# Anomaly Detection for Industrial Control Systems Through Totally Integrated Automation Portal Project History

Laura Hartmann[1,2*] and Steffen Wendzel[1,2]

[1]Fernuniversität in Hagen, Hagen, Germany
steffen.wendzel@fernuni-hagen.de

[2]University of Applied Sciences Worms, Worms, Germany
hartmann@hs-worms.de, wendzel@hs-worms.de

Received: May 25, 2022; Accepted: August 22, 2022; Published: September 30, 2022

**Abstract**

Attacks on industrial control systems (ICS) have been intensively studied during the last decade. Malicious alternations of ICS can appear in several different ways, e.g., in changed network traffic patterns or in modified data stored on ICS components. While several heuristics and machine learning methods have been proposed to analyze different types of ICS data regarding anomalies, no work is known that uses the data of *Totally Integrated Automation (TIA) Portal* for anomaly detection. TIA Portal is a popular software system for organizing the ICS, with which configuration and programming data can be viewed, changed and deleted. By saving the single project datasets historically, old versions of the current system configurations can be restored. This work extends our previous work [1], in which we started to examine real TIA Portal project data of an automotive manufacturer's production line, covering a period of about three years of historical data, for various features that may indicate anomalies. We therefore proposed heuristics that detect timing- and size-based anomalies in the TIA Portal data. Our initial approach is extended by applying machine learning algorithms on top of our built heuristics to improve our detection results. We have also added more details of the given dataset. Additionally, we investigate a further feature set consisting of the different types and a varying amount of code blocks of our given dataset. Our approach covers both, changes to the data caused by infiltrated attacks as well as malicious changes made by employees who have direct access to the machines.

**Keywords**: Industrial Control Systems (ICS), Anomaly Detection, Cyber Physical Systems (CPS) Security, Intrusion Detection Systems (IDS), Machine Learning (ML)

## 1 Introduction

Since cyber security is important in the area of Industrial Control Systems (ICS), many researchers are concerned with the challenge of protecting these. Nevertheless, mistakes made by employees and malicious manipulations by attackers or saboteurs, who have gained access to the machines, can cause harm to people and companies. Exemplary, despite of other factors, people might be harmed by an axis for which the angle at which an axis pivots is manipulated. Noteworthy attacks in the past that could have been detected through a data-based approach include *Stuxnet* in 2010, *Havex* in 2013, *Irongate* in 2016, and *Industroyer* as well as *Triton* in 2017. Recently, the possibility of a malignant change was

*Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 13(3):4-24, Sept. 2022
DOI:*10.22667/JOWUA.2022.09.30.004*

*Corresponding author: Center for Technology and Transfer (ZTT), University of Applied Sciences Worms, Worms, Germany, Tel: +49-(0)6241-509-255, Web: `https://www.hs-worms.de/hartmann/`

demonstrated once again by a series of explosions at the nuclear energy facilities in Iran from June to October 2020 where, in total, 25 explosions and fires mainly on nuclear facilities have been registered[1][2].

Our project MADISA[3] focuses on anomaly detection for data of ICS, i.e., code and configurations as well as their metadata within ICS. In this initial study, we aim to identify anomalies in the data stored on ICS with heuristics and machine learning. Therefore we analyze Totally Integrated Automation (TIA) Portal project data. In detail, we investigate the alternations performed by the programmable logic controller (PLC) programmers of TIA Portal projects over time.

We provide the first analysis of TIA Portal project data for the purpose of anomaly detection and build heuristics based on our investigation. We apply an approach based on data history analysis in the sense that we evaluate new configuration changes on the basis of previous data modifications. Therefore, in this work, we will introduce some stealthy attacker scenarios for TIA Portal data and evaluate the different timestamps and storage variables contained in our data on basis of these scenarios. The extension of this work over [1] also handles the usage of machine learning algorithms in addition to heuristics. Furthermore, we describe another non-stealthy scenario for the given data: the insertion or deletion of code blocks. Additionally, we provide more details of our given dataset.

The remainder of this paper is structured as follows: Sect. 2 describes the related work. Sect. 3 contains the explanation of the given datasets. Our examination of the features consisting of timing and size related variables is placed in Sect. 4, including the attacker scenarios used for our examination (Sect. 4.1) and the analysis of our data (Sect. 4.2). We extend the examination with the different blocks' counts in Sect. 5, with an additional scenario for this feature set (Sect. 5.1) and an analysis of these features (Sect. 5.2). Summarizing Sect. 6 is followed by the concluding Sect. 7, which also provides a prospect of the future work.

## 2 Related Work

Anomaly detection based on historical data from ICS is mostly unexplored. According to an overview of existing approaches by Feng et al. [2], there are different approaches for anomaly detection, but none of them deals with the historical project data in form of code, configurations and the projects' metadata. For example, Kiss et al. [3] use network data from host and security equipment, and F. Zhang et al. [4] and R. Zhang et al. [5] detect anomalies by investigating binary process data. Examining programmable logic controller protocols is done by Yoo et al. [6], and text- and binary-based protocols by Wressnegger et al. [7]. M. Zhang et al. [8] detect anomalies via time shifts of physical operations and executable paths. Das et al. [9] use sensor data for behavior-based anomaly detection with an indication to the occurrence by evaluating the historical sensor measurements. This approach is close to our work, but it only uses historical sensor measurements for evaluation. We use code and configurations and the corresponding metadata of these files, which are not considered in their approach.

It must be noted that this article is based on [1]. We extend the previous work by additional descriptions of the given data, further evaluation of the chosen features of sizes and timings, as well as an additional – non-stealthy – scenario for the different code blocks and their amount over time.

There are methods to detect attacks that have not yet been manipulated the TIA Portal project data: those address the network. Anomaly detection can be performed in two modes here: the monitoring of the traffic and the network protocol analysis. The approaches dealing with the detection of anomalous

[1] "Iran blasts: What is behind mysterious fires at key sites?", `https://www.bbc.com/news/world-middle-east-53305940`

[2] "Stuxnet 2? Iran Hints Nuclear Site Explosion Could Be A Cyberattack", `https://www.forbes.com/sites/kateoflahertyuk/2020/07/04/stuxnet-2-iran-hints-nuclear-site-explosion-could-be-a-cyberattack/?sh=6e5d323c25ad`

[3] `https://madisa.ztt.hs-worms.de/`

behavior after successful infiltration of an attack address the endpoints of malicious changes: the sensor and process data which are secured by monitoring timing invariants, rules for physical states, watermarking of data, and other detection methods [10] like the plausibility check by Krotofil et al. [11]. Intrusion detection systems (IDS) were revised by several works (e.g., [12, 13]) in the near past. Gómez et al. [12] propose a new learning dataset for IDS related to network traffic that is based on their methodology consisting of attack selection, attack deployment, traffic capture, and feature selection. Rubio et al. [13] discuss existing cybersecurity threats and various defense techniques, including anomaly-based approaches using data mining, classification, clustering, and (machine) learning techniques, as well as statistical, knowledge-based and time series analyses, to name a few. Afterwards they present IDS solutions and research from the industrial as well as the academic sector. In their discussion section they name research fields which are not fully examined until now such as advanced persistent threats' behavior, or a holistic defense solution.

A challenging part of research is the detection of *stealthy attacks*. For instance, Hu et al. [14] deal with this topic based on a residual skewness analysis of process data. Genge et al. [15] investigate stealth attacks on aging devices in the Industrial Internet of Things and include the decay of processes in their statistical analyses. Sensor and actuator attacks were examined by Urbina et al. [16] assuming that stealthy attacks evolve with the changes of the machine. Krotofil et al. [17] already examined timestamps of processes to make DoS attacks more stealthy. Referring to this, it can be assumed that the investigation of timestamps of TIA Portal data is a further step in detecting stealthy attacks, and thus essential. Stealthy attacks are on the one hand characterized by perfectly timed malicious changes during the working shifts in which employees are also making their legitimate changes. On the other hand they are implemented with a low frequency of changes since in production one or fewer changes per day are implemented. Such attacks are also characterized by the fact that they are usually implemented by small manipulations in the data (most of the legitimate changes alter only a few bytes). Those attacks match several characteristics of our legitimate data. They can potentially be detected by our approach when looking at the code and configurations of our data, which we will address in future work. Another stealthy approach by Alsabbagh and Langendörfer [18] also deals with TIA Portal code and content. The authors propose an approach where they decompile the PLC's Statement List source code and replace instructions. To make it stealthy towards PLC programmers that compare local against PLC's online code, the authors propose to redirect the PLC programmer's view to a fake PLC copy with uninfected code by a redirect. They add that the PLC programmer can only recognize this if he would, in addition, check the IP address of the PLC. The same scenario is used by the authors in their follow-up work [19], where they decompile Ladder Diagram source code. Since our envisioned approach checks code and content of projects, both of these scenarios would raise an alarm in our system (depending on where one implements our heuristics (cf. Sect. 6, *Deployment of heuristics.*). A third paper by Alsabbagh and Langendörfer [20] also deals with TIA Portal data, i.e., in this case they introduced an attacker's code to the main Organization Block of the TIA Portal. Also, they added another Organization Block to the system that will be activated when the first block gives a signal at a specific time. Those blocks are changed and inserted rarely as they handle startups, interruptions and other basic operations (in our data, we had three new Organization Blocks in four projects over a period of three years in total). Moreover, depending of the size and time of the change, our heuristics would raise an alarm in such a case.

