Deception-Resistant Stochastic Manufacturing for Automated Production Lines

Zeyu Yang
Zhejiang University
zeyuyang@zju.edu.cn

Hongyi Pu
Zhejiang University
hongyipu.zju@gmail.com

Liang He
University of Nebraska-Lincoln
lhe20@unl.edu

Chentao Yao
Zhejiang University
chengtao.yao@outlook.com

Jianying Zhou
Singapore University of Technology and Design
jianying_zhou@sutd.edu.sg

Peng Cheng
Zhejiang University
lunarheart@zju.edu.cn

Jiming Chen
Zhejiang University, Hangzhou Dianzi University
cjm@zju.edu.cn

ABSTRACT

The advancement of Industrial Internet-of-Things (IIoT) magnifies the cyber risk of automated production lines, especially to deception attacks that tamper with the monitoring data to prevent the manipulated operation of production lines from being detected. To address this issue, we propose `Stochastic Manufacturing` (`StoM`), a new paradigm of manufacturing that is resistant to deception by design. `StoM` voids the foundation of deception attacks — i.e., the highly predictable operation data due to the cyclical manufacturing process — by injecting controlled stochasticity into the operation of production lines without degrading manufacturing efficiency or quality. `StoM` then examines if this stochasticity can be observed from the operation data and triggers an alarm of deception attack if not. We have experimentally evaluated `StoM` on two production line platforms, showing `StoM` to detect deception attacks with a detection rate exceeding 99.1%, a false alarm rate below 0.1%, and a latency of less than 1.2 manufacturing cycles. Our empirical analysis also shows that it is highly impractical for attackers to spoof the controlled stochasticity.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**.

KEYWORDS

Production Line Manufacturing; Controlled Stochasticity; Deception Attacks

ACM Reference Format:
Zeyu Yang, Hongyi Pu, Liang He, Chentao Yao, Jianying Zhou, Peng Cheng, and Jiming Chen. 2024. Deception-Resistant Stochastic Manufacturing for Automated Production Lines. In *The 27th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2024), September 30–October 02, 2024, Padua, Italy*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3678890.3678896

RAID 2024, September 30–October 02, 2024, Padua, Italy

ACM ISBN 979-8-4007-0959-3/24/09
https://doi.org/10.1145/3678890.3678896

1 INTRODUCTION

Automated production lines, essential to modern manufacturing processes [30, 42], have revolutionized industries ranging from aircraft to automobiles and beyond [5, 22, 40]. These production lines enhance manufacturing efficiency by systematically organizing machines, including industrial robots, to perform repeatable operations [2, 54]. The repeatable operation in each manufacturing cycle makes the resultant operation data highly predictable, laying the foundation for *deception attacks* [49] — attacks that manipulate the manufacturing process and replace real-time monitoring data with historical (and normal) operation data, deceiving the Supervisory Control and Data Acquisition (SCADA) into believing that the system is operating normally.

As a real-life deception attack, Stuxnet [23] destroyed thousands of centrifuges by (i) intruding into the centrifuge system through the compromise of an engineer's computer (and USB sticks), (ii) manipulating the centrifuge's rotating speed via code injection, and (iii) replaying previously recorded normal data to the SCADA. The risk of deception attacks is magnified further with the advancement of Industrial Internet-of-Things (IIoT), where automated production lines are increasingly networked. This trend voids the ingrained conviction that manufacturing systems are designed to work in isolation. Additionally, operation data becomes easier to eavesdrop on and manipulate [16, 50, 58].

To protect automated production lines from attacks, many Intrusion Detection Systems (IDSs) have been developed based on the SCADA-received monitoring data [26, 29, 45, 62, 67, 69]. For example, Urbina et al. [63] designed an IDS that checks the real-time operation data of ultra-filtration tanks with a physical model to detect potential intrusions. Feng et al. [25] and Adepu et al. [3] identified system invariants of water treatment plants and checked whether the received operation data conflicts with these invariants. However, these IDSs assume the authenticity of the SCADA-received operation data on which the intrusion detection is built, i.e., to serve as a root-of-trust, which does not hold for deception attacks that tamper with the operation data.

Unlike traditional approaches that address cyber attacks by patching IDSs, this paper fundamentally disables deception attacks by introducing a new manufacturing paradigm called *Stochastic Manufacturing* (StoM). StoM disrupts the foundation of deception attacks — namely, the predictable operation of production lines — by injecting controlled stochasticity into the manufacturing process. Specifically, StoM injects randomized control commands into the operation of automated production lines and then cross-validates the empirically observed operation data with the expected system behavior. A mismatch flags a deception attack. StoM does not require any hardware retrofits and can thus be deployed to all existing production lines as a software module of SCADA.

The challenges in the design of StoM are two-fold. First, StoM's randomization of the production line operation must not degrade manufacturing quality or efficiency. StoM addresses this challenge in three steps: (i) identifying control commands that only impact the time between consecutive manufacturing steps, not the processing of an object; (ii) modeling the dependencies between these feasible control commands and the resulting operation time for each manufacturing step; (iii) jointly randomizing the control commands of all manufacturing steps to ensure the total operation time (and hence the manufacturing efficiency) is kept at its original level. Second, when validating the observed manufacturing process, StoM cannot assume any knowledge about the dynamics of manufacturing machines, that is, how the machines' onboard sensor readings change with given control commands. This information often requires proprietary details from equipment manufacturers (such as ABB) and is difficult to identify based solely on operation data [8]. StoM tackles this challenge by validating the observed manufacturing process in the time domain: identifies the time to complete each manufacturing step based on the operation data and cross-validates it with the expected timings.

We have implemented and evaluated StoM on two platforms of automated production lines: the Automated Automobile Manufacturing (AVM) and the Automated Bottle Manufacturing (ABM). The experiments encompassed more than 12,500 manufacturing cycles, demonstrating StoM's ability to (i) detect deception attacks with a detection rate exceeding 99.1%, a false alarm rate below 0.1%, and a latency of less than 1.2 manufacturing cycles, and (ii) maintain manufacturing efficiency with a precision of 97.9%.

In summary, this paper makes the following contributions:

- Revealing a fundamental vulnerability of automated production lines to deception attacks, i.e., the predictable operation;
- Designing StoM, a new paradigm of production line operation that is resistant to deception by design;
- Demonstrating StoM's security and effectiveness via extensive field tests.

The rest of the paper is structured as follows. We discuss the literature in Sec. 2. The background of automated production lines and the motivational observation inspiring StoM are presented in Sec. 3 and Sec. 4, respectively. We introduce the detailed design and implementation of StoM in Sec. 5. The evaluation results of StoM and the empirical analysis of spoofing the controlled stochasticity are presented in Sec. 6. The applicability of StoM is discussed in Sec. 7. Finally, we conclude in Sec. 8.

2 RELATED WORK

The real-life attack *Stuxnet* [23] has raised significant research attention to deception attacks [11, 12, 28, 65, 67]. For example, Pu et al. [49] replayed the normal operation of the industrial robot to deceive human operators that the victim robot is operating normally. Chung et al. [18] injected the faked operation data of surgical robots to mislead the actions of doctors. Davide et al. [52] manipulated the displayed states of industrial robots to deceive human operators into falsely perceiving the robotic arm as safe for approaching. Yang et al. [66] stealthily manipulated the control commands by exploiting the mismatched frequencies between SCADA monitoring and process operation.

Table 1: Comparison between StoM and existing IDSs.

	Resistance to Deception Attack	Requiring No System Dynamics	Application Platforms	Validated Empirically
[29]	✓	–	Production lines	✓
[45]	–	✓	Production lines	–
[51]	–	✓	Production lines	✓
[69]	–	✓	Production lines	✓
[10]	–	✓	Process Control	–
[25]	–	✓	Process Control	✓
[67]	–	✓	Process Control	✓
[63]	–	–	Process Control	✓
[3]	–	✓	Water Plant	✓
[26]	–	–	Power Grid	✓
[70]	✓	–	Power Grid	–
[17]	–	–	Robotic Vehicles	✓
[53]	✓	–	Robotic Vehicles	✓
[24]	✓	–	LTI Systems†	–
StoM	✓	✓	Production lines	✓

† LTI (i.e., Linear Time-Invariant) System defines a model of general systems.

A number of intrusion detection systems (IDSs) have been proposed for production lines to mitigate the potentially life-threatening impacts of attacks. Intuitively, manipulated system operations can be identified by checking if the observed system states (e.g., robot's angle) agree with the legitimate system models. Following this idea, Narayanan et al. [45] modeled the operation of industrial robots by exploring their repetitive behaviors. Pu et al. [51] verified the operation of industrial robots using regular power consumption. Zhang et al. [69] diagnosed industrial robots by exploring the temporal relations among their different actions. To defend against more advanced attacks that can manipulate the monitoring data by replaying historical logs, Ghaeini et al. [29] applied an attestation strategy to authenticate industrial robots based on their predictable operating traces in response to specific challenging signals. However, the dynamics of industrial robots are required to design proper challenging signals, which are not available for all automated production lines.

It is possible to extend and apply the IDSs proposed for other platforms to production lines [39]. For example, Choi et al. [17] and Quinonez et al. [53] built physical dynamic models of unmanned vehicles; Aoudi et al. [10] identified the chemical process using the spectrum of time series; a state-space dynamic model of the ultra filtration tank is constructed in [63]; the invariants of a water

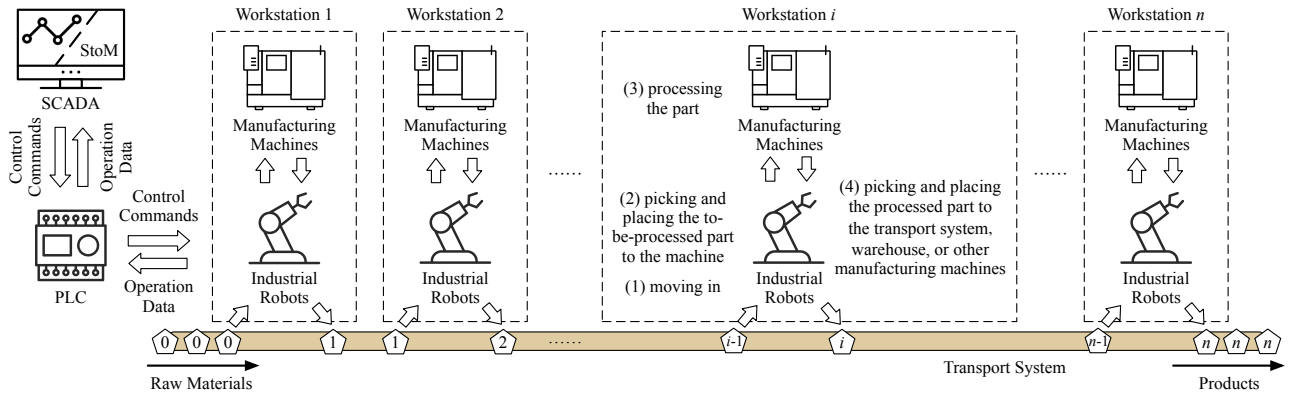


Fig. 1. Typical operation of automated production line.

treatment plant are identified in [3, 25]; a control graph model for the distillation tower is constructed in [67]; the time pattern of communication for power substations is explored in [26]; a command prediction based detection/localization is proposed in [68]. However, all these IDSs assume the authenticity of the monitoring data upon which they are built (i.e., to serve as a root of trust), which may not hold for deception attacks. To bridge this gap, [24, 70] explored moving target defense (MTD)-based attack detection based on the *a priori* knowledge of the dynamics of physical processes. This knowledge is, however, unavailable and difficult to identify for the automated production lines.

