more detailed flowchart describing the detection process, including references to the individual classes, is given in Figure 4.2.

In more detail, the framework is started and awaits connections that will provide it with data. For each incoming connection, it creates a Thread that maintains the connection, stores received information in the Queue and tears down connections when necessary. Because the framework can deal with different types of data, the thread will first receive a StartupPacket, which defines where the data is coming from and what kind of data it is. For simplicity, our prototype also relies on this to start the appropriate internal detectors, although future work may allow the dynamic starting of internal detectors with different parameters.

After the initialization process, whenever new data is received, the merging phase occurs. In this phase, the framework checks whether it has received earlier data from the same source and stores the new data directly in the framework. This is particularly helpful for sensor data in our framework, which has a list of old measurements that can be used for detection. The resulting data item is passed to the detection phase.

In the detection phase, the internal detectors are each called in their own thread and process the given data item. Detectors may have state, which is independent of the framework; each detector is responsible for its internal consistency. Detectors are responsible for filtering which data they rely on; they can be given arguments to monitor data from a specific source (or multiple sources). After the detector's algorithm executes, a subjective logic opinion is created to represent whether the given data is reliable. This information is then stored in the world model, which in our prototype is a list of triplets that represents the graph described in Chapter 3.

After all detectors terminate, the framework uses their results to re-evaluate the data currently in the world model and determines whether or not an alert should be raised. When detectors produce a new opinion on an existing data item (i.e., when new sensor

