electronics

MDPI

Article

Advanced Bad Data Injection Attack and Its Migration in Cyber-Physical Systems

Wenping Deng [1], Ziyu Yang [2,*], Peng Xun [1], Peidong Zhu [1] and Baosheng Wang [1]

[1] College of Computer, National University of Defense Technology, Changsha 410073, China
[2] Institute of Systems Engineering, Academy of Military Sciences, Beijing 100000, China
* Correspondence: zyyang@nudt.edu.cn

Received: 11 July 2019; Accepted: 18 August 2019; Published: 26 August 2019

Abstract: False data injection (FDI) attack is a hot topic in cyber-physical systems (CPSs). Attackers inject bad data into sensors or return false data to the controller to cause the inaccurate state estimation. Although there exists many detection approaches, such as bad data detector (BDD), sequence pattern mining, and machine learning methods, a smart attacker still can inject perfectly false data to go undetected. In this paper, we focus on the advanced false data injection (AFDI) attack and its detection method. An AFDI attack refers to the attack where a malicious entity accurately and successively changes sensory data, making the normal system state continuously evaluated as other legal system states, causing wrong outflow of controllers. The attack can lead to an automatic and long-term system failure/performance degradation. We first depict the AFDI attack model and analyze limitations of existing detectors for detecting AFDI. Second, we develop an approach based on machine learning, which utilizes the k-Nearest Neighbor (KNN) technique and heterogeneous data including sensory data and system commands to implement a classifier for detecting AFDI attacks. Finally, simulation experiments are given to demonstrate AFDI attack impact and the effectiveness of the proposed method for detecting AFDI attacks.

Keywords: false data injection; cyber-physical system; security; detector; heterogeneous data

1. Introduction

Many critical infrastructures, such as smart grid and smart transportation systems, are fundamentally supported by the underlying cyber systems [1]. Efficient and convenient management can be achieved by adopting cyber systems. However, there exist many vulnerabilities in cyber systems, like malformed message attacks [2–4] and denial of service attacks [5]. Therefore, for modern cyber-physical systems (CPSs), many new vulnerabilities from cyberspace have been exposed, and consequently, security has been a crucial factor for these modern CPSs. For example, attackers can delay the data transmission between cyberspace and physical space by intruding the cyberspace to cause the wrong control of physical process [6]. Currently, a vast number of existing studies pay attention to attacks and their migrations in CPSs [6–12]. In particular, FDI is a hot topic in CPSs. FDI attack refers to one kind of attack that attackers modify measurements of sensors or return false sensory data to the controller, causing the wrong state estimation [11,12]. Attackers can launch FDI attacks by injecting bad data into sensors, intruding communication systems to modify feedback data, or modifying time stamps of sensory data.

Previous works [13–19] have extensively studied FDI attacks. However, previous FDI attacks mainly pay attention to masking the wrong system state or disturbing the state estimation by launching just one attack. They did not discuss how to keep going undetected for a long time after an FDI caused the system fault. Especially, continuously falsifying legal states by injecting legal data into the normal systems to cause automatic and long-term disruption/performance degradation,

is a research area with limited work. Legal data mean the measurements that can pass detectors and can be estimated as legal system state. A legal state refers to the state where controllers consider the system is normally running.

On the other hand, many effective detectors of FDI attacks have been proposed. For example, detectors based on machine learning [20,21] and methods based on models [13,22] have been used to detect FDI attacks. The above methods can better detect some special FDI attacks. However, an effective detection method still needs to be explored when attackers elaborately falsify normal system states. Especially, the above detection approaches are ineffective to identify FDI attacks that continuously falsify other legal states when the systems run normally.

In this paper, we focus on FDI attacks and their migrations in large-scale CPSs. First, we depict a complex FDI attack called advanced false data injection (AFDI) attack, where attackers infer system parameters by long-term monitoring sensory data, and continuously inject legal data to modify the normal system state as another legal system state, leading to automatic outflow of mismatched commands. Consequently, long-term harm or performance degradation of physical systems is obtained. Different from previous FDI attacks that only consider how to mask the system exception or disturb the system by launching one FDI attack, AFDI attacks focus on how to cause automatic and successive disruption by only injecting legal measurements under normal situations. Second, we prove that the existing detection approaches, such as BDD, sequence pattern mining-based detectors, and Machine Learning-Based detectors, cannot identify AFDI attacks. Third, bearing in mind that commands from the controllers cause changes in sensory data, and attackers hope to inject false data to generate wrong commands, we propose a novel detection method based on machine learning. The proposed method utilizes the KNN technique and heterogeneous data, including commands from the controller and sensory data to obtain a classifier for detecting AFDI attacks. Finally, our simulation experiments validate the impact of AFDI attacks and the effectiveness of the proposed detection approach.

Our contributions can be summarized as follows.

- We depict the AFDI attack model, which can only directly and successively cause system failure/performance degradation by injecting legal data into the normal system.
- We discuss the limitations of fundamental detectors for detecting AFDI attacks.
- A novel detector based on machine learning, Heterogeneous Data Learning Detector (HeteD), is proposed, which can effectively identify AFDI attacks.

The rest of this paper is organized as follows. In Section 2, we introduce the system model of CPS and FDI attack model. We depict the attack model of AFDI in Section 3. In Section 4, we discuss the limitation of fundamental detectors and describe the proposed detection approach against AFDI attacks. Numerical results are given in Section 5. The related works about FDI attacks and detection are reviewed in Section 6. In Section 7, we conclude this work and propose future work.

2. Preliminaries

2.1. System Model of CPS

The model of a large-scale CPS is shown in Figure 1, which includes a cyber system, communication system, and physical system. The cyber system is composed of the central controller and state estimator. The physical system includes a vast number of PLCs and sensors. The communication system is responsible for transmitting information between the physical system and cyber system. Each cycle, sensors measure the physical process and send sensory data to the cyber system. Sensory data is first transmitted to the state estimator and the state estimator evaluates the system state. If no exceptions about sensory data exist, the estimated system state is sent to the central controller. Then, the central controller issues commands based on the evaluated state to the physical domain by the communication system. When commands reach the physical system, PLCs first receive commands and disaggregate them into many subcommands to control the physical process. After that, the physical process has

a change and sensory data may be different from previous measurements. The above process can be modeled in terms of a 5-tuple

$$P = \{C, T, S, R, F\}$$

where

- $C = \{c_1, ..., c_m\}$ is a finite set of commands from the central controller. c_k is the kth kind of command. $C(k) = \{c_i, ..., c_j\}$ indicates the commands issued by the central controller at time k.
- $T = \{t_1, ..., t_{n_T}\}$ is a finite set of time series. A time series is the measured values of one sensor with the change of time. $t_i = \{t_i(1), ..., t_i(k)\}^T$ means the time series from the ith sensor. $t_i(k)$ denotes the measurement of the ith sensor at time instant k. n_T means the number of sensors.
- $S = \{s_1, ..., s_n\}$ is a finite set of physical system states. $s_j = \{a_1, ..., a_{n_S}\}^T$ denotes the jth state and $a_i \in \mathbb{R}$. Estimators can evaluate the system state at time k, $S(k)$, based on values of sensors, which can be computed by

$$T(k) = C_{matrix} \times \overline{S(k)} \tag{1}$$

  where $C_{matrix} \in \mathbb{R}^{n_T \times n_S}$ is called the Jacobian matrix of the system topology. $\overline{S(k)} \in S$ denotes the evaluated state at time k, and under normal circumstances $\overline{S(k)} = S(k)$ and $T(k) = \{t_1(k), ..., t_{n_T}(k)\}^T$. The system's physical dynamics is described by the following widely adopted discrete-time model [22].

$$T(k) = A \times T(k-1) + B \times g(u(k-1)) \tag{2}$$

  where $A \in \mathbb{R}^{n_T \times n_T}$ and $B \in \mathbb{R}^{n_T \times m}$ are constant matrices. $u(k) = \{u_1(k), u_2(k), ..., u_m(k)\}^T \in \mathbb{R}^{m \times 1}$ denotes the control signals determined by the control commands at time k. The value of $u_i(k)$ is determined by the corresponding command c_i. $g() \in \mathbb{R}^{m \times 1}$ is a function of $u(k)$.
- $R = \{r_1, ..., r_i, ..., r_{n_R}\}$ is a finite set of relationships among states and commands. $r_k = s_i \rightarrow \{c_i, ..., c_j\}$ ($r_k \in R$) indicates that when the system state is s_i, the central controller will issue a set of commands $\{c_i, ..., c_j\}$. These commands may be activated by the corresponding state or be input by operators.
- $F = \{f_1, ..., f_{n_F}\}$ is a subset of set S. Any state in set F is called an illegal state. A disruption or degradation of performance occurs when the system state is f_i.

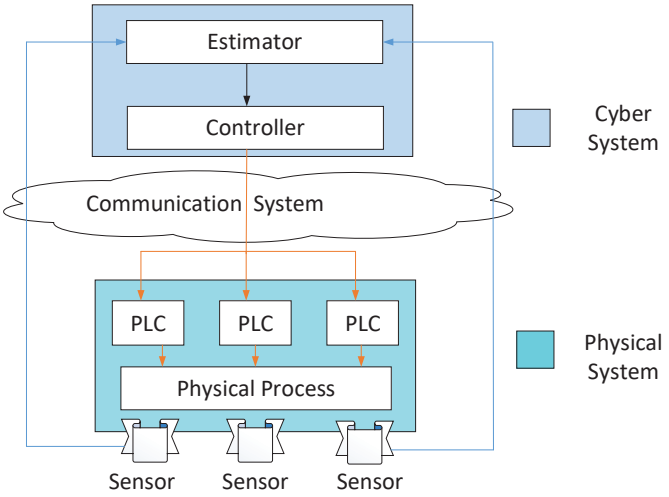


Figure 1. Model of a large-scale cyber-physical system (CPS).