Table 1 summarizes the aforementioned IDSs and compares them with StoM, emphasizing StoM's unique advantage: protecting production lines from deception attacks by randomizing their operation without requiring any knowledge of system dynamics.

Also note that different from the "stochastic manufacturing" in [59], which optimizes the manufacturing process under uncertainties (e.g., electromagnetic disturbances), StoM exploits the actively injected (and controlled) stochasticity in the manufacturing process to enhance system security.

```
MODULE MainModule
// Declaration and initialization of commands
VAR num speed{4} := [50,50,50,50];
VAR num waiting_time{4} := [2,2,2,2];
VAR num acceleration{4} := [100,100,100,100];
VAR num position{4} := [p1,p2,p3,p4];

PROC main()
//Cyclic Manufacturing
WHILE TRUE
    //Updating commands
    update commands;
    //Setting robot speed as speed{1}*max_speed/100
    VelSet speed {1}, max_speed;
    //Setting robot acceleration as acceleration
     {1}*max_acceleration/100
    AccSet acceleration{1}, max_acceleration;
    //Picking up the object from position{1}
    pick_up (position {1});
    WaitTime waiting_time {1}; //Waiting
     waiting_time{1}
    ...
ENDWHILE
```

Fig. 2. Exemplary operation code of AVM's ABB robot.

3 PRELIMINARIES

We first introduce the background of automated production lines and the risk of intrusions thereof, using our scaled-down but fully operational platforms.

3.1 Automated Production Line

Different forms of automated production lines have been discussed and deployed in practice [2, 5, 6, 22, 37, 41, 43]. Fig. 1 summarizes these variations and depicts a generalized model, consisting of workstations, programmable logic controllers (PLCs), a transport system, a SCADA, and other auxiliary devices [30]. Raw materials are transferred by the transport system and fed to different workstations for processing. The completed products leave the production line from the last workstation. The time interval between completing two consecutive products is defined as the *cycle time* of the automated production line, determining the manufacturing efficiency.

Workstation is the fundamental unit of a production line, responsible for performing a specific manufacturing operation. A workstation consists of industrial robots, manufacturing machines (such as milling CNCs), and conveyors, which collaborate sequentially or in parallel. In a typical cycle of sequential operation, there are four execution steps: (i) a to-be-processed part moves into the workstation via the transport system; (ii) the industrial robot of the workstation picks up the part and places it in the manufacturing machine; (iii) the manufacturing machine processes the part; (iv) the industrial robot picks up the processed part from the manufacturing machine and places it to the transport system, warehouse, or other manufacturing machines in the workstation. The workstation might stop for a while each time one of the above execution steps is completed (e.g., to wait for other machines to be ready). For ease of description, we refer to these execution steps and the subsequent waiting period as *operation steps*. It is important to note that the operation time of each workstation is consistent with the cycle time of the production line. Parallel operation in production lines represents a variant of sequential operation, incorporating a primary sequential operation and one or more cooperative parallel sub-operations.

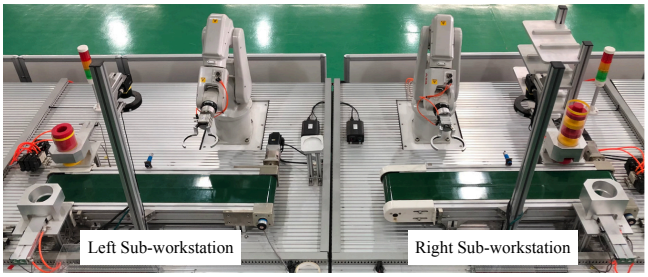


Fig. 3. Testbed of Automated Vehicle Manufacturing.

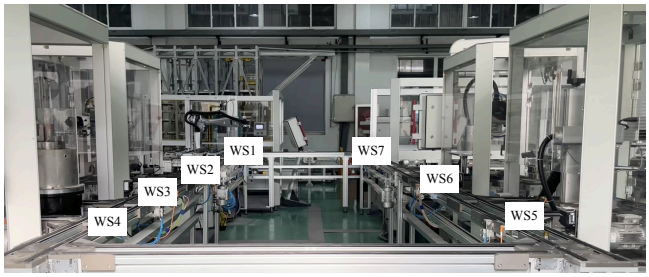


Fig. 4. Testbed of Automated Bottle Manufacturing.

PLCs (Programmable Logic Controllers) govern the operations of automated production lines by processing signals from various devices, such as CNCs and industrial robots, through the industrial network (e.g., PROFINET). For instance, a PLC receives a signal from a CNC indicating the completion of part processing and instructs the industrial robot to pick up the processed part. Additionally, PLCs are responsible for gathering and transmitting operation data from the production line to the SCADA system.

Transport system moves objects between workstations along a pre-deployed delivery path using conveyor belts, rollers, skate wheels, and similar components. Note that the operation of the transport system is controlled by the PLCs.

SCADA is a software system that supervises and oversees the operation of production lines. SCADA controls the production lines by sending specific instructions to the devices thereof (e.g., PLCs, robots, CNCs), monitors the production lines based on the operation data received from PLCs, and detects anomalies in the manufacturing process. Fig. 2 gives an exemplary code of the robot in AVM: the robot updates commands (i.e., Line 12) once receiving instructions from the SCADA and operates accordingly (i.e., Lines 14-19).

3.2 Testbeds: AVM and ABM

According to the abstracted system model in Fig. 1, we build two production line testbeds: Automated Vehicle Manufacturing (AVM) and Automated Bottle Manufacturing (ABM).

As shown in Fig. 3, AVM consists of one workstation (one left sub-workstation and one right sub-workstation). Each sub-workstation has an ABB IRB120 robot, a conveyor belt, and an Omron CP1H PLC. AVM supports the sequential operation of the right sub-workstation (i.e., *"welding"* and *"warehousing"*) and parallel operation of both sub-workstations (i.e., *"assembly"*):

- *"welding"*: picking up and moving the object in the air following a square trajectory to simulate the welding operation;
- *"warehousing"*: picking and placing the object in the warehouse;
- *"assembly"*: assembling two objects into a cylinder by cooperatively operating the two sub-workstations.

The second testbed ABM produces bottles using 7 workstations (see Fig. 4). These workstations are equipped with multiple conveyor belts, Siemens S7-1200/1500 PLCs, COMAU NJ60/Racer7/Rebel-S6 robots, and DMG MORI HV-6/FAVGOL DT300 CNCs, supporting the operation of *"picking and placing"*, *"turning and milling"*, *"inspection"*, *"craving"*, *"inkjeting"*, *"assembly"*, and *"warehousing"*.

Both AVM and ABM are equipped with a SCADA developed with >3, 000 lines of C# code, consisting of three modules: a GUI facilitating the real-time control/monitoring of the production line, an intrusion detector detecting the potential anomalies based on the received operation data [45], and the software implementation of `StoM`.

3.3 Threat Model

We consider adversaries who want to mount deception attacks on production lines, i.e., damaging the manufacturing process without being detected by the SCADA, the sneakiness of which incurs more risk to system operation than traditional manipulation attacks [19, 57]. Adversaries achieve this with the following abilities similar to *Stuxnet* [23], which we have empirically corroborated on AVM and are also commonly adopted in the literature [8, 18, 28, 51, 52, 67].

Ability-1: Invading the industrial network remotely. The industrial network is invadable from the public Internet due to incorrectly configured protection strategies [21, 38] and the frequently exposed zero-day vulnerabilities [13, 33]. For example, adversaries can first invade the corporate network [46, 61] to gain control of the office equipment thereof (e.g., computers, printers) by exploiting vulnerabilities such as MS08-067 [34] and MS17-010 [31]. Then, by pivoting through the compromised equipment, adversaries can gain access to the production line [15, 23].

Ability-2: Manipulating the operation of production lines via code injection. Controllers, such as PLCs, in the production line support the online reprogramming but via weakly authenticated communication [14, 20, 47]. Once connected with targeted controllers, adversaries can upload the malicious code using the development software (e.g., *STEP 7* for Siemens PLCs [7] and *RobotStudio* for ABB robots [1]) without authorization. For example, Stuxnet uploaded the modified code to PLCs by infecting the *STEP 7* development software [23]; PIDS used the *RobotStudio* to upload malicious code to the robot directly [51].

Ability-3: Modifying the operation data reported to SCADA. Recent attacks and exposed malware proved that SCADA systems are not secure [4]. By eavesdropping and manipulating the communication packets, adversaries can send the deceived operation data to SCADA using *Ettercap* [48] or *NetfilterQuque* [27]. For example, PIDS tampered with the operation data by hijacking communication packets sent to the SCADA [51]; Stuxnet manipulated the operation data stored in the PLC's memory by injecting malicious code [23].

Table 2: The cyclical "welding" operation leads to similar operation data per cycle in AVM.

Line Id.	Control Commands			Operation Data	
	Robot Speed (%)	Conveyor Speed (%)	Waiting Time (s)	R^2-Correlation	Cycle Time (s)
1	50	50	2	0.99	27.2 ± 0.1
2	100	60	5	0.98	37.4 ± 0.2
3	90	70	4.5	0.98	34.8 ± 0.2
4	80	80	4	0.98	32.6 ± 0.2
5	70	90	3.4	0.98	30.0 ± 0.2
6	60	100	3.3	0.98	30.0 ± 0.2
7	50	50	2.5	0.98	30.0 ± 0.2
8	40	40	1.4	0.98	30.0 ± 0.3

Empirical Validation of Adversary Abilities. We have empirically corroborated the feasibility of the above adversary abilities and mounted deception attacks to AVM with about 600 lines of Python code. Specifically, we use (i) *metasploit* [36] to remotely invade the industrial network where AVM is deployed, (ii) *arpspoof* [60] to intercept the communication packets and *NetfilterQueue* [27] to eavesdrop and modify the operation data in the packets, and (iii) ABB robot's Software Development Kit (SDK) [55] to upload malicious code to AVM. Note that it is challenging for adversaries to access the production line system in person, and thus, we assume that adversaries cannot deploy any additional devices in the production line to assist the attack. Please see [9] for a demo video of such a deception attack.