## 3 Description of the Given Data

Our given datasets from a real-world environment – a German car manufacturer plant – contain four projects of TIA Portal, a software to simulate, engineer and operate the ICS, where each project coordinates the code and configuration of one machine. These projects contain archive and backup data for

four different components of the production where *archives* are long-term and *backups* are short-term records of the TIA Portal projects.

**The backup folders** Three of our four projects include a backup folder with 40 backups where each day of the last 40 days carries one backup. These daily backups were configured in a job (pre-set schedule is daily) and can deviate if the company specifies other times. However, it is recommended by the versiondog vendor to save them daily to ease data comparison. Some of the given backups are made twice a day while some are left out on certain days, which shifts the time span of existing backups by one day, i.e., some backups contain the last 39 or 41 days instead of 40. These shifts result from a second manual backup – on top of the automatically created one – or the removal of one. Our approach is able to handle these non-continuous backups. In total, there are 120 zip-folders that contain backups.

**The archive folders** Again, only three of the four projects include an archive folder. It consists of project data over a period of about three years. The frequency of the occurrence of archives differs; there are zero to six folders per day. In total, there are 566 zip-folders that contain an archive.

The car manufacturer uploaded these folders to the versioning tool *versiondog*[4]. Versiondog allows to remotely save the ICS' data and shows their historical modifications. The options chosen for our uploaded data are: *1) Software Upload* where a predefined job is uploading the backups automatically with the storage strategy "Always save backup", which allows us to compare the different backup folders – even if they are the same as the previous one. *2)* Since the car manufacturer changed its remote saving storage options, the latest part of the data uses another option: the recommended latest version of *Station Upload* where the different backup folder sizes are smaller than the ones of the software upload strategy. However, this new strategy is supported only by TIA Portal V15.1 and newer, so it is not yet applicable for anyone using older versions. Thus, we consider both upload strategies.

Despite the fact that the time span of existing backups is 40 days and the timestamps in them range back to about three years, the single data entries in the backups and archives span back to 2012 with some additional dates, which are fixed to the year 1601, because they were set in a standardized way once at the beginning and have not changed anymore.

Each archive or backup folder contains a project file, which is also provided to us as a *json* file. These files are structured as follows: There are seven main data paths, i.e., each of the files contains data divided into seven substructures – ControllerDataTypeFolder, ControllerTagsFolder, Device, ProgramBlockFolder, TechnologicalParamFolder, TextLists and TracesFolder – where, for example, "Devices" includes both network and programmable devices, and "ProgramBlockFolder" contains code. The main data paths are also separated into further data paths. Timestamps can be found in two of them: *programBlocksFolder/pro-gramBlocks/ID/Data* and *programBlocksFolder/DataBlocks/ID/PLC_Data*. In total, seven different timing variables were investigated during the work on [1]. Two variables dealing with the size of volatile and non-volatile storage were investigated in this work. They are located under the same data paths as the timing variables.

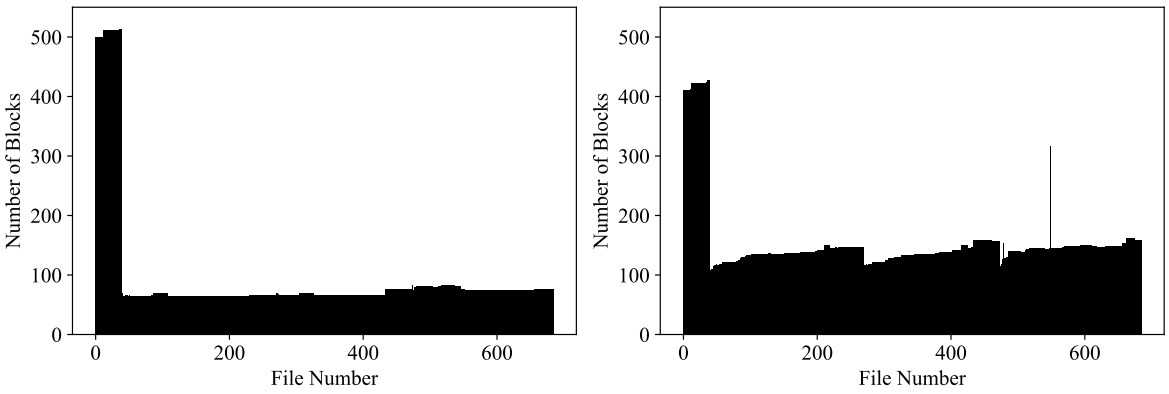
All files include code blocks that are categorized into the following:

- **Organization Blocks (OB)** are responsible for functions such as startup, interruptions, error handling and cyclic execution of code. (First data path)
- **Function Blocks (FB)** include the actual program. They can access internal storage. (First data path)
- **Function Calls (FC)** are also functions. In contrast to FB, FC only have temporary internal storage. (First data path)

[4]versiondog, `https://auvesy-mdt.com/en/versiondog`

- **Data Blocks (DB)** store the data for the code blocks. (Second data path)

The distribution of the blocks shown over all projects, clustered by DataBlocks (DB) and programBlocks (PB, namely FB, FC, OB), is as follows (Fig. 1):




(a) DataBlocks (b) ProgramBlocks

Figure 1: Distribution of Data- and programBlocks.

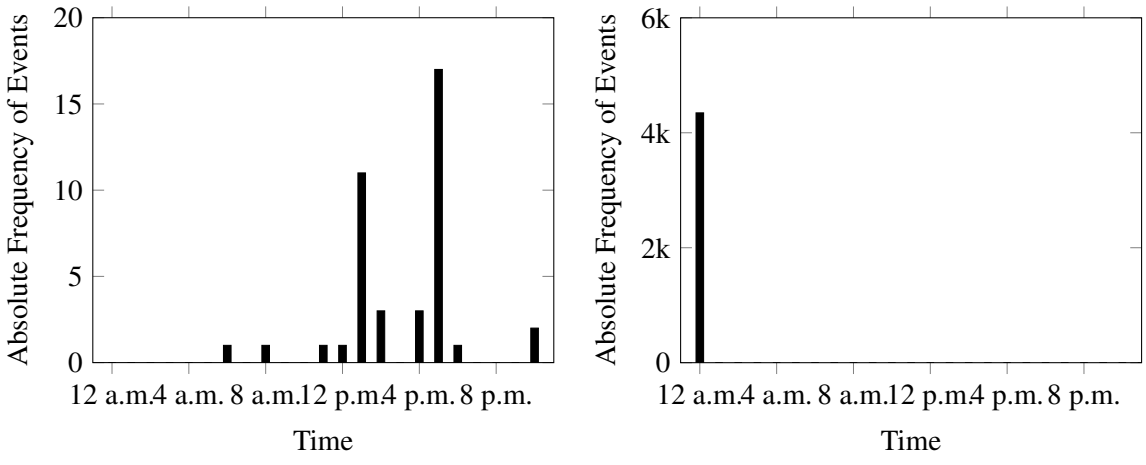
All of these code blocks carry the following time variables:

- **ftCreation**: Variable that represents the time of creation of the block.
- **ftModified**: Variable that indicates the modification time of the block, i.e., every change in code (ftCodeModified), interface (ftInterfaceModified) or in a load-relevant field (ftLoadRelevantModified) is mapped to this.
- **ftCodeModified**: Time variable representing the time of modifications in the code/data itself.
- **ftInterfaceModified**: Variable is updated when modifying the interface.
- **ftLoadRelevantModified**: By editing comment, author, block version, etc., this variable is set to a new timestamp.
- **compileTime**: This indicates the last compilation time of the block.
- **downloadTime**: This value describes the time at which the block was latest loaded onto a device.

The distribution of the timing variables differs per variable and block type. Exemplary for OB and DB, the following Fig. 2 displays the distribution of compileTime in the first of our four projects.