2.2. FDI Attack Model

We describe an FDI attack from three aspects: attack goal, attacker's knowledge, and attack capability. Attacks with different goals, knowledge, and capabilities can obtain different levels of impact.

Attack goal. Injecting false data can make the evaluated state $\overline{S(t)}$ differ from the real state $S(t)$. Based on different intents, the goal can be divided into two categories: (i) Inject bad data to mask the wrong state $S(t) = s_f (s_f \in F)$. (ii) Inject data to cause the outflow of wrong commands $C(t)$ $(S(t) \rightarrow C(t) \notin R)$.

Attacker's knowledge. An attacker can have different levels of knowledge of the target system, including (i) the sensory data, T; (ii) Jacobian matrix, C_{matrix}; (iii) system commands, C; (iv) system parameters A, B, and g; (v) and knowledge of states R and F. Depending on the assumptions made on each of these components, we can envisage different attack scenarios. Typically, two main settings are considered, referred to as attacks with perfect and limited knowledge.

- Perfect knowledge. In this case, everything of the targeted system assumes to be known by the attacker. Components T and C can be obtained by intruding into the communication system. An attacker can know F and C_{matrix} from the public information and system theory. Combining history data and C_{matrix}, R can be obtained. If the attacker is a designer of systems, $\{A, B, g\}$ may be known.
- Limited knowledge. We assume that getting information from the public dataset and collecting data by intruding into the CPS can be implemented, which means in this case, although there exists a wide range of limited knowledge, $\{T, C_{matrix}, C, F, R\}$ can be obtained by the attacker. Considering that the CPS may be complex, $\{A, B, g\}$ cannot be obtained in many situations.

Attack capability. An attacker can have different levels of attack capability, including (i) attackers can inject any value into a part of sensors; (ii) attackers can simultaneously modify sensory data of all sensors, but the injected data is limited; and (iii) attackers can inject any sensory data into any sensor.

- Situation (i) because there exist vulnerabilities in sensors and intruding into some sensors may be easy, and a part of the sensor can be attacked and any bad data can be injected.
- Situation (ii) as described in the work by the authors of [18], attacks, such as time synchronization attacks, can be launched to modify time stamps. All values of sensors can be simultaneously modified; however, injected values of sensory data are in a limited range.
- Situation (iii) there exists a vast number of sensors and these sensors are distributed. Intruding into all sensors is not practical. However, an attacker may intrude into the communication system and modify feedback data transmitted from the sensors to the state estimator. For example, in the work by the authors of [19], sensory data is fed back from PLCs to the state estimator. There exist possibilities that attackers manipulate PLCs to modify all of the feedback data (i.e., attack the website of firmware update of PLCs).

3. AFDI Attack Model

We first characterize the AFDI model from attack goal, attacker's knowledge, and attack capability. Then, we describe the attack process of AFDI.

AFDI Attack Goal. For AFDI attacks, the outflow of mismatched commands is needed to disrupt the physical system. Considering that long-term disruption or performance degradation is the primary target of AFDI attacks, continuously masking the fault state is also needed.

AFDI Attacker's Knowledge. To go undetected for a long time, an attacker should know what sensory data is proper for the current state, which means that knowledge $\{T, C_{matrix}, C, F, R, A, B, g\}$ is needed. By analyzing historical data and public information, $\{C_{matrix}, C, F, T, R\}$ can be obtained. In the latter paragraph, we propose a method that attackers without prior knowledge of $\{A, B, g\}$ can perfectly falsify system states. Launching an AFDI attack only requires limited knowledge.

AFDI Attack Capability. Considering that attacks last for a long time and measurements of any sensor may be influenced, an AFDI attack requires the ability that attackers can modify all sensory data. Therefore, the ability to inject any data into any sensor is needed by an attacker to launch AFDI attacks.

We assume that a high-skilled attacker can own the limited knowledge and the needed attack capability in the follow-up discussion, which is feasible such as attack event stuxnet [23]. The attack process is described as follows.

- Collect information and compute parameter A

A malicious entity collects sensory data $\{T(1), ..., T(N)\}$ and system commands $\{C(1), ..., C(N)\}$. Then, the attacker generates a collection V = null and adds any element $< T(i), T(j) >$, satisfying $i < j$, $S(i) = S(j)$, and $C(i) = C(j)$ into V. Therefore, we can obtain

$$T(j+1) - T(i+1) = A \times (T(j) - T(i)). \tag{3}$$

By utilizing linear regression, $A =$

$$(X^T X)^{-1} X^T Y$$

where $X = (..., T(j) - T(i), ...)$, $Y = (..., T(j+1) - T(i+1), ...)$, and $< T(i), T(j) > \in$ V.

- Search state transition path

By analyzing historical data, a malicious entity can find the existing relationships R among states and commands. Simultaneously, state transition paths are also mined. A state transition path denotes the state sequence, such as $\{s_1, s_2, s_3\}$, where the system state is s_1 at the beginning, and then the state is changed to s_2. With the outflow of commands, the system state becomes s_3.

In this case, an attacker needs to find two normal state transition paths:

$$\textbf{Path} \quad \textbf{1}: \mathrm{G_s} \xrightarrow{\mathrm{C_d}} \mathrm{s_{i_1}} \xrightarrow{\mathrm{C_{i_1}}} \mathrm{s_{i_2}} \xrightarrow{\mathrm{C_{i_2}}} \mathrm{s_{i_k}} ... \xrightarrow{\mathrm{C_{in}}} \mathrm{s_{i_n}}$$

$$\textbf{Path} \quad \textbf{2}: \mathrm{G_s} \xrightarrow{\mathrm{C_f}} \mathrm{s_{j_1}} \xrightarrow{\mathrm{C_{j_1}}} \mathrm{s_{j_2}} \xrightarrow{\mathrm{C_{j_2}}} \mathrm{s_{j_k}} ... \xrightarrow{\mathrm{C_{jm}}} \mathrm{s_{j_m}}$$

where C_d, C_f, C_{i_k}, and C_{j_k} mean different sets of commands. G_s denotes a series of state transitions. $s_k \xrightarrow{C_{i_k}} s_l$ denotes that when the system state is s_k and commands C_{i_k} are issued from the controller, the state becomes s_l.

- Inject continuously bad data

At time l, the system state is s_d ($G_s = \{s_1, s_2, ..., s_d\}$), commands are C_f (bf Path 2), and the attack is launched at time $l+1$. We use $T(k)$ to represent the historical sensory data where the system state is s_d and commands are C_d (Path 1). In this case, attackers will continuously modify sensory data such that the controller considers that the state path is Path 1. The injected bad data $T(l+j)_{bad}$ at time $l+j$ satisfies

$$\begin{cases} T(l+j)_{bad} = T(k+j) + A \times (T(l+j-1) - T(k+j-1)), \\ T(l)_{bad} = T(l), \\ 0 < j \leq t_{bad} \quad j \in \mathbb{I}, \end{cases} \tag{4}$$

where t_{bad} means the duration of bad data injection.

Theorem 1. *When injected bad data satisfies (4), the evaluated state at time $l+j$ is the same to the state at time $k+j$.*

Proof. Based on (1), the estimated state at time $l+j$ is

$$\begin{aligned}
&\overline{S(l+j)} \\
&= M(T(k+j) + A(T(l+j-1) - T(k+j-1))) \\
&= S(k+j) + MA(T(l+j-1) + G - G - T(k+j-1)) \\
&= S(k+j) + S(l+j-1) - S(k+j-1) \\
&= S(k+j),
\end{aligned}$$

where $G = Bg(u(l+j-1)) = Bg(u(k+j-1))$ and $M = (C_{matrix}^T C_{matrix})^{-1} C_{matrix}^T$.

From Theorem 1, we know that by utilizing the historical data, attackers can continuously falsify other legal states by bad data injection satisfying (4). During the continuous attacks, the controller considers that Path 1 is executed; however, the actual situation is shown in Path 3.

$$\text{Path} \quad 3 : \mathrm{G_s} \xrightarrow{\mathrm{C_f}} \mathrm{s_{j_1}} \xrightarrow{\mathrm{C_{i_1}}} \mathrm{s_{h_2}} \xrightarrow{\mathrm{C_{i_2}}} \mathrm{s_{h_k}} \ldots \xrightarrow{\mathrm{C_{in}}} \mathrm{s_{h_n}}$$

Command sequence $\{C_{i_1}, ..., C_{i_n}\}$ is issued when the state sequence is $\{s_{i_1}, ..., s_{i_n}\}$, but for state s_{h_k}, commands C_{i_k} are not proper to manipulate the physical process, and the physical system may go into illegal states $\{..., s_{h_k}, ...\}(s_{h_k} \in F)$ for a long time. □

4. Limitation of Fundamental Detectors and Our Detection Approach

In this section, first, we discuss the limitations of existing fundamental detectors including BDD, detector based on machine learning, and sequence pattern mining-based detector. Then, we propose a novel and effective method to detect AFDI attacks.