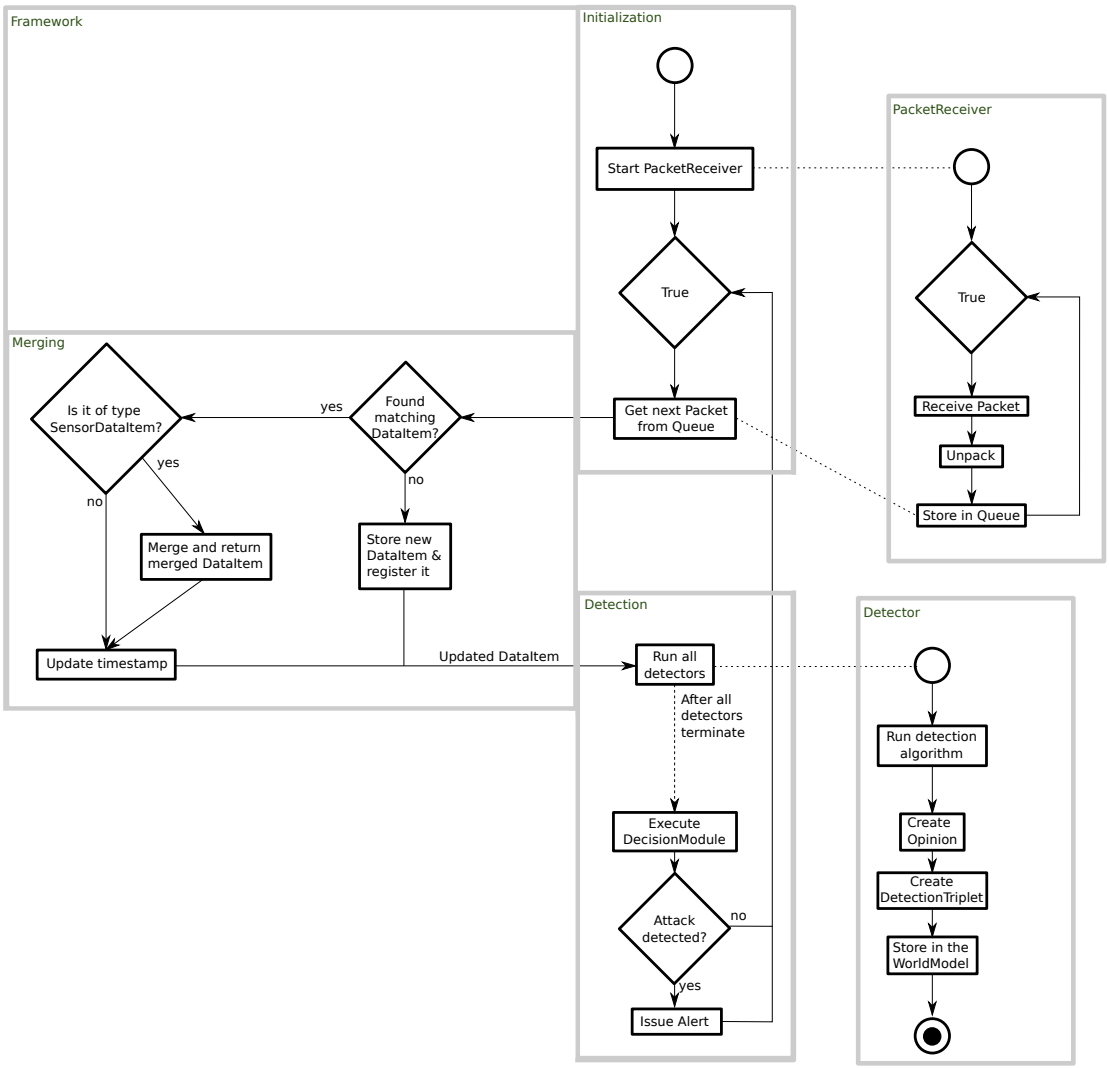


Figure 4.2: Detailed description of the workflow from Figure 4.1.

values are added), our prototype replaces the existing opinion[1]. The different opinions on the same data item are then merged to determine a merged opinion, which in our framework is used only to determine whether an alert is triggered or not[2] and are not stored in the framework.

4.2.1 Implemented Detectors

Figure 4.3 contains an overview of the detectors that were originally planned for implementation, and how they are to be combined. Note that the flexibility of the framework is preserved in our implementation – this just shows which particular detectors are implemented, and what data they will use. In particular, there are three external and three internal detectors: a HIDS, which provides alerts when activity is detected on a server; an S7 parser, which extracts raw voltage readings and forwards them; and a PLC-Command parser, which forwards the command issued by the controlling PLC. The last of these also includes an opinion based on its assessment of the PLC's behavior, which can be imagined as the same control process being implemented in software. This could allow us to detect manipulations of packets that are sent to the controlling PLC, but not to the framework through the S7 parser, for whatever reason. The internal detectors check the plausibility of the voltage (is it within a reasonable range?), correlate the values received from the different PLCs, and correlates activity on the server with changes in the commands the PLC sends. Finally, the alert produced by the HIDS is also converted into an opinion. This produces a total of five types of opinions for the decider, which it uses to create output.

However, as discussed in Section 4.1, issues with the testbed meant that critical information for this process was not available. In particular, we could no longer observe measurement values communicated between the sensing and the controlling PLCs. However, we were able to modify our experiments to use data transmitted to the HMI, which also receives and displays measurements. This meant that we also had to modify the corresponding detectors. The voltage plausiblity detector now checks whether the measurements sent to the HMI are moving away from the set point, using the difference between the last two measurements, after a certain threshold. The set point and the thresholds are a parameters for this algorithm. The voltage correlation algorithm, on the other hand, now checks whether the commands issued by the controlling PLC are

[1] Future work will determine whether the optimal strategy is to replace opinions or to gradually change them by merging new opinions into old ones. Replacing opinions is simpler and more flexible, because gradual change can still be implemented in the detectors themselves.

[2] Future work could store merged opinions, but this may lead to cascading effects in the detection results.

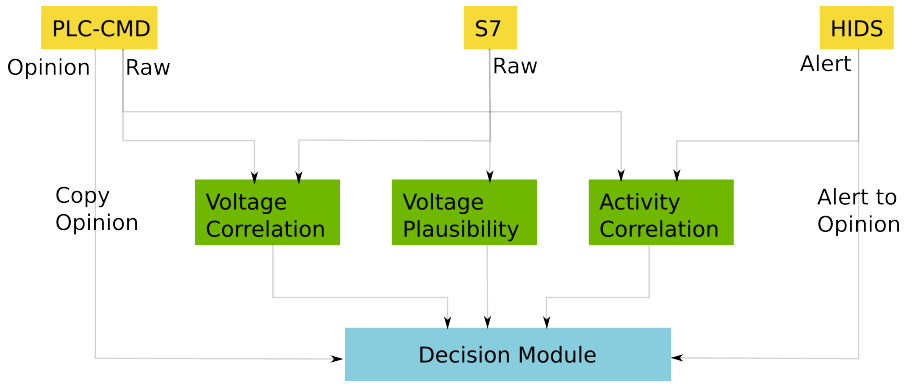


Figure 4.3: This figure shows how the individual detectors can be integrated. On the top, in yellow, the external detectors are listed; the middle parts in green are the internal detectors. The type of external detector is denoted directly below the external detector. Note that it is easy to add new internal detectors, as well as external detectors that rely on opinions; we can simply write the necessary code that specifies which inputs are needed, and which output is generated.

