Incidents Are Meant for Learning, Not Repeating: Sharing Knowledge About Security Incidents in Cyber-Physical Systems

Faeq Alrimawi, Liliana Pasquale, Deepak Mehta, Nobukazu Yoshioka, Bashar Nuseibeh

***Abstract*—Cyber-physical systems (CPSs) are part of many critical infrastructures such as industrial automation and transportation systems. Thus, security incidents targeting CPSs can have disruptive consequences to assets and people. As incidents tend to re-occur, sharing knowledge about these incidents can help organizations be more prepared to prevent, mitigate or investigate future incidents. This paper proposes a novel approach to enable representation and sharing of knowledge about CPS incidents across different organizations. To support sharing, we represent incident knowledge (*incident patterns*) capturing incident characteristics that can manifest again, such as incident activities or vulnerabilities exploited by offenders. Incident patterns are a more abstract representation of specific incident instances and, thus, are general enough to be applicable to various systems - different than the one in which the incident occurred. They can also avoid disclosing potentially sensitive information about an organization's assets and resources. We provide an automated technique to *extract* an incident pattern from a specific incident instance. To understand how an incident pattern can manifest again in other cyber-physical systems, we also provide an automated technique to *instantiate* incident patterns to specific systems. We demonstrate the feasibility of our approach in the application domain of smart buildings. We evaluate correctness, scalability, and performance using two substantive scenarios inspired by real-world systems and incidents.**

***Index Terms*—Cyber-physical systems, Security incidents, Smart building, Knowledge Sharing**

I. Introduction

Cyber-Physical Systems (CPSs) combine computation, communication, and physical processes [1] to augment physical systems with enhanced capabilities, such as real-time monitoring and dynamic control. Nowadays applications of CPS can be found in many domains such as industrial control systems, transportation systems and smart buildings. Thus, security incidents targeting CPSs can have disruptive consequences to their users and the assets they manage.

CPSs enable complex interactions between cyber and physical components. For example, in a smart building, a rise in the measured temperature of a room can trigger a digital process to issue a command to an air conditioner to start cooling the room. Interactions between cyber and physical components can extend the attack surface of a CPS, giving malicious individuals more opportunities to cause harm since they can exploit vulnerabilities from either component to impact the other. Thus, the number of security incidents targeting CPSs has increased over the past years [2]. For example, in the Ukrainian power grid incident [3], offenders used spear phishing to gain a foothold in the distribution companies' computer network. Then, they gained access to the power grid network, to infect and disable some physical devices (e.g., workstations, serial-to-Ethernet) that control the electricity distribution, causing disruption to the normal operation of the grid. Previously, in the German steel-mill incident [4], offenders used spear phishing to gain a foothold in the corporate network, and then gain access to the plant's network in order to shutdown the blast furnace and the alarm system.

Some aspects of prior incidents often tend to re-occur. For example, in the Ukrainian power grid incident an offender obtained access to a private network using spear phishing. A similar activity also happened in the German steel-mill incident. Thus, sharing knowledge about prior security incidents can help organizations being more prepared to prevent, mitigate, or investigate future incidents [5]. However, supporting information sharing about security incidents is still a key open challenge [6], [7]. Besides, knowledge about security incidents targeting CPSs is limited.

In this paper we propose a novel approach to enable representation and sharing of incident knowledge across different organizations. To enable sharing, we represent incident knowledge as *incident patterns* that capture incident characteristics, which can occur again, such as vulnerabilities exploited by offenders. Incident patterns are a more abstract representation of incident instances. Therefore, they can be general enough to be applied to various systems, beside the one in which the incident occurred. They can also avoid revealing potentially critical information about an organization's assets and resources (e.g., physical structure of a building or vulnerable devices). As incident activities can target or exploit system components, we also provide a representation of the system where an incident occurs. This includes cyber and physical components, their structure, dependencies and dynamic behavior. We provide two meta-models to represent incidents and cyber-physical systems, respectively.

We propose an automated technique to *extract* an incident pattern from a specific incident instance. The extraction technique explores the inheritance hierarchy in the cyber-physical system meta-model to abstract specific system characteristics described in the incident instance. To understand how an incident pattern can manifest again in other CPSs, we propose an automated technique to *instantiate* incident patterns to different systems. The instantiation technique uses a representation of the dynamic behavior of the system - expressed

as a labeled transition system - to identify the behavior traces matching the activities in the incident pattern. We demonstrate the feasibility of our approach in the application domain of smart buildings. We evaluate correctness, scalability, and performance using two substantive scenarios inspired by real-world systems and incidents.