4.1. Limitations of Fundamental Detectors

Reviewing the existing FDI detection approaches, we divide detectors into three categories, including residual based BDD, detectors utilizing classifiers based on machine learning, and detectors based on sequence pattern mining. Next, we discuss their limitations for detecting AFDI attacks, respectively.

4.1.1. Residual-Based BDD

Residual-based BDD utilizes the residual between the observed sensory data and estimated sensory data to identify anomalies. Intuitively, the normal measured sensory data is close to the actual values, and the bad data may move the estimated variables away from their true values [13]. Equation (5) represents the detection theory, where l_{BDD} denotes the threshold.

$$T(k) \begin{cases} Bad & \|T(k) - C_{matrix} \times \overline{S(k)}\|_2 > l_{BDD} \\ Normal & \text{Others} \end{cases} \tag{5}$$

Although the residual based BDD can identify many types of bad data, previous studies such as [13] have proved that when the bad data satisfies (6), attacks cannot be detected.

$$T(k)_{bad} = T(k) + \beta \tag{6}$$

where β is a linear combination of the column vectors of C_{matrix} (i.e., $\beta = C_{matrix} \times h$). $T(k)_{bad}$ and $T(k)$ denote the bad data and normal data at time k, respectively.

Following the fore discussion, in Theorem 2, we show that an AFDI attack goes undetected by the residual-based BDD.

Theorem 2. *For the descriptor system (Equations (1) and (2)) and the residual-based BDD, the AFDI attacks are always undetectable.*

Proof. When the AFDI attack is launched at time $t+1$ and the attacker hopes to falsify the state transition path since time $j+1$ ($S(t) = S(j)$), the difference between injected bad data and actual data is

$$\begin{aligned} & T(t+l)_{bad} - T(t+l) \\ & = T(j+l) + A(T(t+l-1) - T(j+l-1)) - T(t+l) \\ & = A \times T(t+l-1) + G - T(t+l) \\ & = T(t+l) - T(t+l) \\ & = C_{matrix} \times 0, \end{aligned}$$

where $G = g(u(k+l-1)) = g(u(k+j-1)$ and $0 < l \leq t_{bad}$.

Therefore, based on (6), the injected bad data can pass the residual based BDD. □

4.1.2. Machine Learning-Based Detectors

In previous studies, the detector based on machine learning can be seen as a binary classification problem. From the perspective of machine learning, there exist three modes including supervised learning, semi-supervised learning, and unsupervised learning. Because methods based on supervised learning provide better detection results [24], we mainly discuss their limitations.

Supervised machine learning techniques utilize a set of labeled training data (i.e., the samples that we have known whether they are abnormal) to generate a classifier. When a new sample without labeling is input into the classifier, the model will show its label. We mainly focus on two methods: Class I: classifier based on sensory data $T(t)$ [21,25]. Class II: classifier based on the difference between two sensory data $T(t) - T(t-l_d)$ [24] , where $l_d \in \mathbb{I}$ is a constant and its value depends on the setting of defenders.

For Class I, samples with labels can be represented as $\{(T(1), y_1), (T(2), y_2), ..., (T(k), y_k)\}$, where y_i is the label of $T(i)$. When bad data is injected at time i, $y_i = 1$, otherwise, $y_i = 0$.

For Class II, samples with labels can be represented as $\{(T(l_d+1) - T(1), y_1), (T(l_d+2) - T(2), y_2), ..., (T(k) - T(k-l_d))\}$. When bad data is injected at time i, $y_{i-l_d} = 1$, otherwise, $y_{i-l_d} = 0$.

Although there exists many machine learning techniques, such as KNN, random forest (RF), support vector machine (SVM), and naive bayes (NB), they cannot provide effective detection results by only using sensory data to train model when attackers inject bad data as described in Section 3. Theorems 3 and 4 show that under some conditions, AFDI attacks go undetected by the supervised learning techniques. Assume that the task is to predict the label of injected bad data $T(t)$ at time t. The machine learning technique, 1-nearest neighbor learning, is seen as an example.

Theorem 3. *For the descriptor system (1) and (2) and Class I-based detector utilizing the 1-nearest neighbor learning technique*

$$\overline{y_t} = \arg\min_{y_j} \|T(t) - T(j)\|_2,$$

the AFDI attacks are always undetectable.

Proof. When bad data is injected at time t, we can obtain the following information. (1) $y_t = 1$; (2) $y_j = \overline{y_j} = y_{j-1} = \overline{y_{j-1}} = y_{t-1} = \overline{y_{t-1}} = 0$, where $\overline{y_i}$ denotes the predicted label and y_i denotes the actual label; (3) $T(t) = T(j) + A \times (T(t-1) - T(j-1))$.

Then, we predict the label of $T(t)$, $\overline{y_t}$, as

$$\begin{aligned} & \arg\min_{y_k} \|T(t) - T(k)\| \\ & = \arg\min_{y_k} \|C_{matrix} \times S(t) - C_{matrix} \times S(k)\| \\ & = \arg\min_{y_k} \|C_{matrix} \times (S(t) - S(k))\|. \end{aligned}$$

When $k = j$, $\overline{y_t} = y_j = 0$ because $S(t) = S(j)$. Therefore, the AFDI attacks can go undetected. □

Theorem 4. *For the descriptor system (Equations (1) and (2)) and Class II-based detector utilizing the 1-nearest neighbor learning technique*

$$\overline{y_t} = \arg\min_{y_j} \|T(t) - T(t - l_d) - (T(j) - T(j - l_d))\|,$$

when attackers can find two paths satisfying Path 1 and Path 2 in the historical data and constant l_d is smaller than the duration of G_s, the AFDI attacks are undetectable.

Proof. Assume that bad data is injected at time t and $T(t) = T(j) + A \times (T(t-1) - T(j-1))$. Because l_d is smaller than the duration of G_s, the evaluated system state $S(t - l_d)$ at time $t - l_d$ is the same to the evaluated system state $S(j - l_d)$. Therefore, we can obtain

$$C_{matrix} \times (\overline{S(t)} - S(t - l_d) - (S(j) - S(j - l_d))) = 0.$$

Based on the above result, it is easy to obtain $\overline{y_t} = y_j = 0$ and AFDI attacks can go undetected.

Because most of machine learning techniques assume that similar samples tend to have similar labels and attack data $T(t)$ can always find the similar benign data $T(j)$, the ability is also similar for the detection of FDI attacks under different machine learning techniques [24]. Therefore, we say that the type of approach is ineffective to identify AFDI attacks. □

4.1.3. Sequence Pattern Mining-Based Detector

The sequence pattern mining-based detector utilizes a series of state transitions to denote the normal system behaviors and abnormal system behaviors. An effective method has been proposed in the works by the authors of [26,27], where defenders compute the number of occurrences of every state transition path and use some methods to determine whether a path is normal. In general, the larger the number of occurrences is, the larger the possibility that it is a normal path. An exception is issued when the current path is not in the historical data. For example, Path 1 and Path 2 are normal, but Path 3 is abnormal.

Although the method can effectively detect many attacks, when attackers inject bad data, as described in Section 3, they cannot provide the effective detection results. Theorem 5 shows that AFDI attacks can be undetectable by the detector based on sequence pattern mining.

Theorem 5. *For the descriptor system (Equations (1) and (2)) and sequence pattern mining-based detector, the AFDI attacks are always undetectable.*

Proof. Assume that Path 1 and Path 2 often occur in the running process. When a malicious entity launches attacks and falsifies the executed Path 2 as Path 1, the real state transition path is Path 3. However, the controller and detector consider that Path 1 is executed. From the detector's point of view, Path 1 is normal and attacks do not occur. Therefore, AFDI cannot be detected by the sequence pattern mining-based detector. □

4.2. Our Methodology: Heterogeneous Data Learning Detector (HeteD) for Detecting AFDI Attacks

From the analysis in the previous subsection, it is clear to see that an AFDI attack can continuously inject bad data to falsify system states, going undetected by the existing detectors. These detectors mainly pay attention to sensory data, but the AFDI can perfectly modify data to be undetectable. By analyzing the state transition path, we observe that although attackers can falsify the changes in states, the commands from the controller are not modified. Actually, the impact of the attack is created because of the improper combination between the system state and issued commands.

Therefore, defenders can utilize the information from commands to enhance the detector based on sensory data learning. By utilizing the heterogeneous data including commands and sensory data, we propose HeteD based on machine learning techniques to identify AFDI attacks.

As shown in Figure 2, HeteD receives sensory data from sensors and commands from the controller every unit time. There are three components in the detector, including "Process Data" component, "Generate Sample" component, and "Classifier" component. Each unit time, commands are transmitted to the component "Process Data". The component changes commands as a series of signals and then sends them to component "Generate Sample". "Generate sample" combines signals and measurements of sensors to obtain a detectable sample and transmits it to "Classifier". Classifier is responsible for detecting the sample and deciding whether the data is abnormal based on KNN technique. Next, we describe the above process in details.

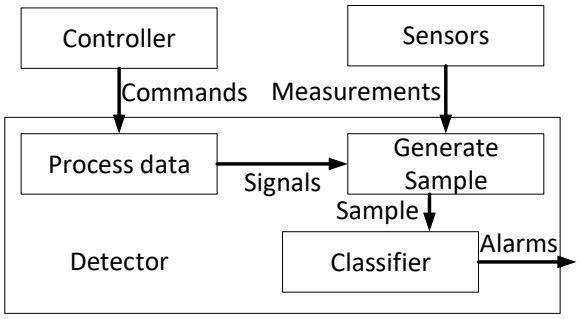


Figure 2. The structure of the Heterogeneous Data Learning Detector (HeteD).