maintaining equilibrium. It assumes that the grid is in a stable state and aims to remain in that state; it verifies whether this is the case by checking that the output of the PLC is the opposite of what was measured. Note that this second detector is only simulated, because as discussed in Section 4.1, we did not obtain access to the operations performed by the PLC. This means that opinion generation at the PLC, meant to demonstrate the distribution capabilities of the framework, we also not implemented. The conversion of HIDS alerts to opinions remained unchanged.

4.2.2 Transfer Protocol

The transfer protocol consists of different packet types; the actual transfer occurs through simple Java Object Streams. A real implementation must use some secure transfer protocol underneath; for this prototype, we will not implement this, but concentrate on the detectors. Our protocol supports three modes of external detectors, which represent different levels of detail and framework integration:

- Raw detectors
- Alert detectors

- Opinion detectors

Raw detectors simply forward raw measurements to the framework as data items. Alert detectors operate on alerts raised by other detection systems, such as a HIDS. Opinion detectors represent the distributed nature of our framework; such a detector independently performs some misbehavior detection on the data available to it, produces an opinion on that data and then forwards it to the framework. In particular, such a detector could be a different instance of the framework at a different location. This can be used to create a hierarchical structure without significant effort.

A potential enhancement for future work is to use a more extensible protocol to implement the transfer protocol, using formats like custom XML or RDF, but this was considered out of scope for this prototype.

4.2.3 Fusion

The implemented framework creates a world model that is comparatively simple, when compared to what is discussed in Chapter 3, because we do not include timestamps in the opinions at the moment. Similarly, merged results are not stored, although the implementation is structured such that these enhancements could be implemented without completely revising the code. Instead, we chose to focus our prototype on evaluating against different attacks and testing detectors with different parameters. The fusion of individual detection results occurs through the consensus operator (which is a commutative operator), and the decision of whether an alert should be raised is made using the expectation of the opinion and a threshold.

4.3 Experimental Setup

We conducted our experiments on the UT testbed. Originally, it was planned to perform a variety of packet injection attacks, but due to the issues discussed in 4.1, this was no longer an option. To still be able to present some initial results, we created a large test trace to work with. The trace contains a stable behavior and several spikes due to users changing their power consumption, and due to the green energy sources changing their production output, for which the PLC should compensate. We use this trace to extract the essential packets that contain the data actually transfered to the HMI and used these to generate a series of attack traces, consisting of falsified measurements. In our case, we need not deal with TCP headers, because we are only interested in the measurements. However, we are still capable of reading directly from the network, because our initial experiments were planned that way.

To simulate the controlling PLC, as mentioned in the previous section, we needed a ground truth of what can be considered normal behavior. Because we were not able to determine the output of the PLC, we have assumed that the control algorithm simply tries to converge to the set point at all times. Through this assumption, we were able to create a trace of 'correct' PLC commands. We use this as a second input for our experiments, together with the network trace. Finally, we have simulated misbehavior of the controlling PLC by altering this second input artificially.

4.3.1 Traces

Figure 4.4 shows a graph of the trace we recorded, as well as the attack traces we used; note that we abstracted away from headers and concentrated on the payload only. Because the attack traces are artificial, they are not suitable for general testing, but the traces and the necessary tools to create and use them are provided with the code, available per request to the authors.

The three attack traces are respectively showing a gradual decrease, a gradual increase, and some deviations from the setpoint upwards. We will test each of these, and the legitimate trace, in combination with an estimation of the PLC behavior. Because we cannot determine the actual behavior of the PLC (see Section 4.1), we used a simple estimation algorithm that determines the behavior based on the previous two measurements. We replay this behavior with a time delay to the framework, together with the measurements taken from the PLC-to-HMI communication. For the purpose of this simple proof of concept, we have assumed that the PLC's output is binary – increase or decrease only.

We also created falsified traces to simulate injection of false PLC behavior (that doesn't affect the actual power output); for this purpose, we have the legitimate trace and four attacks – permanently increase, permanently decrease, and the regular trace with the first 30 messages as an increase or decrease command. We deploy all of these traces and the manipulated network traffic into the framework and observed the results, which we discuss in the next section.

