Yoong *et al. Cybersecurity* (2021) 4:6
https://doi.org/10.1186/s42400-021-00069-7

Cybersecurity

RESEARCH Open Access

Deriving invariant checkers for critical infrastructure using axiomatic design principles

Cheah Huei Yoong[1*], Venkata Reddy Palleti[2], Rajib Ranjan Maiti[3], Arlindo Silva[1] and Christopher M Poskitt[4]

Abstract

Cyber-physical systems (CPSs) in critical infrastructure face serious threats of attack, motivating research into a wide variety of defence mechanisms such as those that monitor for violations of *invariants*, i.e. logical properties over sensor and actuator states that should always be true. Many approaches for identifying invariants attempt to do so automatically, typically using data logs, but these can miss valid system properties if relevant behaviours are not well-represented in the data. Furthermore, as the CPS is already built, resolving any design flaws or weak points identified through this process is costly. In this paper, we propose a systematic method for deriving invariants from an analysis of a CPS *design*, based on principles of the axiomatic design methodology from design science. Our method iteratively decomposes a high-level CPS design to identify sets of dependent *design parameters* (i.e. sensors and actuators), allowing for invariants and invariant checkers to be derived in parallel to the implementation of the system. We apply our method to the designs of two CPS testbeds, SWaT and WADI, deriving a suite of invariant checkers that are able to detect a variety of single- and multi-stage attacks without any false positives. Finally, we reflect on the strengths and weaknesses of our approach, how it can be complemented by other defence mechanisms, and how it could help engineers to identify and resolve weak points in a design before the controllers of a CPS are implemented.

Keywords: Cyber-physical systems, Critical infrastructure, Industrial control systems, Systematic design framework, Axiomatic design, Invariants, Anomaly detection, Supervised machine learning

Introduction

Cyber-physical systems (CPSs), in which software components and physical processes are tightly integrated, are prevalent in the automation of critical infrastructure, e.g. as the industrial control systems of power grids and water purification plants. The potential impact of compromising such systems has made them prime targets for attackers (Hassanzadeh et al. 2020; Leyden 2016). In 2015, for example, the US Department of Homeland Security reported 25 cybersecurity incidents in the water sector and 46 in energy. Internationally, there have been several well-publicised attacks in these sectors too (N. Al-Mhiqani et al. 2018). This situation has motivated the development of multiple different countermeasures for attack detection and prevention, including techniques based on anomaly detection (Cheng et al. 2017; Goh et al. 2017; Harada et al. 2017; Inoue et al. 2017; Pasqualetti et al. 2011; Aggarwal et al. 2018; Aoudi et al. 2018; He et al. 2019; Kravchik and Shabtai 2018; Lin et al. 2018; Narayanan and Bobba 2018; Schneider and Böttinger 2018; Carrasco and Wu 2019; Kim et al. 2019; Adepu et al. 2020; Das et al. 2020; Giraldo et al. 2020; Schmidt et al. 2020), fingerprinting (Ahmed et al. 2018; Ahmed et al. 2018; Formby et al. 2016; Gu et al. 2018; Kneib and Huth 2018; Yang et al. 2020), and fuzzing (Chen et al. 2019; Chen et al. 2020; Wijaya et al. 2020).

Another popular approach is to monitor *invariants* of a CPS (Adepu and Mathur 2016a; Adepu and Mathur

*Correspondence: cheahhuei_yoong@sutd.edu.sg
[1] Singapore University of Technology and Design, 8 Somapah Road, 487372 Singapore, Singapore
Full list of author information is available at the end of the article

2016b; Giraldo et al. 2018), i.e. properties that *always* hold under normal operating conditions, and the violation of which might suggest the presence of an attacker in the system. Invariants are typically relations over the sensor readings and actuator states of a system, a simple example being that "if the tank level is above x, then pump p should be ON". Given the complexity of CPSs in general, several approaches (e.g. (Chen et al. 2016; Chen et al. 2018; Feng et al. 2019)) aim to automatically derive such invariants from sources of data, for instance, the time series of sensor readings and actuator states logged by a supervisory control and data acquisition system (SCADA). There is a risk, however, that viable system behaviours are missed if they are not represented in that data (e.g. rarely occurring), and addressing any design flaws identified is costly as the CPS is already built. Invariants can be derived manually by system engineers (Cárdenas et al. 2011; Adepu and Mathur 2016a; Adepu and Mathur 2016b; Adepu and Mathur 2021; Choi et al. 2018), but if done so in an ad hoc manner, may also lead to properties being missed.

In this paper, we propose a novel and systematic method for deriving invariants and invariant checkers from a *design-level analysis* of a CPS. In doing so, we aim to: (1) find invariants implicit in the design but poorly represented in datasets; (2) ensure that invariants can be contextualised by the specific design iterations and requirements they were derived from; and (3) further integrate security concerns at the design stage, potentially allowing weak points to be identified and fixed before a CPS is built. Our method, inspired by the principles of *axiomatic design* (Suh 2001)—a design science methodology for systems—iteratively decomposes a CPS design to sets of dependent components that can be transformed into invariants. We implement *invariant checkers* using decision tree learning, and use them to monitor CPSs for anomalies, i.e. violations of the invariant properties.

To evaluate the viability of our proposals, we apply our method to the designs of two real-world CPS testbeds. First, Secure Water Treatment (SWaT) (Secure Water Treatment (SWaT) 2020; Mathur and Tippenhauer 2016), a scaled-down version of a modern water purification plant. SWaT is a complex multi-stage CPS involving physical and chemical processes such as ultrafiltration, de-chlorination, and reverse osmosis. Second, Water Distribution (WADI) (Ahmed et al. 2017), a scaled-down version of a water distribution network typical of a city, designed to account for varying patterns of peak and off-peak water demand. Starting from high-level functional requirements, we applied axiomatic design principles to decompose the systems' designs and identify dependencies between their design parameters (i.e. sensors and actuators). Using domain expertise and process graphs, we transformed these into a suite of invariant checkers that were able to detect 13 different single- and multi-stage attacks on the real systems, all without false positives.

Our paper is organised as follows. In our Background section, we present an overview of the SWaT and WADI testbeds, as well as a general attack classification that will be used in the evaluation of our method. In Our Approach, we present the three main steps of our design-level analysis, and show how axiomatic design principles can be used to identify sets of dependent components that can be transformed into invariants. In our Evaluation and discussion section, we assess the effectiveness of decision tree learning for constructing our invariant checkers, their ability to correctly label real SWaT and WADI inputs as normal or anomalous, and then reflect on the strengths and weaknesses of our approach. Finally, we compare our approach against some Related work before drawing some Conclusions and speculating on some future work.

This is a revised and extended version of our position paper, Towards Systematically Deriving Defence Mechanisms from Functional Requirements of Cyber-Physical Systems (Yoong et al. 2020), adding the following new content: (1) an expanded description of the approach, adding details of the training sets used, an algorithm, and additional examples of invariants; (2) the addition of WADI as a second case study; (3) an evaluation of our invariant checkers against 13 different single- and multi-stage attacks; (4) new Discussion and Related work sections offering some critical reflections and comparisons; and (5) significant improvements to all parts of the text, including additional depth, examples, and figures.

Background

This section presents an overview of the two CPS testbeds used to evaluate our proposed approach. First, we present SWaT, a water purification plant that forms our principal case study, followed by our second testbed, the WADI water distribution system. Finally, we clarify what is meant by a CPS attack in the context of such systems.

SWaT testbed

The Secure Water Treatment (SWaT) testbed (Secure Water Treatment (SWaT) 2020; Mathur and Tippenhauer 2016) is a scaled-down version of a modern water purification plant, intended for supporting research into cyber-security solutions for critical infrastructure. SWaT is able to produce up to five gallons of safe drinking water per minute across six distinct co-operating stages (Fig. 1) involving chemical processes like ultrafiltration, de-chlorination, and reverse osmosis. Each stage is controlled by an Allen-Bradley ControlLogix Programmable Logic Controller (PLC), which communicates with sensors and actuators through a field-bus network, and with each other through a 24-port Ethernet switch. A SCADA

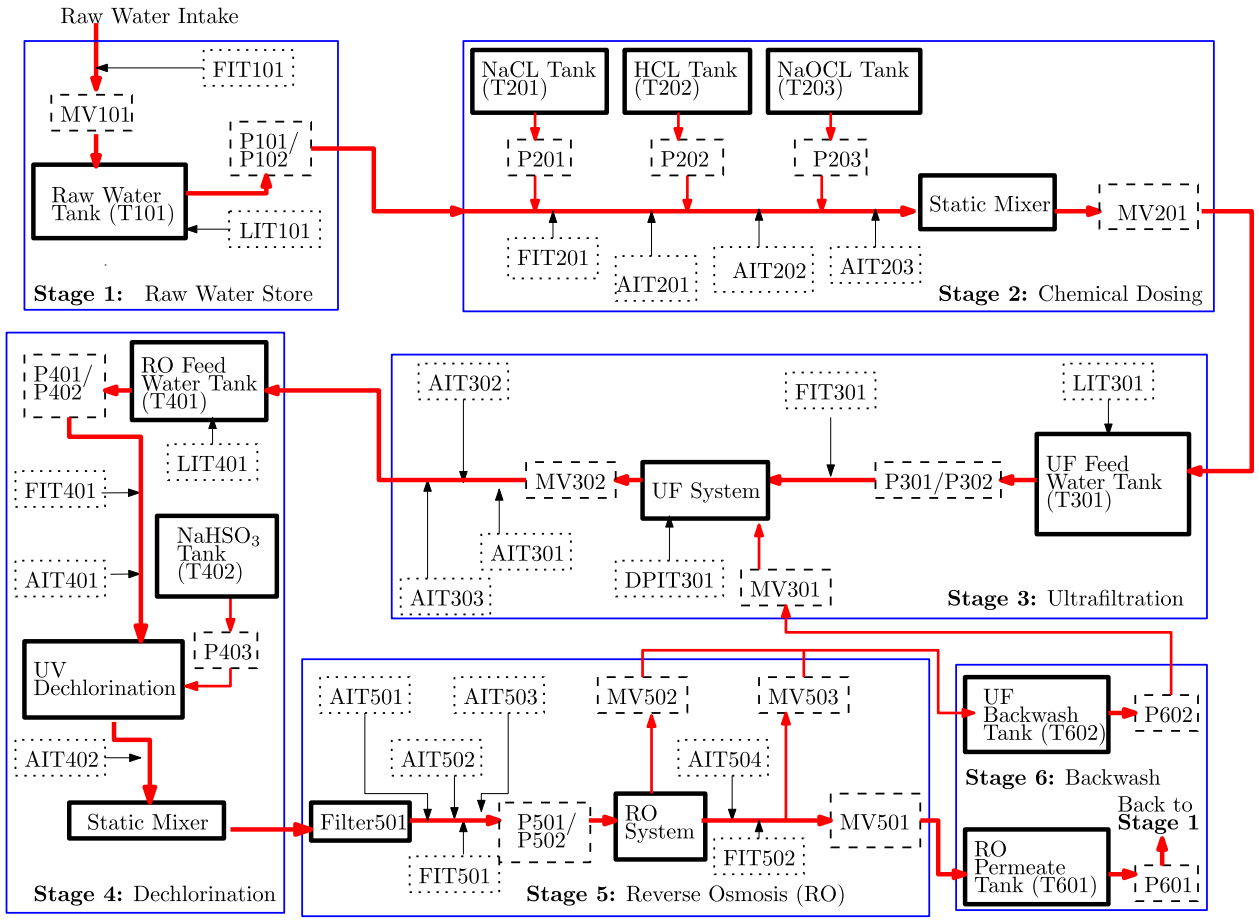


Fig. 1 Overview of the six stages of SWaT. Thick red arrows indicate the flow of water; dotted and dashed rectangles respectively indicate sensors and actuators; thick black rectangles indicate tanks and sub-systems; and blue rectangles indicate the stages

workstation connects a human-machine interface to all of the PLCs, facilitating monitoring and control of the plant by human operators. The physical state of SWaT, as observed by the sensors, is recorded by a historian server at pre-specified intervals. A SWaT dataset is available, consisting of all the data recorded by this server over a period of several days, including a few during which the testbed was subjected to attacks (iTrust Labs: Datasets 2020; Goh et al. 2016).

An overview of the six sub-processes of SWaT is given in Fig. 1. A number of the testbed's 68 sensors and actuators are depicted, with sensors including Flow Indicator Transmitters (FITs), Analyzer Indicator Transmitters (AITs), and Level Indicator Transmitters (LITs). Actuators include Motorised Valves (MVs) for controlling the inflow of water into tanks and Pumps (Ps) for pumping it out. Note that each stage is controlled by a dedicated PLC (not shown in the figure).

Stage One. This is the first stage of SWaT, consisting of a raw water tank (T-101), connected to a motorised valve (MV-101) that controls the inflow of raw water. An electromagnetic flow transmitter (FIT-101) reads the flow rate of this water, and sends it to the PLC. Pump P-101 transfers water from T-101 into the chemical dosing process of stage two. The operation of P-101 is interlocked to the level indicator transmitter (LIT-301) in tank T-301 of stage three.

Stage Two. Chemical dosing is applied in this stage. The chemical properties of the incoming raw water are measured using analyser indicator transmitters AIT-201, AIT-202, and AIT-203. This information is used by the PLC to control pumps P-201, P-202, and P-203, adjusting the dosing and thus the water's chemical properties before it enters stage three.

Stage Three. Ultrafiltration (UF) is performed in this stage. Raw water, after being dosed with chemicals in stage two, is fed into a UF unit. The operation of P-301 is interlocked with the level indicator transmitter LIT-401 for the reverse osmosis (RO) feed water tank (T-401) in stage four. Thus, P-301 is stopped when the water level

in T-401 is high, but when the water level reaches the low marker, P-301 is turned on, and MV-302 is opened. Flow meter FIT-301 measures the incoming flow rate to the UF unit. The differential pressure indicator transmitter (DPIT) continuously monitors the difference in inlet pressure and outlet pressure. If the UF membranes are clogged, the DPIT triggers an alarm, and a backwash sequence begins in stage six. AIT-301, AIT-302, and AIT-303 measure and transmit (to the PLC) various chemical parameters of the water entering the UF feed water tank T-301.

Stage Four. De-chlorination is performed in this stage: any free chlorine in the water coming out of the UF unit is removed using a combination of an ultraviolet dechlorinator and sodium bisulphate. P-401 is started when T-401 reaches the high marker, moving water through the de-chlorinator unit. The hardness analyser (AIT-401) monitors and reports the level of hardness to avoid scaling within the RO system.