4.2.1. Process Heterogeneous Data

In this step, we change every command as a control signal. The control signal has two values "0" and "1". We divide different commands into two classes, including continuous commands and instant commands. A continuous command denotes that when the command occurs, its impact on the system state can last some time or the impact is broken until the next related command occurs. An instant command represents that when the command occurs, its impact on the system state instantly occurs, but cannot be kept. For example, command "turn on or off a valve" is a continuous command, and command "Increase demands" in the smart grid case is an instant command. Data processing of the two types of commands is described as follows.

- Continuous Command Processing

When a continuous command occurs, the value of the signal becomes "1" and it remains invariant for a time interval. The duration depends on the character of the corresponding command. For example, if the influence of the command only lasts a fixed time interval, the value of the signal becomes "0" when the duration of "1" is equal to the fixed time interval. If the impact of a command is stopped when another occurs, the value of the signal becomes "0" after the corresponding command occurs. Figure 3a–c describes the two situations. In Figure 3a, a command that allocates resource for users with effective limited time interval $t_{interval}$ is issued at time 0. When the time is $t_{interval}$, the value of the signal becomes "0". In Figure 3b,c, we show the values of two signals about two commands "turn on the valve" and "turn off the valve". When "turn on the valve" occurs at time 0, the value of the corresponding signal becomes "1". When "turn off the valve" occurs at time $t_{interval}$, the value of the signal about command "turn on the valve" becomes "0" and the value of the signal about command "turn off the valve" is "1".

- Instant Command Processing

When an instant command occurs, the value of the signal becomes "1" from "0" or changes from "1" to "0". The value of the signal remains invariant until its next outflow. In Figure 3d,

a command that increases demand into the smart grid is issued at time 0 and time $t_{interval}$. The value of the corresponding signal changes from "0" to "1" at 0 and changes from "1" to "0" at $t_{interval}$.

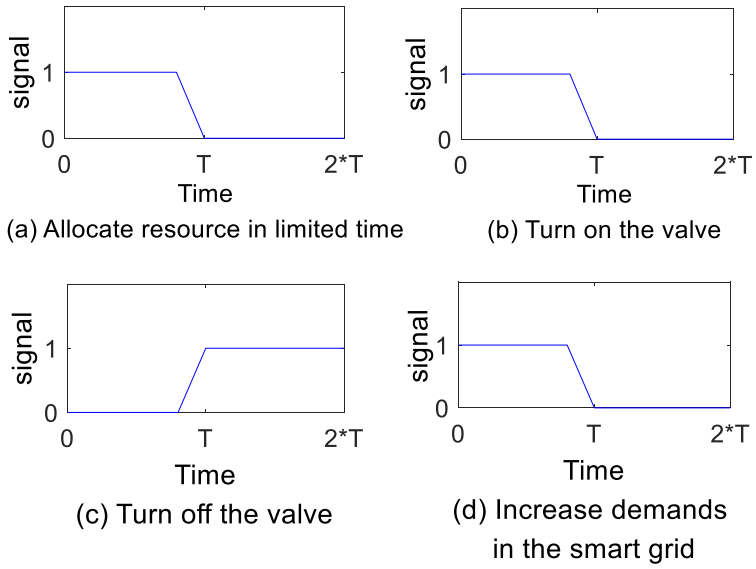


Figure 3. An example of heterogeneous data processing.

4.2.2. Sample Generation

Each unit time, HeteD combines signals and sensory data to generate a sample and sends the sample to "Classifier". Next, we describe the structure of a sample.

We utilize variable $V_S(t) = \{V(t)^T, T(t)^T\}^T$ to represent the combination of signals and time series at time t, where $V(t) = \{v_1(t), v_2(t), ..., v_m(t)\}^T$ denotes the signal vector at time t and $v_i(t)$ denotes the signal of command c_i at time t. To capture the temporal structure of data sequence, we generate samples using the concept of first difference (the work by the authors of [24] used the technique to implement FDML detector, which attributes to Class II introduced in Section 4.1.2). Therefore, the new sample $N_S(t)$ at time t can be described as

$$V_S(t) - V_S(t - l_d) \tag{7}$$

where l_d is the parameter and set by the detector.

4.2.3. Classifier

"Classifier" is implemented by utilizing the KNN technique. We assume that there are many historical data, including normal samples and abnormal samples. If there are fewer abnormal samples, we can obtain them by simulating attacks. Label is annotated on every historical sample. If the sample is normal, the corresponding label is "0", otherwise, the label is "1".

When a new sample $N_S(t)$ is input into "Classifier", the component will search K_c historical samples satisfying

$$\min_{N_S(k)} |N_S(t) - N_S(k)|_2$$

subject to

$$k < t$$

where $K_c \in \mathbb{I}$ is a parameter of the detector.

For K_c selected samples, if the number of samples whose labels are equal to "1" is larger than the number of samples whose labels are equal to "0", the predicted label of $N_S(t)$ is "1", otherwise, the label is "0". When the predicted label is "1", an alarm is issued from the detector.

At last, we discuss the reason that HeteD can obtain better detection effect than FDML. Considering the process of AFDI attacks, when attackers inject bad data satisfying (Equation (4)) since time t, bad sensory data $T'(t)$ causes the outflow of wrong control commands. From HeteD's point of view, sensory data is $T'(t)$, and the control commands are changed from $C(t)$ to $C'(t)$. Next, we prove when AFDI attacks satisfy (Equation (4)), we can obtain

$$||N'_s(t) - N_s(t-l_d) - (N_s(k) - N_s(k-l_d)||_2 > ||N_s(t) - N_s(t-l_d) - (N_s(k) - N_s(k-l_d))||_2 = 0.$$

Proof. Based on Theorem 3, we know $||T(t) - T(t-l_d) - (T(k) - T(k-l_d))||_2 = 0$. Considering $S(t) = S(k)$ and $S(t-l_d) = S(k-l_d)$, we can obtain $C(t) = C(k)$. Therefore, $||N_s(t) - N_s(t-l_d) - (N_s(k) - N_s(k-l_d))||_2 = 0$. Because $N'_s(t) \neq N_s(t)$, we can obtain $||N'_s(t) - N_s(t-l_d) - (N_s(k) - N_s(k-l_d)||_2 > 0$. Therefore, the equation is proved. □

According to the above conclusion, we say there exist possibilities that the normal situation is evaluated as the normal sample and there exist possibilities that the abnormal situation is evaluated as the abnormal sample. Therefore, comparing with FDML (Theorem 4), HeteD can obtain a better detection effect.

5. Numerical Results

In this section, several simulations are given to validate the impact of AFDI attacks and evaluate the performance of HeteD. Considering we have proved that BDD, sequence pattern mining-based detector, and Class I-based detector utilizing machine learning have no ability to identify AFDI attacks, we only compare the effectiveness of our method with the approach, FDML, using the first difference between two sensory data in the work by the authors of [24].

5.1. Simulation on Smart Grid

5.1.1. Scenario

The imbalance between demand and generation can lead to deviation of the frequency from its normal value. If the deviation is larger than a threshold for a long time, some generators may be disconnected from the grid leading to the cascading failure [28]. Direct load control is responsible for recovering the deviation in frequency. The model of the direct load control in the smart grid is shown in Figure 4. The direct load controller can turn off or on electric appliances to change loads of smart grid to decrease the deviation in frequency. When the frequency is lower than the normal frequency, $f_n = 50$ HZ, the direct load controller decreases the use of direct load. When the current frequency is higher than the normal frequency, the direct load controller increases the use of direct load [29]. Users can also randomly turn on or off appliances to increase or decrease loads of the smart grid. Under normal situations, when demands of users have a change, generation of generators has a change and the frequency oscillates until the frequency remains invariant. The generation $G(t)$ changes according to

$$\begin{cases} G_{TAR}(t) = (\dfrac{f_{sp} - f(t)}{0.04 f_n}) \times G_{MAX}, \\ G(t + u_t) = G(t) + (G_{TAR}(t) - G(t)) \times M \times u_t, \end{cases}$$

where f_{sp} is the set point of frequency and G_{MAX} denotes the max power of generators. $M = 3$ s and $u_t = 1$ s.

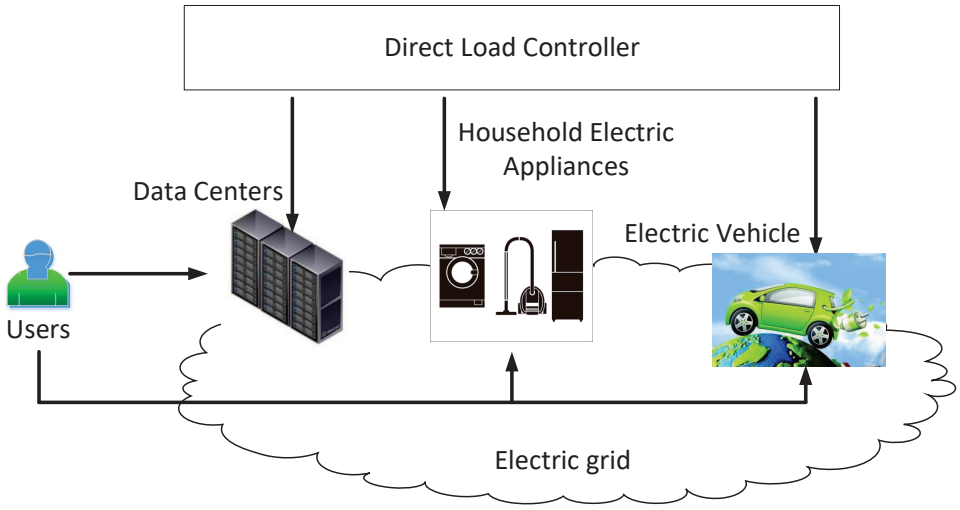


Figure 4. The model of the direct load control.