The novelty of our work lies in the combination of three key elements:

- Incident patterns to support representation and sharing of incident knowledge across different organizations.
- Automated technique to extract an incident pattern from a specific incident instance, in order to facilitate sharing of incident knowledge and avoid disclosing sensitive information about an organization's assets and resources.
- Automated technique to instantiate an incident pattern to specific cyber-physical systems, in order to facilitate assessment about whether and how prior incidents can manifest again.

The remainder of this paper is organized as follows. In Section II, we motivate the need to share information about incidents in CPSs. In Section III, we provide an overview of our approach to share incident knowledge. In Section IV, we present, respectively, our system and incident meta-models. In Section V, we describe the incident pattern extraction technique. In Section VI, we illustrate the incident pattern instantiation technique. In Section VII, we present an evaluation of both techniques, and discuss the results and threats to validity. In Section VIII, we compare our approach with related work. Finally, in Section IX, we conclude and present future work.

II. Motivating Example

Our motivating example is centered on the ACME company that operates across three different smart buildings: a *Research Center*, a *Warehouse*, and a *Manufacturing Plant*. This is depicted in Fig. 1. The plan of the 2nd floor of the Research Center consists of a *Server Room*, a *Control Room*, and a *Toilet*. The *Server Room* contains a *Fire Alarm*, an air conditioning unit (*HVAC*), and some *Server*s, while the *Control Room* contains a *Workstation*. The whole building is equipped with *Smart Light*s. The *HVAC*, the *Fire Alarm* and the *Smart Light*s communicate with the *Workstation* through the *Installation Bus* network, which adopts the KNX protocol [8].

Unfortunately, an incident occurred in the *Research Center*. An offender reached the 2nd floor, entered the *Toilet*, and connected his/her Laptop physically to the *Installation Bus* by replacing the *Smart Light* (*SL1*). After that, s/he was able to detect the *HVAC* and connect to it. Detection of the *HVAC* was possible by eavsdropping on the messages exchanged through the installation bus, which are not encrypted [9]. Then, the offender sent a targeted Malware to disable the *HVAC* (e.g., exploiting the vulnerabilities present in Trane HVACs [10]). This subsequently caused the *Servers* to heat up. The incident actions are listed at the bottom of Fig. 1.

Upon the discovery of the incident, security administrators wrote a report describing how the incident occurred.

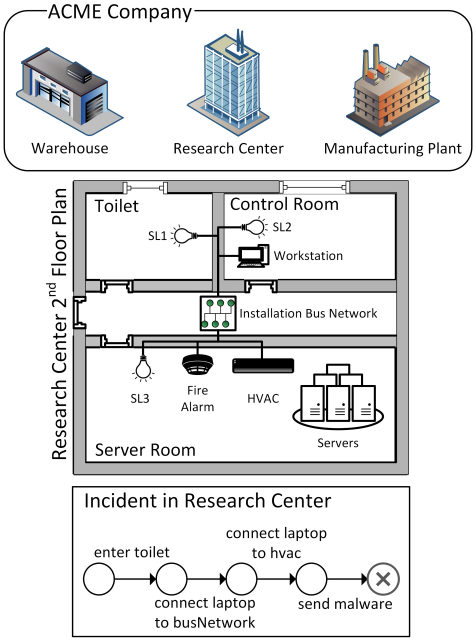

Fig. 1: The ACME Company Motivating Example.

Afterwards, to assess whether similar incidents activities can manifest in the other buildings, security administrators have to identify existing vulnerabilities brought by cyber and physical components in those buildings. This may require to examine the physical structure of each building, as well as the software and network configurations of the digital devices within the buildings.

Security administrators may face the following challenges. First, the approach each organization follows to perform incident reporting is not standardized [11]. Although different templates (e.g., [12]) have been proposed to identify what type of information incident reports should contain, the description of an incident is usually provided in natural language [13], without following a specific structure. Understanding how an incident can re-occur in a different system is arduous. It would require a security administrator to examine the incident description manually, and speculate on all possible ways in which incidents activities can be performed. Second, incident reports may not represent incident activities in CPSs, which can exploit vulnerabilities and dependencies between cyber and physical components. Finally, incident reports may contain sensitive information (e.g., internal network structure) that cannot be disclosed to third parties [11].