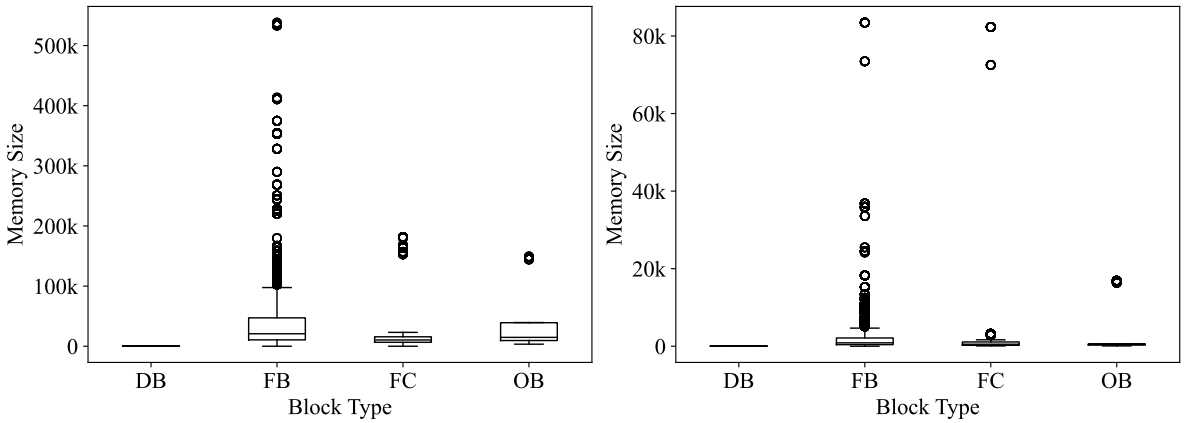
The dataset contains two storage variables:

- **loadMemoryRequired**: This represents the non-volatile storage of the blocks with DB storage being always set to zero. When downloading the project to the CPU, it is first stored in here.
- **workMemoryRequired**: This variable represents the volatile storage of the executable OB, FB, FC or DB; again with DB storage always set to zero. While running the project, this storage is used.




(a) compileTime of OB (b) compileTime of DB

Figure 2: Distribution of timing variables.




(a) Boxplot of loadMemoryRequired variable. (b) Boxplot of workMemoryRequired variable.

Figure 3: Boxplots of size variables.

Fig. 3 summarizes the load- and workMemoryRequired variables:

As one can recognize, and as mentioned, DBs always have their storages set to zero. FBs have the highest usage of memory, followed by FCs. OBs use only a small amount of storage in comparison to the other block types.

Fig. 4 summarizes the given data described. The gray-patterned paths will be investigated in future work, the white ones are those that are considered in this work.

Other features that we can locate in the data, and that will be investigated in the future, are the code languages (such as FUP (*Function Plan*) or SCL (*Structured Control Language*), the textual and graphical code (investigated linguistically), and the configurations of the devices as well as the devices themselves that can be found in the internal data. The sizes of the individual files in the zip-folders, the encryption, file names and file types that we can pull from the external data will also be examined.

In this work, we decided to start with the examination of size- and time-based features, as well as counts of blocks, as such features almost necessarily change when a manipulation in the data is made.

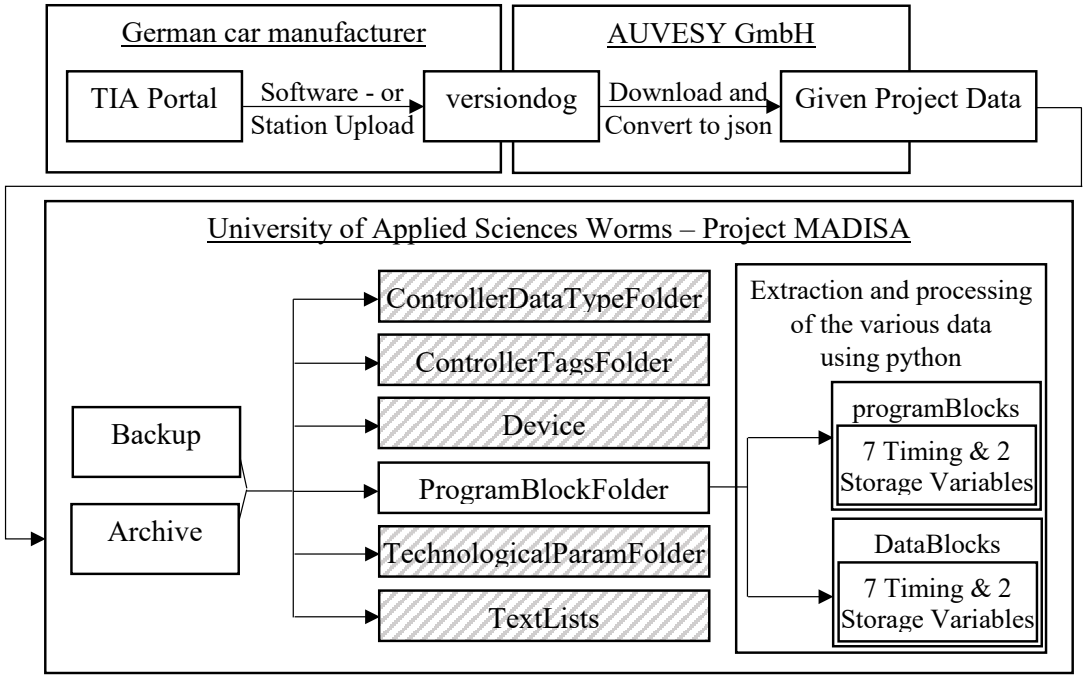



Figure 4: Summary of the given dataset – gray-patterned paths will be investigated in future work, the white ones are those that are considered in this work.

Code, configurations, names and other data is planned to be investigated in the future. Here, one will need to linguistically analyze the data; that is why we delimit our work in this step from those features.

Another aspect of the data given is the target files. Here, one needs to examine existing attacks and their main targets. This also leaves room for further work.

In a previous work [21], we already focused on the "Device" path, where we studied changes over time. The values of features were rarely changing, rendering this path mostly unsuitable for information hiding and stealthy attacks.

**Features to be investigated in other work:** Our examination shows that there are multiple features of the given data that could be exploited and therefore should be investigated (Fig. 5): The gray-patterned can be investigated in future work, or have already been examined in related work (e.g., "Devices" [21]).

# 4 Time- and Size-based Features Examination

This section handles the first of two feature sets that are examined in this work. First, we present attacker scenarios for this evaluation, followed by the dataset analysis. The latter also includes the comparison of results and the improvement of our results by applying a machine learning algorithm.

## 4.1 Attacker Scenario

For our examination, we simulated three different attackers:

*a)* **PLC Programmer**
The first attacker is an internal employee, a PLC programmer, that exactly knows the working shifts and, thus, the timing of legitimate manipulations of the PLC code, since he himself programs the

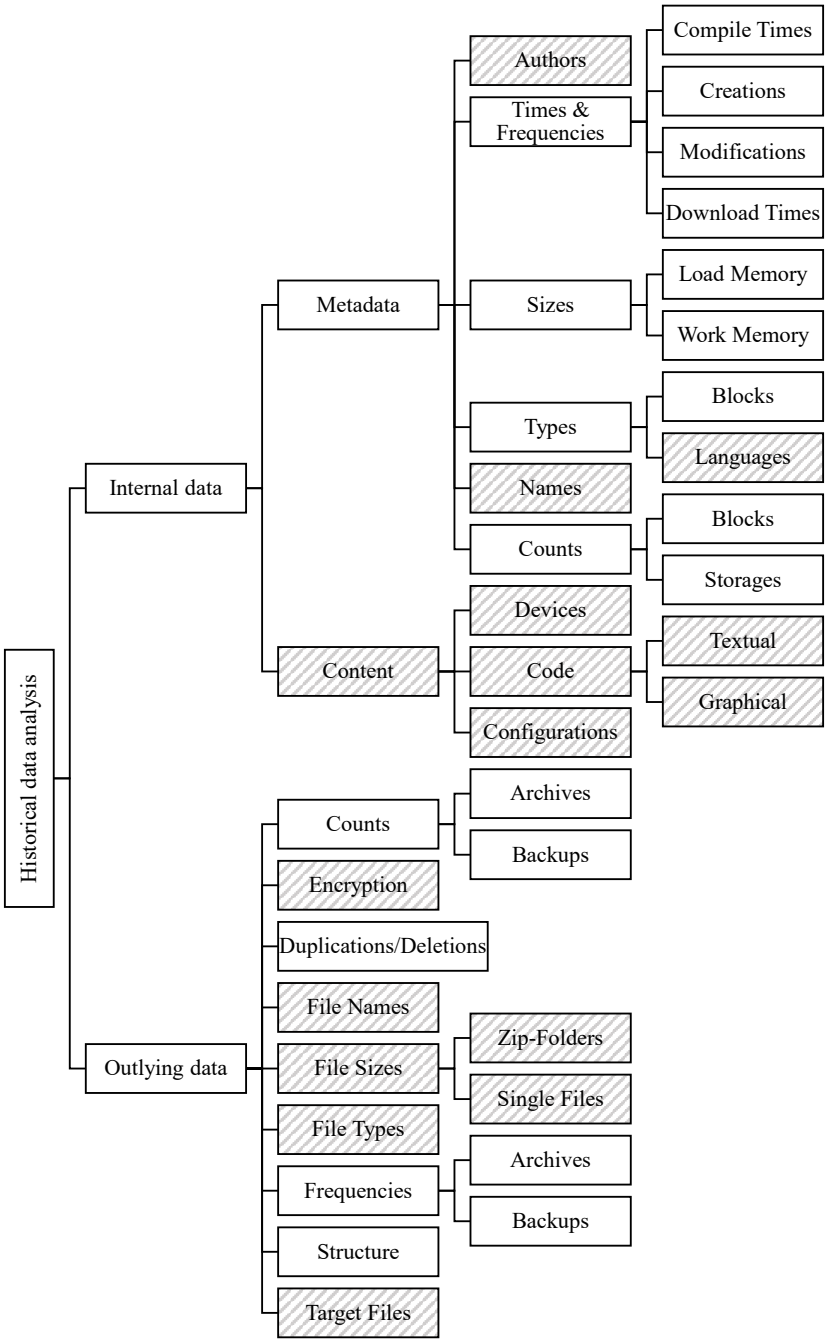



Figure 5: Features to be investigated – the gray-patterned features can be investigated in future work, or have already been examined in related work (e.g., "Devices" [21]).