4.4 Results

Here we briefly present how our framework behaves when it is given the traces discussed in the previous section. Note that, because the framework is deterministic except for message re-ordering, and because the traces are only partially realistic, we did not repeat these experiments. We organize this section by network traffic; each set of graphs contains a specific network trace (i.e., all voltage values are the same), but a different

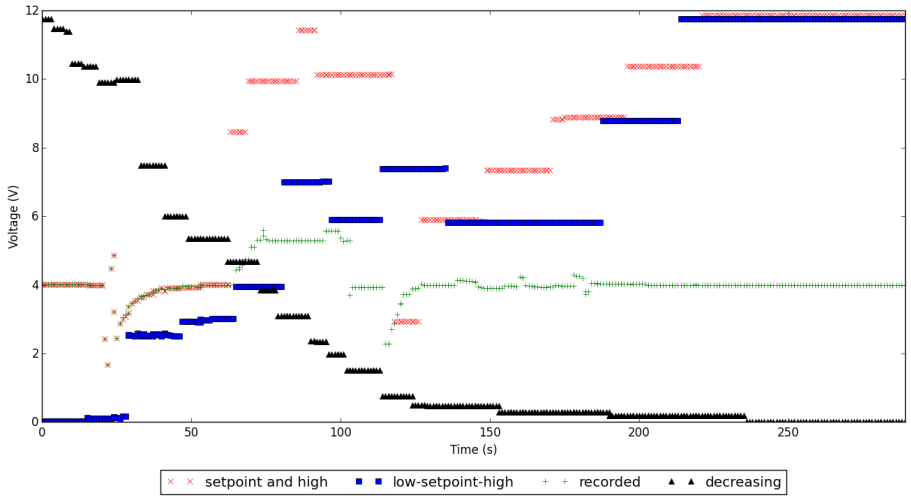
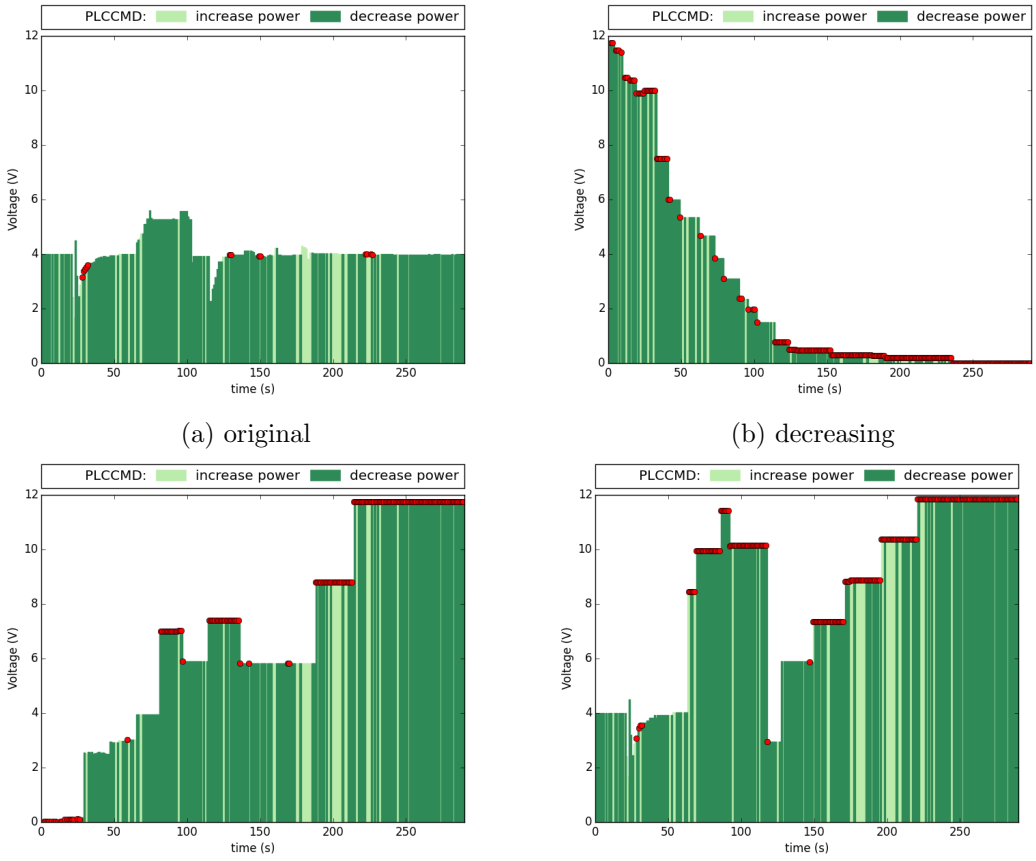