Therefore, it is necessary to provide an approach to represent incidents in a more structured form, which can capture activities that can use cyber and physical components. Moreover, it is necessary to provide a modeling technique to represent incident information in an abstract form, in order to avoid disclosing specific sensitive information about the system in which the incident occurred. Finally, automated techniques should be provided to analyze incident knowledge and assess whether and how prior incidents can re-occur. This can help organizations be more prepared to prevent, mitigate or investigate future incidents by, for example, updating security

and auditing measures depending on potential incidents.

III. Sharing Incident Knowledge

To address the challenges highlighted in the previous section, we propose our approach to share incident knowledge across different systems and organizations. Our approach provides representations of incidents and the cyber-physical systems in which they can occur. It also provides two automated techniques. One technique extracts potential incident patterns from specific incidents. The other technique instantiates incident patterns in CPSs, in order to assess whether and how an incident can reoccur. Our approach is shown in Fig. 2.

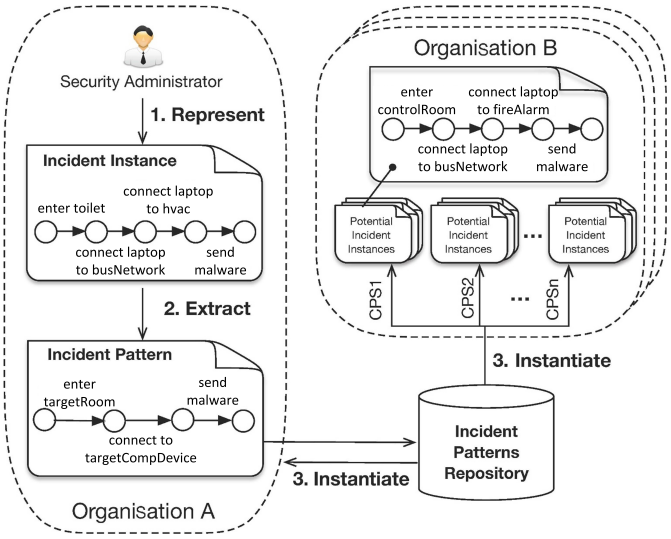

Fig. 2: Our approach for sharing incident knowledge.

After an incident occurs a security administrator operating within an organization (*A*) can *represent* (1) the activities of an incident (*incident instance*). These activities refer to specific cyber and physical system components (e.g., *SL1* or *Bus Network of the ACME Research Centre*). Subsequently, an *incident pattern* can be *extracted* (2) from an incident instance automatically. An incident pattern includes activities that refer to a more abstract representation of cyber-physical system components. For example, activity *"enter toilet"* in the incident instance can be abstracted to *"enter targetRoom"* in the incident pattern. This avoids disclosing specific information indicating, for example, location of smart devices exploited in the incident. Differently from the traditional notion of *pattern*, in this paper an incident pattern is extracted from a single incident instance. However, an incident pattern can potentially be instantiated in various ways in different systems.

The incident pattern is subsequently *stored* (3) in a shared repository. Each time the repository is updated, a set of subscribed organizations is notified. They can access incident patterns and *instantiate* (4) them automatically w.r.t a set of specific CPSs they manage. This allows system administrators to assess all possible ways in which incident patterns can reoccur again in those CPSs. For example, the incident pattern extracted from the Research Center incident can be instantiated to another organization's CPS (e.g., *CPS1 managed by organization B*). Incident activity *"connect to targetCompDevice"* in the incident pattern is instantiated to 2 subsequent activities *"connect laptop to busNetwork"* and *"connect laptop to fireAlarm"*.

In our previous work [14], we briefly introduced our approach for sharing incident knowledge and provided a preliminary description of the models adopted to represent an incident and the cyber-physical system in which it can occur. In this paper, we extend our previous work, by describing in more detail our models and how they can be used to represent cyber-physical systems, incident instances and patterns. Differently from our previous work, in this paper we also provide two automated techniques to support extraction and instantiation of incident patterns, respectively. Finally we evaluate our techniques on a substantive, large-scale example.

IV. Modeling Systems & Incidents

To model incidents in cyber-physical systems, we provide two meta-models. First, a cyber-physical system meta-model represents CPSs where an incident can occur, focusing on their components, structure and dynamic behavior. Second, an incident meta-model is proposed to represent incident patterns and instances. The full meta-models are implemented as Eclipse plugins that are available publicly[1].