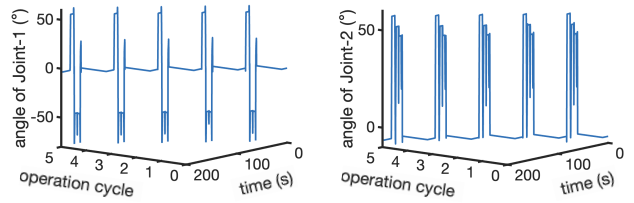
4 MOTIVATING EXAMPLE

The fact that automated production lines commonly carry out manufacturing operations *cyclically* lays the foundation of deception attacks. To demonstrate this, we operate AVM to perform 600 times the *"welding"* operation, with the speeds of the robot/conveyor belt set as 50% of the maximum speed and the waiting time after each execution step as 2s. Fig. 5 depicts the thus-collected operation data, where the angles of robot joints behave similarly in each operation cycle. The average R^2-correlation [32] among the operation data of different cycles is 0.99[1] (see Line 1 of Table 2). We have repeated the above experiments with different control commands, each 50 times. Table 2 summarizes the obtained results, demonstrating, again, the similar operation data in each cycle. These observations corroborate that the cyclical operation of production lines leads to highly similar operation data in each cycle, laying the foundation of deception attacks — adversaries can record the normal operation data and then replay them to deceive the SCADA into concluding a normal manufacturing process is underway, even when the production line is actually being manipulated.

The above observations also reveal two other important facts: (i) each setting of control commands leads to a determined cycle time, with an average standard deviation of 0.59% (refer to Lines 1-8 in Table 2); (ii) different combinations of control commands achieve the same cycle time and manufacturing efficiency (refer to Lines 5-8 in Table 2). The first fact motivated us to detect deception attacks by verifying dependencies between control commands and the resulting operation time. The second fact highlights the opportunity to randomize control commands of each manufacturing machine while maintaining the overall cycle time.

Note that the commands we explored (i.e., speed and waiting time) do not affect how the object is processed. Varying these commands will not impact the production quality.

[1]A close-to-1 R^2 indicates a higher similarity among different operations.



(a) Robot's angle of Joints-1 (b) Robot's angle of Joints-2

Fig. 5. The exemplary cyclical operation data in the platform of AVM: the robot's joints perform similarly among different operation cyclic.

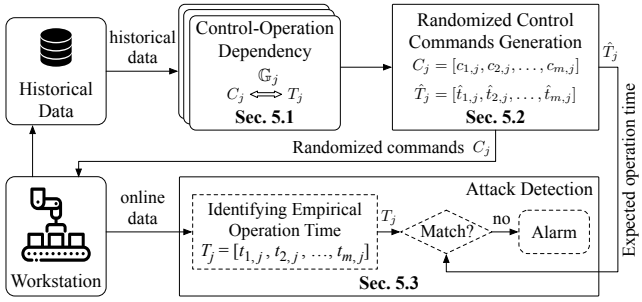
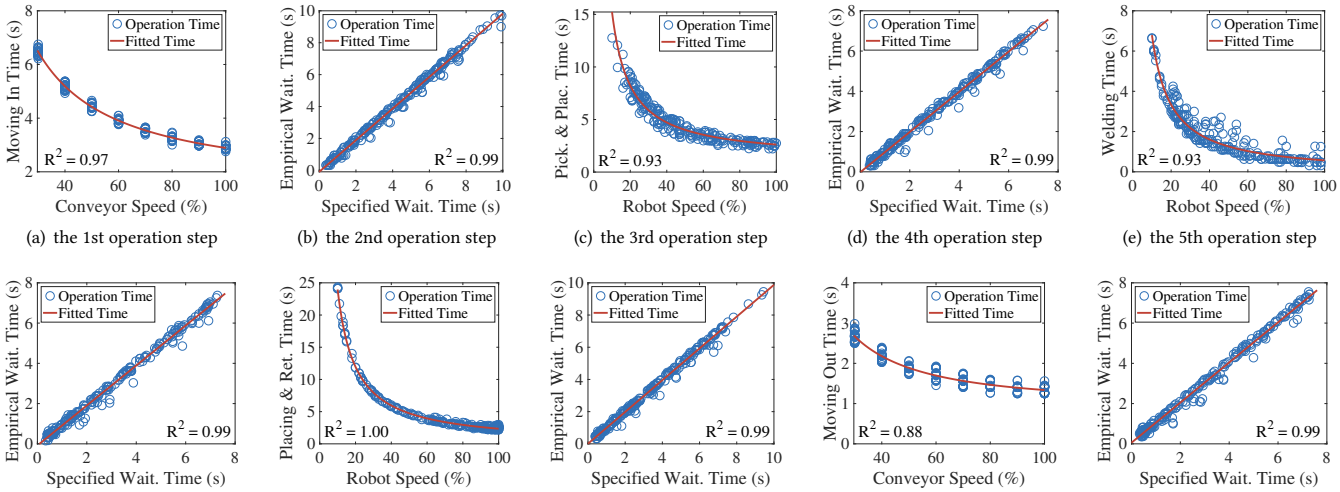


Fig. 6. Workflow of StoM.

5 DESIGN OF STOM

Building upon the above insights, we have developed a groundbreaking manufacturing paradigm called StoM, which fundamentally neutralizes deception attacks by disrupting the cyclical operation of production lines through randomized control commands. In every manufacturing cycle, StoM generates and sends randomized commands to the manufacturing devices in the production line, specifying their operations for the subsequent cycle, and detects deception attacks by examining if the observed operation data of the production line matches the randomized commands — replaying pre-recorded normal operation data will not deceive StoM. StoM ensures that the same cycle time and quality are maintained when generating the randomized commands. We deliver StoM as a software module of the SCADA system, which can be deployed without requiring any hardware retrofits.

Fig. 6 depicts an overview of StoM, consisting of three steps: capturing the dependency between control commands and operation data, generating randomized control commands, and detecting deception attacks at runtime. We will next explain each of these steps in detail.



(a) the 1st operation step (b) the 2nd operation step (c) the 3rd operation step (d) the 4th operation step (e) the 5th operation step

(f) the 6th operation step (g) the 7th operation step (h) the 8th operation step (i) the 9th operation step (j) the 10th operation step

Fig. 7. The dependency between control commands and the resultant operation time can be described by Eq. (2).

5.1 Identify Control-Operation Dependency

StoM's randomization of the manufacturing process is built on the dependency between the issued control commands and the concomitantly resultant operation time. Let us consider that a robot in the production line receives the commands specifying its maximum speed v and waiting time t_w. The robot moves for distance d by accelerating from speed 0 to v with acceleration a, moving with v, decelerating from v to 0 with acceleration $-a$, and waiting for time t_w. The time t to complete this operation is:

$$t = \frac{v}{a} + \frac{d - 2v^2/(2a)}{v} + \frac{v}{a} + t_w = d \cdot v^{-1} + \frac{1}{a} \cdot v + t_w, \quad (1)$$

showing the operation time can be described in the form of:

$$t = G(c) = \omega_1 \cdot c^{-1} + \omega_2 + \omega_3 \cdot c, \quad (2)$$

where c is the control command of a given operation step specifying the speed of a device or the waiting time after an execution step, and the function $G(\cdot)$ with coefficients of $\{\omega_1, \omega_2, \omega_3\}$ denotes the dependency between command c and operation time t. We next use 200 cycles of AVM's *"welding"* operation to empirically examine this dependency. Specifically, we vary the speed of different execution steps, i.e., {moving-in, picking-and-placing, welding, placing-and-returning, moving-out}, as {30~100%, 10~100%, 10 ~ 100%, 10~100%, 30~100%} of the maximum speed, and specify the waiting time after each of the above execution step as {0~10s, 0~7.5s, 0~7.5s, 0~10s, 0~7.5s}. Fig. 7 shows the relationship between the control commands and the corresponding time to complete different operation steps, which can be well fitted using the function $G(\cdot)$ in Eq. (2): the average (or worst case) R^2 of 0.97 (or 0.88) corroborates the high goodness of fit.

StoM identifies function $G(\cdot)$ based on the historical control commands. Let us consider the historical data with n manufacturing cycles, each containing m operation steps. The dependency between historical control commands and the time to complete the ith operation step ($i \in \{1, 2, \cdots, m\}$) can be written as:

$$T_i = X_i \cdot \omega_i, \quad (3)$$

where $T_i = [t_{i,1}, t_{i,2}, \ldots, t_{i,n}]^{\mathrm{T}}$ denotes the operation time vector in the ith operation step, $\omega_i = [\omega_{1,i}, \omega_{2,i}, \omega_{3,i}]^{\mathrm{T}}$ denotes the coefficient vector of function $G_i(\cdot)$ in the ith operation step, and X_i is the control command matrix in the ith operation step:

$$X_i = \begin{bmatrix} c_{i,1}^{-1} & 1 & c_{i,1} \\ c_{i,2}^{-1} & 1 & c_{i,2} \\ \vdots & \vdots & \vdots \\ c_{i,n}^{-1} & 1 & c_{i,n} \end{bmatrix}. \quad (4)$$

StoM then uses the least squares method [35] to identify the coefficient vector ω_i in $G_i(\cdot)$:

$$\omega_i = (X_i^{\mathrm{T}} X_i)^{-1} X_i^{\mathrm{T}} T_i. \quad (5)$$

The empirical results show that 3 cycles of operation data resultant from different commands (i.e., $n \geq 3$) are sufficient for StoM to identify the function $G_i(\cdot)$ in each operation step, which can be further updated/refined using the newly collected operation data at runtime, as we will elaborate in Sec. 6.3.

5.2 Generate Randomized Commands

In practice, manufacturing machines rarely operate at full speed [37, 41], offering StoM the chance of adjusting the operation speed of each manufacturing step without affecting the overall cycle time. With the control-operation dependency captured in Sec. 5.1, StoM injects the controlled stochasticity to the production line by issuing, in each manufacturing cycle of a workstation, randomized control commands specifying the speed and waiting time of devices thereof. Note that StoM will not randomize the acceleration/position of devices (as defined in Lines 5-6 of Fig. 2), because (i) only part of manufacturing machines support the online management of acceleration, and (ii) the changing of positions may degrade the product quality. Also, in view of the fact that different workstations in a production line operate independently, StoM generates commands for each workstation individually.

Algorithm 1 Randomization of Sequential Operations

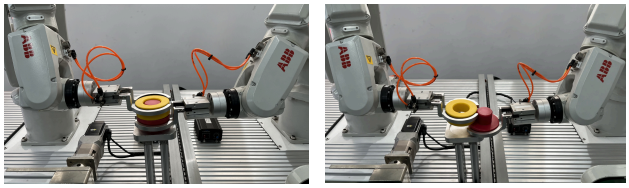
Input: T, $\mathbb{L} = \{L_1, L_2, \cdots, L_m\}$, $\mathbb{U} = \{U_1, U_2, \cdots, U_m\}$, $\mathbb{G} = \{G_1, G_2, \cdots, G_m\}$;
Output: $C_j = \{c_{1,j}, c_{2,j}, \cdots, c_{m,j}\}$, $\hat{T}_j = \{\hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m,j}\}$;
$T_{\min} = \sum_{i=1}^{m} L_i$;
$\Delta T = T - T_{\min}$; *// total time that can be randomized*
for $(i = 1; i \leqslant m; i++)$ **do**
  $a_i = U_i - L_i$; *// time that can be randomized for each step*
end for
$\mathbb{I} = \arg\_sort(a_1, a_2, \cdots, a_m)$;
for $(i = 0; i < m - 1; i++)$ **do**
  $\Delta t = \text{random}\left(0, \min\left(a_{\mathbb{I}[i]}, \Delta T\right)\right)$;
  $\hat{t}_{\mathbb{I}[i],j} = L_{\mathbb{I}[i]} + \Delta t$; *// the randomized time for each step*
  $\Delta T = \Delta T - \Delta t$;
end for
$\Delta t = \Delta T$; *// the randomized time for the last step*
$\hat{t}_{\mathbb{I}[m-1],j} = L_{\mathbb{I}[m-1]} + \Delta t$;
for $(i = 1; i \leqslant m; i++)$ **do**
  $c_{i,j} = G_i^{-1}\left(\hat{t}_{i,j}\right)$; *// the randomized control commands*
end for
return C_j and $\hat{T}_j$.