Figure 4.4: Voltage values over time contained within our traces

(malicious) configuration of the PLC behavior (referred to as a *PLCCMD trace*); in addition, we have one set of graphs that combines all traces with the legitimate description of the PLC behavior (simulated, as discussed in the previous section). In each graph, the behavior is shown by the color of the area below the graph, and a red dot indicates that our framework triggered an alert at this time.

Figure 4.5 shows the simulated behavior and the various network traces, along with the output of our framework. Both attacks that start at unusual voltages, far away from the set point (shown in Figures 4.5b and 4.5c) immediately detect the attack; given the correct set point, it is easy to detect outliers. The last attack concentrates on moving away from the set point; in contrast to the others, it starts with real measurements, and thus is detected only when it jumps into an unstable state (see Figure 4.5d). In this trace, the attack also stops at roughly $t = 120$, which the framework correctly recognizes. Note that the original trace in Figure 4.5a shows some false positives also present in the spike trace. For future work, we will aim to optimize the framework to reduce the false positive rate; currently the opinion output by each mechanism is binary.

Figure 4.6 shows the results for different attack traces. Most of the detection is independent of the PLC command sequence, because the values measured are so far outside the legitimate value space. However, as expected, once the voltage measurements drop below a threshold, it takes a long time for the attack to be counted as an actual attack