Stage Five. Reverse osmosis (RO) is applied in stage five. The RO system is designed to provide bulk reduction of inorganic impurities. The RO permeate stream is channelled to the RO permeate tank (T-601) when MV-501 is opened. Before reaching the tank, the conductivity analyser (AIT-504) measures water conductivity, and if above the threshold, water is diverted to a reject tank T-602 by opening valve MV-503. The rejected water is used to clean the UF membranes in the backwash process. RO permeate pump P-601 recycles water from T-601 back to T-101.

Stage Six. Finally, stage six consists of a backwash process. UF membranes need cleaning to remove solid particles. This cleaning is achieved through the backwash process, which is programmed to start every 30 min. It is also started when the pressure drop across the membrane goes above a pre-set threshold. The rejected RO water from tank T-602 is moved through the UF unit by starting pump P-602.

WADI testbed

The Water Distribution (WADI) testbed (Ahmed et al. 2017) is a scaled-down version of a typical water distribution network, designed to account for varying patterns of water demands (e.g. peak vs off-peak), and support research into ways of mitigating attacks that might otherwise cut off the water supplies of real consumers. WADI consists of three distinct stages, each controlled by a National Instruments PLC. The first stage consists of two 2500 litre water tanks which receive treated water from an external source. In the second stage, this water is fed through to two elevated reservoirs, which are configured to supply six consumer tanks based on a pre-set pattern of demand. Finally, in the third stage, unused water is fed into a return water tank, which can then be pumped back to the first stage to be re-used.

Figure 2 provides an overview of the WADI's three stages, as well as the main sensors and actuators involved. While the electronics involved are all based on different hardware from that of SWaT, we use a similar naming convention: LT for Level Transmitters, AIT for Analyser Indication Transmitters, FIT for Flow Indication Transmitters, PIT for Pressure Indication Transmitters, LS for Level Switches, P for Pumps, MV for Motorised Valves, MCV for Modulating Control Valves, and SV for Solenoid Valves. Each component is named according to its stage, type, and index: `stage-type-index`. For example, component 2-MV-001 is a motorised valve in stage two.

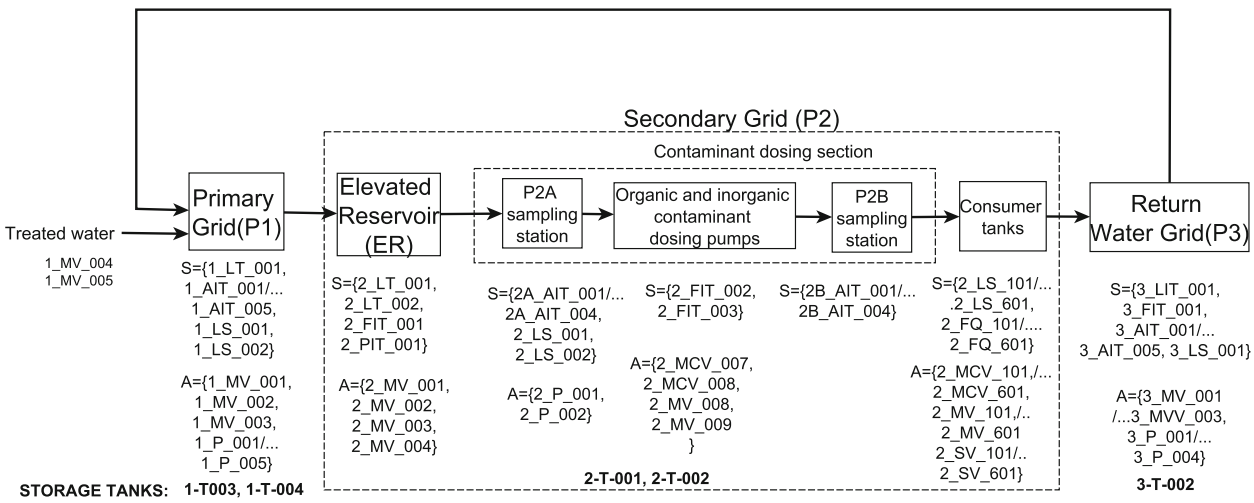


Fig. 2 Overview of the three stages of WADI. Solid arrows indicate the flow of water and the sequence of processes; component names are of the form `stage-type-index`, e.g. 1-LT-001 is a Level Indication Transmitter in stage one

Attacks

To evaluate the defence mechanisms developed in this work, we must first clarify exactly what we mean by a CPS *attack*. We define an attack as a tuple $\langle T, C, O, D, L \rangle$ of five components: T, a *type*; C, the subset of components being targeted (e.g. pump P-101); O, the objective; D, the description of the attacker's steps (or actions); and L, the initial (or launch) state of the attack. In the context of SWaT and WADI, we consider four types of attacks:

- *Single attack, one stage (SAOS)*: one component is attacked, and the impact is observable within one stage only;
- *Single attack, multiple stages (SAMS)*: one component is attacked, but the impact is observable within more than one stage;
- *Multiple attacks, one stage (MAOS)*: multiple components are attacked, affecting one stage only;
- *Multiple attacks, multiple stages (MAMS)*: multiple components are attacked, affecting multiple stages of the CPS.

Thus, $T \in \{\text{SAOS}, \text{SAMS}, \text{MAOS}, \text{MAMS}\}$, and we aim to cover attacks of multiple different types to test the effectiveness of our defence mechanisms across a variety of scenarios. For simplicity, we will describe the steps of attackers (D) informally, but precisely, using natural language in our case studies.

Our attack classification is similar to that of Adepu and Mathur (Adepu and Mathur 2016), in that we distinguish between single- and multi-point attacks. However, our model emphasises the *results* of the attacks (rather than just the steps themselves), classifying attacks according to whether they impact devices in one or multiple stages of the CPS. We shall use this classification for the attacks we consider in our Evaluation.

Our approach: a design-level analysis

The overarching goal of our work is to define a systematic *design-level* method for identifying—possibly before a system is built—dependencies between CPS components, helping designers to understand the potential impact of compromised components, and to identify weak points that should be redesigned or mitigated by other means (e.g. security policies, access rights, physical keys). In this paper, we focus on one particular application of this analysis: deriving *invariants*, i.e. mathematical relations over the dependent components. These can be included as part of the implemented CPS's defence mechanisms, in the form of *invariant checkers*.

Figure 3 summarises the steps of our approach. First, a design-level analysis based on *axiomatic design principles* is used to identify groups of dependent components in the CPS design. Second, *invariants are derived* for those groups of components, guided by domain knowledge and/or process graphs. Finally, we *construct invariant checkers* that can be used as defence mechanisms for an implementation of the CPS design. We expand upon these broad steps in the following three subsections, demonstrating them on SWaT, our principal case study, as well as WADI.

Step one: axiomatic design process

Our analysis is based on the principles of *axiomatic design*, a systems design methodology developed by Nam Pyo Suh (Suh 2001), that uses matrix methods to systematically analyse the transformation of customer needs (e.g. "build a six-stage water treatment plant") into functional requirements (e.g. "track water level of tanks"), design parameters (e.g. sensing mechanisms), and process variables (e.g. value ranges). The objective of the theory is to create a scientific base for the design

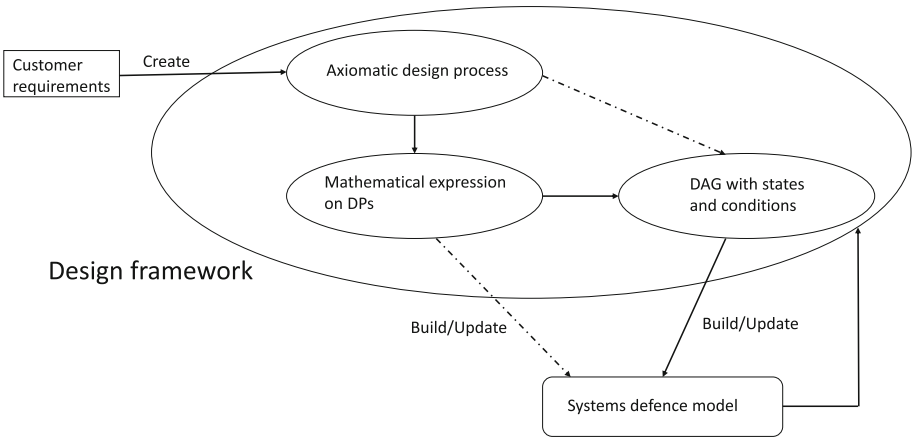


Fig. 3 High-level overview of our design-level analysis for obtaining invariant checkers

process by building upon a suite of fundamental theories from logic and rationale thinking. Researchers have applied this theory in areas such as manufacturing (Matt 2012; Zhu et al. 2008) and software development (Kandjani et al. 2015; Mohsen and Cekecek 2000).

In axiomatic design, functional requirements (FRs) express *what* we want to achieve, i.e. the specific behaviours we want from the design. Design parameters (DPs) are elements of the physical design that are chosen to *realise* the FRs. Finally, process variables (PVs) are elements of the process design controlling the DPs (e.g. continuous or discrete values that are characterising the process). Matrix methods are used by the designer to map FRs to DPs in the physical domain. For example, suppose that the top-level of a design involved two FRs and two DPs. These can then be related using the following matrix:

$$\begin{bmatrix} FR_1 \\ FR_2 \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \begin{bmatrix} DP_1 \\ DP_2 \end{bmatrix}$$

The square matrix is a binary (or Boolean) matrix, indicating the coupling between FRs and DPs. After identifying the couplings at a high level (e.g. where one DP might represent *all* pumps), the designer would decompose the FRs and DPs further (e.g. with one DP representing exactly *one* of the pumps) until achieving a fine-grained set of dependencies in the design. The decomposed matrices can then be subjected to analyses to assess and mitigate the effects of coupling.

In our work, rather than using axiomatic design to build a CPS from the ground up, we take the core principles of the approach and apply them to an *existing* CPS design in order to identify dependencies. We take DPs to be CPS components such as tank level sensors or motorised valves, each of which can function within specific values of PVs. Furthermore, we differ from conventional axiomatic design in the type of coupling: instead of considering physical coupling between FRs and DPs, we consider *information state coupling* when decomposing the matrix equations. This paves the way for a simple and high-level design analysis to uncover the relations that exist between DPs in normal CPS behaviour.

Applied to SWaT

Based on the requirements of SWaT, a top-level design decomposition is given in Table 1. By the axiomatic design principles, this first level should be a functionally *uncoupled* design guaranteeing that each DP satisfies exactly one FR. This is reflected by the matrix of Eq. (1), a diagonal matrix in which each FR is related only to its given DP from Table 1.

$$\begin{bmatrix} FR_1 \\ FR_2 \\ FR_3 \\ FR_4 \\ FR_5 \\ FR_6 \\ FR_7 \\ FR_8 \end{bmatrix} = \begin{bmatrix} X & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & X & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & X & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & X & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & X & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & X & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & X & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & X \end{bmatrix} \begin{bmatrix} DP_1 \\ DP_2 \\ DP_3 \\ DP_4 \\ DP_5 \\ DP_6 \\ DP_7 \\ DP_8 \end{bmatrix} \quad (1)$$

Next, the engineer analyses the DPs against the FRs and updates the corresponding Boolean value of the matrix if there is an *information state coupling* between them. Inserting this information state coupling into Eq. (1) results in Eq. (2), where $\otimes$ (or X on the diagonal) indicates some dependencies, and a zero (0) denotes the absence of them. Note that for simplicity, we assume these dependencies to be symmetric, e.g. if DP7 is (information state) coupled with FR2, then FR2 is coupled with DP7.

$$\begin{bmatrix} FR_1 \\ FR_2 \\ FR_3 \\ FR_4 \\ FR_5 \\ FR_6 \\ FR_7 \\ FR_8 \end{bmatrix} = \begin{bmatrix} X & \otimes & \otimes & \otimes & 0 & \otimes & \otimes & 0 \\ \otimes & X & \otimes & 0 & 0 & 0 & \otimes & 0 \\ \otimes & \otimes & X & 0 & 0 & 0 & \otimes & 0 \\ \otimes & 0 & 0 & X & \otimes & 0 & 0 & 0 \\ 0 & 0 & 0 & \otimes & X & 0 & 0 & \otimes \\ \otimes & 0 & 0 & 0 & 0 & X & \otimes & 0 \\ \otimes & \otimes & \otimes & 0 & 0 & \otimes & X & 0 \\ 0 & 0 & 0 & 0 & \otimes & 0 & 0 & X \end{bmatrix} \begin{bmatrix} DP_1 \\ DP_2 \\ DP_3 \\ DP_4 \\ DP_5 \\ DP_6 \\ DP_7 \\ DP_8 \end{bmatrix} \quad (2)$$

Equation (2) shows that FR7 ("direct flow of water") is coupled with DP1–3, DP6, and DP7. This is justified by a number of different behaviours in the design. For example, if a tank level (DP2) is low and the corresponding pump (DP1) is on, then a motorised valve (DP7) is opened. Note however that Eq. (2) presents design information that is at a very high and broadly defined level. For instance, FR3—"track flow rate of water"—relates to

Table 1 Top-level decomposition of SWaT

Functional Requirements (FRs)	Design Parameters (DPs)	Process Variables (PVs)
FR1: Feed water to water tanks/systems	DP1: DOL/VSD Pumps	Features - Switch (On/Off) and Speed
FR2: Track level of water in tanks	DP2: Sensing mechanisms	Value range
FR3: Track flow rate of water	DP3: EMF sensors	Value range
FR4: Monitor chemical properties of water	DP4: Chemical properties sensors	Value range
FR5: Feed chemicals to water	DP5: Dosing Pumps	Switch (On/Off)
FR6: Track water pressure	DP6: Pressure sensors	Value range
FR7: Direct flow of water	DP7: Motorised valves	Switch (On/Off)
FR8: Track level of chemicals in tanks	DP8: Level switch	Value range

multiple different locations and flow sensors (DP3) in SWaT. Another example is FR1—"feed water to water tanks/systems"—when in reality, there are multiple water pumps (DP1) in six different stages of SWaT. In order to derive meaningful invariants that relate concrete components of the CPS, our method requires that the top-level design of Eq. (2) is iteratively decomposed towards a *point-to-point mapping* between each FR and DP. For illustration purposes, such a mapping is shown in the third-level decomposition of Table 2.

For simplicity of presentation, rather than use a full point-to-point mapping, we decompose the eight FRs of Eq. (2) into the 30 FRs of the second-level decomposition in Table 3. This is much more concrete than the top-level decomposition as it factors in particular sensors and actuators from different stages, but groups some of them together for convenience (e.g. P-101 and P-102 are the same DP, as the latter pump is simply a backup for the former).