A. Modeling Cyber-Physical Systems

We tailor our system meta-model to represent smart buildings, which are a specific application domain of CPSs. So far the analysis of security incidents in smart buildings [9] has received little attention and this has motivated our focus on this domain.

A simplified version of the smart building meta-model is shown in Fig. 3. Note that this meta-model can be extended to represent other time-discrete CPSs in domains different than smart buildings. Our meta-model includes *Asset*s, which can represent physical and cyber components in a smart building. Each Asset instance is identified by its *name*. *PhysicalAssets* represent any physical component, such as *Actor*, *PhysicalStructure*, and *ComputingDevice*. *Actor* can be a person in the smart building such as a *Visitor* or an *Employee*. *PhysicalStructure* represents part of the smart building physical layout, which includes *Room* and *Floor*. *ComputingDevice* represents any computing device, such as *Laptop*, *FireAlarm*, *SmartLight*, *Server*, *HVAC*, and *Workstation*. *DigitalAsset*s represent any digital data that can be processed or software that can be installed in a digital device inside the smart building. A DigitalAsset can also represent a network (e.g., *BusNetwork*) installed in the smart building.

An instance of the smart building meta-model representing the Research Centre of the ACME Company is shown in Fig. 4. For example, *sl1* and *toilet* are instances of *SmartLight* and *Room*, respectively.

To model the structure of a CPS, the meta-model also represents containment and connectivity relations between components. The *containedAssets* relation denotes the Asset(s) contained in a PhysicalAsset. The *containedDigitalAssets* relation

[1] https://tinyurl.com/yd9k6zhe

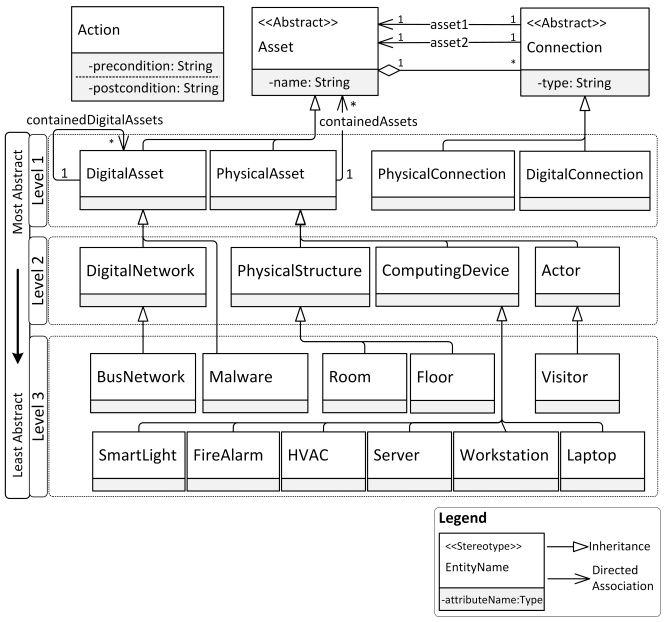

Fig. 3: Smart Building meta-model (simplified).

denotes the DigitalAsset(s) contained in another DigitalAsset. For example, as shown in Fig. 4, *sl1* is contained in the *toilet*, *sl2* and the *workstation* are contained in the *controlRoom*, and *sl3*, the *fireAlarm*, the *server* and the *hvac* are contained in the *serverRoom*. *Connection* represents connectivity between two components (*asset1* and *asset2*) and can be described by a *type* (e.g., wired). Digital connectivity between assets (e.g., through a network) is expressed as a *DigitalConnection*, while physical connectivity between assets (e.g., two rooms are connected through a door) is expressed as a *PhysicalConnection*. For example, as shown in Fig. 4, the *toilet* and the *serverRoom* are connected physically to the *hallway*, while *sl1-sl3*, the *fireAlarm* and the *workstation* are connected physically to the *busNetwork*. The *workstation* is also connected digitally to the *hvac* and the *fireAlarm*, to which it sends control commands.