(a) original

(b) decreasing

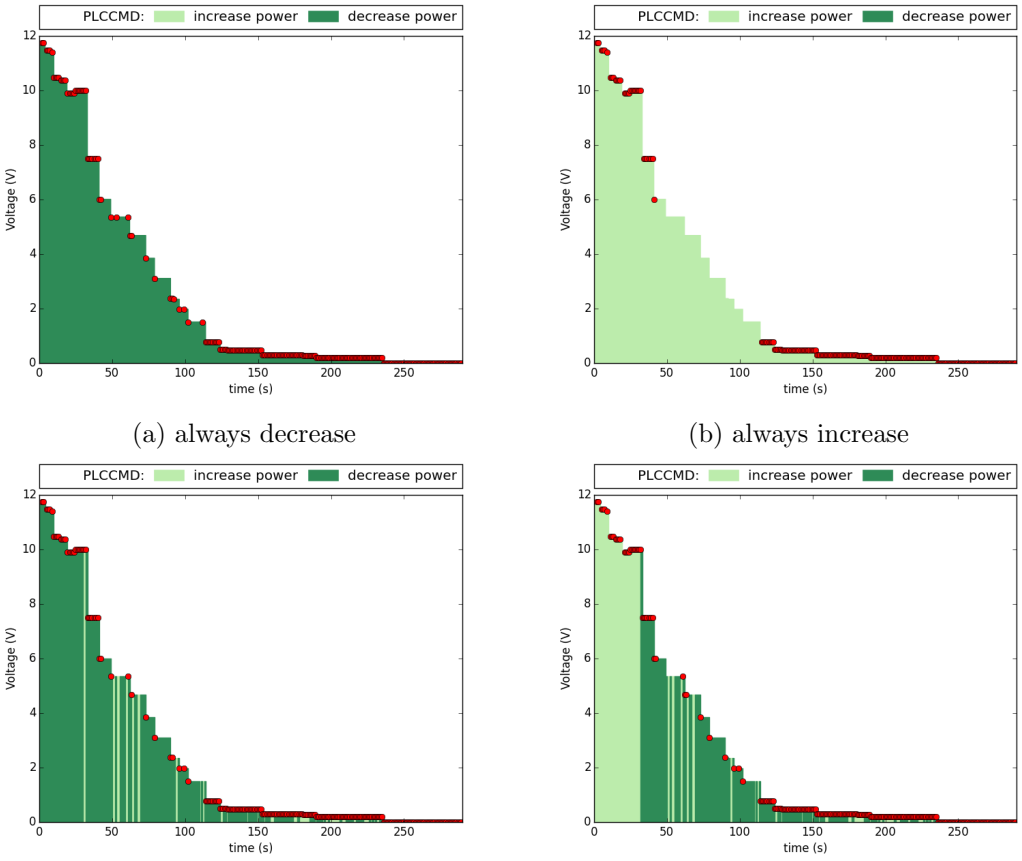
(c) low-setpoint-high

(d) spikes

Figure 4.5: With the original PLCCMD trace

if the PLC claims to be increasing power (shown in Figure 4.6b), because the voltage measurements are in the error surrounding the set point. This is unavoidable, as this is exactly the behavior that the PLC should have in response to these measurements. Detecting this case could be possible through other detection mechanisms, or by deploying additional sensors that can verify the measured voltages. A similar pattern, but in reverse, can be observed in Figure 4.7, where detection for increments does not work within a specific range. Note also that, because the detectors look at the most recent received values only, the detection does not trigger when the voltage remains constant, unless it is far outside the tolerance. Figure 4.8 is very similar, but shows a stronger variation and demonstrates that injecting values near the set point does not necessarily have an effect on how the attack is detected.

Finally, Figure 4.9 shows the legitimate trace we made, accompanied by the different incorrect behavior of the PLC. As shown here, the detector that verifies the PLC behavior is much less reliable (or rather, more subject to false positives) than the other detector. However, unlike the other detector, which only runs when the measurements are far enough away from the set point, the detector looking at PLC behavior can actually detect smaller attacks that have a limited influence. However, it can also be seen that the majority of these detection events are very similar to the false positives shown in Figure 4.5a. Because we are using a simplistic simulation of the PLC, it is unclear what the cause is exactly. Future work could attempt to employ machine learning to study legitimate PLC behavior, or it could incorporate a more precise model (for example, using a different control algorithm). This could also allow modern control algorithms to be tested and used for early warnings without replacing the PLCs.



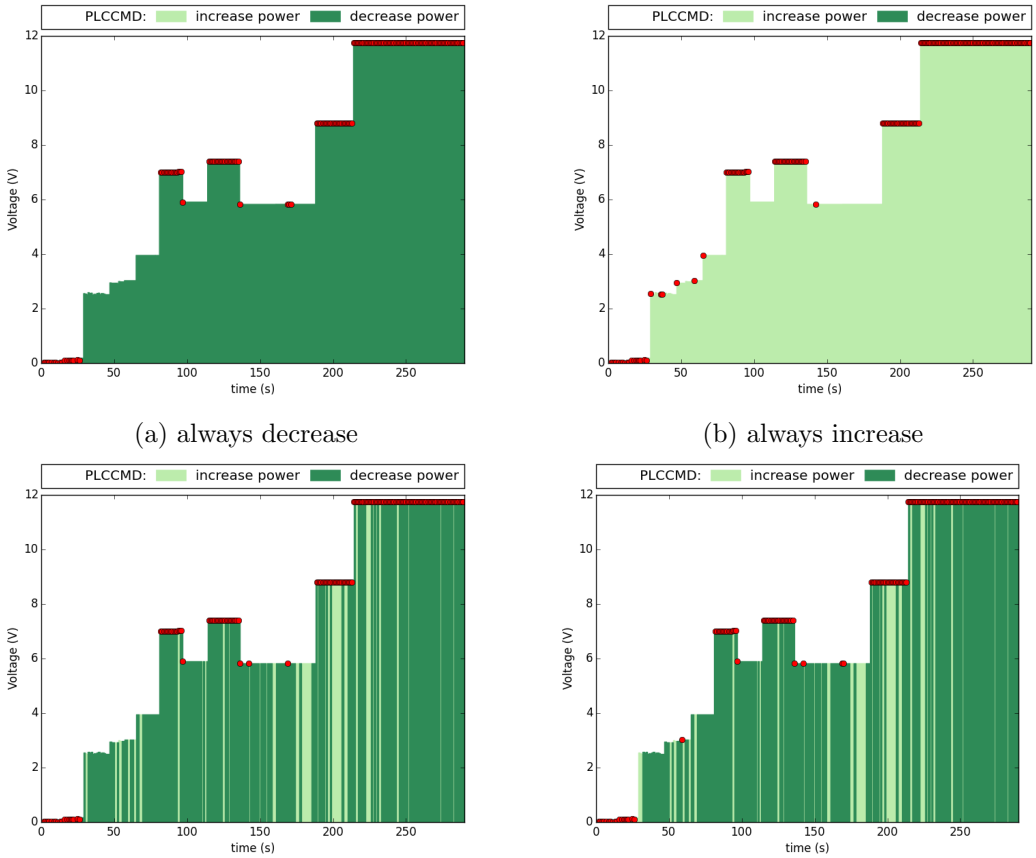
(a) always decrease

(b) always increase

(c) first 30 increase

(d) first 30 decrease

Figure 4.6: From high to low, for different malicious PLC behaviors.