Next, the equation in Fig. 4 is constructed by mapping down the information-state coupling from Eq. (2) and adjusting according to the FRs of Table 3. At this second level, we use the notational format FR*i.j* and DP*i.j* with *i* denoting the number from the top-level design and *j* the number from the second-level.

Finally, Table 4 presents the dependencies between DPs for each second-level FR of Table 3, using the information-state coupling as identified by the CPS designer in the equation of Fig. 4. These sets of dependencies identified in the design can then be used to construct invariants (see Step Two and Three).

Applied to WADI

Similar to SWaT, our method requires a functionally uncoupled top-level decomposition in which each DP satisfies exactly one FR. Following the design of Palleti et al. (Palleti et al. 2018), we decompose WADI into eight FRs and DPs, which are presented together in Table 5. An analysis of the requirements enables the designer to derive the matrix below (Eq. 3), in a similar way to SWaT.

$$\begin{bmatrix} FR_1 \\ FR_2 \\ FR_3 \\ FR_4 \\ FR_5 \\ FR_6 \\ FR_7 \\ FR_8 \end{bmatrix} = \begin{bmatrix} X & \otimes & 0 & 0 & 0 & 0 & 0 & \otimes \\ \otimes & X & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & X & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & X & 0 & 0 & 0 & \otimes \\ 0 & 0 & 0 & 0 & X & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & X & \otimes & \otimes \\ 0 & 0 & 0 & 0 & 0 & \otimes & X & 0 \\ \otimes & 0 & 0 & \otimes & 0 & \otimes & 0 & X \end{bmatrix} \begin{bmatrix} DP_1 \\ DP_2 \\ DP_3 \\ DP_4 \\ DP_5 \\ DP_6 \\ DP_7 \\ DP_8 \end{bmatrix} \quad (3)$$

Further detailed, lower levels of decomposition follow the same procedure as in SWaT, and enable the designer to flesh out the relations across the different components of the system. For WADI, a second-level decomposition leads to 25 FRs and DPs.

Step two: deriving invariants

Having identified sets of dependent DPs, the second step of our approach is to identify the *invariant relationships* that exist between them, i.e. relational properties that always hold under normal operating conditions. For example, if two DPs have been identified as dependent, then we are seeking to identify the combinations of states they are permitted to be in, with any other combinations representing anomalous behaviour.

Design engineers can derive invariants for these sets of DPs in a number of different ways. We propose a systematic approach supported by visual aids. In particular, we propose the construction of mathematical expressions over the DP states (e.g. "if DP_1 is on and DP_2 is low then the system is anomalous"). These expressions can be constructed directly (e.g. in a tabulated format) by the designer or engineer using domain expertise, or can be guided by visual aids such as *process graphs*. These are based on the directed acyclic graph (DAG) concept of graph theory (Bondy and Murty 2008) and can assist the designer/engineer to visually analyse the relationship between states of DPs (represented as nodes) and the conditions that cause them to change (represented as edges).

Applied to SWaT

To illustrate this step, consider, row FR7.1 of Table 4: here, DP7.1.1 expresses that motorised valve MV-101 has a dependency on LIT-101 (and vice versa). Using knowledge of the plant's design, we then construct state expressions to characterise their invariant relationship, i.e. the combinations of states they will always be in during normal operation. These state expressions consist of combinations of DP states and the corresponding labels of anomalous or non-anomalous. In the case of LIT-101, we use the low/high thresholds to determine two discrete states of interest, then relate them against the possible discrete values of MV-101 (open or closed). These equations are given in Table 6 (Invariant #1), where MV-101 denotes that the valve is open, !MV-101 denotes that the valve is closed, LIT-101 denotes that the tank level is high, and !LIT-101 denotes that the tank is low. The table also contains labels reflecting the judgement of an engineer as to which of these four combinations reflect anomalous configurations. For example, if the valve is open and the tank level is high, this is anomalous as it could cause the tank to overflow.

Figure 5 shows how the normal and anomalous cases of Invariant #1 would be depicted as process graphs. The arrows in parts (b) and (c) indicate changes of state, here triggered by LIT-101 reporting a reading below one of its low thresholds (Low or LowLow), or above one of its high thresholds (High or HighHigh). The idea is that the designer traverses through the different paths to explore

Table 2 Third-level decomposition of SWaT's FRs and DPs

Functional Requirements (FRs)	Design Parameters (DPs)	Process Variables (PVs)
FR1.1.1: Feed raw water from stage one to UF feed tank in stage three using pump P-101	DP1.1.1: P-101	On/Off
FR1.1.2: Feed raw water from stage one to UF feed tank in stage three using pump P-102	DP1.1.2: P-102	On/Off
FR1.2.1: Feed water from stage three to RO feed tank in stage four using pump P-301	DP1.2.1: P-301	On/Off
FR1.2.2: Feed water from stage three to RO feed tank in stage four using pump P-302	DP1.2.2: P-302	On/Off
FR1.3.1: Pump water from stage four through de-chlorination system using pump P-401	DP1.3.1: P-401	On/Off
FR1.3.2: Pump water from stage four through de-chlorination system using pump P-402	DP1.3.2: P-402	On/Off
FR1.4.1: Pump (VSD) water from stage five to tanks in stage six using pump P-501	DP1.4.1: P-501	On/Off
FR1.4.2: Pump (VSD) water from stage five to tanks in stage six using pump P-502	DP1.4.2: P-502	On/Off
FR3.5.1: Compute RO membrane inlet flow meter in stage five	DP3.5.1: FIT-501	$0 \leq \alpha \leq \max_{k1}$
FR3.5.2: Compute RO permeate flow meter in stage five	DP3.5.2: FIT-502	$0 \leq \alpha \leq \max_{k2}$
FR3.5.3: Compute RO reject flow meter in stage five	DP3.5.3: FIT-503	$0 \leq \alpha \leq \max_{k3}$
FR3.5.4: Compute RO re-circulation flow meter in stage five	DP3.5.4: FIT-504	$0 \leq \alpha \leq \max_{k4}$
FR4.1.1: Calculate chemical dosing conductivity of water in stage two	DP4.1.1: AIT-201	$0 \leq \alpha \leq \max_{m1}$
FR4.1.2: Calculate chemical dosing pH of water in stage two	DP4.1.2: AIT-202	$0 \leq \alpha \leq \max_{m2}$
FR4.1.3: Calculate chemical dosing ORP of water in stage two	DP4.1.3: AIT-203	$0 \leq \alpha \leq \max_{m3}$
FR4.1.4: Calculate UF permeate pH of water in stage three	DP4.1.4: AIT-301	$0 \leq \alpha \leq \max_{m4}$
FR4.1.5: Calculate UF permeate ORP of water in stage three	DP4.1.5: AIT-302	$0 \leq \alpha \leq \max_{m5}$
FR4.1.6: Calculate UF permeate conductivity of water in stage three	DP4.1.6: AIT-303	$0 \leq \alpha \leq \max_{m6}$
FR4.1.7: Calculate RO feed hardness of water in stage four	DP4.1.7: AIT-401	$0 \leq \alpha \leq \max_{m7}$
FR4.1.8: Calculate RO ORP of water in stage four	DP4.1.8: AIT-402	$0 \leq \alpha \leq \max_{m8}$
FR4.1.9: Calculate RO feed pH of water in stage five	DP4.1.9: AIT-501	$0 \leq \alpha \leq \max_{m9}$
FR4.1.10: Calculate RO feed ORP of water in stage five	DP4.1.10: AIT-502	$0 \leq \alpha \leq \max_{m10}$
FR4.1.11: Calculate RO feed conductivity of water in stage five	DP4.1.11: AIT-503	$0 \leq \alpha \leq \max_{m11}$
FR4.1.12: Calculate RO permeate conductivity of water in stage five	DP4.1.12: AIT-504	$0 \leq \alpha \leq \max_{m12}$
FR5.1.1: Feed NaCl dosing in stage two using pump P-201	DP5.1.1: P-201	On/Off
FR5.1.2: Feed NaCl dosing in stage two using pump P-202	DP5.1.2: P-202	On/Off
FR5.1.3: Feed HCl dosing in stage two using pump P-203	DP5.1.3: P-203	On/Off
FR5.1.4: Feed HCl dosing in stage two using pump P-204	DP5.1.4: P-204	On/Off
FR5.1.5: Feed NaOCl dosing in stage two using pump P-205 (FAC)	DP5.1.5: P-205	On/Off
FR5.1.6: Feed NaOCl dosing in stage two using pump P-206 (FAC)	DP5.1.6: P-206	On/Off
FR5.1.7: Feed NaOCl dosing to stage three UF cleaning using pump P-207 (UF)	DP5.1.7: P-207	On/Off
FR5.1.8: Feed NaOCl dosing to stage three UF cleaning using pump P-208 (UF)	DP5.1.8: P-208	On/Off
FR5.1.9: Feed NaHSO3 dosing in stage four using pump P-403	DP5.1.9: P-403	On/Off
FR5.1.10: Feed NaHSO3 dosing in stage four using pump P-404	DP5.1.10: P-404	On/Off
FR7.1.1: Direct raw water inlet in stage one	DP7.1.1: MV-101	On/Off
FR7.1.2: Direct water flow in stage two	DP7.1.2: MV-201	On/Off
FR7.1.3: Direct UF backwash in stage three	DP7.1.3: MV-301	On/Off
FR7.1.4: Direct UF feed water in stage three	DP7.1.4: MV-302	On/Off
FR7.1.5: Direct UF backwash drain in stage three	DP7.1.5: MV-303	On/Off
FR7.1.6: Direct UF drain in stage three	DP7.1.6: MV-304	On/Off
FR7.1.7: Direct RO permeate in stage five	DP7.1.7: MV-501	On/Off
FR7.1.8: Direct RO backwash in stage five	DP7.1.8: MV-502	On/Off
FR7.1.9: Direct RO permeate reject in stage five	DP7.1.9: MV-503	On/Off
FR7.1.10: Direct RO reject in stage five	DP7.1.10: MV-504	On/Off

Table 3 Second-level decomposition of SWaT's FRs and DPs

Functional Requirements (FRs)	**Design Parameters (DPs)**	Process Variables (PVs)
FR1.1: Pump raw water from stage one to UF feed tank in stage three	DP1.1: P-101,P-102	On/Off
FR1.2: Pump water from stage three to RO feed tank in stage four	DP1.2: P-301,P-302	On/Off
FR1.3: Pump water from stage four through de-chlorination system	DP1.3: P-401,P-402	On/Off
FR1.4: Pump (VSD) water from stage five to tanks in stage six	DP1.4: P-501,P-502	On/Off
FR1.5: Pump water from RO permeate tank to raw water tank in stage one	DP1.5: P-601	On/Off
FR1.6: Pump water for UF backwash system	DP1.6: P-602	On/Off
FR1.7: Pump water for RO/UF cleaning	DP1.7: P-603	On/Off
FR2.1: Determine water level in raw water tank of stage one	DP2.1: LIT-101	$0 \leq \alpha \leq \max_a$
FR2.2: Determine water level in UF feed tank of stage three	DP2.2: LIT-301	$0 \leq \alpha \leq \max_b$
FR2.3: Determine water level in RO feed tank of stage four	DP2.3: LIT-401	$0 \leq \alpha \leq \max_c$
FR2.4: Determine water level in RO permeate tank of stage six	DP2.4: LS-601	$\text{Low}_d \leq \alpha \leq \text{High}_d$
FR2.5: Determine water level in UF backwash tank of stage six	DP2.5: LS-602	$\text{Low}_e \leq \alpha \leq \text{High}_e$
FR2.6: Determine water level in CIP tank of stage six	DP2.6: LS-603	$\text{Low}_f \leq \alpha \leq \text{High}_f$
FR3.1: Measure raw water flow rate in stage one	DP3.1: FIT-101	$\text{Low}_g \leq \alpha \leq \text{High}_g$
FR3.2: Measure water flow rate in stage two	DP3.2: FIT-201	$\text{Low}_h \leq \alpha \leq \text{High}_h$
FR3.3: Measure water flow rate in stage three	DP3.3: FIT-301	$\text{Low}_i \leq \alpha \leq \text{High}_i$
FR3.4: Measure water flow rate in stage four	DP3.4: FIT-401	$\text{Low}_j \leq \alpha \leq \text{High}_j$
FR3.5: Measure water flow rate in stage five	DP3.5: FIT-501,FIT-502,FIT-503,FIT-504	$\text{Low}_k \leq \alpha \leq \text{High}_k$
FR3.6: Measure water flow rate in stage six	DP3.6: FIT-601	$\text{Low}_l \leq \alpha \leq \text{High}_l$
FR4.1: Calculate chemical properties of water	DP4.1: AIT-201,AIT-202,AIT-203,AIT-301,AIT-302,AIT-303 AIT-401,AIT-402,AIT-501,AIT-502,AIT-503,AIT-504	$\text{Low}_m \leq \alpha \leq \text{High}_m$
FR5.1: Pump chemicals to water	DP5.1: P-201,P-202,P-203,P-204,P-205,P-206,P-207,P-208, P-403,P-404	On/Off
FR6.1: Measure UF filter differential pressure	DP6.1: DPIT-301	$0 \leq \alpha \leq \max_n$
FR6.2: Measure RO membrane inlet pressure	DP6.2: PIT-501	$0 \leq \alpha \leq \max_o$
FR6.3: Measure RO membrane pressure	DP6.3: PIT-502	$0 \leq \alpha \leq \max_p$
FR6.4: Measure RO reject pressure	DP6.4: PIT-503	$0 \leq \alpha \leq \max_q$
FR7.1: Control water flow direction	DP7.1: MV-101,MV-201,MV-301,MV-302,MV-303,MV-304, MV-501,MV-502,MV-503,MV-504	On/Off
FR8.1: Determine NaCl level in NaCl tank of stage two	DP8.1: LS-201	$\text{Low}_r \leq \alpha \leq \max_r$
FR8.2: Determine HCl level in HCl tank of stage two	DP8.2: LS-202	$\text{Low}_s \leq \alpha \leq \max_s$
FR8.3: Determine NaOCl level in NaOCl tank of stage two	DP8.3: LS-203	$\text{Low}_t \leq \alpha \leq \max_t$
FR8.4: Determine NaHSO3 level in NaHSO3 tank of stage four	DP8.4: LS-401	$\text{Low}_u \leq \alpha \leq \max_u$

$$
\begin{bmatrix} FR1.1 \\ FR1.2 \\ FR1.3 \\ FR1.4 \\ FR1.5 \\ FR1.6 \\ FR1.7 \\ FR2.1 \\ FR2.2 \\ FR2.3 \\ FR2.4 \\ FR2.5 \\ FR2.6 \\ FR3.1 \\ FR3.2 \\ FR3.3 \\ FR3.4 \\ FR3.5 \\ FR3.6 \\ FR4.1 \\ FR5.1 \\ FR6.1 \\ FR6.2 \\ FR6.3 \\ FR6.4 \\ FR7.1 \\ FR8.1 \\ FR8.2 \\ FR8.3 \\ FR8.4 \end{bmatrix} = \begin{bmatrix}
X & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & \otimes & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & X & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & \otimes & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & \otimes & 0 & \otimes & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & 0 & X & \otimes & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & \otimes & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \otimes & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & \otimes & 0 & 0 & 0 & \otimes & \otimes & \otimes & \otimes & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & X & 0 & 0 & \otimes & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\otimes & 0 & 0 & 0 & \otimes & 0 & 0 & X & 0 & 0 & \otimes & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
\otimes & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & \otimes & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & \otimes & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \otimes & 0 & 0 & \otimes & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
\otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \otimes & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
\otimes & \otimes & \otimes & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & X & 0 & 0 & 0 & 0 & 0 & \otimes & \otimes & \otimes & \otimes \\
0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X & 0 & \otimes & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X & \otimes & 0 & 0 & 0 & 0 \\
\otimes & \otimes & 0 & \otimes & 0 & \otimes & 0 & \otimes & \otimes & \otimes & 0 & \otimes & \otimes & \otimes & \otimes & \otimes & 0 & \otimes & \otimes & 0 & 0 & \otimes & 0 & \otimes & \otimes & X & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & X & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X
\end{bmatrix} \begin{bmatrix} DP1.1 \\ DP1.2 \\ DP1.3 \\ DP1.4 \\ DP1.5 \\ DP1.6 \\ DP1.7 \\ DP2.1 \\ DP2.2 \\ DP2.3 \\ DP2.4 \\ DP2.5 \\ DP2.6 \\ DP3.1 \\ DP3.2 \\ DP3.3 \\ DP3.4 \\ DP3.5 \\ DP3.6 \\ DP4.1 \\ DP5.1 \\ DP6.1 \\ DP6.2 \\ DP6.3 \\ DP6.4 \\ DP7.1 \\ DP8.1 \\ DP8.2 \\ DP8.3 \\ DP8.4 \end{bmatrix}
$$