When the frequency becomes stable, the frequency may not be equal to the normal frequency. Then, as described in the work by the authors of [30], the direct load controller is activated and the output load L_D satisfies

$$\begin{cases} \min\{L_{terminal}(t) - \dfrac{f_{sp} - f_n}{2} * G_{MAX}, L_U(t)\} & f(t) < f_n \\ \min\{\dfrac{f_{sp} - f_n}{2} * G_{MAX} - L_{terminal}(t), L_R(t)\} & f(t) > f_n \end{cases}$$

where the output load refers to the load that the direct load controller will turn on or off. $L_{terminal}(t)$ denotes the sum of terminal loads at time t. $L_U(t)$ denotes the number of loads that can be turned off at time t and $L_R(t)$ means the number of loads that can be turned on at time t.

We use Matlab to implement the direct load control. We only pay attention to the sensors that measure the frequency of the smart grid and demands of users. The frequency can be computed by the relationship between demands and generation [30], described as

$$\begin{cases} \omega(t) = 2\pi \times f(t), \\ \alpha = \dfrac{2 \times G_{MAX} \times h}{\omega_{nor}^2}, \\ \dfrac{1}{2}\alpha \times \omega^2(t + u_t) = \dfrac{1}{2}\alpha \times \omega^2(t) + (G(t) - L_{terminal}(t)) \times u_t, \\ f(t + u_t) = \dfrac{\omega(t + u_t)}{2\pi}, \end{cases} \tag{8}$$

where ω_{nor} denotes the rotating frequency at normal frequency and $h = 4$.

The system has 14 commands, shown in Table 1, where commands $C1$–$C8$ are automatically issued from the direct load controller and commands $C9$–$C16$ are from the changes in users' demands. The direct load controller issues commands based on the frequency and current demands of users. Users' demands can be changed randomly by users. When the frequency becomes higher than 50.2 HZ or lower than 49.8 HZ and lasts for 40 s, generators or electrical appliances will be broken.

Table 1. Description of commands in the smart grid.

Command	Description
C_1/C_{16}	Turn on a load of $X \in$ (0 MW, 120 MW)
C_2/C_{15}	Turn off a load of $X \in$ (0 MW, 120 MW)
C_3/C_{14}	Turn on a load of $X \in$ [120 MW, 240 MW)
C_4/C_{13}	Turn off a load of $X \in$ [120 MW, 240 MW)
C_5/C_{12}	Turn on a load of $X \in$ [240 MW, 480 MW)
C_6/C_{11}	Turn off a load of $X \in$ [240 MW, 480 MW)
C_7/C_{10}	Turn on a load of $X \in$ [480 MW, $+\infty$)
C_8/C_9	Turn off a load of $X \in$ [480 MW, $+\infty$)

We randomly change users' demands, $L_U(t)$ and $L_R(t)$, to collect a series of normal historical data.

5.1.2. Attack Effect on the Smart Grid

In the subsection, we pay attention to two normal situations:

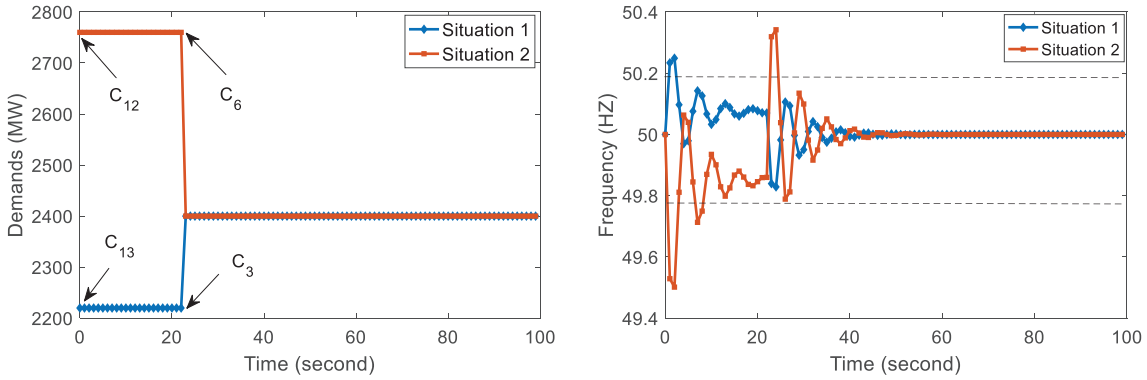
Situation 1: As shown in Figure 5, at time $t = 0$, demands change from 2400 MW to 2220 MW and the frequency begins to oscillate. Until the frequency becomes stable at time $t = 22$, the direct load controller issues command C_3 and the frequency returns to the normal value.

Situation 2: As shown in Figure 5, at time $t = 0$, demands change from 2400 MW to 2760 MW and the frequency begins to oscillate. Until the frequency becomes stable at time $t = 22$, the direct load controller issues command C_6 and the frequency returns to the normal value.

Based on the two situations, we study two AFDI attack cases:

Case 1: When situation 1 occurs, attackers use measurements of demands and frequency under situation 2 to replace the real measurements of demands and frequency, and the attack lasts for 100 s.

Case 2: When situation 2 occurs, attackers use measurements of demands and frequency under situation 1 to replace the real measurements of demands and frequency, and the attack lasts for 100 s.



(a) Demands under normal situations (b) Frequency under normal situations

Figure 5. Measurements under normal situations

The real measurements under two attack cases are shown in Figure 6. For case 1, comparing Figure 6a with Figure 5a, due to false data injection using measurements of situation 2, the direct load controller thinks that the frequency is lower than the normal frequency; therefore, command "turn off loads" is issued at time $t = 22$. However, the real demand is smaller than the generation and the operation makes the deviation of frequency larger as shown in Figure 6b. At last, the frequency is higher than 50.2 HZ and remains invariant for a long time. Until time $t = 100$, generators will be disconnected and electrical appliances may be broken.

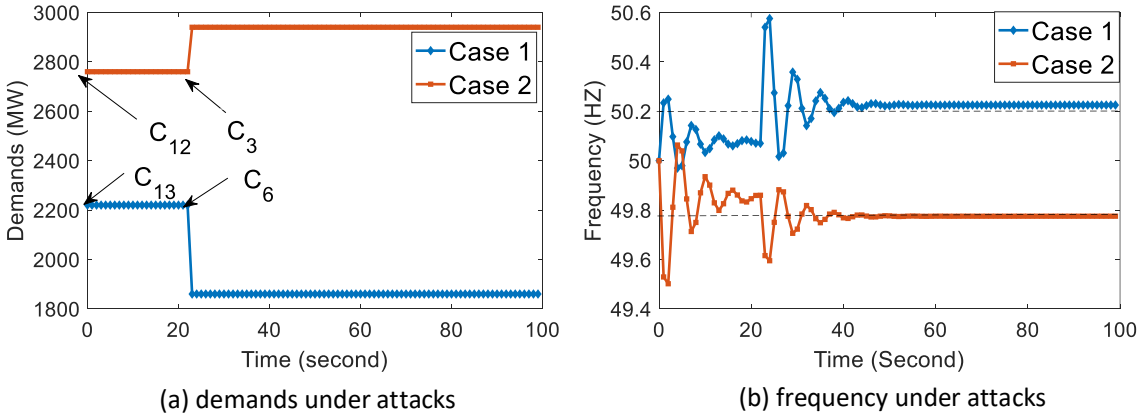


Figure 6. Real measurements under two attack cases.

For case 2, comparing Figure 6a with Figure 5a, due to false data injection using measurements of situation 1, the direct load controller thinks that the frequency is higher than the normal frequency; therefore, command "turn on loads" is issued at time $t = 22$. However, the real demand is larger than the generation and the operation makes the deviation of frequency larger as shown in Figure 6b. At last, the frequency is lower than 49.8 HZ and remains invariant for a long time. Until time $t = 100$, generators will be disconnected and electrical appliances may be broken.

From the above results, we can know AFDI attacks in the smart gird can obtain large disruption.

5.1.3. Performance of HeteD on the Smart Grid

We first introduce two performance metrics, including false positive ratio and false negative ratio.