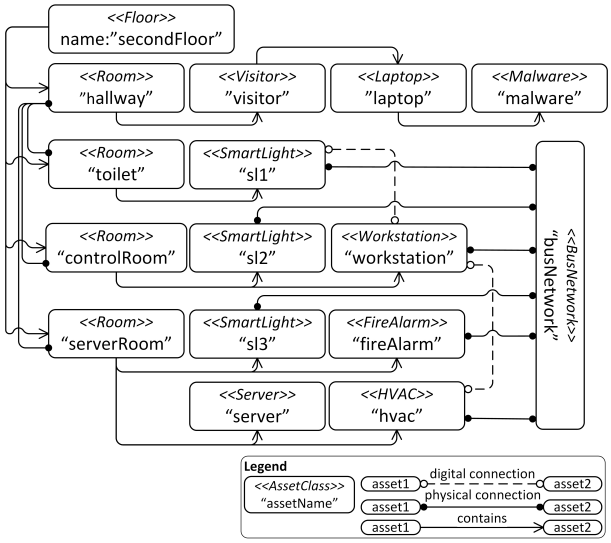

Fig. 4: Research Center instance (simplified).

To specify the dynamic behaviour of a CPS, the meta-model allows representing *Action*s. For example, Actions can represent a person entering a room or connecting his/her laptop to a computing device via the bus network. An Action is expressed as a re-writing rule, where a portion of the system matching a *pre-condition* is re-written with the sub-system represented in the *post-condition*. Pre- and post-conditions are expressed using a custom notation inspired by Bigraphical Reactive Systems (BRS) [15], which allows representing cyber and physical components and their connectivity and containment relations.

Table I represents pre- and post-conditions of actions "*enter Room*", "*connect Laptop to BusNetwork physically*", and "*connect Laptop to ComputingDevice via BusNetwork*".

TABLE I: Pre- & post-conditions of some actions of the smart building example.

enter Room
pre: $(Room_1\{phys\} \cdot Actor) \mid (Room_2\{phys\})$ **post:** $Room_1\{phys\} \mid (Room_2\{phys\} \cdot Actor)$
connect Laptop (Lap) to BusNetwork (Bus) physically
pre: $((Actor \cdot Lap) \mid Dev\{phys\}) \mid\mid Bus\{phys\}$ **post:** $((Actor \cdot Lap\{phys\}) \mid Dev) \mid\mid Bus\{phys\}$
connect Laptop to ComputingDevice (Dev) via BusNetwork
pre: $Actor \cdot Lap\{phys\} \mid\mid Bus\{phys\} \mid\mid Dev\{phys, dig\}$ **post:** $Actor \cdot Lap\{phys, dig\} \mid\mid Bus\{phys\} \mid\mid Dev\{phys, dig\}$

The precondition of Action "*enter Room*" means that two different rooms ($Room_1$ and $Room_2$) are connected physically ($\{phys\}$) and are contained in the same physical structure (see operator '|'), for example, the same floor. An $Actor$ is inside $Room_1$ (see operator '.'). As a result of Action "*enter Room*", the $Actor$, who was previously contained in $Room_1$, is now inside $Room_2$.

Action "*connect Laptop to BusNetwork physically*" indicates that an actor establishes a physical connection of a laptop to a bus network by replacing a computing device, which was previously connected to the bus network. As indicated in the pre-condition, an *Actor* who carries a laptop (Lap) (see operator '.') is initially co-located ('|') with a computing device (Dev). This device is in turn connected physically ($\{phys\}$) to the bus network (Bus). Also $Actor$ and Dev are not necessarily contained in the same location as Bus (see operator '||'). In the post-condition Lap is connected physically to Bus, replacing Dev.

Action "*connect Laptop to ComputingDevice via BusNetwork*" indicates that an actor connects a laptop digitally to a computing device through the bus network. The pre-condition indicates that an $Actor$ carries a laptop (Lap) (see operator '.'). The laptop (Lap) and the computing device (Dev) are connected physically ($\{phys\}$) to a bus network (Bus). The post-condition indicates that Lap establishes a digital connection (dig) with Dev. Note that $Actor$, Bus and Dev are not necessarily contained in the same physical structure (see operator '||').

CPS components can be defined at different levels of abstraction in the meta-model. Level 3 includes concrete entities, while Levels 1 and 2 include more abstract entities. A CPS, such as the one represented in Fig. 4, is described by instances

of the most concrete entities of the smart building meta-model (i.e. those in Level 3 in Fig. 3).