Fig. 4 Equation characterising the second-level information-state couplings

the different behaviours, before mapping them down to equivalent state expressions.

We exemplify the process of deriving invariants with two more examples. Consider DP3.5.1, in row FR3.5 of Table 4, which denotes a set of dependencies between flow sensor FIT-501, pump P-401, and pump P-402. We identify two states of interest for each component: water flowing (FIT-501>0) versus no water flowing; and pump on (P-401) versus pump off (!P-401). Using knowledge of the design, we enumerate the mathematical state expressions in Table 7 (Invariant #2).

Finally, consider the more complex case of DP7.1.2, in row FR7.1 of Table 4. The four devices are found across *different stages* of SWaT: the pumps P-101 and P-102 in stage one, the valve MV-201 in stage two, and the tank level sensor LIT-301 in stage three. The communication between these components is summarised in Fig. 6a. The sensor reading of LIT-301 (in tank T-301) is transmitted from PLC3 to PLC2. When the water level in the tank is low or very low, PLC2 sends a signal to close valve MV-201. Next, it transmits a signal to start pump P-101 or P-102 via PLC1. By doing so, water is pumped from stage one of SWaT through MV-201 in stage two through to tank T-301 in stage three. These are typical normal cases, i.e. without an attack. The process graph in Fig. 6b presents exactly three different paths to represent normal behaviour, whereas Fig. 6c depicts other, anomalous cases. For example, if the tank is falsely reported as being in a low state (e.g. by an attacker), MV-201 is anomalously opened, and P-101 or P-102 are switched on. The water starts flowing into the tank, but since its *actual* state is high, it may start to overflow and cause damage. The mathematical state expressions corresponding to these paths are summarised in Table 8 (Invariant #3).

Note that for SWaT, we are able to assume that all DPs have one of two states, meaning that the total number of state expressions to analyse will be 2^n, where n is the number of components involved. Though n is typically of a manageable magnitude for SWaT, for other systems, the number of equations may grow too large to manually complete. We address this in Step Three by allowing automated approaches (e.g. decision tree learning) to generalise from a strict subset of the state expressions that engineers have been able to identify.

Applied to WADI

After decomposing the design of WADI using axiomatic design principles, we are able to identify the following sets

Table 4 Linking SWaT's second-level decomposition of FRs to dependent DPs

Functional Requirements (FRs)	Design Parameters (DPs)
FR1.1: Pump raw water from stage one to UF feed tank in stage three	DP1.1: P-101, P-102
	DP1.1.1: P-101; Other DPs: DP2.1(LIT-101), DP2.2(LIT-301), DP7.1(MV-201)
	DP1.1.2: P-102; Other DPs: DP2.1(LIT-101), DP2.2(LIT-301), DP7.1(MV-201)
FR1.2: Pump water from stage three to RO feed tank in stage four	DP1.2: P-301, P-302
	DP1.2.1: P-301; Other DPs: DP2.2(LIT-301), DP2.3(LIT-401), DP7.1(MV-302)
	DP1.2.2: P-302; Other DPs: DP2.2(LIT-301), DP2.3(LIT-401), DP7.1(MV-302)
FR1.3: Pump water from stage four through de-chlorination system	DP1.3: P-401, P-402
	DP1.3.1: P-401; Other DPs: DP1.4(P-501,P-502), DP2.3(LIT-401)
	DP1.3.2: P-402; Other DPs: DP1.4(P-501,P-502), DP2.3(LIT-401)
FR1.4: Pump (VSD) water from stage five to tanks in stage six	DP1.4: P-501, P-502
	DP1.4.1: P-501; Other DPs: DP1.3(P-401,P-402), DP7.1(MV-501)
	DP1.4.2: P-502; Other DPs: DP1.3(P-401,P-402), DP7.1(MV-501)
FR1.5: Pump water from RO permeate tank to raw water tank in stage one	DP1.5: P-601; Other DPs: DP2.1(LIT-101), DP2.4(LS-601)
FR1.6: Pump water for UF backwash system	DP1.6: P-602; Other DPs: DP2.5(LS-602), DP7.1(MV-301)
FR1.7: Pump water for RO/UF cleaning	DP1.7: P-603; Other DPs: DP2.6(LS-603)
FR2.1: Determine water level in raw water tank of stage one	DP2.1: LIT-101; Other DPs: DP1.1(P-101,P-102), DP1.5(P-601), DP2.4(LS-601), DP7.1(MV-101)
FR2.2: Determine water level in UF feed tank of stage three	DP2.2: LIT-301; Other DPs: DP1.1(P-101,P-102), DP1.2(P-301,P-302), DP7.1(MV-201)
FR2.3: Determine water level in RO feed tank of stage four	DP2.3: LIT-401; Other DPs: DP1.2(P-301,P-302), DP1.3(P-401,P-402), DP7.1(MV-302)
FR2.4: Determine water level in RO permeate tank of stage six	DP2.4: LS-601; Other DPs: DP1.5(P-601), DP2.1(LIT-101)
FR2.5: Determine water level in UF backwash tank of stage six	DP2.5: LS-602; Other DPs: DP1.6(P-602), DP7.1(MV-301)
FR2.6: Determine water level in CIP tank of stage six	DP2.6: LS-603; Other DPs: DP1.7(P-603), DP7.1(MV-301)
FR3.1: Measure raw water flow rate in stage one	DP3.1: FIT-101; Other DPs: DP2.1(LIT-101), DP7.1(MV-101)
FR3.2: Measure water flow rate in stage two	DP3.2: FIT-201; Other DPs: DP1.1(P-101,P-102), DP2.2(LIT-301), DP7.1(MV-201)
FR3.3: Measure water flow rate in stage three	DP3.3: FIT-301; Other DPs: DP1.2(P-301,P-302), DP2.3(LIT-401), DP7.1(MV-302)
FR3.84: Measure water flow rate in stage four	DP3.4: FIT-401; Other DPs: DP1.3(P-401,P-402), DP2.3(LIT-401)
FR3.5: Measure water flow rate in stage five	DP3.5: FIT-501,FIT-502,FIT-503,FIT-504
	DP3.5.1: FIT-501; Other DPs: DP1.3(P-401,P-402)
	DP3.5.2: FIT-502; Other DPs: DP1.4(P-501,P-502)
	DP3.5.3: FIT-503; Other DPs: DP1.4(P-501,P-502)
	DP3.5.4: FIT-504; Other DPs: DP1.3(P-401,P-402)
FR3.6: Measure water flow rate in stage six	DP3.6: FIT-601; Other DPs: DP1.6(P-602), DP2.5(LS-602), DP7.1(MV-301)
FR4.1: Calculate chemical properties of water	DP4.1: AIT-201,AIT-202,AIT-203,AIT-301,AIT-302,AIT-303,
	AIT-401,AIT-402,AIT-501,AIT-502,AIT-503,AIT-504
	DP4.1.1: AIT-201; Other DPs: DP1.1.1(P-101), DP1.1.2(P-102)
	DP4.1.2: AIT-202; Other DPs: DP1.1.1(P-101), DP1.1.2(P-102)
	DP4.1.3: AIT-203; Other DPs: DP1.1.1(P-101), DP1.1.2(P-102)
	DP4.1.4: AIT-301; Other DPs: DP1.2.1(P-301), DP1.2.2(P-302)

Table 4 Linking SWaT's second-level decomposition of FRs to dependent DPs (*Continued*)

Functional Requirements (FRs)	Design Parameters (DPs)
	DP4.1.5: AIT-302; Other DPs: DP1.2.1(P-301), DP1.2.2(P-302)
	DP4.1.6: AIT-303; Other DPs: DP1.2.1(P-301), DP1.2.2(P-302)
	DP4.1.7: AIT-401; Other DPs: DP1.3.1(P-401), DP1.3.2(P-402)
	DP4.1.8: AIT-402; Other DPs: DP1.3.1(P-401), DP1.3.2(P-402)
	DP4.1.9: AIT-501; Other DPs: DP1.3.1(P-401), DP1.3.2(P-402)
	DP4.1.10: AIT-502; Other DPs: DP1.3.1(P-401), DP1.3.2(P-402)
	DP4.1.11: AIT-503; Other DPs: DP1.3.1(P-401), DP1.3.2(P-402)
	DP4.1.11: AIT-504; Other DPs: DP1.4.1(P-501), DP1.4.2(P-502)
FR5.1: Pump chemicals to water	DP5.1: P-201,P-202,P-203,P-204,P-205,P-206,P-207,P-208,P-403,P-404
	DP5.1.1: P-201; Other DPs: DP4.1.1(AIT-201), DP7.1.2(MV-201)
	DP5.1.2: P-202; Other DPs: DP4.1.1(AIT-201), DP7.1.2(MV-201)
	DP5.1.3: P-203; Other DPs: DP4.1.2(AIT-202), DP7.1.2(MV-201)
	DP5.1.4: P-204; Other DPs: DP4.1.2(AIT-202), DP7.1.2(MV-201)
	DP5.1.5: P-205; Other DPs: DP4.1.3(AIT-203), DP7.1.2(MV-201)
	DP5.1.6: P-206; Other DPs: DP4.1.3(AIT-203), DP7.1.2(MV-201)
	DP5.1.7: P-207; Other DPs: DP4.1.5(AIT-302)
	DP5.1.8: P-208; Other DPs: DP4.1.5(AIT-302)
	DP5.1.9: P-403; Other DPs: DP4.1.8(AIT-402)
	DP5.1.10: P-404; Other DPs: DP4.1.8(AIT-402)
FR6.1: Measure UF filter differential pressure	DP6.1: DPIT-301; Other DPs: DP1.2.1(P-301), DP1.2.2(P-302), DP7.1(MV-302)
FR6.2: Measure RO membrane inlet pressure	DP6.2: PIT-501; Other DPs: DP1.4.1(P-501), DP1.4.2(P-502)
FR6.3: Measure RO membrane pressure	DP6.3: PIT-502; Other DPs: DP1.4.1(P-501), DP1.4.2(P-502), DP7.1.7(MV-501),
	DP7.1.9(MV-503)
FR6.4: Measure RO reject pressure	DP6.4: PIT-503; Other DPs: DP1.4.1(P-501), DP1.4.2(P-502), DP7.1.8(MV-502),
	DP7.1.10(MV-504)
FR7.1: Control water flow direction	DP7.1: MV-101,MV-201,MV-301,MV-302,MV-303,MV-304,MV-501,MV-502,MV-503,MV-504
	DP7.1.1: MV-101; Other DPs: DP2.1(LIT-101)
	DP7.1.2: MV-201; Other DPs: DP1.1.1(P-101), DP1.1.2(P-102), DP2.2(LIT-301)
	DP7.1.3: MV-301; Other DPs: DP1.6(P-602), DP2.5(LS-602), DP2.6(LS-603)
	DP7.1.4: MV-302; Other DPs: DP1.2.1(P-301), DP1.2.2(P-302), DP2.3(LIT-401)
	DP7.1.5: MV-303; Other DPs: DP1.2.1(P-301), DP1.2.2(P-302)
	DP7.1.6: MV-304; Other DPs: DP1.2.1(P-301), DP1.2.2(P-302)
	DP7.1.7: MV-501; Other DPs: DP1.4.1(P-501), DP1.4.2(P-502)
	DP7.1.8: MV-502; Other DPs: DP1.4.1(P-501), DP1.4.2(P-502)
	DP7.1.9: MV-503; Other DPs: DP1.4.1(P-501), DP1.4.2(P-502)
	DP7.1.10: MV-504; Other DPs: DP1.4.1(P-501), DP1.4.2(P-502)
FR8.1: Determine NaCl level in NaCl tank of stage two	DP8.1: LS-201; Other DPs: DP5.1.1(P-201), DP5.1.2(P-202)
FR8.2: Determine HCl level in HCl tank of stage two	DP8.2: LS-202; Other DPs: DP5.1.3(P-203), DP5.1.4(P-204)
FR8.3: Determine NaOCl level in NaOCl tank of stage two	DP8.3: LS-203; Other DPs: DP5.1.5(P-205), DP5.1.6(P-206), DP5.1.7(P-207), DP5.1.8(P-208)
FR8.4: Determine NaHSO3 level in NaHSO3 tank of stage four	DP8.4: LS-401; Other DPs: DP5.1.9(P-403), DP5.1.10(P-404)

Table 5 Top-level decomposition of WADI

Functional Requirements (FRs)	Design Parameters (DPs)
FR1: Supply water to the elevated tanks	DP1: Pumps
FR2: Monitor water level of elevated tanks	DP2: Level sensors
FR3: Monitor water flow rate	DP3: Flow sensors
FR4: Monitor the water quality	DP4: Water quality sensors
FR5: Monitor the dosing agent	DP5: Dosing pumps
FR6: Supply water to the consumer tanks	DP6: Methods of distribution
FR7: Measure and monitor the pressure of water	DP7: Pressure meters
FR8: Control the direction of the water flow	DP8: Control valves

of dependent components (depicted in Fig. 2). These sets of dependencies identified in the design include two limited to a single stage of WADI, and two involving multiple stages.