5.2.1 Randomization of Sequential Operations. Denote $C_j = \{c_{1,j}, c_{2,j}, \cdots, c_{m,j}\}$ as the m commands to configure an m-step operation of a workstation at the jth manufacturing cycle and $\hat{T}_j = \{\hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m,j}\}$ as the corresponding operation time. StoM randomizes C_j and estimates $\hat{T}_j$ as:

$$\begin{aligned} \text{randomizing} \quad & C_j = \{c_{1,j}, c_{2,j}, \cdots, c_{m,j}\}, \\ \text{s.t.} \quad & \hat{t}_{i,j} = G_i(c_{i,j}), && (6) \\ & \sum_{i=1}^{m} \hat{t}_{i,j} = T, && (7) \\ & \hat{t}_{i,j} \in [L_i, U_i], && (8) \end{aligned}$$

where T is the expected cycle time of workstation; L_i and U_i are the lower and upper bounds of the time to complete the ith operation step; $G_i(\cdot)$ is the identified function of Eq. (2). Note that Eqs. (7) and (8) are two requirements that the randomized command generation must satisfy: (i) T has to be maintained to ensure the same manufacturing efficiency; (ii) the time to complete each operation step has to be lower-bounded (or upper-bounded) by L_i (or U_i), which depends on the fastest (or slowest) execution speed or the smallest (or longest) waiting time of devices in the workstation.

Alg. 1 summarizes StoM's randomized command generation for sequential operations, following the basic idea of (i) identifying the shortest time to complete all operation steps based on Eq. (8), (ii) stochastically slowing them down — in ascending order of the deviation between U_i and L_i — to ensure that the thus-resultant time to complete operation steps satisfies Eq. (7), and (iii) converting the expected operation time to control commands using Eq. (6). Note that StoM will have non-unique solutions for the randomized control commands C_j, as long as not all manufacturing machines operate at full speed — a common practice in production lines. Because the workstation will not operate according to the received commands until the next manufacturing cycle, StoM has enough time to generate the randomized commands C_j.



(a) normal assembly (b) collision of robots

Fig. 8. The left robot needs to arrive at the assembly position later than the right robot to avoid collision.

5.2.2 Randomization of Parallel Operations. In addition to the operating time constraints imposed by sequential operations, the command randomization for parallel operations must conform to the physical/space constraints between the sequential and parallel sub-operations. Taking AVM's "*assembly*" operation shown in Fig. 8 as an example, the robot on the right side has to arrive at the assembly position earlier than the left robot; otherwise, they will collide when the left robot moves down to assemble with the right robot. To avoid introducing physical conflicts (e.g., collision) to the production lines, we add additional physical/space constraints into StoM's command randomization. Let us consider a parallel operation with $(m + r)$ operation steps (including a main sequential operation with m steps and additional parallel sub-operations with a total of r steps). StoM generates randomized $C_j = \{c_{1,j}, c_{2,j}, \cdots, c_{m+r,j}\}$ for the jth manufacturing cycle by:

$$\begin{aligned} \text{randomizing} \quad & C_j = \{c_{1,j}, c_{2,j}, \cdots, c_{m+r,j}\}, \\ \text{s.t.} \quad & \hat{t}_{i,j} = G_i(c_{i,j}), && (9) \\ & \sum_{i=1}^{m} \hat{t}_{i,j} = T, && (10) \\ & \hat{t}_{i,j} \in [L_i, U_i], && (11) \\ & \forall k \in [1, q], \ \sum_{i=1}^{m+r} a_{i,k} \cdot \hat{t}_{i,j} > 0, && (12) \end{aligned}$$

where $a_{i,k} \in \{-1, 0, 1\}$ specifies the physical/space constraints for parallel operations according to the operation manual, and q denotes the total number of physical/space constraints.

Still taking AVM's "*assembly*" operation as an example, the right and left robots arrive at the assembly position via steps of $\{1, 2, 3\}$ and $\{11, 12, 13\}$, respectively, and taking a time of $\sum_{i=1}^{3} \hat{t}_{i,j}$ and $\sum_{i=11}^{3} \hat{t}_{i,j}$. To avoid the collision, the right robot needs to arrive earlier than the left robot, i.e., $\sum_{i=1}^{3} \hat{t}_{i,j} < \sum_{i=11}^{13} \hat{t}_{i,j}$, and thus the $a_{i,k}$s in Eq. (12) can be written as:

$$a_{i,k} = \begin{cases} -1, & i \in \{1, 2, 3\} \\ 0, & i \in \{4, 5, 6, 7, 8, 9, 10, 14, 15, 16, 17, 18\} \\ 1, & i \in \{11, 12, 13\} \end{cases}. \quad (13)$$

Alg. 2 summarizes StoM's randomized command generation for parallel operations, following the basic idea of: (i) randomly generating the control commands for the main sequential operation using Alg. 1, (ii) randomly generating the control commands for the parallel sub-operation steps that should not be previously assessed and marked as infeasible control commands, and (iii) determining if the generated control commands satisfy Eq. (12) — the control commands will be marked as infeasible and then generated again if not. Note that the manufacturing process still operates at the

Algorithm 2 Randomization of Parallel Operations

Input: T, $\mathbb{L} = \{L_1, L_2, \cdots, L_{m+r}\}$, $\mathbb{U} = \{U_1, U_2, \cdots, U_{m+r}\}$, $\mathbb{G} = \{G_1, G_2, \cdots, G_{m+r}\}$, $a_{i,k} (i \in [1, m+r], k \in [1, q])$;
Output: $C_j = \{c_{1,j}, c_{2,j}, \cdots, c_{m+r,j}\}$, $\hat{T}_j = \{\hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m+r,j}\}$;

flag = false;
while not flag **do**
  $\{c_{1,j}, c_{2,j}, \cdots, c_{m,j}\}$, $\{\hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m,j}\}$ = Alg. 1;
  for $(i = 1; i \leqslant r; i++)$ **do**
    $\hat{t}_{m+i,j}$ = random (L_{m+i}, U_{m+i});
  end for
  flag = true;
  for $(k = 1; k \leqslant q; k++)$ **do**
    if $\sum_{i=1}^{m+r} a_{i,k} \cdot \hat{t}_{i,j} \leqslant 0$ **then**
      flag = false; // *physical/space constrains not satisfied*
    end if
  end for
end while
for $(i = 1; i \leqslant r; i++)$ **do**
  $c_{m+i,j} = G_{m+i}^{-1}(\hat{t}_{m+i,j})$; // *the randomized control commands*
end for
return C_j and $\hat{T}_j$.

expected efficiency even without generating newly randomized control commands. We have also empirically examined how many trials of command generation are needed before a valid solution can be identified. Fig. 9 shows that StoM only needs an average of 1.7 trials to generate the commands for AVM's "*assembly*" operation and the first trial successes with a chance of 60%.

5.2.3 Security Analysis of StoM's Randomness. Next, we examine the possibility of adversaries guessing the randomized control commands for sequential operations, which also forms the security foundation for randomization of parallel operations. We measure such possibility using the entropy of the operation time randomization [64]: a larger entropy indicates a higher level of randomness and hence more secure, as it will be more challenging for adversaries to guess the corresponding randomized control commands.

Specifically, the entropy of the expected operation time of an m-step sequential operation can be calculated as:

$$H = -\int_{L_1}^{U_1}\int_{L_2}^{U_2}\cdots\int_{L_{m-1}}^{U_{m-1}} p\left(\hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m,j}\right) \times \log_2\left(p\left(\hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m,j}\right)\right) \mathrm{d}\hat{t}_{\mathrm{m-1,j}} \mathrm{d}\hat{t}_{\mathrm{m-2,j}} \cdots \mathrm{d}\hat{t}_{\mathrm{1,j}},$$

where $p(\hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m,j})$ denotes the probability distribution of the expected operation time:

$$\begin{aligned} & p\left(\hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m,j}\right) \\ =& p\left(\hat{t}_{1,j}\right) \times p\left(\hat{t}_{2,j} \mid \hat{t}_{1,j}\right) \times \cdots \times p\left(\hat{t}_{m,j} \mid \hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m-1,j}\right), \\ =& \prod_{i=1}^{m-1} \frac{1}{\min\left\{U_i - L_i, \Delta T - \sum_{k=1}^{i-1}(\hat{t}_{k,j} - L_k)\right\}}, \end{aligned}$$

where $\Delta T = T - \sum_{i=1}^{m} L_i$ denotes the total time that StoM can randomize and the precision of randomized time is 1 second. Note that $p(\hat{t}_{m,j} \mid \hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m-1,j}) = 1$ because $\hat{t}_{m,j}$ can be determined completely by $\{\hat{t}_{1,j}, \hat{t}_{2,j}, \cdots, \hat{t}_{m-1,j}\}$ according to Eq. (7). Taking

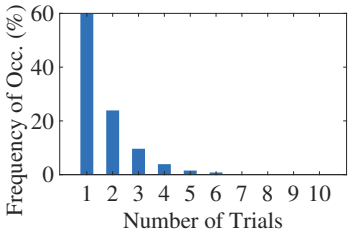
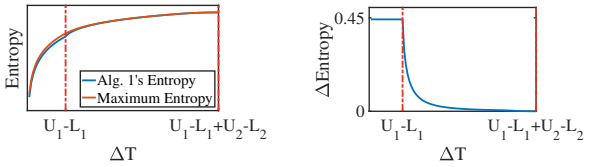


Fig. 9. StoM needs only 1.7 trials on average to randomize steps in parallel operation of "*assembly*" in AVM.



(a) entropy (b) the deviation of two entropies

Fig. 10. The entropy of the expected operation time returned by StoM is close to its theoretical maximum.

the case of $m = 3$ as an example, Fig. 10 shows that the entropy H of the operation time randomized by StoM is close to its maximum $H_{\max}$ (achieved when the operation time is uniformly distributed): (i) both H and $H_{\max}$ increase, while their difference decreases, with a larger ΔT, and (ii) the entropy H resulted by StoM achieves the maximum when $\Delta T = (U_1 - L_1) + (U_2 - L_2)$.

5.3 Detect Deception Attack in Runtime

After sending the randomized commands to the production line, StoM detects deception attacks by comparing the empirically observed operation time with the expected ones.