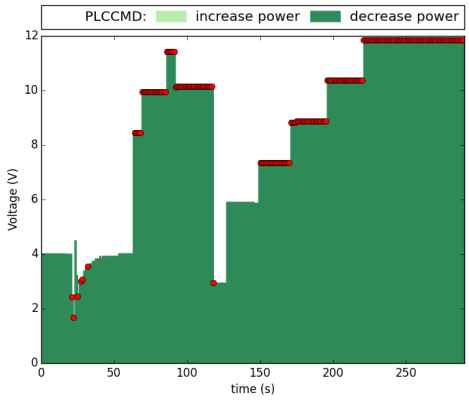
(a) always decrease

(b) always increase

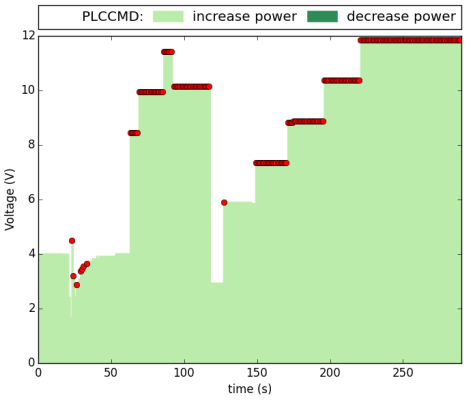
(c) first 30 increase

(d) first 30 decrease

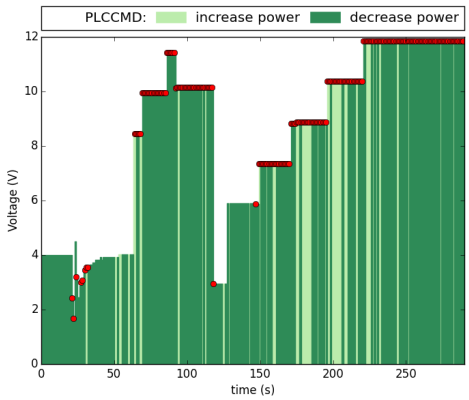
Figure 4.7: From low to high values, for different malicious PLC behaviors.



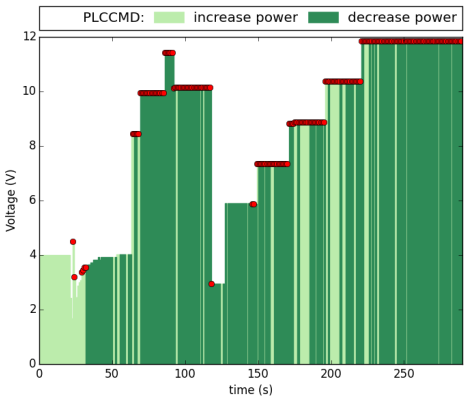
(a) always decrease



(b) always increase



(c) first 30 increase



(d) first 30 decrease

Figure 4.8: From stable to high values, for different malicious PLC behaviors.