- Motorised valves (1-MV-001, 1-MV-005) allowing treated water to flow into the stage one tank, and the level indicator transmitter (1-LT-001) in that tank;
- Motorised valve (2-MV-001) in an elevated reservoir of stage two, and the associated level indicator transmitter (2-LT-001);
- Pump 1-P-005 in stage one, and a motorised valve (2-MV-003) and level indicator transmitter (2-LT-002) in stage two;
- Pumps 1-P-005 and 1-P-006 in stage one, and a motorised valve (2-MV-003) and level indicator transmitter (2-LT-002) in stage two.

With these sets of dependencies extracted, their invariant relationships can be derived by the designer/engineer in much the same way as SWaT.

Step three: building invariant checkers

In this final step, we incorporate the identified invariants into the implemented CPS as *invariant checkers*, i.e. defence mechanisms that monitor for any violations of the properties. Intuitively, an invariant checker takes live sensor readings and actuator states from a CPS, maps them to the appropriate state expression of the invariant (discretising continuous values where necessary), then returns the corresponding label of anomalous or non-anomalous.

Implementing invariant checkers can be done in at least two ways. First, if a complete set of state expressions is available, they can be programmed explicitly, e.g. as a control structure. Alternatively, an invariant checker can be constructed automatically using a supervised algorithm such as *decision tree learning* (Breiman et al. 1984). This latter approach has the advantage that the set of expressions need not be complete, as the learning algorithm will attempt to generalise from the samples presented.

Applied to SWaT and WADI

Consider the training set given in Table 9, which corresponds to the state expressions and labels of Invariant #3 (Table 8). The inputs are discretised representations of the sensor and actuator states, whereas the labels y_k are discrete values between 1 and 5 of which 2 and 4 indicate anomaly cases (of course, one could simply use two labels—anomaly, normal—but this helps differentiate exactly *which* anomaly occurred). Note that the training set is incomplete in comparison to Table 8, but is still enough to learn an accurate classifier (i.e. invariant checker), as detailed in the next section. This allows for the possibility of a design engineer to focus on enumerating the most important cases, using decision trees to generalise the rest, followed by some validation (see Evaluation and discussion) to ensure that the resulting classifier is correct.

Once the invariant checker is constructed or trained, Algorithm 1 summarises how it can be deployed as an anomaly detector—whether on simulators, datasets, or real systems and testbeds. In SWaT and WADI, the invariant checkers are implemented using values from

Table 6 State expressions of Invariant #1 for SWaT

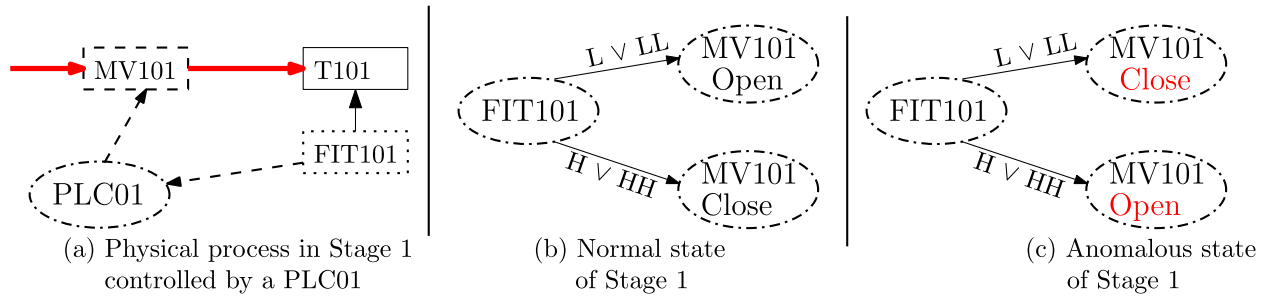
State expression	Label
!MV-101 and !LIT-101	Anomaly
!MV-101 and LIT-101	No anomaly
MV-101 and !LIT-101	No anomaly
MV-101 and LIT-101	Anomaly

Algorithm 1: Detecting anomalies

Input: Ordered set of dependent components DP; invariant checker M_{DP}; mapping F from raw inputs of DP to discretised inputs of M_{DP}; set of anomalous labels L_{DP}

Output: Anomalous values of DP

```
anomalyDetected := False;
while ! anomalyDetected do
    Read the current values of DP and store in the sequence S;
    if actuator states in S are stable then
        label := M_DP(F(S));
        if label ∈ L_DP then
            anomalyDetected := True;
Return S;
```



(a) Physical process in Stage 1 controlled by a PLC01 (b) Normal state of Stage 1 (c) Anomalous state of Stage 1

Fig. 5 Process graphs for Invariant #1 **a** Subsystem in stage one of SWaT; **b** a process graph for normal cases; **c** a process graph for anomalous cases

the historian server. Intuitively, the states of the sensors/actuators relevant to a particular invariant are constantly read, discretised, then labelled (as anomalous or non-anomalous) by the invariant checker. As soon as an anomalous label is returned, an alarm is raised and the relevant values are sent to the plant operator. Note that the algorithm requires actuator states to be 'stable', i.e. not in the process of transitioning from one state to another (such as a closed valve moving into a fully open position). These transient states can be handled either by expanding the mathematical state expressions to cover them, or by using complementary defence mechanisms alongside our invariant checkers (see our Discussion section).

The classifiers (i.e. invariant checkers) embody key characteristics of the design as identified by the engineer through a systematic method. We envisage that this has the potential to complement defence mechanisms based on data mining (e.g. (Pal et al. 2017; Chen et al. 2018; Umer et al. 2020)), where invariant relationships are based only on observable data after the system has been implemented, and which might not reflect all invariants implied by the design. Furthermore, as our invariants are constructed at the design stage, it may be possible to involve them in early simulations of the processes, and to iteratively modify the system design before it is implemented (Fig. 3) if any weak points or large sets of dependencies are identified.

Table 7 State expressions of Invariant #2 for SWaT

State expression	Label
!(FIT-501>0) and !P-401 and !P-402	No anomaly
!(FIT-501>0) and !P-401 and P-402	Anomaly
!(FIT-501>0) and P-401 and !P-402	Anomaly
!(FIT-501>0) and P-401 and P-402	Anomaly
FIT-501>0 and !P-401 and !P-402	Anomaly
FIT-501>0 and !P-401 and P-402	No anomaly
FIT-501>0 and P-401 and !P-402	No anomaly
FIT-501>0 and P-401 and P-402	Anomaly

Evaluation and discussion

Having derived invariants and invariant checkers from the designs of SWaT and WADI, in this section, we evaluate how effective they are at detecting real attacks.

Experiments and results

We derived eight invariants and invariant checkers from the designs of SWaT and WADI by following the steps given in the previous section. These invariants relate groups of dependent level/flow/analytical sensors, pumps, and motorised valves from both single stages and multiple stages of their respective testbeds (Table 10). As described, we used process graphs and domain knowledge to determine state expressions for the invariants, i.e. combinations of discretised states and their corresponding labels. The full sets of expressions for Invariants #1–#3 are given in Tables 6, 7, and 8 (with the expressions for the others omitted for brevity).

Table 8 State expressions of Invariant #3 for SWaT

State Expression	Label
!LIT-301 and !MV-201 and !P-101 and !P-102	Anomaly
!LIT-301 and !MV-201 and (!P-101 or P-102)	Anomaly
!LIT-301 and !MV-201 and (P-101 or !P-102)	Anomaly
!LIT-301 and !MV-201 and P-101 and P-102	Anomaly
!LIT-301 and MV-201 and !P-101 and !P-102	Anomaly
!LIT-301 and MV-201 and (!P-101 or P-102)	No anomaly
!LIT-301 and MV-201 and (P-101 or !P-102)	No anomaly
!LIT-301 and MV-201 and P-101 and P-102	Anomaly
LIT-301 and !MV-201 and !P-101 and !P-102	No anomaly
LIT-301 and !MV-201 and (!P-101 or P-102)	Anomaly
LIT-301 and !MV-201 and (P-101 or !P-102)	Anomaly
LIT-301 and !MV-201 and P-101 and P-102	Anomaly
LIT-301 and MV-201 and !P-101 and !P-102	Anomaly
LIT-301 and MV-201 and (!P-101 or P-102)	Anomaly
LIT-301 and MV-201 and (P-101 or !P-102)	Anomaly
LIT-301 and MV-201 and P-101 and P-102	Anomaly

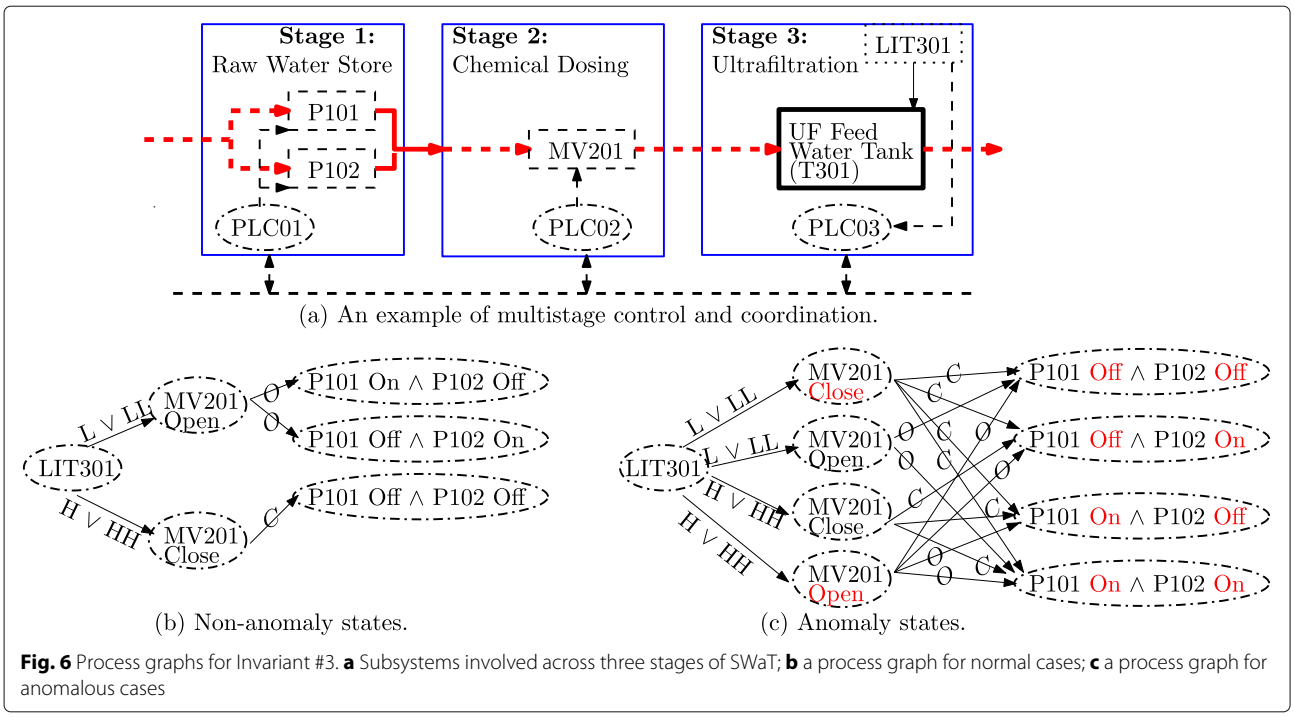


Fig. 6 Process graphs for Invariant #3. **a** Subsystems involved across three stages of SWaT; **b** a process graph for normal cases; **c** a process graph for anomalous cases

We experimentally assess the effectiveness of our design-derived invariant checkers on the SWaT and WADI testbeds. First, we run a pre-study to assess the *suitability of decision trees* for implementing invariant checkers from our (possibly partial) sets of mathematical state expressions. Second, we describe a study to *validate* that our invariant checkers properly classify normal system behaviour as non-anomalous (i.e. without raising false alarms). Finally, in our main study, we assess their effectiveness at *detecting attacks*, before discussing how designers might have mitigated them in the first place.

Table 9 Training set of Invariant #3 for SWaT

LIT-301	MV-201	P-101	P-102	y_k
1	1	1	1	1 (normal)
1	1	2	1	1 (normal)
1	1	2	2	1 (normal)
1	2	1	1	1 (normal)
1	2	1	2	2 (anomaly)
1	2	2	1	2 (anomaly)
1	2	2	2	3 (normal)
2	1	1	1	4 (anomaly)
2	1	1	2	5 (normal)
2	1	2	1	5 (normal)
2	1	2	2	5 (normal)
2	2	1	1	5 (normal)
2	2	2	2	5 (normal)

Suitability of decision trees

As previously discussed, after identifying a group of related components and some mathematical state expressions, our approach uses decision tree learning to convert this information (e.g. Table 9) into a classifier that can be used as an invariant checker. To assess the suitability of decision trees for this purpose, we designed a simple *pre-study* to ascertain that the learnt classifiers correctly label states as anomalous or non-anomalous, using complete sets of state expressions as our oracles. Our decision trees were implemented in Python using scikit-learn (no maximum tree depth; minimum sample of two to split a node; minimum sample of one to be leaf).

For each invariant, we generated 1000 copies of every possible combination of inputs (thus totalling 4000 tests for Invariant #1 and 16,000 for Invariant #3). First, we fed these to decision trees trained on the complete sets of state expressions for Invariants #1–#8 and found that the classifiers labelled them correctly 100% of the time. This is unsurprising, but an important sanity check before deploying our classifiers. Second, we fed the same inputs but to decision trees trained on partial sets of state expressions, in particular, the training set of Table 9 which covered all three non-anomalous cases but only 10 of the anomalous cases. Here too the classifiers labelled inputs correctly 100% of the time, suggesting that decision trees may be useful in cases where we want to generalise from partially completed analyses, e.g. if the number of dependent components is larger than 2–4. This should be investigated further, although for SWaT and WADI, is

Table 10 Dependent sensors and actuators of some design-derived invariants for SWaT and WADI

Invariant ID	CPS	Dependent Components	Notes
#1	SWaT	LIT-101, MV-101	Single-stage
#2	SWaT	P-401/402, FIT-501	Multi-stage
#3	SWaT	P-101/102, MV-201, LIT-301	Multi-stage
#4	SWaT	P-203, MV-201, AIT-202	Single-stage
#5	WADI	1-MV-001, 1-MV-005, 1-LIT-001	Single-stage
#6	WADI	2-MV-001 and 2-LIT-001	Single-stage
#7	WADI	1-P-005, 2-MV-003, 2-LIT-002	Multi-stage
#8	WADI	1-P-005, 1-P-006, 2-MV-003, 2-LIT-002	Multi-stage

less important as the number of dependent components involved for each invariant typically remains in this range.