Identifying Empirical Operation Time. Workstations report only the data describing the real-time operations of devices thereof (e.g., the robot's joint angle), but without the time with which the operations are completed. StoM thus will identify the operation time from the data log. Taking the "*welding*" operation of AVM as an example, all devices (including industrial robots, PLCs, infrared sensors, conveyor belt, and welding guns) in the workstation report their operation data to StoM: the curves ❶❷ in Fig. 11 show the angle of robots' Joints-1 and Joints-2, and the curves ❸❹❺ in Fig. 11 show digital signals indicating whether the workstation operates, the object moves in, and the robot performs welding, respectively. Taking the above 5 time series as inputs, StoM identifies the time for the conveyor to complete the object moving-in, the robot to complete the object welding, and the waiting time after each execution step as follows.

(1) Identifying the start/stop events of device operation. The operation data consists of both analog signals (e.g., the robot's joint angle) and digital signals (e.g., power on/off), requiring different treatments to identify events. StoM first determines if the collected signal is digital or analog by checking whether the data series only contains "0" and "1" — i.e., the signal is digital (or analog) if yes (or not). StoM then transforms the operation

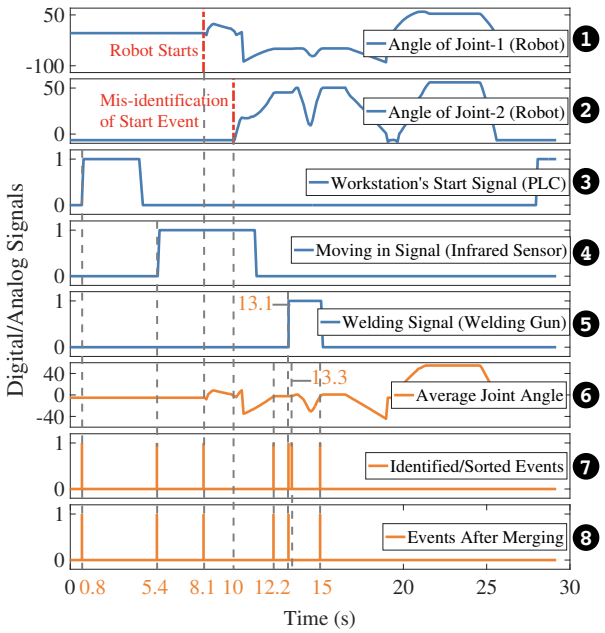


Fig. 11. AVM's "*welding*" operation data reported to SCADA.

data to events based on the fact that when the event happens, the digital signal changes from "0" to "1", and the differential of an analog signal varies from a large value to 0 (or from 0 to a large value). Specifically, StoM searches the data x_j when $x_j = 1$ and $x_{j-1} = 0$ for a digital signal series x_j, and the data x_j when the deviation $|x_j - x_{j-1}| < \gamma$ and $|x_j - x_{j+1}| > \gamma$ (or $|x_j - x_{j-1}| > \gamma$ and $|x_j - x_{j+1}| < \gamma$) for an analog signal series x_j. The identified timestamp of x_j will then be treated as the time an event happened. This way, StoM searches in all the time series to identify events $\mathbb{E} = \{e_1, e_2, \cdots, e_h\}$, where h is the number of the identified events. Note that γ is the threshold that should be larger than the maximal deviation of analog signals when the device does not operate, while smaller than the minimal deviation of analog signals when the device operates at the slowest speed.

(2) Classifying events $\mathbb{E} = \{e_1, e_2, \cdots, e_h\}$ into p categories (i.e., $\mathbb{E} = \{E_1, E_2, \cdots, E_p\}$) according to the operation they belong to, where $p = 1$ (or $p \geqslant 2$) in the sequential (or parallel) operations denotes the number of sub-operations thereof. Denote $E_k = \{e'_1, e'_2, \cdots, e'_{m_k}\}$ as the set of m_k events in the kth category.

(3) Sorting the kth category of events E_k to $\{e''_1, e''_2, \cdots, e''_{m_k}\}$ according to their timestamps, i.e., $t(e''_1) \leqslant t(e''_2) \leqslant \cdots \leqslant t(e''_{m_k})$, where $t(e''_i)$ denotes the timestamp of event e''_i.

(4) Calculating the operation time based on the consecutive events in the kth category, i.e., $t_i = t(e''_i) - t(e''_{i-1})$.

When applying the above operation time identification, StoM needs to address the following two practical challenges. First, the robot/CNC might operate using only part of the joints/motors, which may cause the mis-identification of events. For example, the angle of robot's Joint-2 (the curve ❷ in Fig. 11) shows that the robot starts at 10s, while the robot actually starts at 8.1s according to the angle of Joint-1 (the curve ❶ in Fig. 11). To address this, StoM considers all joints/motors of a manufacturing device when identifying the events. Specifically, StoM first identifies the source of each data series based on the IP address of the communication packets containing the operation data, and then calculates the average value of the data series from the same device, i.e., for an analog signal with S number of data series of $\{x_1, x_2, \cdots, x_S\}$, StoM calculates the average value by $\bar{x} = (1/S) \cdot \sum_{i=1}^{S} x_i$, and uses $\bar{x}$ to identify the events operated by the corresponding manufacturing device. Secondly, the same event may induce multiple varying signals, leading to the redundant identification of events and wrong calculation of operation time. For example, both the events at 13.1s and 13.3s identified from data series (the curves ❺❻ in Fig. 11) indicate the starting of the welding. StoM will mis-identify the welding event and mis-conclude the corresponding operation time as 0.2s if not properly treated. StoM addresses this issue by merging the two events when the gap of their timestamps is too small, i.e., for a sorted event series $\{e''_1, e''_2, \cdots, e''_{m_k}\}$ (see the curve ❼ in Fig. 11), an event e_j will be deleted if $|t(e_j) - t(e_{j-1})| < \delta$, where δ is the threshold that is empirically set as twice the average time interval for the workstation to report the operation data.

Taking AVM's "*welding*" operation shown in Fig. 11 as an example where γ is 0.2s and δ is 0.3s, StoM first calculates the average angle of the Joint-1 and Joint-2 (the curve ❻ in Fig. 11). Then, StoM transforms the operation data to events, sort them, and find that: (i) the workstation starts at 0.8s; (ii) the object moves in at 5.4s; (iii) the robot starts at 8.1s and 13.3s; (iv) the welding process starts at 13.1s. After that, StoM merges events whose happened interval is small enough, e.g., the start time difference between the robot and the welding signal is $|13.3 - 13.1| = 0.2$s (see the curve ❽ in Fig. 11 for the events after merging). Finally, StoM calculates the time to complete these operation steps based on the identified events, e.g., the time to complete the moving-in is $5.4 - 0.8 = 4.6$s, the waiting time after the moving-in is $8.1 - 5.4 = 2.7$s, and the time to complete the welding is $15 - 13.1 = 1.9$s.

Detecting Deception Attacks. StoM compares the empirical and expected operation time to detect deception attacks with the following 4 steps.

(1) calculating the residual (Δ_j) between the empirical and expected operation time at the jth manufacturing cycle:

$$\Delta_j = \sum_{i=1}^{m} |t_{i,j} - \hat{t}_{i,j}|, \tag{14}$$

where $t_{i,j}$ and $\hat{t}_{i,j}$ are empirical and expected time to complete the ith operation step in the jth manufacturing cycle;

(2) calculating the expected value (δ) and the standard deviation (σ) of the time series of residual Δ_j during normal operation:

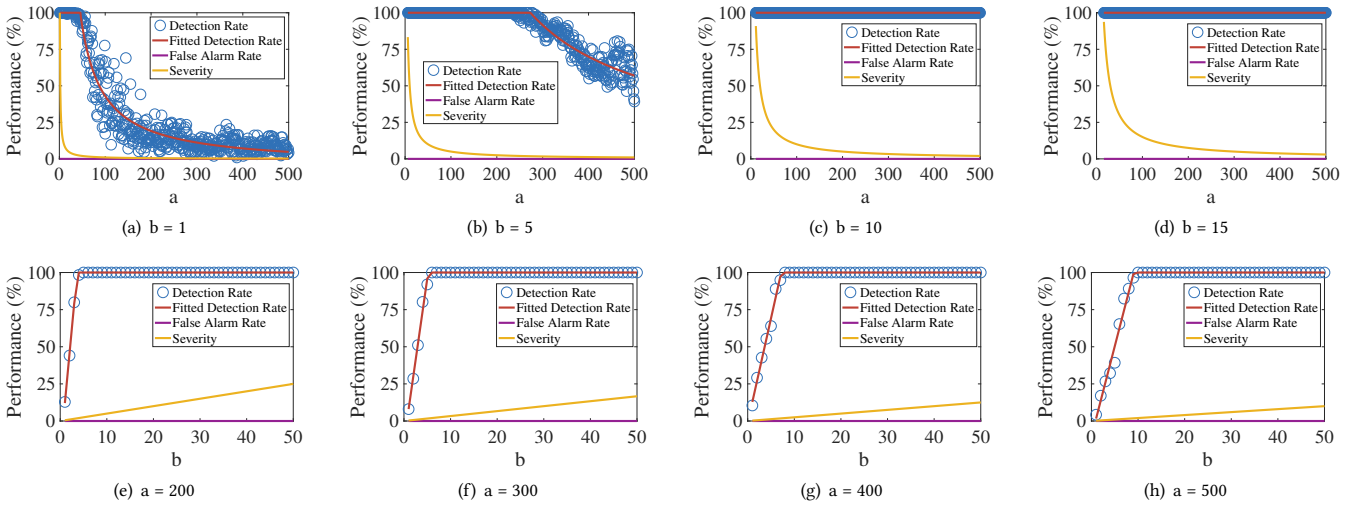
$$\delta = \frac{1}{n} \sum_{j=1}^{n} \Delta_j \quad \text{and} \quad \sigma = \left(\frac{1}{n} \sum_{j=1}^{n} \Delta_j^2\right)^{\frac{1}{2}}; \tag{15}$$

(3) calculating the residual sum as $s_j = \max(s_{j-1} + \Delta_j - \delta, 0)$;

(4) raising an alarm if $s_j > \tau$, where τ is the alarming threshold and is typically set as 4σ [44].

6 EVALUATION AND ANALYSIS

We have experimentally evaluated StoM on AVM and ABM, focusing on three key aspects: its resistance to deception attacks, its ability



(a) b = 1 (b) b = 5 (c) b = 10 (d) b = 15

(e) a = 200 (f) a = 300 (g) a = 400 (h) a = 500

Fig. 12. DR, FA, and ς with different a and b for intermittent deception attacks.

Table 3: Configuration of StoM on AVM.