PLCs. The way legitimate PLC programmers work is as follows: If a programmer wants to change something in an existing code, he can view the last backup in comparison to the current state – here he can check if he overwrites something with his new code by mistake. Also he can directly view the changes made since the backup.

If a malicious saboteur were to make changes during the working shifts and one of his colleagues, who is ideally trained, and, thus, compares versions before editing the code or configurations, were to

access the code of the machine, he would most likely notice the malicious change. However, because of this, the malicious programmer maintains a *new* piece of code that is not connected to a PLC. The colleague would not recognize the new code due to unawareness of it and because the code is not displayed for comparison since it is not connected to the device, and it would accordingly not be noticed. The next day, this code snipped is in the backup – it is now seemingly a legitimate code snipped. Since it includes a timer that connects it to the environment some days later at a legitimate time, but shortly before the worker shift ends, almost no one would be able to detect the malicious code which then executes itself. From this, the assumptions listed below can be assumed to be plausible:

- *Timing*: inserting malicious code in legitimate times — the PLC programmer has the ability to insert the malicious code snippet during the work shifts, since he is working on the machines at this time anyway. For activation of the snippets, he needs to insert the connection as late as possible during the working shifts, as the possibility of another employee inserting something in the code is lower during that time. Some of those late inserted changes might be recognizable by our heuristic.
- *LoadMemory and WorkMemory*: inserting malicious data with an overlap of 50% with the legitimate sizes — since the programmer first inserts the malicious code without activating it, the load and work storages are set to zero. By activating the next day, the sizes grow to a seemingly normal value; however, since we measure the differences, the detection of the second step, the embedding of the code to the system, is recognized by our heuristics.

*b)* **Backup Coordinator**
The second attacker is also an internal employee, who is responsible for the controlling of backups and archives, and, thus, aware of the sizes of the backup and archive folders that are delivered by the versioning tool (versiondog).

He places his attack with legitimate sizes — small changes spread over several days, but almost completely overnight and before the new backup is made for two reasons: a) the detection of a malicious change can then not be detected by the employees working on the machines since the new backup is the standard for comparison of the code by PLC programmers the next day; b) his office colleagues are not around and accordingly do not notice anything. According to this, the following values can be assumed plausible:

- *Timing*: inserting malicious data outside the legitimate times — as he wants his colleagues not to recognize his malicious insertions, he starts his sabotage after they left. Another reason for the late night insertions is the awareness of the saboteur that no one of the PLC programmers will see his changes since he knows the timings by checking the earlier backup data. Thus, he inserts his changes at 11 p.m. — right before the backups are saved.
- *LoadMemory and WorkMemory*: inserting malicious data in legitimate sizes — since the saboteur is also aware of the sizes through the knowledge gained from the older backups, he is only updating values in a limited manner, increasing their limit by 1 in every step, which prevents load and work storage from being exceeded.

*c)* **Various Attacker**
We also designed a third, non-stealthy scenario in which the saboteur attacks evenly distributed and multiple times throughout the day and does so at different sizes.

## 4.2 Dataset Analysis

This section analyzes the detectability of anomalies in the provided data. This is done based on the distribution of the data within a feature for each block type and variable, which allows outliers to be detected even in the legitimate given data (Sect. 4.2.1), and the differences of individual parameters for the single blocks, with outliers considered as anomalous values (Sect. 4.2.2). Also, all heuristics are combined (Sect. 4.2.3). Afterwards, in Sect. 4.2.4, we compare our results followed by an extension of the features' investigation by applying a machine learning algorithm instead of our heuristics to compare results (Sect. 4.2.5).

The features studied were chosen for the following reasons: They are always present, i.e., no matter whether code or configurations are considered. Another reason why we chose these metadata is that the timing of an attack is crucial for its stealthiness. Thus, we try to detect attacks in temporal series or in combination with other values, such as the examined sizes.

Our clustering of the timing- and size-based variables' values into legitimate and anomalous times and sizes is based on an analysis of the PLC programmers' behavior in each project, i.e., the times in which the PLC programmers are working and the sizes of data that are typically generated by the PLC programmers. Our attackers were simulated closely to the given data to underline our behavioral clustering and our resulting heuristics.

As a result of our clustering, we also have up to 19.87% false positives (feature-dependent) in our legitimate data. Based on discussions with employees of the company who provided us with the data, the presence of anomalous values for the representatively interrogated test nodes could be confirmed. Nevertheless, we continued to treat these anomalies in our data as legitimate and did *not* categorize them as true negatives to determine the performance of our heuristics under realistic conditions. In other words, if we would remove all these anomalies to build our model and our heuristics, every user of it would have to remove all anomalous values before using it, which would be impractical.

*Dataset Training and Testing.* To create our heuristics, we used the first and second project for training. For testing, we applied the heuristics to projects 3 and 4.

### 4.2.1 Time-based Anomalies

Starting with the time variables of the first project, the first data path contains FB, FC and OB, the second one carries the DB. Based on the distribution of the individual time variables, ranges can be defined for which changes in the data are permitted. For FB, all variables are set from 12:00 a.m. to 1:59 a.m. or 8:00 a.m. to 8:59 p.m. The FC are mainly in the processing period from 7:00 a.m. to 9:59 p.m. OB starts and ends for the 7 variables from 7:00 a.m. to 10:59 p.m. Five of seven DB variables are set between 7:00 a.m. and 8:59 p.m. as well as 12:00 a.m. to 1:59 a.m., but the compile- and downloadTime variables are always set at exactly 12:53:28 a.m.

The knowledge gained from the distribution-based clustering of the first project (backup) and second project (archive) was subsequently applied to the other data from projects 3 and 4 (each project consisting of backup and archive files). To present the results, we use *Receiver Operating Characteristic* (ROC) curves, which illustrate the false positive rate versus the true positive rate calculated with different thresholds in different attacker scenarios, and *Area Under Curve* (AUC), which is used to express a value for the area that lies under the ROC curve and implies that the closer the area to 1.0, the better the detection of the anomalies.

Fig. 6 illustrates the different ROC curves, where the timings were clustered using various narrow and broad ranges of times, see Tab. 1 for the clustering details.

For scenario *c)* (Various Attacker), the narrower times clustering provides us slightly better values in comparison to our preferred clustering (F-Measure +0.017), but also doubles the false positives. There-

| Threshold | Times |
|---|---|
| narrower | FB 12:00 a.m. to 1:59 a.m. and 9:00 a.m. to 7:59 p.m.<br>FC 8:00 a.m. to 8:59 p.m.<br>OB 8:00 a.m. to 9:59 p.m.<br>DB 12:00 a.m. to 1:59 a.m. and 7:00 a.m. and 8:59 p.m.<br>and compile- and downloadTime at 12:00 a.m. to 12:59 a.m. |
| our clustering | FB 12:00 a.m. to 1:59 a.m. and 8:00 a.m. to 8:59 p.m.<br>FC 7:00 a.m. to 9:59 p.m.<br>OB 7:00 a.m. to 10:59 p.m.<br>DB 12:00 a.m. to 1:59 a.m. and 8:00 a.m. and 7:59 p.m.<br>and compile- and downloadTime at 12:00 a.m. to 12:59 a.m. |
| broader | FB 12:00 a.m. to 1:59 a.m. and 7:00 a.m. to 9:59 p.m.<br>FC 6:00 a.m. to 9:59 p.m.<br>OB 6:00 a.m. to 10:59 p.m.<br>DB 12:00 a.m. to 1:59 a.m. and 6:00 a.m. and 9:59 p.m.<br>and compile- and downloadTime at 12:00 a.m. to 12:59 a.m. |
| even broader | FB 12:00 a.m. to 1:59 a.m. and 6:00 a.m. to 10:59 p.m.<br>FC 5:00 a.m. to 10:59 p.m.<br>OB 5:00 a.m. to 10:59 p.m.<br>DB 12:00 a.m. to 1:59 a.m. and 5:00 a.m. and 10:59 p.m.<br>and compile- and downloadTime at 12:00 a.m. to 12:59 a.m. |
| broadest | 12:00 a.m. to 10:59 p.m. for all block types |

Table 1: Thresholds for time variable clustering.

fore, our clustering is preferable. Our time-based clustering shows us that the detection with only this feature, i.e., the timing variables, is not sufficient for scenarios *a)* (PLC Programmer) and *c)* at this point. Only for scenario *b)* (Backup Coordinator) we are able to detect the attack with an AUC of 0.999.