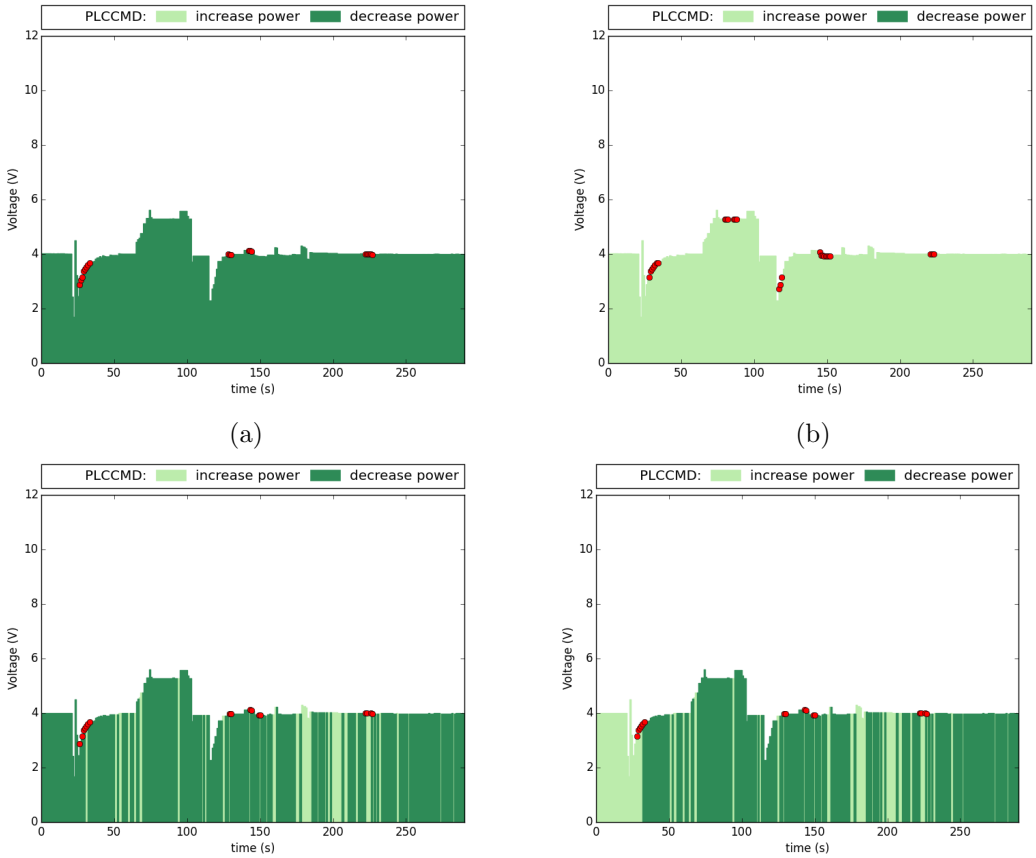


Figure 4.9: Real PCAP trace, for different malicious PLC behaviors.

5 Conclusion

This deliverable has discussed the design and a proof of concept implementation of a framework for misbehavior detection in ICS. Despite some unexpected obstacles in the testbed, our implementation managed to show the potential for detecting attacks by entities that have access to the network, in particular the ability to modify, inject or remove packets. In future work, we will extend our framework and perform more rigorous experiments to validate our approach. Our work is largely orthogonal to existing work on intrusion detection, such as the mechanisms discussed throughout WP6 and WP7 of CRISALIS, and thus offers a complementing approach that will leverage the benefits of the different detection approaches in one joint framework.

Bibliography

[1] M. Caselli, E. Zambon, and F. Kargl. Sequence-aware intrusion detection in industrial control systems. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, pages 13–24. ACM, 2015.

[2] M. Caselli, E. Zambon, J. Petit, and F. Kargl. Modeling message sequences for intrusion detection in industrial control systems. In *Critical Infrastructure Protection IX*. Springer, 2015. To appear.

[3] S. Dietzel, R. W. van der Heijden, H. Decke, and F. Kargl. A flexible, subjective logic-based framework for misbehavior detection in v2v networks. In *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on*, pages 1–6. IEEE, 2014.

[4] A. Jøsang. The consensus operator for combining beliefs. *Artif. Intell.*, 141(1-2):157–170, Oct. 2002.

[5] A. Jøsang. Subjective Logic. Technical report, University of Oslo, 2015. http://folk.uio.no/josang/.

[6] M. Raya, P. Papadimitratos, V. D. Gligor, and J.-P. Hubaux. On Data-Centric Trust Establishment in Ephemeral Ad Hoc Networks. In *2008 IEEE INFOCOM - The 27th Conference on Computer Communications*, pages 1238–1246. IEEE, 2008.

[7] R. W. van der Heijden, S. Dietzel, and F. Kargl. Misbehavior detection in vehicular ad-hoc networks. In *1st GI/ITG KuVS Fachgespräch Inter-Vehicle Communication*, pages 23–25. University of Innsbruck, 2013.

[8] R. W. van der Heijden and F. Kargl. Open issues in differentiating misbehavior and anomalies for vanets. In *2nd GI/ITG KuVS Fachgespräch Inter-Vehicle Communication*, pages 24–26. University of Luxembourg, 2014.