Parameters	*"Welding"*	*"Warehousing"*	*"Assembly"*
Number of operation steps	10	6	18
Sampling rate (s)	0.15	0.15	0.15
Upper bounds (s)	$\{30\} \times 10$	$\{30\} \times 6$	$\{30\} \times 18$
Lower bounds (s)	{2.8, 0.3, 2.6, 0.3, 0.7, 0.3, 2.3, 0.3, 1.8, 0.3}	{2.8, 0.3, 6.6, 0.3, 1.0, 0.3}	{2.8, 0.3, 2.6, 0.3, 0.7, 0.3, 2.3, 0.3, 1.3, 0.3, 2.8, 0.3, 2.6, 0.3, 0.7, 0.3, 2.3, 0.3}
Constraints/ conditions	–	–	$a_{1,k}s$ = {−1, −1, −1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0}, $a_{2,k}s$ = {1, 1, 1, 1, 1, 1, 1, 1, 1, −1, −1, −1, −1, −1, −1, −1, −1, −1}

to control and maintain manufacturing efficiency, and the amount of historical data required for StoM to identify the control-operation dependency. Table 3 lists the configuration of StoM when implemented on AVM, where the lower bounds are identified (and then empirically validated) based on the data-sheet of devices thereof (e.g., ABB robots [56]). Constraint $a_{1,k}$ is to avoid the collision and $a_{2,k}$ is to ensure that the left sub-workstation completes the operation before the completion of each manufacturing cycle. Table 4 summarizes the configuration of StoM when implemented on ABM.

6.1 Detecting Deception Attacks

We classify deception attacks into three categories based on how adversaries manipulate the operation data.

- *Continuous deception attack:* adversaries record the normal cyclic operations and replay them repeatedly.
- *Intermittent deception attack:* adversaries mount the attack intermittently to be more stealthy. Specifically, the adversary mounts the attack (i.e., manipulating the production line's operation and replaying the operation data) every a cycles, and each attack lasts for b cycles ($a \geqslant b \geqslant 1$). Clearly, this attack reverts to the above continuous attack when $a == b$. Note that although a larger a and smaller b make the attack stealthier, the severity of the attack is alleviated because only b/a of manufacturing cycles are compromised.
- *Advanced deception attack:* adversaries who acquire the randomized control commands generated by StoM, coupled with detailed knowledge about the specific dynamics of the production line, can manipulate the operation data to align with the randomized control commands. This manipulation voids StoM's ability to detect deception attacks.

We have experimentally evaluated StoM against the continuous and intermittent deception attacks, and uncovered the infeasibility of the advanced deception attacks in practice.

All the 3 operations of AVM (i.e., "*welding*", "*warehousing*", and "*assembly*") and the 7 operations of ABM (i.e., *"picking and placing"*, *"turning and milling"*, *"inspection"*, *"craving"*, *"inkjeting"*, *"assembly"*, and *"warehousing"*) have been mounted with the deception attacks to evaluate StoM. Specifically, for each operation, the testbed first operates normally for 100∼300 cycles, after which the deception attacks are mounted to make the testbed operate abnormally for the same number of cycles. Each test is repeated 10 times. We evaluate StoM's detection of deception attacks using detection rate (DR) and false alarm rate (FA):

$$\text{DR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100\% \quad \text{and} \quad \text{FA} = \frac{\text{FP}}{\text{TN} + \text{FP}} \times 100\%,$$

where TP (or FN) is the number of manipulated manufacturing cycles that are classified as attacked (or normal), and FP (or TN) is the number of normal manufacturing cycles that are classified as being manipulated (or normal). We also examine the detection latency to evaluate how quickly StoM detects the attack.

Table 4: Configuration of StoM on ABM.

Parameters	*"Picking and Placing"*	*"Turning and Milling"*	*"Inspection"*	*"Craving"*	*"Inkjeting"*	*"Assembly"*	*Warehousing*
# of operation steps	4	16	4	7	4	9	4
Sampling rate (s)	1	1	1	1	1	1	1
Upper bounds (s)	$\{325\} \times 4$	$\{325\} \times 16$	$\{325\} \times 4$	$\{325\} \times 7$	$\{325\} \times 4$	$\{325\} \times 9$	$\{325\} \times 4$
Lower bounds (s)	{30, 3, 10, 3}	{5, 3, 34, 3, 34, 3, 30, 3,} 31, 3, 42, 16, 39, 3, 10, 25}	{10, 3, 12, 4}	{3, 31, 3, 3, 3, 21, 3}	{5, 3, 10, 2}	{7, 3, 4, 3, 5, 4,} 2, 3, 17}	{32, 3, 16, 3}
Constraints/ conditions	–	$a_{1,k}s = \{0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, -1, 0\}$, $a_{2,k}s = \{0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, -1\}$	–	–	–	$a_{1,k}s = \{-1, -1, -1, -1, 0, 0, 0, 1\}$	–

Table 5: Summary of experimental results of StoM on AVM and ABM.

Testbeds	Operations	Sequential or Parallel	Total Number of Cycles	StoM's Performance in Attack Detection								
				Detection of Continuous Attack			Detection of Intermittent Attack				Estimation	Control
				DR (%)	FA (%)	Latency	DR (%)	FA (%)	Latency	ς (%)	Accuracy (%)	Precision (%)
AVM	"*welding*"	Sequential	150 × 10	99.6	0	1.6	99.8	0	1.3	5	93.0	97.5
	"*warehousing*"	Sequential	300 × 10	99.9	0.03	1.4	99.8	0.03	1.2	5	96.2	97.9
	"*assembly*"	Parallel	100 × 10	99.7	0	1.3	99.7	0	1.3	5	93.7	96.6
ABM	"*picking and placing*"	Sequential	100 × 10	100.0	0	1	100.0	0	1	5	99.9	99.0
	"*turning and milling*"	Parallel	100 × 10	100.0	0	1	86.0	0	1	5	97.7	97.8
	"*inspection*"	Sequential	100 × 10	100.0	0	1	100.0	0	1	5	97.3	98.3
	"*craving*"	Sequential	100 × 10	100.0	0.4	1	100.0	0.4	1	5	99.9	99.2
	"*inkjecting*"	Sequential	100 × 10	100.0	0	1	100.0	0	1	5	97.6	97.2
	"*assembly*"	Parallel	100 × 10	100.0	0	1	98.0	0	1	5	93.0	96.7
	"*warehousing*"	Sequential	100 × 10	100.0	0.6	1	100.0	0.6	1	5	99.9	99.2

StoM vs. Continuous Deception Attacks. As summarized in Table 5, StoM detects continuous deception attacks to AVM (or ABM) with an average DR of 99.7% (or 100%), FR of 0.01% (or 0.1%), and latency of 1.4 (or 1.0) manufacturing cycles.

StoM vs. Intermittent Deception Attacks. StoM detects intermittent deception attacks (with attack cycle $a = 100$ and attack duration $b = 5$) to AVM with an average DR of 99.8%, FR of 0.01%, and a latency of 1.3 manufacturing cycles (see Table 5). StoM also performs well on ABM, achieving 100% DR, no false alarms, and a latency of 1 manufacturing cycle for most of the 7 operations, except {86%, 98%} DR for the {"turning and milling", "assembly"} operations and {0.4%, 0.6%} FA for the {"craving", "warehousing"} operations.

We have also evaluated the impacts of replaying cycle a and attacking cycle b on StoM's detection for intermittent deception attacks, by (i) setting b as {1,5,10,15} and varying a from $b \sim 500$, and (ii) setting a as {200,300,400,500} and varying b from $0 \sim 50$. Different a and b lead to different attack severity, which we quantify using how many manufacturing cycles are affected, i.e., $\varsigma = b/a \times 100\%$. Fig. 12 shows the results obtained with AVM: (i) both the detection rate DR and the harmfulness ς decrease (or increase) with a larger a (or b), and (ii) a 100% DR and no false alarm is achieved/triggered when $b > 10$ or $\varsigma > 2\%$.

StoM vs. Advanced Deception Attacks. Sophisticated adversaries — i.e., adversaries with full knowledge of the production line and StoM — can deliver advanced deception attacks to void StoM in the following two ways.

Attack-1: Adversaries who know the specific dynamics of the production line and the randomized control commands can carefully forge the operation data to make them consistent with the control commands randomized by StoM.

First of all, it is the operation data from which StoM identifies the operation time of each manufacturing step and further detects deception attacks. To evade StoM, adversaries have to generate proper operation data based on the dynamics of production lines, which is, however, challenging: (i) the parameters of each machine in production lines, such as the Denavit-Hartenberg parameters of robot arms, are proprietary to equipment manufacturers (e.g., ABB, KUKA), preventing adversaries from deriving theoretical dynamics of the production lines; (ii) the barely triggered operation status limits adversaries from accurately and completely identifying the non-linear dynamics from historical operation data.

Even if they have learned the dynamics of production lines, adversaries have to obtain the randomized control commands further. The randomized control commands can be protected by applying a mask, such as the *format-preserving mask*. After receiving the masked control commands from SCADA, the controller can obtain

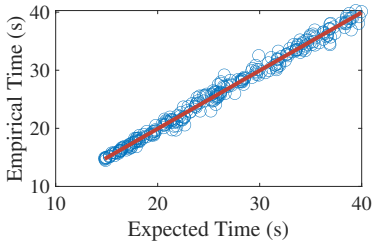


Fig. 13. Empirical and expected cycle time of the "*welding*" operation of AVM.

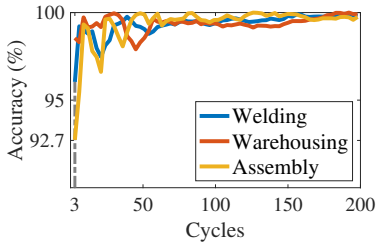


Fig. 14. Accuracy of estimating the operation time vs. the number of runtime manufacturing cycles.

the original randomized control commands based on specific mappings. This prevents adversaries from eavesdropping the randomized control commands via man-in-the-middle attacks. Also, even if adversaries can compromise the controllers, it is still non-trivial to identify the unmasking function because adversaries need to disassemble the binaries of firmware and control programs, which are proprietary and diverse among controller vendors. Note that unmasking the control commands only incurs small overhead to controllers. For example, we deployed a linear unmaking function in a Siemens S7-1500 PLC, showing that only 0.02ms overhead is induced to the original calculation cycles ($\geq$ 2ms).

Attack-2: Adversaries who have the knowledge on all possible control commands to be randomized by StoM *and the corresponding operation data, can launch advanced deception attacks. Specifically, whenever a newly generated command by* StoM *has been eavesdropped, adversaries search the records to find the one closest to the new command and replay the corresponding operation data to SCADA.*

We have examined such an advanced attack on AVM, showing that a huge storage space is needed to record all possible variations of control commands and operation data, exceeding the storage capacity of devices in the current production line and thus renders the advanced attack impractical. Specifically, let us consider a workstation with m operation steps, x sensors (with 4-Bytes for each sensory sample), sampling rate s, cycle time T, and alarm threshold τ. To void StoM, the adversary needs to make sure $\Delta_j < \delta$ in Eq. (14), otherwise the corresponding s_j will increase gradually and trigger $s_j > \tau$. Such constraints require the adversary to record at least $(U_i - L_i)/\delta$ control commands for each operation step i. The required storage space is thus:

$$Q = 4x \cdot T/s \cdot \prod_{i=1}^{m} (U_i - L_i)/\delta. \tag{16}$$

According to the settings of AVM in Table 3, the operations of "*welding*", "*warehousing*" and "*assembly*" require adversaries to store 16PB, 3TB and 10^8PB data. Because the adversary is hard to deploy additional computational/storage devices to the production line, these huge data can only be stored on the existing devices of the production line, whose storage capacity is way too small to satisfy the requirements, such as 40GB in ABB/COMAU robots, 100MB in Siemens/Omrom PLCs, and 500GB in general purpose computers.