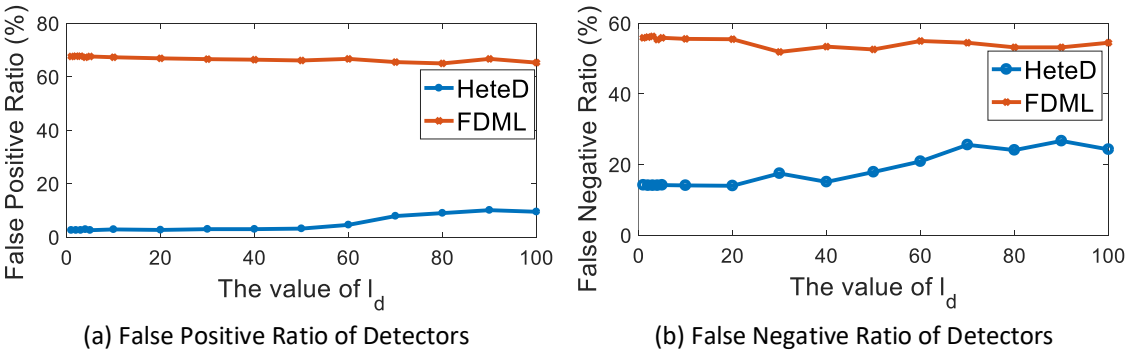
False positive ratio refers to the ratio of samples that are secure and classified as anomalies to all secure samples. False negative ratio is the ratio of samples that are anomalies, but are classified as normal samples to all abnormal samples.

For commands C1–C16, we see them as instant commands. To compare the detection performance of our approach with FDML, we launch 50,000 AFDI attacks at different time points to obtain 50,000 attack samples. Every attack lasts 120 s. We use the same 50,000 situations without attacks as normal samples. Forty-nine-thousand attack samples and 49,000 normal samples are used as training samples, and extra samples are testing samples.

We implement two detectors—HeteD and FDML—by python. The detection performance of two detectors on the corresponding dataset is shown in Figure 7 where K_c=5. With different values of l_d, we can see the false positive ratio of HeteD is lower than FDML in Figure 7a. With different values of l_d, we can also clearly observe that the false negative ratio of HeteD remains lower than FDML in Figure 7b. Comparing the detection results of HeteD and FDML in the smart grid, we can know that when we use HeteD and set $l_d = 1$, the false positive ratio is very small, the false negative ratio is lower than 20%, and good detection effect can be achieved.

Next, we analyze the reason that the two detection methods have different detection effects. When we use FDML to detect exceptions, the samples are constructed by demands and frequency. As described in Equation (4), when l_d is small, there exists a vector $T(k)$ satisfying Theorem 4. Therefore, most of abnormal situations may be seen as normal samples and many normal situations are seen as abnormal samples. For example, when case 1 in Figure 6 occurs, the bad data (frequency and demands under situation 2) can be seen as normal samples and FDML can not detect anomalies. When case 2 in Figure 6 occurs, the same conclusion is obtained. Actually, malicious samples and normal samples have the same vectors. Therefore, the detection effect is very poor. When we use HeteD to detect exceptions, the samples are constructed by sensory data and control commands. The abrupt changes in control commands lead to the situation that most of attacks can not be seen as normal samples and most of normal running situations are not seen as attack samples. For example, when case 1 in Figure 6

occurs, the bad data (frequency demands under situation 2) cannot match with the command sequence $\{C_{13}, C_6\}$ because $\{C_{12}, C_6\}$ are issued under normal situations. When case 2 in Figure 6 occurs, the bad data (frequency, demands under situation 1) can not match with command sequence $\{C_{12}, C_3\}$ because $\{C_{13}, C_3\}$ are issued under normal situations. When $l_d = 1$, the above two examples can be identified. Therefore, HeteD can provide the better detection effect.



(a) False Positive Ratio of Detectors (b) False Negative Ratio of Detectors

Figure 7. Detection performance of detectors on the smart grid.

5.2. Simulation on Tank System

5.2.1. Scenario

A tank system [7,31] is simulated by using Matlab/Simulink, and its structure is shown in Figure 8. The controller of the tank system receives requests from users to produce liquid C, E, F, and G by the neutralization process of ingredient A, ingredient B, and ingredient D. Liquid C is produced when the ratio of ingredient A to ingredient B is 1. When the ratio is 3, liquid G is produced. Liquid E is produced when the ratio of ingredient A to ingredient D is 1, and when the ratio is 3, liquid F is obtained. Every time the tank system only generates one kind of product. Every ingredient can be supplied by three tanks and flows out from tanks by 3 mL/second. Every tank has a sensor to measure the current amount of ingredient or product. Moreover, from the tank with the ingredient to the tank neutralizing product, there exist pumps controlling the input of product tank. When a product is neutralized, the corresponding valves of outputting liquid are open and the liquid flows out from their tanks by 6 mL/second. The tank system also provides a sensor to measure which service is needed by users. Services include service 1: $60 \times 3 \times 2$ mL liquid C, service 2: $60 \times 3 \times 4$ mL liquid C, service 3: $60 \times 3 \times 4$ mL liquid G, service 4: $60 \times 3 \times 2$ mL liquid E, service 5: $60 \times 3 \times 4$ mL liquid E, and service 6: $60 \times 3 \times 4$ mL liquid F.

Producing liquid C with $60 \times 3 \times 4$ mL is described as an example to demonstrate the process. The initial state is s_0. When service 2 is requested, the state becomes s_2 and the controller issues commands to turn on two pumps that input ingredient A to tank P2 (i.e., p11 and p12). Then the state changes from s_2 to s_8. Until the system state becomes s_{14}, commands which close pumps are issued. After 60 s, the system automatically issues commands to turn on the valve $V11$. Then the system state becomes s_{19}. Tables 2 and 3 show the system commands and system states. We randomly request different services for a long time and obtain a set of normal data.

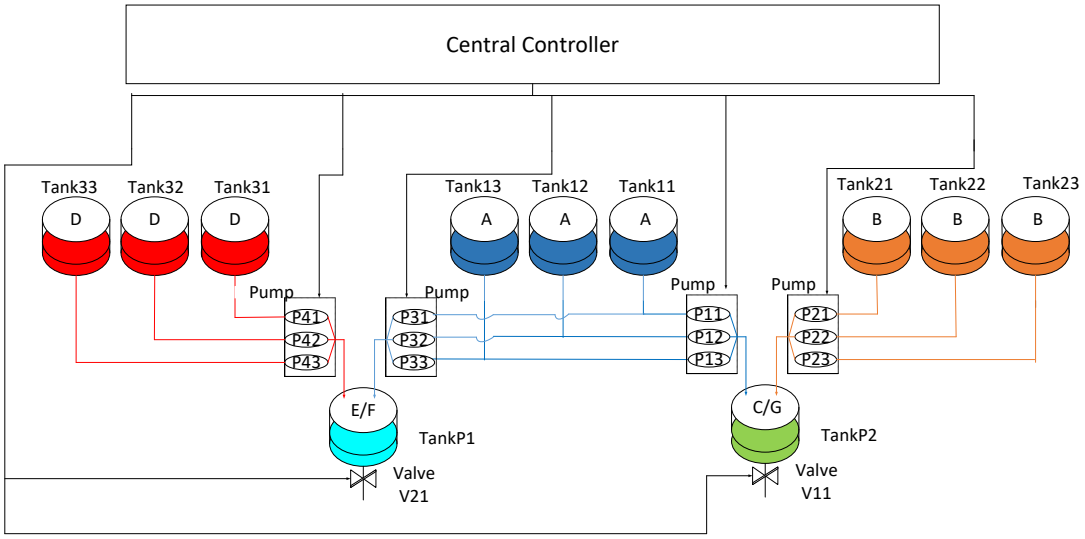


Figure 8. The structure of the tank system.

Table 2. Description of commands in the tank system.

Command	Description
P11o/P11f	Switch on/off Pump P11
P12o/P12f	Switch on/off Pump P12
P13o/P13f	Switch on/off Pump P13
P21o/P21f	Switch on/off Pump P21
P22o/P22f	Switch on/off Pump P22
P23o/P23f	Switch on/off Pump P23
P31o/P31f	Switch on/off Pump P31
P32o/P32f	Switch on/off Pump P32
P33o/P33f	Switch on/off Pump P33
P41o/P41f	Switch on/off Pump P41
P42o/P42f	Switch on/off Pump P42
P43o/P43f	Switch on/off Pump P43
V11o/V11c	Open/Close Valve V11
V21o/V21c	Open/Close Valve V21
Rs 1-6	Request Service 1–6

Table 3. Description of states in the tank system.

State	Description
s_0	No request
s_1/s_2	Request service 1/service 2
s_3/s_4	Request service 3/service 4
s_5/s_6	Request service 5/service 6
s_7/s_8	Output ingredient under service 1/service 2
s_9/s_{10}	Output ingredient under service 3/service 4
s_{11}/s_{12}	Output ingredient under service 5/service 6
s_{13}/s_{14}	The product is obtained under service 1/service 2
s_{15}/s_{16}	The product is obtained under service 3/service 4
s_{17}/s_{18}	The product is obtained under service 5/service 6
s_{19}	Valve V11 is open
s_{20}	Valve V12 is open

5.2.2. Attack Impact on the Tank System

In this subsection, we use an attack case to illustrate the attack effect. By analyzing the history data, we first obtain parameter $A = E_{11\times11}$, where E denotes the identify matrix. Every time users request a service, the system state is s_0 at the beginning. We select two state transition paths from the historical data to generate the attack case

$$\textbf{Path} \quad \textbf{4}: s_0 \xrightarrow{RS5} s_5 \xrightarrow{p33o,p32o,p42o,p41o} s_{11} \rightarrow s_{17} \xrightarrow{p33f\,p32f,p41f,p42f,V21o} s_{20}$$

$$\textbf{Path} \quad \textbf{5}: s_0 \xrightarrow{RS3} s_3 \xrightarrow{p11o,p12o,p13o,p21o} s_9 \rightarrow s_{15} \xrightarrow{p11f,p12f,p13f,p21f,V11o} s_{19}$$

Since time $t = 2580$, the system state is s_0 and users need liquid G. At $t = 2581$, an AFDI attack is launched and false data is injected to tell the controller that service 5 is needed. Attackers hope the controller will execute Path 4. Figure 9a,b shows the situation about production in TankP2 and TankP1 under the normal situation and attacked situation. In Figure 9a, we can clearly observe that liquid G and liquid F are successively needed. However, in Figure 9b, the actual production is liquid E (service 5). The above results illustrate the AFDI attack impact on the tank system.