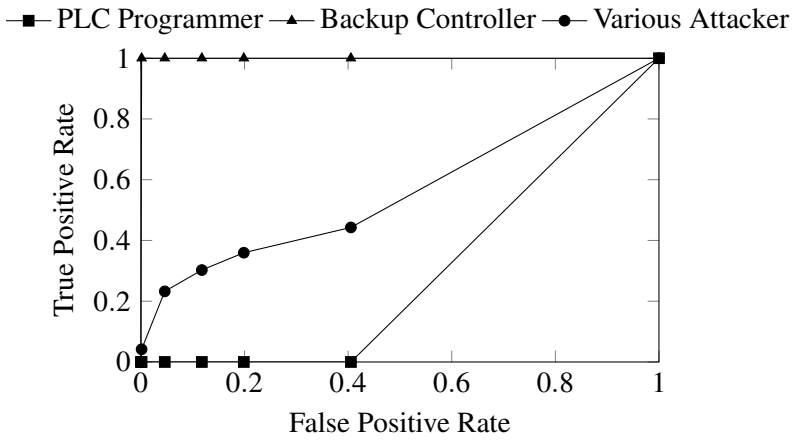
#### 4.2.2 Size-based Anomalies

The second feature that we analyze in this work is the size-related variables. These are separated into Load- and WorkMemory. The first aspect to notice is that DB always use zero bytes of both storage types, therefore we would not see a malicious modification in this block type. Nevertheless, we also simulated attacks on this type of data. For the other blocks (OB, FB, FC), we noticed that it is not sufficient to divide them only by blocks. Here we also had to include the different names of the blocks. For each name's first appearance the difference is set to zero; for each following block with the same name, we use Eqn. 1, with $x$ being the load storage, $n$ the new block, $p$ the previous one. If $p_x$ is 0, i.e., the loadMemoryRequired difference is null, the formula switches to Eqn. 2.

$$f(x) = \begin{cases} 1 & \text{if } \left| \frac{100}{p_x} \cdot (n_x - p_x) \right| < 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$f(x) = \begin{cases} 1 & \text{if } \left| \frac{100}{1} \cdot (n_x - p_x) \right| < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

For the workMemoryRequired variable, the detection threshold shifts to 0.1 instead of 1. Exemplary for both storage types, we examined the loadMemoryRequired blocks (Fig. 8) with the resulting ROC curve shown in Fig. 8a with thresholds below following values *1)* $< 0.1$, *2)* $< 1$, *3)* $< 10$, *4)* $< 50$, and



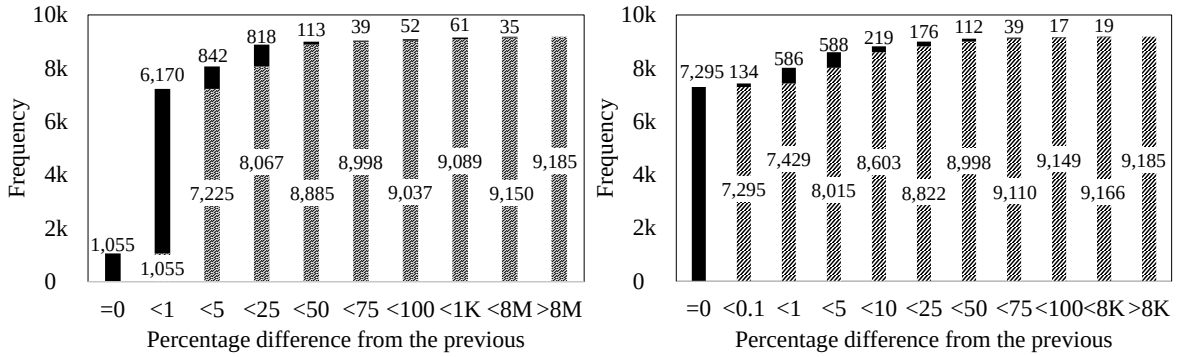

(a) ROC curve

| Scenario | Prec. | Recall | Acc. | F-Measure | AUC |
|---|---|---|---|---|---|
| PLC Programmer | 0.000 | 0.000 | 0.401 | — | 0.297 |
| Backup Coordinator | 0.834 | 1 | 0.901 | 0.910 | 0.999 |
| Various Attacker | 0.576 | 0.270 | 0.536 | 0.368 | 0.506 |

(b) Our preferred grouping

Figure 6: ROC curve for timing feature; underlying values for timing variables clustered by our preferred grouping.

*5)* $< 100$. The given dataset contains 37,118 entries in the first project but only 682 entries that have differences in their loadMemoryRequired variable, i.e., 36,436 entries do not have any changes in their deposited load storage. Fig. 7 illustrates the different distributions.



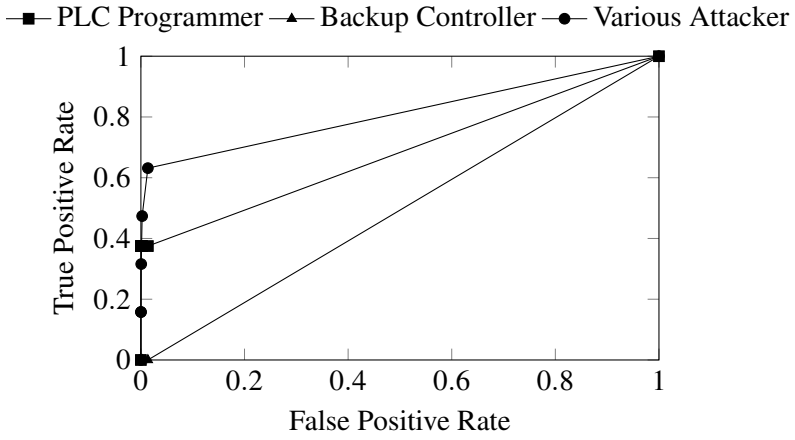

(a) Partitioning of the loadMemoryRequired difference values into ranges.

(b) Partitioning of the workMemoryRequired difference values into ranges.

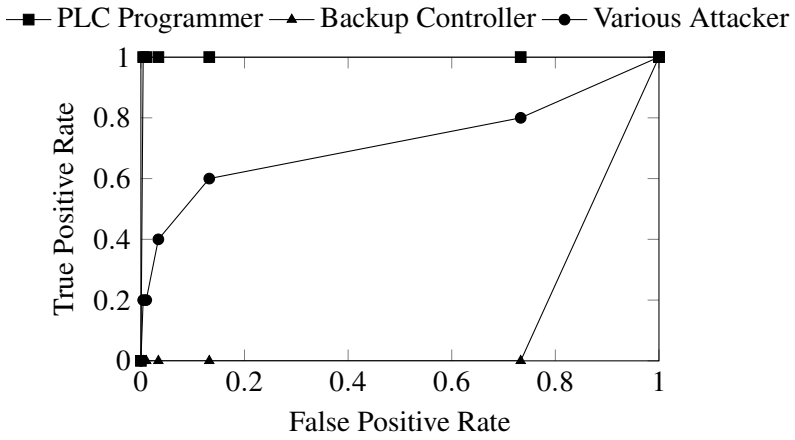
Figure 7: Partitioning of size variables' difference values into ranges.

For this reason, we have based our formula on the modified entries of the legitimate data which is illustrated in Fig. 8b. The values underlying the second plot for loadMemoryRequired and our Eqns. 1 and 2 are located in Tab. 8c.

By regarding the storage, it is possible to exactly remark from which block the anomalous value comes from. Blocks that were recently inserted must be white-listed and can be neglected until they have a modification within their data. As long as blocks have not changed, no previous block is given, so




(a) ROC curve for entire dataset




(b) ROC curve with $load\_difference = 0$ excluded

| Scenario | Prec. | Recall | Acc. | F-Measure | AUC |
|---|---|---|---|---|---|
| PLC Programmer | 0.994 | 0.375 | 0.686 | 0.545 | 0.683 |
| Backup Coordinator | 0.000 | 0.000 | 0.499 | – | 0.493 |
| Various Attacker | 0.995 | 0.474 | 0.736 | 0.642 | 0.812 |

(c) Eqn. 1/2

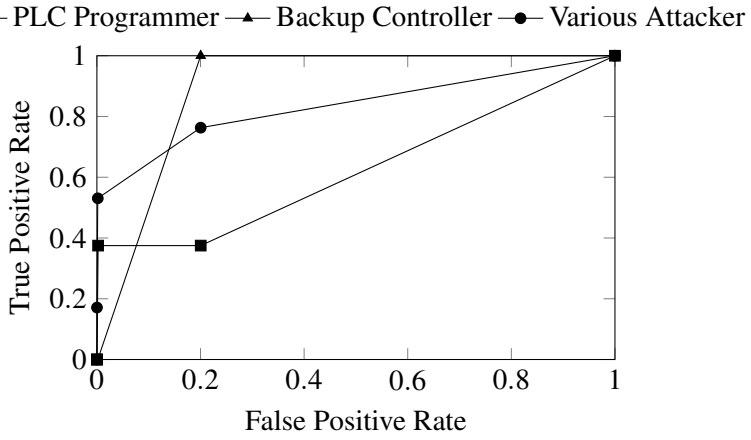
Figure 8: ROC curve for our entire dataset; ROC curve with $differences = 0$ excluded; LoadMemoryRequired difference for single blocks clustered by Eqn. 1/2 for ROC curve 8a.

the difference used for the formula is set to zero. Only with the 2nd, 3rd, ..., $n^{th}$ entry, differences to the respective previous ones are taken into account. Nevertheless, it is not enough to consider this feature alone for scenarios *b)* and *c)*, since they are still able to insert malicious values and remain mostly undetected. As with the previous feature, it is not recommended to use it as a stand-alone feature.