Validating the invariant checkers

In our pre-study, we validated our classifiers (i.e. invariant checkers) against inputs we *labelled ourselves*. In practice, we need confidence that our invariant checkers perform correctly for inputs from the real systems too: real normal behaviour should be classified as non-anomalous, whereas real behaviours under attack scenarios should be classified as anomalous.

We designed a simple study to assess the first of these two requirements, i.e. to validate that our SWaT and WADI classifiers *actually do* characterise invariant properties of the testbeds. In other words, we want to validate that whenever the system is behaving normally, our invariant checkers label this behaviour as such and do not mislabel it as anomalous (detecting anomalies will be addressed in our third study).

To assess this, we made use of the SWaT and WADI datasets (iTrust Labs: Datasets 2020; Goh et al. 2016), which respectively contain seven and 14 days of data from continuous normal operation of the testbeds, i.e. without interruption from any faults or attacks. In particular, the dataset contains the readings of all sensors and the states of all actuators as logged every one second during this period. For every log in this normal dataset, we mapped the (continuous) sensor values and actuator states to the (discretised) inputs of our invariant checkers, and noted whether the classification was anomalous or non-anomalous. We found that all of our invariant checkers correctly labelled all stable states extracted from the logs as non-anomalous, i.e. 100% of the time. In other words, *no false positives* were reported at any point while analysing these 21 days' worth of logged data.

Effectiveness at detecting attacks

While our second study suggests that our invariant checkers are unlikely to raise false alarms, we must also investigate their ability to detect true positives, i.e. *actual* attacks. In this third study, we launched several attacks on the SWaT testbed that targeted components covered by the invariants, and observed whether or not the checkers were able to successfully detect them based on values obtained from the testbeds' historians. For WADI, we evaluated our invariant checkers against real data extracted from the system while it was under attack (iTrust Labs: Datasets 2020).

Tables 11 and 12 list several different SWaT and WADI attack scenarios, targeting sensors and actuators covering all of the dependent components monitored by our invariant checkers. Furthermore, the attacks cover all four categories of attacks as introduced earlier. For each SWaT attack in turn, we used the SCADA controls to bring the given testbed into the stated launch state, before starting the attack as described, extracting the logs from the historian, and mapping the (continuous) sensor values and actuator states to the (discretised) inputs of our invariant checkers (for WADI, we used existing attack data (iTrust Labs: Datasets 2020)). We found that for *every attack*, at least one of the invariant checkers (i.e. the one covering the affected dependent components) was able to correctly *detect the anomaly* and raise an alarm for the plant engineer. However, it should be noted that the anomalies were only detected once actuator states had stabilised, e.g. a valve is either open or closed, but not in the process of changing from one state to the other. Nonetheless, in a water plant the processes typically evolve slowly (e.g. filling up a tank takes time), so despite the small delay, the alarm is still likely to be raised well before an unsafe state is reached.

Attack mitigation

All of the attacks presented in Table 11 exploit the same dependencies between components that we identified in a design-level analysis to develop invariant checkers. By identifying dependencies in the design early, the designer can take steps either to minimise them (i.e. feedback to and adjust the design) or identify other means of attack mitigation.

For example, consider Attack #1, visualised in Fig. 7. The dependency exploited is given in DP7.1.1, row FR7.1 of Table 4. SWaT's defences could be strengthened against this attack by overriding attempts to manually turn on MV-101 if the water level reported by LIT-101 is above its high (H) or critically high (HH) thresholds. This would help to prevent the original attack, although would rely on the assumption that the value of LIT-101 is correct and can be trusted (additional invariant checkers concerning LIT-101 could help to mitigate this threat).

Consider Attacks #2 through to #5, as visualised in Fig. 8. The dependencies exploited are given in DP7.1.2, row FR7.1 of Table 4. Attacks #2 and #3 could be mitigated by overriding attempts to change the states of the pumps

Table 11 Attacks launched on SWaT to test our invariant checkers

No.	Test category	Target component(s)	Launch state	Attack intent and description
1	SAOS	MV-101	At time *t*, LIT-101 is above 900mm and MV-101 is turned off (Fig. 7)	Damage or reduce reliability of MV-101 At time *t*++, an attacker begins manually turning MV-101 on and off several times.
2	SAOS	P-101	At time *t*, LIT-301 is below 450mm, MV-201 is opened, and P-101 is turned on (Fig. 8)	Damage or reduce reliability of P-101 At time *t*++, an attacker manually stops P-101 for 10 seconds. Then the attacker manually turns on P-101 for another 10 seconds. The whole scenario is repeated a few times.
3	MAOS	Water pipes	At time *t*, LIT-301 is above 900mm, MV-201 is closed, P-101 is switched off and P-102 is switched off. (Fig. 8)	Damage water pipes at stage two At time *t*++, an attacker manually turns P-101 on. Next, the attacker manually turns pump P-102 on.
4	MAMS	P-101, P-102, MV-201	At time *t*, LIT-301 is above 900mm, MV-201 is closed, and P-101 and P-102 are switched off. (Fig. 8)	Damage or reduce reliability of P-101, P-102, and MV-201 At *t*++, an attacker manually turned on MV-201. Next, the attacker manually starts pump P-102. Then the attacker manually starts pump P-101. The test scenario is repeated several times.
5	SAMS	LIT-301,MV-201 P-101	At time *t*, LIT-301 is above 900mm, MV-201 is closed and P-101 and P-102 are switched off. (Fig. 8)	Damage or reduce reliability of LIT-301 sensors, P-101, and MV-201. At *t*++ where $x > 3$, an attacker changes the LIT-301 value to 400mm. At $t + 2x$ the value of LIT-301 is set to above 800mm. This procedure is repeated several times. These causes the LIT-301 sensors to sense sudden change in readings, MV-201 to open and close repeatedly, and P-101 to start and off many times.
6	SAOS	P-203	At time *t*, MV-201 is opened, P-203 is turned on, AIT-202 analyses water pH is above 8. (Fig. 9)	Damage or reduce reliability of P-203 At time *t*++, an attacker turns off and on P-203 several times.
7	SAOS	P-203 AIT-202	At time *t*, MV-201 is opened, P-203 is turned on, AIT-202 analyses water pH is above 8. (Fig. 9)	Damage or reduce reliability of P-203 and AIT-202 At time *t*++, an attacker changes pH value from above 8 to 6 in AIT-202. This causes P-203 to switch off. The attacker repeatedly changes these pH values. These caused P-203 to switch on and off several times. The sensor in AIT-202 needs to calculate the sudden change in pH several times.

Table 11 Attacks launched on SWaT to test our invariant checkers (*Continued*)

No.	Test category	Target component(s)	Launch state	Attack intent and description
8	SAOS	Water pipes	At time *t*, P-401 and P501 are switched on. (Fig. 10)	Damage water pipes at stage four At time *t*++, an attacker physically turns P-402 on. Now both P-401 and P-402 are pumping water to stage five which add extra water pressure to the pipes.
9	SAOS	FIT-501	At time *t*, P-401 is switched on. (Fig. 11)	Damage or reduce reliability of FIT-501. At time *t*++, an attacker drastically increases the value of FIT. This procedure is repeated several times.

Table 12 WADI attacks used to test our invariant checkers

No.	Test category	Target component(s)	Launch state	Attack intent and description
10	SAOS	Water pipes	At time *t*, 1-LT-001 is at Low or LowLow, 1-MV-005 is open and 1-MV-001 is turned off (Fig. 12)	Damage the pipes of WADI by increasing water pressure. At time *t*++, an attacker manually turns on 1-MV-001
11	SAOS	2-LT-001	At time *t*, 2-LT-001 is at High or HighHigh, and 2-MV-001 is closed (Fig. 13)	Overflow the tank, 2-LT-001. At time *t*++, an attacker opens 2-MV-001.
12	MAOS	2-LT-002	At time *t*, 2-LT-002 is at High or HighHigh, 1-P-005 is not turned on, and 2-MV-003 is not opened (Fig. 13)	Overflow the tank, 2-LT-002. At time *t*++, an attacker turns on 1-P-005 and 2-MV-003
13	MAOS	2-LT-002	At time *t*, 2-LT-002 is at High or HighHigh, 1-P-005 and 1-P-006 are not turned on, and 2-MV-003 is closed (Fig. 13)	Overflow the tank, 2-LT-002. At time *t*++, an attacker turns on 1-P-006 and 2-MV-003

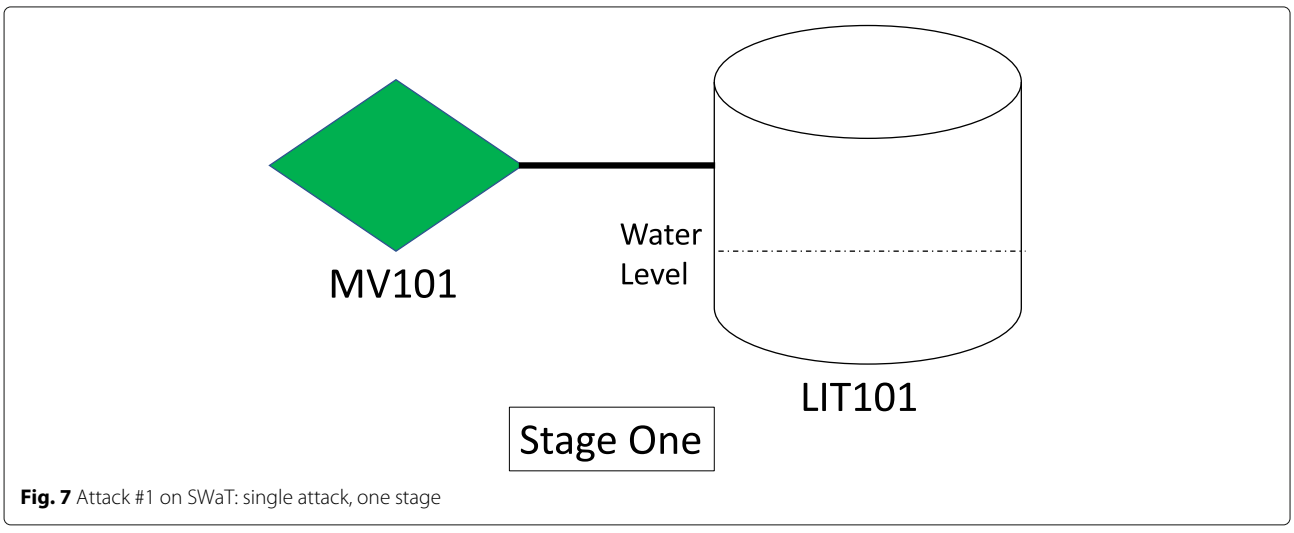


Fig. 7 Attack #1 on SWaT: single attack, one stage

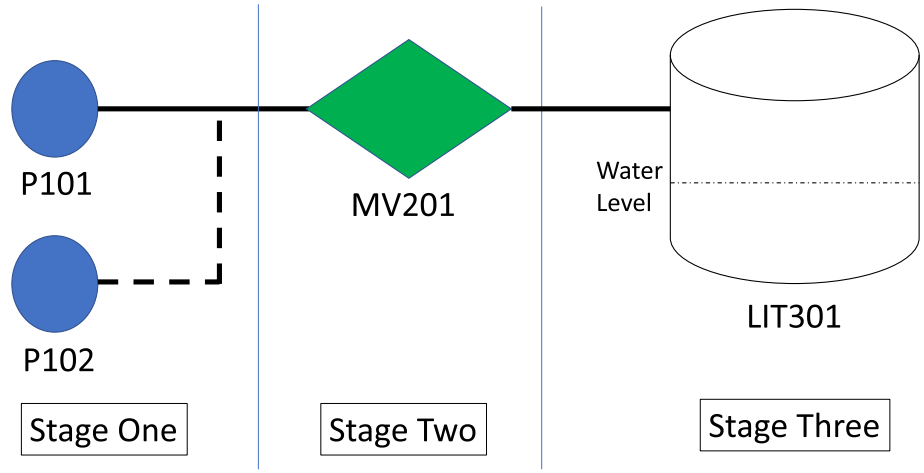


Fig. 8 Attacks #2-#5 on SWaT: including SAOS, MAOS, and MAMS attacks

when the LIT is in its high (H) or critically high (HH) thresholds. A key lock could even be introduced to minimise the possibility of the manipulation being carried out by an insider. Attack #4 could be mitigated by similar measures for MV-201. Finally, Attack #5 could be mitigated by a monitor that checks for vast changes in continuous values, e.g. a sudden increase in a tank from level z at time t to level $z + 3$ at time $t + 1$.

Consider Attacks #6 and #7, as visualised in Fig. 9. The dependencies exploited are given in DP5.1.3, row FR5.1 of Table 4. Attack #6 could be mitigated by preventing P-203 from being switched off when the water pH is above eight, e.g. by requiring a password to digitally switch it off or requiring a key to physically turn it off. Attack #7 could be mitigated using similar solutions to those suggested for #5.

For Attack #8 (Fig. 10), which exploits dependencies DP1.3.1 and DP1.3.2, row FR1.3 of Table 4, similar mitigations to those suggested for Attacks #2 and #3 could be used. For Attack #9 (Fig. 11), which exploits dependency DP3.5.1, row FR3.5 of Table 4, a similar solution to that of #5 can be again be used.

Discussion

This evaluation has shown that our *design-level analysis* for identifying sets of dependent components can also be used to derive invariants and invariant checkers. These invariant checkers are able to classify a range of real system inputs as anomalous or non-anomalous without any false positives. Our work has focused on the *method* of deriving invariants: axiomatic design principles provide systematic guidance for this purpose, without the need for any complex mathematical modelling (e.g. Petri-nets (Liu et al. 2017), Bayesian networks (Hadjsaid et al. 2009)) or even the implemented control logic itself. Ideally, the method would be applied earlier in the design process so that the system and its invariants can be derived together.