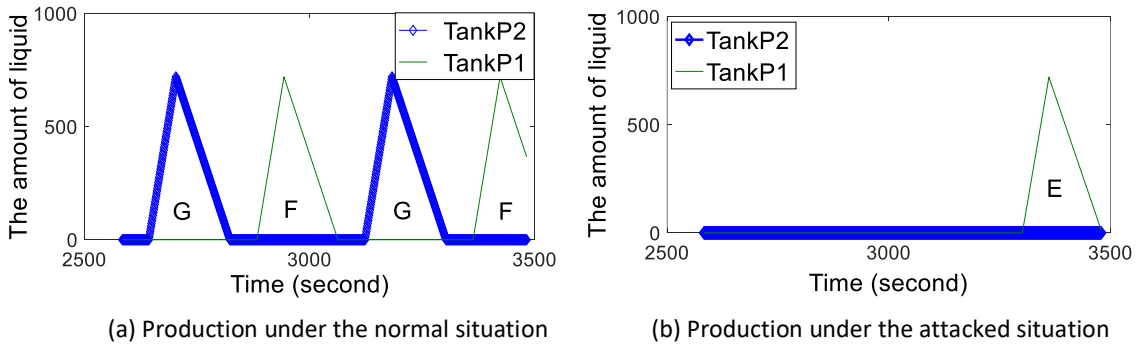
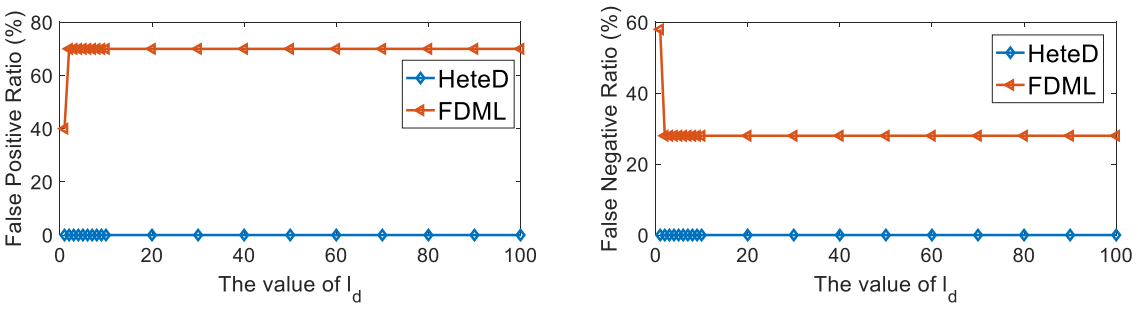


Figure 9. Production under the normal situation and the attacked situation.

5.2.3. Performance of HeteD on the Tank System

Commands $RS1 \sim RS6$ are seen as instant commands and others are continuous commands. We randomly launch 50,000 AFDI attacks at different time points. Every FDI attack lasts 120 s. We also select the same data without attacks as the normal samples. 49,000 attack samples and 49,000 normal samples are used as the training samples, and residual 1000 attack samples and 1000 normal samples are testing samples.

In Figure 10, we show the detection performance with changes in the values of l_d under two detectors, where $K_c = 5$. With the increase of l_d, we can see that the false positive ratio and false negative ratio of HeteD keeps invariant and all of attacks can be identified. The detection results are very good and all of attacks can be detected. For FDML, when $l_d = 1$, the best detection results can be achieved. FDML is ineffective to detect AFDI attacks because of extremely high false positive ratio and extremely high false negative ratio. Next, we analyze the reason that HeteD has better detection effect than FDML. When AFDI attacks are launched, measurements of tanks are modified and the bad data can be evaluated as normal system states. According to Theorem 4, normal samples and abnormal samples have the same vectors and the detection model based on machine learning will provide poor results. For HeteD, taking the attack case as an example, when attackers modify the state from s_3 to s_5, control command $RS3$ is not modified as $RS5$. When the detection sample is constructed, the abnormal sample is not the same as the normal sample. Therefore, HeteD can provide better detection results.



(a) False Positive Ratio (a) False Negative Ratio

Figure 10. Detection performance of detectors on the tank system.

Based on the attack cases in the smart grid and tank system, we can see that AFDI attacks can successively degrade the system performance or disrupt the physical system for a long time. By analyzing the detection effect of FDML and HeteD in the smart grid and tank system, the proposed detection method is better than FDML when detecting AFDI attacks. Although our method, in effect, detects AFDI attacks in many CPSs, the following issues, not explored in this paper, should be considered in the case of real attacks. (1) A vast number of commands and sensors may decrease the performance of HeteD, which may be solved by decreasing the dimension of the sample. (2) Noise of measurements may impact the injected bad data and detection performance. (3) External input of control commands may impact the detection results.

6. Related Work

In this section, we review the previous works from two aspects, including FDI attacks and detection methods.

6.1. FDI Attack

In the work by the authors of [13], FDI attack was first introduced and attackers could inject proper bad data to be undetectable by BDD of the smart grid. Attackers only need partial knowledge to launch FDI attacks for masking the system exception or disturbing the normal running. In the work by the authors of [14], FDI attacks with perfect knowledge were proposed to disturb the system running. However, the above two works only discussed a single attack and does not concern how to launch successive attacks. In the work by the authors of [15], attackers with perfect knowledge injected false data to successively mask transmission line outages leading to a serious situation without awareness. With further research, attack strategies have been improved, for example, optimal FDI attack actions are studied to mask the exceptional frequency leading to the largest disruption of generators in the work by the authors of [16]. In the work by the authors of [17], FDI attacks with limited knowledge are described to mask the line outages. In the work by the authors of [18], time synchronization attack was introduced. Attackers falsified the GPS signal to generate sensory data with bad time stamps, masking the system faults. Different from previous works, this method does not need system knowledge. In the work by the authors of [19], FDI attacks with perfect knowledge were used to cooperate with false command injection attacks. False command injection caused the system failure and FDI could delay the time of attack detection and develop impact of attacks. Based on the above discussion, the characters of FDI attacks [13–19] are summarized in Table 4. We can clearly see that the FDI attack becomes more complex, hides the traces of attacks better, and causes greater disruption. However, previous FDI attacks mainly pay attention to masking system exceptions or conducting the system exception by only launching one FDI. Previous works do not discuss how to keep going undetected for a long time after an FDI causes the exception. Especially, continuously falsifying legal states by injecting legal data into the normal systems to cause the automatic and long-term disruption/performance degradation, is a research area with limited work.

Table 4. Characters of previous FDI attacks.

Works	Attack Goal	Knowledge	Successive Attack
[13]	Mask wrong system state or cause the system fault	Perfect	No
[14]	Cause the system fault	Perfect	No
[15]	Mask wrong system state	Perfect	Yes
[16]	Mask wrong system state	Perfect	Yes
[17]	Mask wrong system state	Limited	Yes
[18]	Mask wrong system state	Limited	No
[19]	Mask wrong system state	Perfect	No
HeteD	Cause system faults and mask wrong system state	Limited	Yes

6.2. Detection of FDI Attacks

For FDI detection, many effective approaches have been proposed. In the work by the authors of [11], the method based on correlation mining between data was proposed. However, when attackers can know the correlations among different types of data and modify multiple types of sensory data like time synchronization attack, the detection results are poor. In works by the authors of [12,20,21,25], the method based on machine learning utilizing classification models was depicted. Although the above approaches can detect some FDI attacks escaping from BDD, other complex FDI attacks, such as an elaborate time synchronization attack, cannot be detected. In the work by the authors of [24], a new machine learning method based on first difference was utilized to detect time synchronization attacks. In the work by the authors of [26], the authors built a system model and used sequential pattern mining to analyze state paths and detect anomalies. However, AFDI can not be detected by this method. The above methods can better detect some special FDI attacks. However, an effective detection method still needs to be explored when attackers elaborately falsify normal system states. Especially, the above detection approaches are ineffective to identify AFDI attacks. The characters of the above detection methods are summarized in Table 5, where time cost refers to the duration from analyzing data to obtaining detection results. The difference between HeteD and previous works is that HeteD utilizes the control commands and sensory data to train the detection model.

Table 5. Characters of Previous FDI Attacks.