### 4.2.3 Combined Utilization of Features

In order to increase the quality of our heuristics, we now evaluate their combined utilization. None of our two features (time and storage) was suitable to recognize stealthy attacks properly. Thus, we first categorized them by their ability of identifying stealthy attacks: Since small changes like a 1-bit content change are not detectable with only one of our two features, the timestamps and sizes combination are

important. A small change during an unusual time could still be recognized as well as a large change at a legitimate time. For this purpose, we combined the timing and size variables with each other, i.e., we tested how many anomalous values in one element of the dataset have to exist to raise an alarm. Fig. 9 exemplifies the ROC curves and metrics for the scenarios with *1)* at least one feature is anomalous, *2)* at least two features are anomalous, *3)* all three features are anomalous.




(a) ROC curve

| Scenario | Prec. | Recall | Acc. | F-Measure | AUC |
|---|---|---|---|---|---|
| PLC Programmer | 0.652 | 0.375 | 0.587 | 0.476 | 0.624 |
| Backup Coordinator | 0.833 | 1 | 0.900 | 0.909 | 0.899 |
| Various Attacker | 0.792 | 0.763 | 0.781 | 0.777 | 0.834 |

(b) Underlying values of the feature combination

Figure 9: ROC curve for the combination of timing and sizes features; underlying values of the combined heuristics.

One should choose our heuristics in combination, since an attack does not necessarily have to include all three anomalies, e.g., an attacker implementing large changes at a legitimate time that can still be detected in the LoadMemory. For the PLC programmer's evaluation, when two features trigger an alarm we obtain slightly better results than when one of three features is anomalous, since we have more false positives in the second case. For the other two scenarios, the various attacker and the backup coordinator, we obtain better results when already one anomalous feature triggers an alarm.

#### 4.2.4 Comparison of Results

When we jointly utilized the heuristics, we were able to increase the AUC for scenario *a)* by 32.7% instead of only using the timing feature. In case of scenario *b)*, the AUC increased by 40.6% instead of only using the loadMemoryRequired size variable. For scenario *c)*, the AUC increased by 32.8% instead of only using the timing feature, or 2.2% instead of only using the loadMemoryRequired size variable. For the other constellations, i.e., the comparison of our combination to the storage clustering for scenario *a)* and the timing clustering for scenario *b)*, the AUC decreases which is caused by the increasing number of false positives. We consider the combined utilization of features as slightly preferable.

#### 4.2.5 Extended Examination of Features Using Machine Learning Algorithm

In this section, we investigate the performance of a machine learning classifier and compare the results with the ones achieved using our heuristics. Therefore, we used the weka tool[5].

**Sizes and Times** In this section, we examine the combination of our features, i.e., sizes and timing variables, using the IBk classifier[6] in combination with the SpreadSubsample function[7] and a 10-fold cross-validation. The IBk classifier is a k-Nearest-Neighbor (kNN) algorithm which we deployed with neighbors $k = 1$. More neighbors led to slightly worse results. We also tested the utilization of REPTree[8], a decision tree classifier which builds a pruned decision tree; it is faster on higher amount of data, and insensitive to outliers. Our data used in this examination is: 400 entries of the data of PLC programmer and backup coordinator. For the various attacker, we used 456 samples. For the legitimate data we have 37,118 entries, the timestamps and size variables of the first project (axis).

We used both classifiers on these data: *1)* for 400 samples of each of the four classes, the legitimate, the various, the PLC, and the backup data. By utilizing the SpreadSubsample functionality, the tool creates random subsamples of our dataset; *2)* with only two classes, the legitimate and the anomalous data – here we have 1,256 subsamples each.

**1)** The best results were achieved by using kNN (IBk) with neighbors $k = 1$. The outcoming confusion matrix and the calculated values look as follows (Tab. 2):

| **Classified as →** | **Legit.** | **Various Att.** | **PLC Progr.** | **Backup Coord.** |
|---|---|---|---|---|
| Legitimate | 387 | 10 | 3 | 0 |
| Various Attacker | 6 | 377 | 10 | 7 |
| PLC Programmer | 0 | 0 | 400 | 0 |
| Backup Coordinator | 0 | 0 | 0 | 400 |

| **Scenario** | **Prec.** | **Recall** | **Acc.** | **F-Measure** | **AUC** |
|---|---|---|---|---|---|
| Legitimate | 0.985 | 0.968 | 0.988 | 0.976 | 0.991 |
| Various Attacker | 0.974 | 0.943 | 0.979 | 0.958 | 0.960 |
| PLC Programmer | 0.969 | 1.000 | 0.992 | 0.984 | 0.992 |
| Backup Coordinator | 0.983 | 1.000 | 0.996 | 0.991 | 0.997 |

Table 2: Confusion matrix of sizes and timing variables clustered by type of attacker; supplementary values.

In comparison to using weka's REPTree, IBk (kNN) provides slightly better results.

As this clustering scenario means that we are aware of the type of attacker, in a further step we only cluster for legitimate and anomalous data.

**2)** The best results for that mix of data was given by kNN (IBk) with $k = 3$ (cf. Tab. 3): These results are significantly better then those without using machine learning algorithms, i.e., the kNN algorithm, but come with higher computing cost for big datasets as used before where one single project already had

[5]weka tool: `https://waikato.github.io/weka-wiki/`

[6]IBk classifier: `https://weka.sourceforge.io/doc.dev/weka/classifiers/lazy/IBk.html`

[7]SpreadSubsample function: `https://weka.sourceforge.io/doc.dev/weka/filters/supervised/instance/SpreadSubsample.html`

[8]REPTree classifier: `https://weka.sourceforge.io/doc.dev/weka/classifiers/trees/REPTree.html`

| **Classified as →** | **Legit.** | **Anomalous** |
|---|---|---|
| Legitimate | 1,246 | 10 |
| Anomalous | 11 | 1,245 |

| **Scenario** | **Prec.** | **Recall** | **Acc.** | **F-Measure** | **AUC** |
|---|---|---|---|---|---|
| Legitimate | 0.991 | 0.992 | 0.992 | 0.992 | 0.992 |
| Anomalous | 0.992 | 0.991 | 0.992 | 0.992 | 0.992 |

Table 3: Confusion matrix of sizes and timing variables clustered by legitimate or anomalous data; supplementary values.

37,118 different data entries instead of using 400 in this part of the evaluation. In comparison to the above heuristics, we used only one project here. The heuristics were built across all two given projects, tested on all 4 afterwards. One should consider using kNN over heuristics but with more detailed segmentation in advance (not over all projects but per unit) and longer times to build the output.

# 5 Examination of DataBlocks and ProgramBlocks

So far we have covered the first feature set consisting of timing and size variables. This section handles the second of two feature sets that are examined in this work: the Data- and programBlocks' counts. Again, we first present an attacker scenario for this evaluation, followed by the dataset analysis.

## 5.1 Attacker Scenario

Another scenario – given by AUVESY GmbH – is one where we have given an attacker deleting or inserting a huge amount of code blocks. This is no stealthy scenario, as it can easily be recognized – but one has to monitor it. For this case, we extracted the programBlocks (PB, namely FB, FC, OB) and DataBlocks (DB) counts for each file of all four projects. We decided not to further cluster by different programBlocks as our results already are satisfying. Also, there were no big differences in the blocks insertions and deletions.