6.2 Maintaining Manufacturing Efficiency

We next examine StoM's ability of maintaining manufacturing efficiency, by checking whether StoM can (i) accurately estimate the expected time to complete each operation step, and based on which, (ii) precisely control the cycle time to the expected level.

We evaluate StoM's estimation of the expected operation time by comparing the expected and empirical time to complete the ith operation step of the jth manufacturing cycle, i.e., comparing $t_{i,j}$ and $\hat{t}_{i,j}$ where $i \in \{1, 2, \cdots, m\}$ and $j \in \{1, 2, \cdots, n\}$:

$$\text{Acc} = 1 - \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{|t_{i,j} - \hat{t}_{i,j}|}{t_{i,j}} \times 100\%. \tag{17}$$

We log the control commands and operation data of 100~300 cycles of AVM's 3 operations and ABM's 7 operations. We repeat this for each operation step 10 times, and then apply StoM to estimate the time to complete these operation steps. The results are given in Table 5, showing that StoM achieves an average estimation accuracy of 94.3% for AVM's 3 operations and 97.9% for ABM's 7 operations.

We next examine StoM's precision of controlling the cycle time. Specifically, we set the cycle time of AVM and ABM as 30s and 325s, respectively, and then perform StoM to control each of AVM's 3 operations and ABM's 7 workstations for 100~300 cycles. Table 5 shows that StoM achieves an average 97.3% control precision for AVM's 3 operations and 98.2% precision for ABM's 7 operations. We have also randomly changed the expected cycle time of AVM's "*welding*" operation from 15s to 40s, and performed the same operations for 300 cycles. Fig. 13 corroborates that StoM controls AVM's cycle time with an average precision of 97.9%.

6.3 Required Runtime Operation Data

Lastly, we examine how many cycles of operation data is needed for StoM to capture the dependency between the control commands and operation time (i.e., $G_i(\cdot)$ in Eq. (2), using AVM's 3 operations as examples. Fig. 14 summarizes StoM's accuracy of estimating the operation time with the number of manufacturing cycles increases from 1-200, showing that StoM needs only 3 cycles of operation data to capture the control-operation dependency with an >92% accuracy, which increases further when more data is available. Such less operation data required for training $G_i(\cdot)$ help StoM, at runtime, update/refine the control-operation dependencies of a production line using the latest collected operation data.

7 DISCUSSION

Below we present several further discussions of StoM.

Broader Applications. StoM has broader applications than just protecting production lines from deception attacks. First, StoM

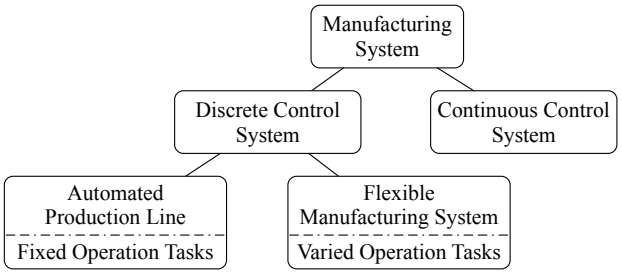


Fig. 15. Classification of manufacturing systems.

can be applied to diagnose any anomaly that violate the control-operation dependency, e.g., incorrect operation data due to sensor failure or communication error. Also, StoM can be applied to other manufacturing systems than production lines. As depicted in Fig. 15, manufacturing systems can be classified into two categories, i.e., discrete and continuous control systems, where the discrete control systems can be further represented by two typical implementations according to their operation tasks: automated production lines and flexible manufacturing systems. Continuous control systems are commonly used to manufacture fluid/gas/powder products (e.g., purifying the alcohol from a mixture), where the materials are processed continuously without the concept of *"manufacturing cycle"*. However, StoM's basic idea — i.e., injecting stochasticity to system operation based on the relationship between the control commands and operation data — can still be applied to continuous control systems. For example, StoM can randomize the continuous control by periodically (and randomly) updating the control set-points that determine the production efficiency but not the quality, and controlling such stochasticity to make the physical process achieve the same production efficiency in the long run. Flexible manufacturing systems support customized production, for which StoM can be applied with a similar appraoch as for production lines. Note that the operation tasks of flexible manufacturing systems may vary based on the products to be manufactured, causing varying cycle time and thus requiring more fine-tuned algorithms for StoM to randomize the control commands.

Configuration for Parallel Operations. The deployment of StoM for parallel operations requires the configuration of $a_{i,k}$ in Eq. (12), which can be done by supporting engineers who have the domain knowledge of a given production line (i.e., as we do for AVM and ABM). Besides, StoM can also be configured using system twins/simulators, often provided in production lines nowadays. For example, we can use the twin/simulator to identify the control commands that cause (or do not cause) collisions, based on which we can identify the valid configurations of $a_{i,k}$ via data-driven approaches.

Further Improvement of Control Precision. The precision of StoM in controlling the cycle time (i.e., averaged at 97.9% for AVM and ABM) can be improved further by mitigating the inherent/slight variance in the operation of the production line. Take AVM as an example, where the object is placed onto the conveyor belt by the robot's gripper. The sticky gripper (due to the aged rubber ring) of the robot causes a slight variance in the object's position on the conveyor belt, which causes, in turn, variances in the time to complete the moving-out operation steps, as observed in Fig. 7(i). Regularly maintaining the gripper helps reduce this physically induced variance and thus improves StoM's control precision. Besides, StoM's precision can be further enhanced by capturing and incorporating this variance into generating the randomized commands, which we will explore in our future work.

Manufacturing efficiency/quality vs. randomized commands. As explained in Sec. 3.1, the operation steps within any workstation consist of (i) the movements of the to-be-processed parts, (ii) the processing of parts in manufacturing machines, and (iii) the waiting period for coordination with other workstations. By collaboratively adjusting the speeds of these steps within each workstation, StoM maintains the execution cycle and thus preserves manufacturing efficiency. In addition, although specific processes, such as cooling, may be time-critical, the movement of parts (via conveyors or robot arms) and the stopping of machines do not affect product quality. By not randomizing commands for the time-critical processes, StoM can ensure consistent product quality. Specifically, StoM can be applied by setting the upper/lower bound of the time-critical process as its original time t_i, i.e., setting $L_i = U_i = t_i$ in Eqs. (8)(11).

8 CONCLUSION

This paper presents StoM, a new paradigm of manufacturing that is resistant to deception by design. StoM stochastically controls the production line operation without degrading the manufacturing efficiency/quality, and checks if the injected stochasticity can be observed from the operation data. We have experimentally evaluated StoM on two automated production line testbeds.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant 62303411, 62293510/62293511 and 62273305, the Zhejiang Provincial Natural Science Foundation under Grant LZ22F030010, and the Fundamental Research Funds for the Central Universities 226-2024-00004 and 226-2023-0011.

REFERENCES

[1] ABB. 2023. Detailed information for: RobotStudio. https://new.abb.com/products/3HAC031177-001/robotstudio. [Online; Accessed May 2023].
[2] ABB. 2023. How do we implement Robotic Process Automation? https://new.abb.com/news/detail/101946/how-do-we-implement-robotic-process-automation. [Online; Accessed June 2023].
[3] Sridhar Adepu and Aditya Mathur. 2018. Distributed Attack Detection in a Water Treatment Plant: Method and Case Study. *IEEE Transactions on Dependable and Secure Computing* 18, 1 (2018), 86–99.
[4] ICS Advisory. 2022. APT Cyber Tools Targeting ICS/SCADA Devices. https://www.cisa.gov/uscert/ncas/alerts/aa22-103a. [Online; Accessed June 2023].
[5] Airbus AG. 2019. Airbus inaugurates new A320 structure assembly line in Hamburg. https://www.airbus.com/en/newsroom/press-releases/2019-10-airbus-inaugurates-new-a320-structure-assembly-line-in-hamburg. [Online; Accessed June 2023].
[6] KUKA AG. 2019. Hier sind die Industrie-4.0-Roboter: Intelligente Automatisierung im KUKA Werk. https://www.youtube.com/watch?v=-PGrqcNtB7M. [Online; Accessed June 2023].
[7] Siemens AG. 2017. Programming with STEP 7. https://cache.industry.siemens.com/dl/files/825/109751825/att_933142/v1/STEP_7_-_Programming_with_STEP_7.pdf. [Online; Accessed June 2023].
[8] Homa Alemzadeh, Daniel Chen, Xiao Li, Thenkurussi Kesavadas, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. 2016. Targeted Attacks on Teleoperated Surgical Robots: Dynamic Model-Based Detection and Mitigation. In *International Conference on Dependable Systems and Networks (DSN)*. IEEE, 395–406.
[9] Anonymous. 2023. Demo: Mounting Deception Attacks to Automated Production Lines. https://youtu.be/zgOkaqLvPyE. [Online; Accessed May 2023].

[10] Wissam Aoudi, Mikel Iturbe, and Magnus Almgren. 2018. Truth Will Out: Departure-Based Process-Level Detection of Stealthy Attacks on Control Systems. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 817–831.

[11] Dillon Beresford. 2011. Exploiting Siemens Simatic S7 PLCs. *Black Hat USA* 16, 2 (2011), 723–733.

[12] Eli Biham, Sara Bitan, Aviad Carmel, Alon Dankner, Uriel Malin, and Avishai Wool. 2019. Rogue7: Rogue Engineering Station Attacks on S7 Simatic PLCs. In *BlackHat USA*.

[13] Nathan Brubaker, Keith Lunden, Ken Proska, Muhammad Umair, Daniel Kapellmann Zafra, Corey Hildebrandt, and Rob Caldwell. 2022. INCONTROLLER: New State-Sponsored Cyber Attack Tools Target Multiple Industrial Control Systems. https://www.mandiant.com/resources/blog/incontroller-state-sponsored-ics-tool. [Online; Accessed June 2023].

[14] Siemens Security Advisory by Siemens ProductCERT. 2020. SSA-381684: Improper Password Protection during Authentication in SIMATIC S7-300 and S7-400 CPUs and Derived Products. https://cert-portal.siemens.com/productcert/pdf/ssa-381684.pdf. [Online; Accessed June 2023].

[15] Defense Use Case. 2016. Analysis of the Cyber Attack on the Ukrainian Power Grid. *Electricity Information Sharing and Analysis Center* (2016).

[16] Cesar Cerrudo and Lucas Apa. 2017. Hacking Robots Before Skynet. *IOActive Website* (2017), 1–17.

[17] Hongjun Choi, Wen-Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Xinyan. 2018. Detecting Attacks Against Robotic Vehicles: A Control Invariant Approach. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 801–816.

[18] Keywhan Chung, Xiao Li, Peicheng Tang, Zeran Zhu, Zbigniew T Kalbarczyk, Ravishankar K Iyer, and Thenkurussi Kesavadas. 2019. Smart Malware that Uses Leaked Control Data of Robotic Applications: The Case of Raven-II Surgical Robots. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. 337–351.