Our work has focused on the problem of *unearthing* invariants, rather than *deploying* them. While our invariant checkers perform well on the inputs that they

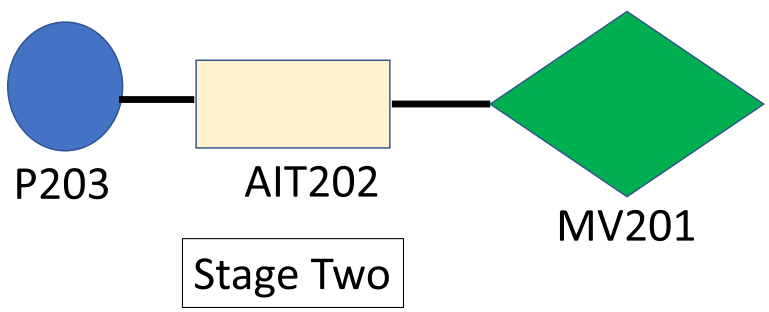


Fig. 9 Attacks #6 and #7 on SWaT: single attack, one stage

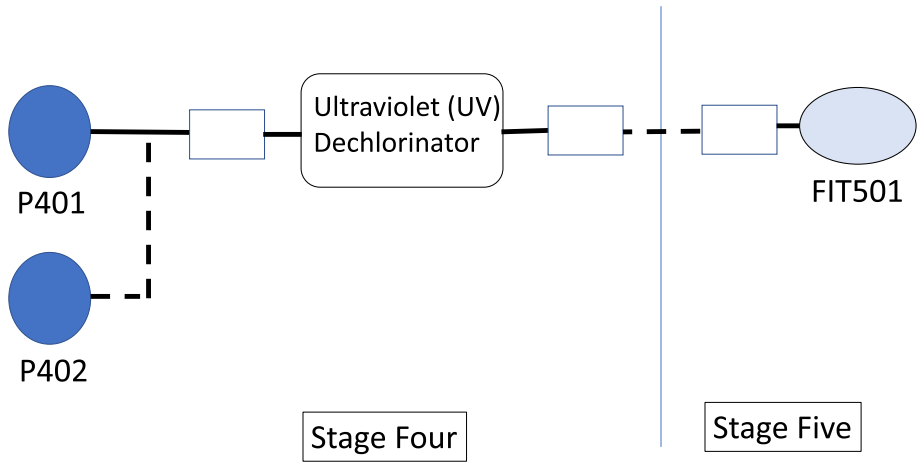


Fig. 10 Attack #8 on SWaT: single attack, one stage

analyse, we have not addressed the problem—shared by *all* invariant-based defence mechanisms—of making sure that those inputs have not been spoofed by an attacker. For this, we recommend combining our invariants with secure deployment strategies as suggested in the literature, e.g. trusted embedded devices connected to PLCs (Alves and Felton 2004; Abera et al. 2016).

Even if deployed securely, no invariant-based defence mechanism alone is enough to secure a CPS against the full range of attacker profiles. For example, invariant checkers would not be able to detect a stealthy attacker changing the reading of LIT-101 from 820mm to 830mm: no anomaly would be detected as LIT-101 remains in the same state of High. However, this kind of attack could be detected by the auto-regression technique (Yoong and Heng 2019), the state estimation method (Adepu and Mathur 2021), or by analysis of sensor and process noise (Ahmed et al. 2018). These various approaches (as well as invariant-based methods) have their own strengths and weaknesses, but are complementary and should be deployed together.

While our case studies have shown viability of deriving dependent components and invariants from a design, they have also highlighted some limitations of the approach, including: (1) manual effort is required; (2) the process requires the designer to have some domain expertise; (3) as SWaT and WADI were already built, we had limited opportunity to investigate how invariants derived from our process could be used to improve the design; and (4) the invariant checkers only detect anomalies once component states have stabilised (e.g. a valve has finished the process of opening). Some of these limitations are inherent from the goals of the approach: we *want* to manually analyse the CPS design in part to ensure we uncover invariants missed by data-driven approaches (e.g. invariants involving components rarely used). However, we are keen to explore ways of mitigating this by automating the extraction of simpler invariants (e.g. from data or

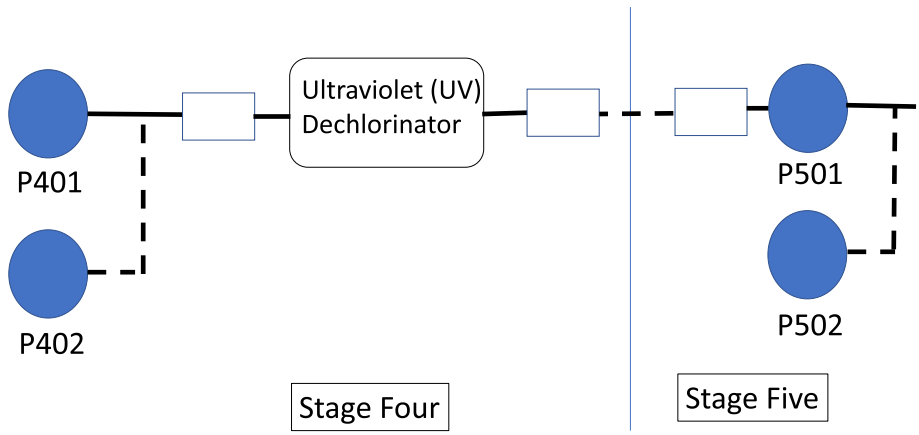


Fig. 11 Attack #9 on SWaT: single attack, one stage

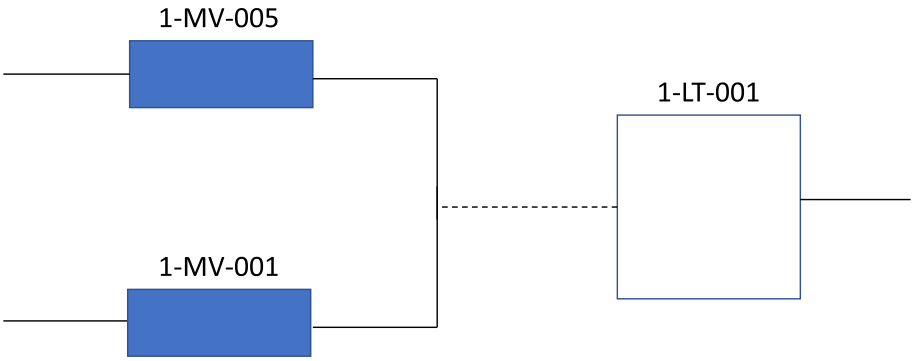


Fig. 12 Attack #10 on WADI: single attack, one stage

simulations), focusing the design-level analysis on others, and supporting the decomposition of matrices using tools.

By applying our axiomatic design analysis to two CPS testbeds, we have increased our confidence in the generality of the method. However, while the testbeds involve different processes, stages, and components, they are both from the domain of water, so we plan to explore the applicability of our work to other types of CPSs as well (e.g. power). As our approach is applied at the design level, it can avoid the safety and resource problems faced by others that rely on guided data generation (e.g. (Chen et al. 2018)).

Related work

In this section, we highlight and compare against some related work that addresses the main themes of this paper: defending CPSs and deriving CPS invariants.

Several different, complementary approaches have emerged in recent years for detecting and preventing attacks on CPSs and critical infrastructure. These include techniques based on *anomaly detection*, in which the logs of the physical data are analysed to identify suspicious events and anomalous behaviours (Cheng et al. 2017; Harada et al. 2017; Inoue et al. 2017; Pasqualetti et al. 2011; Aggarwal et al. 2018; Aoudi et al. 2018; He et al. 2019; Kravchik and Shabtai 2018; Lin et al. 2018; Narayanan and Bobba 2018; Schneider and Böttinger 2018; Carrasco and Wu 2019; Kim et al. 2019; Adepu et al. 2020; Das et al. 2020; Giraldo et al. 2020; Schmidt et al. 2020); digital fingerprinting, in which sensors are checked for spoofing by monitoring time and frequency domain features from sensor and process noise (Ahmed et al. 2018; Ahmed et al. 2018; Formby et al. 2016; Gu et al. 2018; Kneib and Huth 2018; Ahmed et al. 2020; Yang et al. 2020); and attestation, in which unauthorised changes to control logic can be detected (Valente et al. 2014; Abera et al. 2016).

Our work falls into another category of defence mechanisms: *invariant-based defences* (Cárdenas et al. 2011; Adepu and Mathur 2016a; Adepu and Mathur 2016b; Chen et al. 2016; Adepu and Mathur 2021; Chen et al. 2018; Choi et al. 2018; Giraldo et al. 2018; Umer et al. 2020), in which a plant is constantly monitored for violations of properties over the processes or control states (these violations possibly indicating an ongoing attack or fault). In particular, we focus on the problem of *deriving*

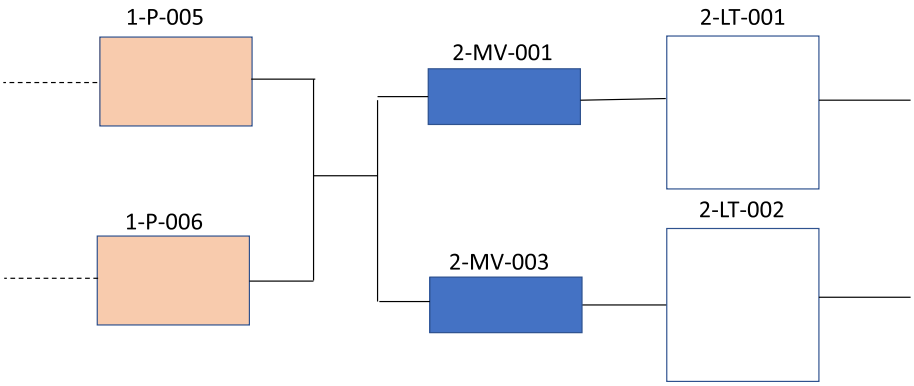


Fig. 13 Attacks #11-#13 on WADI: including SAOS and MAOS attacks

invariants for these defence mechanisms to check. Invariants can be derived from the system's design (e.g. documentation or ladder logic), from data sources (e.g. a historian server), or both (Umer et al. 2020). Our work follows the first approach, in that we start from functional requirements and systematically decompose them to invariants using axiomatic design theory; an approach that can (in principle) be performed before the plant has been implemented. This is in contrast to design-based approaches that start with the laws of physics, e.g. deriving invariant equations from the dynamics of water flow, then using state estimation methods to measure them (Adepu and Mathur 2016a; Adepu and Mathur 2016b).

Data-driven approaches to invariant generation mitigate the manual effort and expertise required of design-based approaches. Feng et al. (2019), for example, use data mining and learning to extract invariants from logs while taking into account noise in sensor measurements; and Chen et al. (2018) learn invariants over sensor readings by seeding control software with faults and observing the outcomes. Nonetheless, data-driven approaches have shortcomings (Ahmed et al. 2020) that could be complemented by design-driven approaches such as ours. First, they succeed or fail based on the quality of data: unbalanced or incomplete data can lead to false positives and incomplete invariant sets. Second, black-box anomaly detectors may be susceptible to adversarial synthetic sensor manipulations (Erba and Tippenhauer 2020). Third, translating data-driven invariants from a testbed to a real plant is difficult due to the policies of typical civil infrastructure operators. While fairly representative, testbeds are not like-for-like, and solutions working perfectly on the former may need significant re-engineering to work on the latter (e.g. due to different design parameters). Some of the data-driven approaches also come with safety concerns (e.g. (Chen et al. 2018)) which would prevent them from being re-trained on a real plant.

Once invariants are obtained (by our approach or others), they need to be deployed in the plant as invariant checkers. In our work, invariant checkers were deployed as decision tree classifiers that monitored live data from the historian server. However, it is also possible for our invariants to be deployed in a more distributed manner, with checkers placed in the PLCs, as described by Adepu and Mathur (2021). We could also deploy our checkers *orthogonally* to existing defence mechanisms, i.e. on an independent network that unobtrusively monitors data extracted from several sources, as proposed by Shrivastava et al. (2018). This would allow our approach to be deployed alongside complementary defence mechanisms that focus on attacks that invariant-based approaches may miss, e.g. the injection of false data (Beg et al. 2017).

Conclusion

We proposed a novel and systematic method for deriving CPS invariant checkers through a *design-level analysis* based on axiomatic design principles. Our method iteratively analyses dependencies in the CPS design to construct mathematical state expressions (or process graphs) that represent invariant relations between sensor readings and actuator states. In contrast to mining-based approaches for identifying invariants, our method aims to: (1) find invariants that are implicit in the design but not well-represented in datasets; (2) ensure that invariants can be contextualised by tracing them back to specific design iterations and requirements; and (3) further integrate security concerns at the design stage, potentially allowing weak points to be identified before a CPS is built. This is achieved using a step-by-step approach from requirements through to the implementation of invariant checkers, in a process that does not require any complex mathematical modelling or dataset-based training.

We evaluated our approach on two real-world CPS testbeds: SWaT, a water purification system, and WADI, a water distribution network for consumer supplies. Starting from high-level functional requirements, we applied axiomatic design principles to decompose the systems' designs and identify dependencies between design parameters (i.e. sensors and actuators). Using domain expertise and process graphs, we derived mathematical state expressions for eight of these sets of dependencies, then generalised them into invariant checkers using decision tree learning. We found that these checkers were able to detect all 13 attacks we launched, covering both single and multi-stage attacks and often multiple components. Finally, we found that our invariant checkers operated *without false positives*, i.e. without incorrectly raising any alarms on normal operational data.

In ongoing work, plan to compare the effectiveness of our invariants against those derived by other approaches (both automated and manual ones) in order to better quantify and understand the potential payoff of analysing the CPS design directly. We also plan to evaluate our approach on other case studies, for example, other industrial control systems (e.g. power grids), and potentially CPSs from other domains such as building management or healthcare. Finally, we are interested in exploring the role that simulation can play, especially for validating invariants at early stages of the design process, before the real system has been implemented. Simulation may play an important role in reducing the effort required by design-centric approaches: we could use a data-driven approach to derive a first set of invariants from logs (e.g. (Feng et al. 2019)), leaving design engineers to focus their axiomatic design analysis on unearthing any invariants that are implicit in the CPS design but not well-represented in the data.

Acknowledgements
We are grateful to the support of the iTrust technicians and NSoE office for helping to facilitate the experiments reported in this research.

Authors' contributions
CHY performed the axiomatic design analysis, implemented the experiments, and wrote the first draft of this paper. VRP, RRM, AS, and CMP all provided technical feedback throughout the project. Furthermore, AS had the initial idea, and CMP made substantial contributions to the text. All authors reviewed the final manuscript. All authors read and approved the final manuscript.

Funding
This research / project is supported by the National Research Foundation, Singapore, under its National Satellite of Excellence Programme "Design Science and Technology for Secure Critical Infrastructure" (Award Number: NSoE DeST-SCI2019-0004). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

Availability of data and materials
The datasets used in this research are available online (iTrust Labs: Datasets 2020).