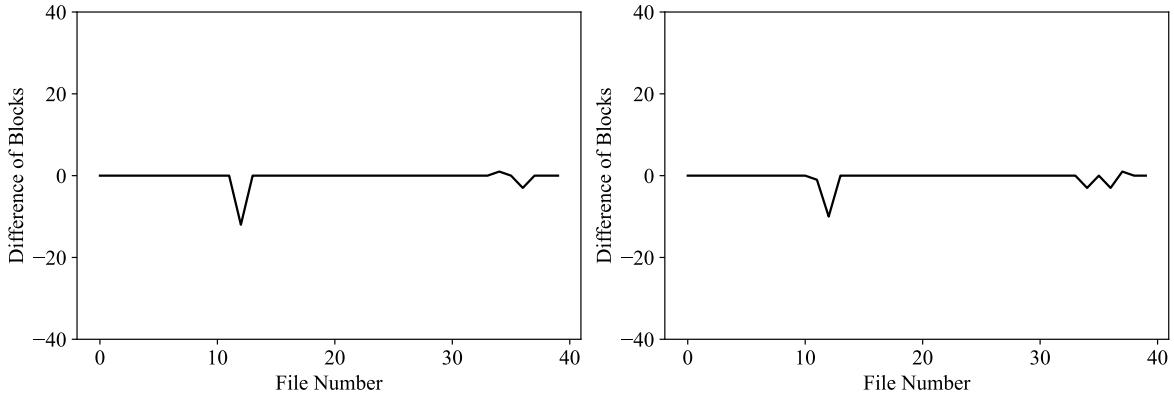
## 5.2 Dataset Analysis

What we had to differentiate between are DB and PB as their values deviate considerably. In Fig. 1 we already illustrated their counts per file. As one can recognize, the first files' amount of blocks is much bigger than the block count of the files. This is because the first bars reflect the first project - the axis - where the other three projects are of the same kind (all soldering tips). This leads to the conclusion that we have to differentiate between projects of different kinds. In the next steps, we therefore regard each project separately.

If we now look at the changes in the number of blocks compared to the previous file, we can see that the absolute number (mostly) only changes between 0 and 12 for DB and 0 to 10 for PB (exemplary for two projects, see Fig. 10).

For each of the projects, i.e., axis and three soldering tips (ST), the values are shown in Tab. 4.

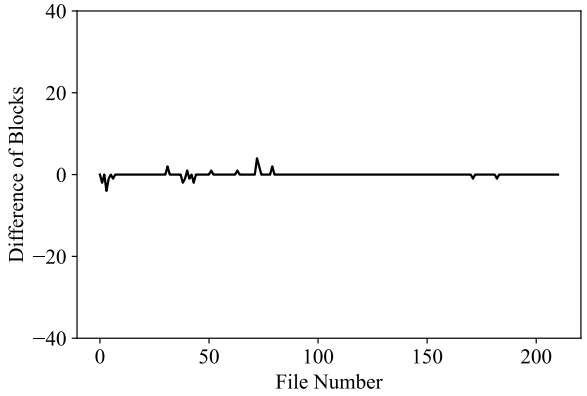
As one can see, the axis project (project 1) only has 40 backups wherein only three changes in the amount of DataBlocks appeared. The number of programBlocks changed five times in the same period. Changes in the number of DataBlocks $> 10$ should be considered anomalous here (Eqn. 3). For



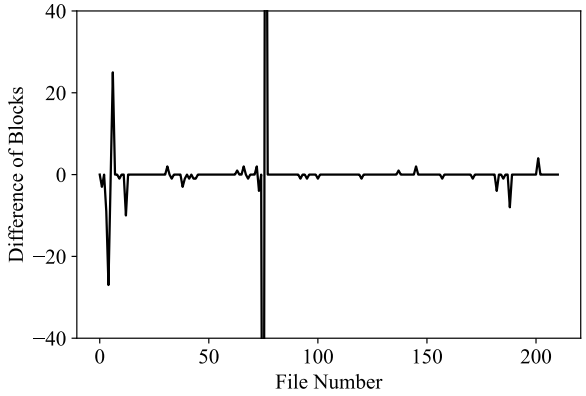

(a) DataBlocks of project 1 (backup)




(b) ProgramBlocks of project 1 (backup)




(c) DataBlocks of project 4 (archive)




(d) ProgramBlocks of project 4 (archive)

Figure 10: Differences in block count.

| **Project** | **Type** | **\|Value Changes\| (DB / PB)** | **# of Changes (DB / PB)** | **Period Length (Days)** |
|---|---|---|---|---|
| Axis | backup | 0-12 / 0-10 | 3 / 5 | 40 |
| ST 1 | archive | 0-5 / 0-8 | 12 / 26 | 190 |
| ST 2 | backup | 0 / 0-1 | 0 / 1 | 40 |
| | archive | 0-3 / 0-8 | 7 / 22 | 163 |
| ST 3 | backup | 0 / 0-1 | 0 / 1 | 40 |
| | archive | 0-11 / 0-10 (170*) | 18 / 33 | 212 |

* If we ignore the four outbursts (-25, 27, 170, -170), the remaining are in between 0 to 10.

Table 4: Evaluation of DataBlocks and programBlocks.

programBlocks, we set the threshold to $> 12$ to be ranked as an anomalous change in numbers (Eqn. 4). In both cases, the first block's difference is set to zero.

$$f(x) = \begin{cases} 1 & \text{if } |(n_x - p_x)| < 12 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$f(x) = \begin{cases} 1 & \text{if } |(n_x - p_x)| < 10 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

If we look at the percentages of files with and without changes, we can recognize that for DB the changes happen in up to 8.5% of the files. For PB, depending on the consideration of outbursts, it is 13.7% and 15.6% of the files that include a change in the block counts. For archives, it can be stated that there is a change in PB every 7 to 8 files, and for DB a change after between 11 and 24 files. Since backups do not provide us enough data (too few changes during 40 days), we cannot draw conclusions here.

Note that the overall number of changes in the number of blocks over a period of three years (i.e., of the archive files) is 37 for DataBlocks and 81 for programBlocks. This means, on average 1.03 changes in DataBlocks per month, and 2.25 for programBlocks. Again, the data is not sufficient to draw a conclusion about the backups.

By using the Decision Tree classifier, we achieved the following results (cf. Tab. 5) when classifying the DB and PB for anomalous and legitimate block difference quantities:

| Scenario | Prec. | Recall | Acc. | F-Measure | AUC |
|---|---|---|---|---|---|
| legitimate | 0.996 | 1.000 | 0.998 | 0.998 | 0.996 |
| anomalous | 1.000 | 0.996 | 0.998 | 0.998 | 0.996 |

Table 5: Decision Tree results for Data- and programBlocks.

At this point, we also experimented with the kNN classifier. We obtained one false negative result instead of one false positive result mentioned above. The company that implements the classifier can decide whether it prefers the risk of a false negative, and thus a potentially undetected malicious change, or a false alarm due to a false positive. As we got an outburst of 170 for the PB, we recommend to use the Decision Tree classifier over kNN.

# 6 Discussion of Detectability and Limitations

**Timing and size-based features.** Since attackers or saboteurs must not necessarily perform malicious modifications outside the legitimate times, the approach of only using the time-based feature is limited. At this point, we recommend a combination of the examined timing and size features, since the barrier to stay stealthy is increased this way. Nevertheless, it is possible for the attacker to tailor his modifications in a way perfectly fitting legitimate characteristics. In this case, the code and content must also be taken into account. Even considering a saboteur who is equally proficient in all features, the use of the combination is worthwhile, as this also increases the detection rate. Since this approach allows clustering across all projects, there are no restrictions on inserting or changing the data, which means that even the creation of a new project can be processed directly. For the storage-based features, individual blocks must be considered, i.e., the creation of a new block is not sufficient, in order to directly recognize an anomaly. At least one change must be made to be able to compare it with the previous block. Again, the limits of detectability are set: If an attacker solely introduces minimal changes, reference must be made to code and content. By combining the timing and sizes features, we obtain improved detection results.

**Block counts.** For the second examination, the DB and PB counts, it is clear that we can see larger changes in the data. The insertion of only one new block cannot be recognized. Only if a sequence of

inserted blocks appears in short time, or if there are many blocks deleted or inserted, we can directly recognize this anomalous behavior.

**Combination of heuristics.** The consideration of combining heuristics also shows that there is no visible correlation between the variables, since an attack can match one of these features and still leave the others unaffected. The data should thus be compared to the exact type of change (e.g., configuration or code) in future work. By including the contents of the files for further investigation, it could become feasible to detect historical attacks where, for example, only one limit value, e.g., the angle at which an axis pivots, the rotor speed, or process pressure, is increased by one every day until it assumes a malicious value. Also, we want to cover the case where a company's PLC programmers are less well-trained, i.e., they do not compare the TIA Portal data for changes between current version and backup version when inserting updates in code or content, which can make changes at work times unnoticeable. This task should be simplified by examining code and content for anomaly detection.

**Deployment of heuristics and machine learning algorithms.** For every upload of short- or long-term backups into the versioning tool, a validation of the data must be performed. This means that there is no real-time check at this point, since, for example, short-term backups are typically saved once a day. It would only be imaginable if every change would be uploaded into the tool in near real-time. This would theoretically be possible via manually uploading backups or archives, i.e., long-term backups. Another limitation of the data given by versiondog is that when a saboteur changes a value and reverses the change before the new backup is made, we will not see this change in our data as the presented status is the same as before the change was made. In such a case, one can consider implementing these checks in the TIA Portal for each push and pull of data from PLC to TIA Portal and vice versa.