[19] Kevin Collier. 2021. In Florida, a near-miss with a cybersecurity worst-case scenario. https://www.nbcnews.com/tech/security/florida-near-miss-cybersecurity-worst-case-scenario-n1257091. [Online; Accessed May 2023].

[20] Cybersecurity and Infrastructure Security Agency. 2020. Rockwell Automation MicroLogix Controllers and RSLogix 500 Software. https://us-cert.cisa.gov/ics/advisories/icsa-20-070-06. [Online; Accessed June 2023].

[21] Nicholas DeMarinis, Stefanie Tellex, Vasileios P Kemerlis, George Konidaris, and Rodrigo Fonseca. 2019. Scanning the Internet for ROS: A View of Security in Robotics Research. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 8514–8521.

[22] FANUC Europe. 2017. FANUC Industrial Robots at AUDI. https://www.youtube.com/watch?v=rbki4HR41-4. [Online; Accessed June 2023].

[23] Nicolas Falliere, Liam O Murchu, and Eric Chien. 2011. W32. Stuxnet Dossier. *White paper, Symantec Corp., Security Response* 5, 6 (2011), 29.

[24] Chongrong Fang, Yifei Qi, Peng Cheng, and Wei Xing Zheng. 2020. Optimal periodic watermarking schedule for replay attack detection in cyber–physical systems. *Automatica* 112 (2020), 108698.

[25] Cheng Feng, Venkata Reddy Palleti, Aditya Mathur, and Deeph Chana. 2019. A Systematic Framework to Generate Invariants for Anomaly Detection in Industrial Control Systems. *Symposium on Network and Distributed System Security (NDSS)* (2019).

[26] David Formby, Preethi Srinivasan, Andrew Leonard, Jonathan Rogers, and Raheem A Beyah. 2016. Who's in Control of Your Control System? Device Fingerprinting for Cyber-Physical Systems. In *Symposium on Network and Distributed System Security (NDSS)*.

[27] Matthew Fox. 2023. Python bindings for libnetfilter_queue. https://pypi.org/project/NetfilterQueue. [Online; Accessed May 2023].

[28] Luis Garcia, Ferdinand Brasser, Mehmet Hazar Cintuglu, Ahmad-Reza Sadeghi, Osama A Mohammed, and Saman A Zonouz. 2017. Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit. In *Symposium on Network and Distributed System Security (NDSS)*.

[29] Hamid Reza Ghaeini, Matthew Chan, Raad Bahmani, Ferdinand Brasser, Luis Garcia, Jianying Zhou, Ahmad-Reza Sadeghi, Nils Ole Tippenhauer, and Saman Zonouz. 2019. PAtt: Physics-based Attestation of Control Systems. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. 165–180.

[30] Mikell P. Groover. 2016. *Automation, Production Systems, and Computer-Integrated Manufacturing*.

[31] Equation Group. 2018. MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption. https://www.rapid7.com/db/modules/exploit/windows/smb/ms17_010_eternalblue. [Online; Accessed May 2023].

[32] Damodar N. Gujarati and Dawn C. Porter. 2009. *Basic Econometrics*.

[33] Dragos Inc. 2022. CHERNOVITE's PIPEDREAM Malware Targeting Industrial Control Systems (ICS). https://www.dragos.com/blog/industry-news/chernovite-pipedream-malware-targeting-industrial-control-systems/. [Online; Accessed June 2023].

[34] Leon Johnson. 2020. Exploitable vulnerabilities #1 (MS08-067). https://www.rapid7.com/blog/post/2014/02/03/new-ms08-067. [Online; Accessed May 2023].

[35] Takeaki Kariya and Hiroshi Kurata. 2004. *Generalized Least Squares*.

[36] David Kennedy, Jim O'gorman, Devon Kearns, and Mati Aharoni. 2011. *Metasploit: The Penetration Tester's Guide*.

[37] Ali Keshvarparast, Daria Battini, Olga Battaia, and Amir Pirayesh. 2023. Collaborative robots in manufacturing and assembly systems: literature review and future research agenda. *Journal of Intelligent Manufacturing* (2023), 1–54.

[38] Qiang Li, Xuan Feng, Haining Wang, and Limin Sun. 2018. Understanding the Usage of Industrial Control System Devices on the Internet. *IEEE Internet of Things Journal* 5, 3 (2018), 2178–2189.

[39] Efrén López-Morales, Ulysse Planta, Carlos Rubio-Medrano, Ali Abbasi, and Alvaro A Cardenas. 2024. SoK: Security of Programmable Logic Controllers. In *33rd USENIX Security Symposium (USENIX Security)*.

[40] Reeco Automation Ltd. 2019. Electronics Assembly - Nidec. https://www.youtube.com/watch?v=jdL_2Y5Q6S4&t=10s. [Online; Accessed June 2023].

[41] Daniele Marrone. 2022. The implementation of collaborative robots in production environments: a review of academic literature and industrial applications. (2022).

[42] Jeremy A Marvel, Roger Bostelman, and Joe Falco. 2018. Multi-Robot Assembly Strategies and Metrics. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 1–32.

[43] InGear Media. 2022. BMW iFactory — iX1 Production. https://www.youtube.com/watch?v=bML5ZUyEiQw. [Online; Accessed June 2023].

[44] Douglas C. Montgomery. 2007. *Introduction to Statistical Quality Control*.

[45] Vedanth Narayanan and Rakesh B. Bobba. 2018. Learning Based Anomaly Detection for Industrial Arm Applications. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*. 13–23.

[46] Emma Newburger. 2021. Ransomware attack forces shutdown of largest fuel pipeline in the U.S. https://www.cnbc.com/2021/05/08/colonial-pipeline-shuts-pipeline-operations-after-cyberattack.html. [Online; Accessed June 2023].

[47] Schneider Electric Security Notification. 2021. Security Notification - Modicon M100/M200/M221 Programmable Logic Controller (V3.0). https://www.se.com/ww/en/download/document/SEVD-2020-315-05/. [Online; Accessed June 2023].

[48] Alberto Ornaghi and Marco Valleri. 2020. Ettercap Home Page. https://www.ettercap-project.org. [Online; Accessed May 2023].

[49] Hongyi Pu, Liang He, Peng Cheng, Jiming Chen, and Youxian Sun. 2023. COMMAND2: A Deception Attack Against Industrial Robots. *Engineering* (2023).

[50] Hongyi Pu, Liang He, Peng Cheng, Mingyang Sun, and Jiming Chen. 2022. Security of Industrial Robots: Vulnerabilities, Attacks, and Mitigations. *IEEE Network* 37, 1 (2022), 111–117.

[51] Hongyi Pu, Liang He, Chengcheng Zhao, David KY Yau, Peng Cheng, and Jiming Chen. 2020. Detecting Replay Attacks against Industrial Robots via Power Fingerprinting. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys)*. 285–297.

[52] Davide Quarta, Marcello Pogliani, Mario Polino, Federico Maggi, Andrea Maria Zanchettin, and Stefano Zanero. 2017. An Experimental Security Analysis of an Industrial Robot Controller. In *Symposium on Security and Privacy (S&P)*. 268–286.

[53] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. 2020. SAVIOR: Securing Autonomous Vehicles with Robust Physical Invariants. In *29th USENIX Security Symposium (USENIX Security)*. 895–912.

[54] Ravi Rao. 2023. What are Robotic Assembly Lines? History, Components, Advantages, Limitations, Applications, and Future. https://www.wevolver.com/article/what-are-robotic-assembly-lines-history-components-advantages-limitations-applications-and-future. [Online; Accessed June 2023].

[55] ABB Robotics. 2015. Application Manual PC SDK. *ABB AB Robotic products* (2015), 204–217.

[56] ABB Robotics. 2019. Product specification - IRB 120. https://library.e.abb.com/public/6aed5e91083f4fceb358eea2fe4c1bab/3HAC035960%20PS%20IRB%20120-en.pdf?x-sign=edIex5StIjpgmJBJJ95tak9NHdyuuut6mzzJHESGKt5i1JG8dhLRBdvggKptgBJn. [Online; Accessed May 2023].

[57] Nabil Sayfayn and Stuart Madnick. 2017. Cybersafety Analysis of the Maroochy Shire Sewage Spill (Preliminary Draft). (2017).

[58] Dimitrios Serpanos and Marilyn Wolf. 2018. *Internet-of-Things (IoT) Systems Architectures, Algorithms, Methodologies*.

[59] Suresh P Sethi and Qing Zhang. 2012. *Hierarchical Decision Making in Stochastic Manufacturing Systems*.

[60] Dug Song. 2000. dsniff. https://www.monkey.org/~dugsong/dsniff. [Online; Accessed May 2023].

[61] Catherine Stupp. 2021. Energy Tech Firm Hit in Ransomware Attack: Oslo-based Volue is working to restore systems and customer software after incident. https://www.wsj.com/articles/energy-tech-firm-hit-in-ransomware-attack-11620764034. [Online; Accessed June 2023].

[62] Zhichuang Sun, Bo Feng, Long Lu, and Somesh Jha. 2020. OAT: Attesting Operation Integrity of Embedded Devices. In *2020 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 1433–1449.

[63] David I Urbina, Jairo A Giraldo, Alvaro A Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg.

2016. Limiting the Impact of Stealthy Attacks on Industrial Control Systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[64] Alfred Wehrl. 1978. General Properties of Entropy. *Reviews of Modern Physics* 50, 2 (1978), 221.

[65] Marilyn Wolf and Dimitrios Serpanos. 2020. *Safe and Secure Cyber-Physical Systems and Internet-of-Things Systems*.

[66] Zeyu Yang, Liang He, Peng Cheng, and Jiming Chen. 2024. Mismatched Control and Monitoring Frequencies: Vulnerability, Attack, and Mitigation. *IEEE Transactions on Dependable and Secure Computing* (2024), 1–18. https://doi.org/10.1109/TDSC.2024.3384146

[67] Zeyu Yang, Liang He, Peng Cheng, Jiming Chen, David KY Yau, and Linkang Du. 2020. PLC-Sleuth: Detecting and Localizing PLC Intrusions Using Control Invariants. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. 333–348.

[68] Zeyu Yang, Liang He, Yucheng Ruan, Peng Cheng, and Jiming Chen. 2024. Unveiling Physical Semantics of PLC Variables Using Control Invariants. *IEEE Transactions on Dependable and Secure Computing* (2024), 1–18.

[69] Mu Zhang, James Moyne, Z Morley Mao, Chien-Ying Chen, Bin-Chou Kao, Yassine Qamsane, Yuru Shao, Yikai Lin, Elaine Shi, Sibin Mohan, et al. 2019. Towards Automated Safety Vetting of PLC Code in Real-World Plants. In *Symposium on Security and Privacy (S&P)*.

[70] Zhenyong Zhang, Ruilong Deng, David KY Yau, Peng Cheng, and Jiming Chen. 2019. Analysis of Moving Target Defense Against False Data Injection Attacks on Power Grid. *IEEE Transactions on Information Forensics and Security* 15 (2019), 2320–2335.