Competing interests
The authors declare that they have no competing interests.

Author details
[1] Singapore University of Technology and Design, 8 Somapah Road, 487372 Singapore, Singapore. [2] Indian Institute of Petroleum and Energy, 2nd Floor, AU Engg College Main Block, Andhra University, 530003 Visakhapatnam, India. [3] Birla Institute of Technology and Science, Pilani, Hyderabad Campus Jawahar Nagar, Kapra Mandal Medchal District, 500078 Telangana, India. [4] Singapore Management University, 80 Stamford Road, 178902 Singapore, Singapore.

Received: 18 August 2020 Accepted: 1 January 2021
Published online: 02 April 2021

References

Abera T, Asokan N, Davi L, Ekberg J, Nyman T, Paverd A, Sadeghi A, Tsudik G (2016) C-FLAT: control-flow attestation for embedded systems software. In: Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS 2016). ACM. pp 743–754. https://doi.org/10.1145/2976749.2978358
Adepu S, Brasser F, Garcia L, Rodler M, Davi L, Sadeghi A, Zonouz S (2020) Control behavior integrity for distributed cyber-physical systems. In: Proc. ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS 2020). IEEE, New York. pp 30–40
Adepu S, Mathur A (2016) Using process invariants to detect cyber attacks on a water treatment system. In: Proc. International Conference on ICT Systems Security and Privacy Protection (SEC 2016), IFIP AICT, vol. 471. Springer, Cham. pp 91–104
Adepu S, Mathur A (2016) Distributed detection of single-stage multipoint cyber attacks in a water treatment plant. In: Proc. ACM Asia Conference on Computer and Communications Security (AsiaCCS 2016). ACM, New York. pp 449–460
Adepu S, Mathur A (2021) Distributed attack detection in a water treatment plant: Method and case study. IEEE Trans Dependable Secure Comput 18(1):86–99
Adepu S, Mathur A (2016) Generalized attacker and attack models for cyber physical systems. In: Proc. IEEE Annual Computer Software and Applications Conference (COMPSAC 2016). IEEE Computer Society. pp 283–292
Aggarwal E, Karimibiuki M, Pattabiraman K, Ivanov A (2018) CORGIDS: A correlation-based generic intrusion detection system. In: Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018). ACM, New York. pp 24–35
Ahmed C, Ochoa M, Zhou J, Mathur A, Qadeer R, Murguia C, Ruths J (2018) *NoisePrint*: Attack detection using sensor and process noise fingerprint in cyber physical systems. In: Proc. Asia Conference on Computer and Communications Security (AsiaCCS 2018). ACM, New York. pp 483–497
Ahmed C, Palleti V, Mathur A (2017) WADI: a water distribution testbed for research in the design of secure cyber physical systems. In: Proc. International Workshop on Cyber-Physical Systems for Smart Water Networks (CySWATER@CPSWeek 2017). ACM, New York. pp 25–28
Ahmed C, R G, Mathur A (2020) Challenges in machine learning based approaches for real-time anomaly detection in industrial control systems. In: Proc. ACM Workshop on Cyber-Physical System Security (CPSS 2020). ACM, New York. pp 23–29
Ahmed C, Zhou J, Mathur A (2018) Noise matters: Using sensor and process noise fingerprint to detect stealthy cyber attacks and authenticate sensors in CPS. In: Proc. Annual Computer Security Applications Conference (ACSAC 2018). ACM, New York. pp 566–581
Alves T, Felton D (2004) TrustZone: Integrated hardware and software security. ARM white paper 3(4):18–24
Aoudi W, Iturbe M, Almgren M (2018) Truth will out: Departure-based process-level detection of stealthy attacks on control systems. In: Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS 2018). ACM, New York. pp 817–831
Beg O, Johnson T, Davoudi A (2017) Detection of false-data injection attacks in cyber-physical DC microgrids. IEEE Trans Ind Inform 13(5):2693–2703
Bondy J, Murty U (2008) Graph Theory. Springer
Breiman L, Friedman J, Stone C, Olshen R (1984) Classification and Regression Trees. Wadsworth and Brooks
Cárdenas A, Amin S, Lin Z, Huang Y, Huang C, Sastry S (2011) Attacks against process control systems: risk assessment, detection, and response. In: Proc. ACM Asia Conference on Computer and Communications Security (AsiaCCS 2011). ACM, New York. pp 355–366
Carrasco M, Wu C (2019) An unsupervised framework for anomaly detection in a water treatment system. In: Proc. IEEE International Conference On Machine Learning And Applications (ICMLA 2019). IEEE, New York. pp 1298–1305
Cheng L, Tian K, Yao D (2017) Orpheus: Enforcing cyber-physical execution semantics to defend against data-oriented attacks. In: Proc. Annual Computer Security Applications Conference (ACSAC 2017). ACM, New York. pp 315–326
Chen Y, Poskitt C, Sun J, Adepu S, Zhang F (2019) Learning-guided network fuzzing for testing cyber-physical system defences. In: Proc. IEEE/ACM International Conference on Automated Software Engineering (ASE 2019). IEEE Computer Society, New York. pp 962–973
Chen Y, Poskitt C, Sun J (2016) Towards learning and verifying invariants of cyber-physical systems by code mutation. In: Proc. International Symposium on Formal Methods (FM 2016), LNCS. Springer, Cham Vol. 9995. pp 155–163
Chen Y, Poskitt C, Sun J (2018) Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system. In: Proc. IEEE Symposium on Security and Privacy (S&P 2018). IEEE Computer Society, New York. pp 648–660
Chen Y, Xuan B, Poskitt C, Sun J, Zhang F (2020) Active fuzzing for testing and securing cyber-physical systems. In: Proc. ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2020). ACM, New York
Choi H, Lee W, Aafer Y, Fei F, Tu Z, Zhang X, Xu D, Xinyan X (2018) Detecting attacks against robotic vehicles: A control invariant approach. In: Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS 2018). ACM, New York. pp 801–816
Das T, Adepu S, Zhou J (2020) Anomaly detection in industrial control systems using logical analysis of data. Comput Secur 96:1–13
Erba A, Tippenhauer N (2020) No need to know physics: Resilience of process-based model-free anomaly detection for industrial control systems. CoRR abs/2012.03586:1–18
Feng C, Palleti V, Mathur A, Chana D (2019) A systematic framework to generate invariants for anomaly detection in industrial control systems. In: Proc. Annual Network and Distributed System Security Symposium (NDSS 2019). The Internet Society, Reston. pp 1–15
Formby D, Srinivasan P, Leonard A, Rogers J, Beyah R (2016) Who's in control of your control system? device fingerprinting for cyber-physical systems. In: Proc. Annual Network and Distributed System Security Symposium (NDSS 2016). The Internet Society, Reston. pp 1–15
Giraldo J, Urbina D, Cardenas A, Valente J, Faisal M, Ruths J, Tippenhauer N, Sandberg H, Candell R (2018) A survey of physics-based attack detection in cyber-physical systems. ACM Comput Surv 51(4):76–17636
Giraldo J, Urbina D, Tang C, Cárdenas A (2020) The more the merrier: adding hidden measurements to secure industrial control systems. In: Proc. Annual Symposium on Hot Topics in the Science of Security (HotSoS 2020). ACM. pp 3–1310. https://doi.org/10.1145/3384217.3385624

Goh J, Adepu S, Junejo K, Mathur A (2016) A dataset to support research in the design of secure water treatment systems. In: Proc. International Conference on Critical Information Infrastructures Security (CRITIS 2016). Springer Vol. 10242. pp 88–99
Goh J, Adepu S, Tan M, Lee Z (2017) Anomaly detection in cyber physical systems using recurrent neural networks. In: Proc. International Symposium on High Assurance Systems Engineering (HASE 2017). IEEE, New York. pp 140–145
Gu Q, Formby D, Ji S, Cam H, Beyah R (2018) Fingerprinting for cyber-physical system security: Device physics matters too. IEEE Secur Priv 16(5):49–59
Hadjsaid N, Tranchita C, Rozel B, Viziteu M, Caire R (2009) Modeling cyber and physical interdependencies - application in ICT and power grids. In: Proc. IEEE/PES Power Systems Conference and Exposition (PSCE 2009). IEEE, New York. pp 1–6
Harada Y, Yamagata Y, Mizuno O, Choi E (2017) Log-based anomaly detection of CPS using a statistical method. In: Proc. International Workshop on Empirical Software Engineering in Practice (IWESEP 2017). IEEE, New York. pp 1–6
Hassanzadeh A, Rasekh A, Galelli S, Aghashahi M, Taormina R, Ostfeld A, Banks M (2020) A review of cybersecurity incidents in the water sector. J Environ Eng 146(5):03120003
He Z, Raghavan A, Hu G, Chai S, Lee R (2019) Power-grid controller anomaly detection with enhanced temporal deep learning. In: Proc. IEEE International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom 2019). IEEE, New York. pp 160–167
Inoue J, Yamagata Y, Chen Y, Poskitt C, Sun J (2017) Anomaly detection for a water treatment system using unsupervised machine learning. In: Proc. IEEE International Conference on Data Mining Workshops (ICDMW 2017): Data Mining for Cyberphysical and Industrial Systems (DMCIS 2017). IEEE, New York. pp 1058–1065
iTrust Labs: Datasets (2020). https://itrust.sutd.edu.sg/itrust-labs_datasets/. Accessed December 2020
Kandjani H, Tavana M, Bernus P, Wen L, Mohtarami A (2015) Using extended axiomatic design theory to reduce complexities in global software development projects. Comput Ind 67:86–96
Kim J, Yun J, Kim H (2019) Anomaly detection for industrial control systems using sequence-to-sequence neural networks. In: Proc. International Workshop on the Security of Industrial Control Systems and Cyber-Physical Systems (CyberICPS 2019), LNCS. Springer Vol. 11980. pp 3–18
Kneib M, Huth C (2018) Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks. In: Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS 2018). ACM, New York. pp 787–800
Kravchik M, Shabtai A (2018) Detecting cyber attacks in industrial control systems using convolutional neural networks. In: Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018). ACM, New York. pp 72–83
Leyden J (2016) Water treatment plant hacked, chemical mix changed for tap supplies. The Register. Accessed December 2020
Lin Q, Adepu S, Verwer S, Mathur A (2018) TABOR: A graphical model-based approach for anomaly detection in industrial control systems. In: Proc. Asia Conference on Computer and Communications Security (AsiaCCS 2018). ACM, New York. pp 525–536
Liu X, Zhang J, Zhu P (2017) Modeling cyber-physical attacks based on probabilistic colored petri nets and mixed-strategy game theory. Int J Crit Infrastruct Prot 16:13–25
Mathur A, Tippenhauer N (2016) SWaT: a water treatment testbed for research and training on ICS security. In: Proc. International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater@CPSWeek 2016). IEEE Computer Society, New York. pp 31–36
Matt D (2012) Application of axiomatic design principles to control complexity dynamics in a mixed-model assembly system: a case analysis. Int J Prod Res 50:1850–1861
Mohsen H, Cekecek E (2000) Thoughts on the use of axiomatic designs within the product development process. In: Proc. International Conference on Axiomatic Design (ICAD 2000). Institute for Axiomatic Design. pp 188–195
Narayanan V, Bobba R (2018) Learning based anomaly detection for industrial arm applications. In: Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018). ACM, New York. pp 13–23
N. Al-Mhiqani M, Ahmad R, Mohamed W, Hassan A, Zainal Abidin Z, Ali N, Abdulkareem K (2018) Cyber-security incidents: A review cases in cyber-physical systems. Int J Adv Comput Sci Appl 9:499–508
Pal K, Adepu S, Goh J (2017) Effectiveness of association rules mining for invariants generation in cyber-physical systems. In: Proc. IEEE International Symposium on High Assurance Systems Engineering (HASE 2017). IEEE Computer Society, New York. pp 124–127
Palleti V, Joseph J, Silva A (2018) A contribution of axiomatic design principles to the analysis and impact of attacks on critical infrastructures. Int J Crit Infrastruct Prot 23:21–32
Pasqualetti F, Dorfler F, Bullo F (2011) Cyber-physical attacks in power networks: Models, fundamental limitations and monitor design. In: Proc. IEEE Conference on Decision and Control and European Control Conference (CDC-ECC 2011). IEEE, New York. pp 2195–2201
Secure Water Treatment (SWaT) (2020). https://itrust.sutd.edu.sg/itrust-labs-home/itrust-labs_swat/. Accessed December 2020
Schmidt T, Hauer F, Pretschner A (2020) Automated anomaly detection in CPS log files - A time series clustering approach. In: Proc. International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2020), LNCS, vol. 12234. Springer. pp 179–194. https://doi.org/10.1007/978-3-030-54549-9_12
Schneider P, Böttinger K (2018) High-performance unsupervised anomaly detection for cyber-physical system networks. In: Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018). ACM, New York. pp 1–12
Shrivastava S, Adepu S, Mathur A (2018) Design and assessment of an orthogonal defense mechanism for a water treatment facility. Robot Auton Syst 101:114–125
Suh N (2001) Axiomatic Design: Advances and Applications. Oxford University Press
Umer M, Mathur A, Junejo K, Adepu S (2020) Generating invariants using design and data-centric approaches for distributed attack detection. Int J Crit Infrastruct Prot 28:100341
Valente J, Barreto C, Cárdenas A (2014) Cyber-physical systems attestation. In: Proc. IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2014). IEEE Computer Society. pp 354–357. https://doi.org/10.1109/DCOSS.2014.61
Wijaya H, Aniche M, Mathur A (2020) Domain-based fuzzing for supervised learning of anomaly detection in cyber-physical systems. In: Proc. International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS 2020). ACM, New York. pp 237–244
Yang K, Li Q, Lin X, Chen X, Sun L (2020) iFinger: Intrusion detection in industrial control systems via register-based fingerprinting. IEEE J Sel Areas Commun 38(5):955–967
Yoong C, Heng J (2019) Framework for continuous system security protection in SWaT. In: Proc. International Symposium on Computer Science and Intelligent Control (ISCSIC 2019). ACM, New York. pp 60–1606
Yoong CH, Palleti VR, Silva A, Poskitt CM (2020) Towards systematically deriving defence mechanisms from functional requirements of Cyber-Physical Systems. In: Proc. ACM Cyber-Physical System Security Workshop (CPSS 2020). ACM. pp 11–22. https://doi.org/10.1145/3384941.3409589
Zhu X, Hu S, Koren Y, Marin S (2008) Modeling of manufacturing complexity in mixed-model assembly lines. J Manuf Sci Eng 130:1–10

Publisher's Note