**Transferring the heuristics and machine learning algorithms to other TIA Portal projects' data.** In order to transfer our procedures of the timing variables to other companies' data, they need to cluster their data to their respective times. We are currently investigating whether the heuristics of the sizes can be transferred directly and without further adjustments. For the blocks' counts, there are no precautions that need to be taken, despite one has to monitor new insertions/deletions over time and in their quantity. Within the various given projects of the car manufacturer, we were able to transfer the procedures successfully.

# 7 Conclusion

We presented extended initial results on an anomaly detection system using historical TIA Portal project data. While the detectability of the provided stealthy attacker scenarios appears to be challenging, we were able to gain utilizable results for the combined utilization of temporal and storage features. By using the kNN classifier we further improved our results. In comparison to our heuristics, one has to regard each project separately when applying machine learning algorithms. The non-stealthy scenario for the DB and PC counts is easy to detect if it is monitored. Smaller changes in blocks' quantity are not recognizable at all. Large or high-frequently reoccurring changes can be detected properly.

**Future work.** Our work identifies some more features from TIA Portal project data that can be investigated (cf. Fig. 5). Exemplary, the examination of code and configuration data seems to be worthwhile, although linguistic and graphical evaluation will require increased computational effort. For this reason, we plan to investigate such additional features to further increase the detectability.

## Acknowledgments

Part of this research was funded by the European Union from the European Regional Development Fund (ERDF) and the State of Rhineland-Palatinate (MWG), Germany. Funding content: P1-SZ2-7 F&E: Wissens- und Technologietransfer (WTT), Application number: 84003751 (MADISA); the Programm zur Förderung des Forschungspersonals, Infrastruktur und forschendem Lernen (ProFIL) of the University of Applied Sciences Worms; AUVESY GmbH, Landau (Rhineland-Palatinate), which also provided us the datasets; and the Center for Technology and Transfer (ZTT) at the University of Applied Sciences, Worms.

## References

[1] L. Hartmann and S. Wendzel. Detection of anomalous values within TIA project data history for industrial control systems. In *Proc. of the 2021 European Interdisciplinary Cybersecurity Conference (EICC'21), Virtual Event, Romania*, pages 91—-97. ACM, November 2021.

[2] C. Feng, V.R. Palleti, A. Mathur, and D. Chana. A systematic framework to generate invariants for anomaly detection in industrial control systems. In *Proc. of the 27th Network and Distributed System Security Symposium (NDSS'19), San Diego, California, USA*, pages 1–15. NDSS, February 2019.

[3] I. Kiss, B. Genge, P. Haller, and G. Sebestyén. Data clustering-based anomaly detection in industrial control systems. In *Proc. of the 10th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP'14), Cluj-Napoca, Cluj, Romania*, pages 275–281. IEEE, October 2014.

[4] F. Zhang, H.A.D.E. Kodituwakku, J.W. Hines, and J. Coble. Multilayer data-driven cyber-attack detection system for industrial control systems based on network, system, and process data. *IEEE Transactions on Industrial Informatics*, 15(7):4362–4369, July 2019.

[5] R.B. Zhang, L.H. Xia, and Y. Lu. Anomaly detection of ICS based on EB-OCSVM. *Journal of Physics: Conference Series*, 1267:012054, July 2019.

[6] H. Yoo and I. Ahmed. Control logic injection attacks on industrial control systems. In *Proc. of the 34th IFIP TC 11 International Conference on ICT Systems Security and Privacy Protection (SEC'19), Lisbon, Portugal*, pages 33–48. Springer, Cham, June 2019.

[7] C. Wressnegger, A. Kellner, and K. Rieck. Zoe: Content-based anomaly detection for industrial control systems. In *Proc. of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'18), Luxembourg City, Luxembourg*, pages 127–138. IEEE, July 2018.

[8] M. Zhang, C.-Y. Chen, B.-C. Kao, Y. Qamsane, Y. Shao, Y. Lin, E. Shi, S. Mohan, K. Barton, J. Moyne, and Z. M. Mao. Towards automated safety vetting of plc code in real-world plants. In *Proc. of the 19th IEEE Symposium on Security and Privacy (S&P'19), San Francisco, California, USA*, pages 522–538. IEEE, September 2019.

[9] T.K. Das, S. Adepu, and J. Zhou. Anomaly detection in industrial control systems using logical analysis of data. *Computers & Security*, 96:101935, September 2020.

[10] C.M. Ahmed and J. Zhou. Challenges and opportunities in CPS security: A physics-based perspective. *IEEE Security & Privacy*, 18(6):14–22, November–December 2020.

[11] M. Krotofil, J. Larsen, and D. Gollmann. The process matters: Ensuring data veracity in cyber-physical systems. In *Proc. of the 10th ACM Symposium on Information, Computer and Communications Security (CCS'15), Singapore, Republic of Singapore*, pages 133–144. ACM, April 2015.

[12] A.L. Perales Gómez, L. Fernández Maimó, A. Huertas Celdrán, F.J. García Clemente, C. Cadenas Sarmiento, C.J. Del Canto Masa, and R. Méndez Nistal. On the generation of anomaly detection datasets in industrial control systems. *IEEE Access*, 7:177460–177473, December 2019.

[13] J.E. Rubio, C. Alcaraz, R. Roman, and J. Lopez. Current cyber-defense trends in industrial control systems. *Computers & Security*, 87:101561, November 2019.

[14] Y. Hu, H. Li, H. Yang, Y. Sun, L. Sun, and Z. Wang. Detecting stealthy attacks against industrial control systems based on residual skewness analysis. *EURASIP Journal on Wireless Communications and Networking*, 74:1–14, March 2019.

[15] B. Genge, P. Haller, and C. Enăchescu. Anomaly detection in aging industrial internet of things. *IEEE Access*, 7:74217–74230, June 2019.

[16] D.I. Urbina, J.A. Giraldo, A.A. Cardenas, N.O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg. Limiting the impact of stealthy attacks on industrial control systems. In *Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16), Vienna, Austria*, pages 1092–1105. ACM, October 2016.

[17] M. Krotofil, A. Cárdenas, J. Larsen, and D. Gollmann. Vulnerabilities of cyber-physical systems to stale data—determining the optimal time to launch attacks. *International Journal of Critical Infrastructure Protection*, 7(4):213–232, December 2014.

[18] W. Alsabbagh and P. Langendörfer. A stealth program injection attack against S7-300 PLCs. In *Proc. of the 21th IEEE International Conference on Industrial Technology (ICIT'21), Valencia, Spain*, pages 986–993. IEEE, June 2021.

[19] W. Alsabbagh and P. Langendörfer. A control injection attack against S7 PLCs -manipulating the decompiled code. In *Proc. of the 47th Annual Conference of the IEEE Industrial Electronics Society (IECON'21), Toronto, Ontario, Canada*, pages 1–8. IEEE, November 2021.

[20] W. Alsabbagh and P. Langendörfer. Patch now and attack later - exploiting S7 PLCs by time-of-day block. In *Proc. of the 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS'21), Victoria, British Columbia, Canada*, pages 144–151. IEEE, July 2021.

[21] L. Hartmann and S. Wendzel. How feasible are steganographic and stealth attacks on TIA project metadata of ICS: A case study with real-world data. In *Proc. of the 21th European Interdisciplinary Cybersecurity Conference (EICC'21), Virtual Event, Romania*, pages 83—84. ACM, November 2021.

---

## Author Biographies

**Laura Hartmann** is a Ph.D. student at FernUniversität in Hagen, Germany, since 2019. She worked for the project MADISA (*Machine learning for attack detection using data of industrial control systems*) (till 2021) and is now a researcher at the Center for Technology and Transfer (ZTT). Her research interests include network steganography and anomaly detection for data of industrial control systems. Laura (co-)authored eight publications so far. Website: `https://madisa.ztt.hs-worms.de`.

**Steffen Wendzel** is a professor of information security and computer networks at Hochschule Worms, Germany, where he is also the scientific director of the Center for Technology and Transfer (ZTT). In addition, he is a lecturer at the Faculty of Mathematics & Computer Science at the FernUniversität in Hagen, Germany, from which he also received his Ph.D. (Dr. rer. nat., 2013) and Habilitation (2020). Before joining Hochschule Worms, he led a smart building security research team at Fraunhofer FKIE in Bonn, Germany. Steffen (co-)authored more than 170 publications. He served as the chair for IWSMR'19-'22 and other workshops, was PC co-chair for EICC'20 and '22 as well as GI Sicherheit'16, and served as a guest editor for special issues of major journals, such as IEEE Security & Privacy (S&P), IEEE Transactions Industrial Informatics (TII) and Future Generation Computer Systems (FGCS). He is editorial board member for the Journal of Universal Computer Science (J.UCS) and the Journal of Cybersecurity & Mobility (JCSM). His website: `https://www.wendzel.de`.