Works	Using Data	AFDI Detection Accuracy	Time Cost
[11]	sensory data	0	Instant
[12,20,21,25]	sensory data	20% 45%	Instant
[24]	first-difference sensory data	30% 55%	Instant
[26]	sensory data	0	Instant
HeteD	first-difference sensory data and control commands	80% 100%	Instant

7. Conclusions

We have described the AFDI attack model, which can directly cause long-term disruption of physical systems by injecting continuously data into the normal system. For example, in the smart grid, with the injection of bad sensory data, the frequency largely changes and the traditional detectors cannot identify. The deviation of frequency exceeds its threshold and the smart grid may encounter cascading failure. For the tank system, with the injection of bad sensory data, the production process is broken and wrong produce is achieved; however, traditional detectors cannot identify these situations. We also prove that the traditional attack detectors are ineffective to detect AFDI attacks. Most importantly, we propose a novel and effective ML-based method by utilizing commands and sensory data to identify AFDI attacks. The simulation results show that our detector, HeteD,

can effectively identify AFDI attacks with low false positive ratio and low false negative ratio. For example, in the scenario of smart grid, our detector can identify most of AFDI attacks with 2.3% false positive ratio and 13.8% false negative ratio. The FDML provides worse detection results with 60% false positive ratio and 50% false negative ratio. In the scenario of the tank system, our detector identified all AFDI attacks with 0% false negative ratio. The FDML provides worse detection results with 70% false positive ratio and 28% false negative ratio. From the above results, we can know that the proposed detection method is very effective. In the future, a detailed analysis of the convergence of detecting other attacks in real-world applications will be provided.

Author Contributions: Conceptualization , W.D. and Z.Y.; Methodology, P.X.; Formal Analysis, W.D. and P.Z. and P.X.; Data Curation, P.X.; Draft Preparation, Z.Y.; Project Administration, B.W.

Funding: The authors would like to thank support from the National Natural Science Foundation of China under Grant No. 61572514, Changsha Science and Technology Program under Grant K1705007, and Science and Technology Planning Project of Changsha under Grant ZD1601042.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Xu, J.; Wei, L.; Wu, W.; Wang, A.; Zhang, Y.; Zhou, F. Privacy-preserving data integrity verification by using lightweight streaming authenticated data structures for healthcare cyber-physical system. *Future Gener. Comput. Syst.* **2018**, in press. [CrossRef]
2. Tsiatsikas, Z.; Kambourakis, G.; Geneiatakis, D.; Wang, H. The Devil is in the Detail: SDP-Driven Malformed Message Attacks and Mitigation in SIP Ecosystems. *IEEE Access* **2019**, *7*, 2401–2417. [CrossRef]
3. Geneiatakis, D.; Kambourakis, G.; Lambrinoudakis, C.; Dagiuklas, T.; Gritzalis, S. A Framework for Protecting a SIP-based Infrastructure Against Malformed Message Attacks. *Comput. Netw.* **2007**, *51*, 2580–2593. [CrossRef]
4. Aziz, S.; Gul, M. A self learning model for detecting SIP malformed message attacks. In Proceedings of the 2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), Beijing, China, 26–28 October 2010; pp. 744–749.
5. Yuan, Y.; Yuan, H.; Ho, D.W.C.; Guo, L. Resilient Control of Wireless Networked Control System Under Denial-of-Service Attacks: A Cross-Layer Design Approach. *IEEE Trans. Cybern.* **2018**, 1–13. [CrossRef] [PubMed]
6. Deng, R.; Zhuang, P.; Liang, H. CCPA: Coordinated Cyber-Physical Attacks and Countermeasures in Smart Grid. *IEEE Trans. Smart Grid* **2017**, *8*, 2420–2430. [CrossRef]
7. Li, W.; Xie, L.; Deng, Z.; Wang, Z. False Sequential Logic Attack on SCADA System and Its Physical Impact Analysis. *Comput. Secur.* **2016**, *58*, 149–159. [CrossRef]
8. Mishra, S.; Li, X.; Pan, T.; Kuhnle, A.; Thai, M.T.; Seo, J. Price Modification Attack and Protection Scheme in Smart Grid. *IEEE Trans. Smart Grid* **2016**, *8*, 1864–1875. [CrossRef]
9. Rahman, M.S.; Mahmud, M.A.; Oo, A.M.T.; Pota, H.R. Multi-Agent Approach for Enhancing Security of Protection Schemes in Cyber-Physical Energy Systems. *IEEE Trans. Ind. Inform.* **2017**, *13*, 436–447. [CrossRef]
10. Han, Q.; Nguyen, P.; Eguchi, R.T.; Hsu, K.L.; Venkatasubramanian, N. Toward an Integrated Approach to Localizing Failures in Community Water Networks (DEMO). In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 2505–2506.
11. Wang, Y.; Zhang, Z.; Xu, L.; Gu, G. SRID: State Relation Based Intrusion Detection for False Data Injection Attacks in SCADA. In Proceedings of the European Symposium on Research in Computer Security, Wroclaw, Poland, 6–10 September 2014; pp. 401–408.
12. Gao, W.; Morris, T.; Reaves, B.; Richey, D. On SCADA control system command and response injection and intrusion detection. In Proceedings of the 2010 eCrime Researchers Summit, Dallas, TX, USA, 18–20 October 2010; pp. 1–9.
13. Liu, Y.; Ning, P.; Reiter, M.K. False data injection attacks against state estimation in electric power grids. In Proceedings of the 16th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 9–13 November 2009; pp. 21–32.

14. Liang, J.; Sankar, L.; Kosut, O. Vulnerability Analysis and Consequences of False Data Injection Attack on Power System State Estimation. *IEEE Trans. Power Syst.* **2016**, *31*, 3864–3872. [CrossRef]
15. Liu, X.; Li, Z.; Liu, X.; Li, Z. Masking Transmission Line Outages via False Data Injection Attacks. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 1592–1602. [CrossRef]
16. Tan, R.; Nguyen, H.H.; Foo, E.; Dong, X.; Yau, D.K.; Kalbarczyk, Z.; Iyer, R.K.; Gooi, H.B. Optimal False Data Injection Attack against Automatic Generation Control in Power Grids. In Proceedings of the 2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS), Vienna, Austria, 11–14 April 2016; pp. 1–10.
17. Chung, H.; Li, W.; Yuen, C.; Chung, W.; Zhang, Y.; Wen, C. Local Cyber-Physical Attack for Masking Line Outage and Topology Attack in Smart Grid. *IEEE Trans. Smart Grid* **2019**, *10*, 4577–4588. [CrossRef]
18. Zhang, Z.; Gong, S.; Dimitrovski, A.D.; Li, H. Time Synchronization Attack in Smart Grid: Impact and Analysis. *IEEE Trans. Smart Grid* **2013**, *4*, 87–98. [CrossRef]
19. Gacia, L.; Brasser, F.; Cintuglu, M.; Sadeghi, A. Hey, My Malware Knows Physics Attacking PLCs with Physical Model Aware Rootkit. In Proceedings of the Network & Distributed System Security Symposium, San Diego, CA, USA, 16–17 February 2017; pp. 1–15.
20. Esmalifalak, M.; Nguyen, N.T.; Zheng, R.; Han, Z. Detecting stealthy false data injection using machine learning in smart grid. In Proceedings of the 2013 IEEE Global Communications Conference (GLOBECOM), Atlanta, GA, USA, 9–13 December 2013; pp. 808–813.
21. Ozay, M.; Esnaola, I.; Vural, F.T.Y.; Kulkarni, S.R.; Poor, H.V. Machine Learning Methods for Attack Detection in the Smart Grid. *IEEE Trans. Neural Netw. Learn. Syst.* **2016**, *27*, 1773–1786. [CrossRef] [PubMed]
22. Vu, Q.D.; Tan, R.; Yau, D.K.Y. On applying fault detectors against false data injection attacks in cyber-physical control systems. In Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016; pp. 1–9.
23. Karnouskos, S. Stuxnet worm impact on industrial cyber-physical system security. In Proceedings of the IECON 2011—37th Annual Conference of the IEEE Industrial Electronics Society, Melbourne, Australia, 7–10 November 2011; pp. 4490–4494.
24. Wang, J.; Tu, W.; Hui, L.C.K.; Yiu, S.M.; Wang, E.K. Detecting Time Synchronization Attacks in Cyber-Physical Systems with Machine Learning Techniques. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 2246–2251.
25. Yan, J.; Tang, B.; He, H. Detection of false data attacks in smart grid with supervised learning. In Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada, 24–29 July 2016; pp. 1395–1402.
26. Pan, S.; Morris, T.; Adhikari, U. Developing a Hybrid Intrusion Detection System Using Data Mining for Power Systems. *IEEE Trans. Smart Grid* **2015**, *6*, 3104–3113. [CrossRef]
27. Pan, S.; Morris, T.; Adhikari, U. Classification of Disturbances and Cyber-Attacks in Power Systems Using Heterogeneous Time-Synchronized Data. *IEEE Trans. Ind. Inform.* **2015**, *11*, 650–662. [CrossRef]
28. Short, J.A.; Infield, D.G.; Freris, L.L. Stabilization of grid frequency through dynamic demand control. *IEEE Trans. Power Syst.* **2007**, *22*, 1284–1293. [CrossRef]
29. Mohsenian-Rad, A.H.; Leon-Garcia, A. Distributed Internet-Based Load Altering Attacks Against Smart Power Grids. *IEEE Trans. Smart Grid* **2011**, *2*, 667–674. [CrossRef]
30. Xun, P.; Zhu, P.D.; Maharjan, S.; Cui, P.S. Successive direct load altering attack in smart grid. *Comput. Secur.* **2018**, *77*, 79–93. [CrossRef]
31. Xun, P.; Zhu, P.; Hu, Y.; Cui, P.; Zhang, Y. Command Disaggregation Attack and Mitigation in Industrial Internet of Things. *Sensors* **2017**, *17*, 2408. [CrossRef] [PubMed]