B. Modeling Incidents

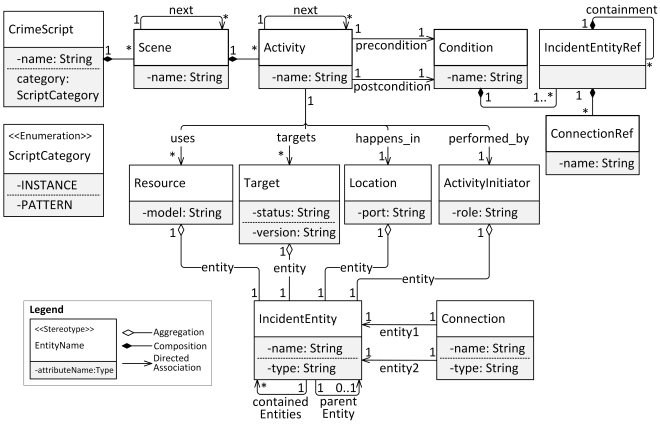

Fig. 5: Incident meta-model (simplified).

We take inspiration from *Crime Script*s to model security incidents. *Crime Script*s are used in criminology to describe the sequence of activities of physical crimes [16]. Despite their adoption for understanding the incident commission process and identifying incident prevention techniques, to the best of our knowledge, there exists no model that can be used to represent and process a *Crime Script* systematically. Thus, we have developed a meta-model that captures the characteristics of *Crime Script*s. To represent incidents that occur in CPSs our meta-model extends the original use of *Crime Script*s to refer to cyber components of the system explicitly. Our meta-model can be used to represent incident instances and incident patterns. An incident instance represents an incident that has occurred or may occur in a specific CPS, such as the *Research Center* in our motivating example. Therefore, incident instances can only refer to concrete CPS entities. An incident pattern is a more abstract representation of an incident, which can occur in various CPSs sharing common characteristics. Thus, incident patterns can only refer to entity types (classes) of the CPS meta-model.

A simplified version of the incident meta-model is shown in Fig. 5. A *CrimeScript* entity is characterized by a *name* and a *category*. A category indicates whether the incident model represents an incident *INSTANCE* or a *PATTERN*. A *CrimeScript* includes a set of partially ordered *Scenes*, which represent the phases of a security incident (e.g., preparation and execution scenes). Each scene, in turn, includes a set of *Activities*. An *Activity* is characterized by a *name* and is linked to the next subsequent activities in chronological order.

An *IncidentEntity* represents any entity that can be relevant to an incident, such as an offender or an asset, and it is characterized by a unique *name* and a *type*. An *IncidentEntity* can —not necessarily— play a role within an *Activity*. It can be a *Target*, a *Resource*, an *ActivityInitiator*, or a *Location*. A *Target* represents a component that can be harmed during an incident. It is characterized by a *status* (e.g., open, connected), and a *version* that indicates a version of a product (e.g., Windows 10). A *Resource* represents a component needed to perform an activity, such as a laptop or a malware, and is also characterized by a *model*. An *Activity* can be performed by an *ActivityInitiator* that may also play a *role* (e.g., offender, victim). A *Location* represents a place where an activity or a sequence of activities of a scene is performed. A *Location* can be physical or digital, depending on the *IncidentEntity* it refers to. A physical location represents a place in the physical space (e.g., a *Room*). A digital location represents a place in the cyberspace, such as an IP address or a digital folder. A *Location*'s *port* defines an access point to the location (e.g., a door for a physical location or port 80 for a digital location). We also allow specifying connectivity and containment relations of an *IncidentEntity* that are relevant for an incident instance. An *IncidentEntity* can contain other entities through relation *containedEntities*; the containing entity —if present— is represented through relation *parentEntity*. A *Connection* represents a physical or digital connectivity relation between two *IncidentEntity* objects.

An incident activity is also characterized by a *pre-* and a *post-condition*, which represent how a portion of the system evolved or should evolve as a consequence of the execution of the activity. Pre- and post-conditions are expressed by referring directly to incident entities and their connectivity and containment relations. More precisely, a *Condition* refers to a set of incident entities (*IncidentEntityRef*). An *IncidentEntityRef* is characterized by the *name* of the *IncidentEntity* it refers to and some of the incident entities it contains (*containment*), which are relevant to the activity. An *IncidentEntityRef* can also refer to a set of connections (*ConnectionRef*), where the referred incident entity is involved and which are relevant to the activity. A *ConnectionRef* refers to a *Connection* using its *name*. In the rest of this section we describe how to use the incident meta-model to represent incident instances and patterns.

1) Modeling Incident Instances: When the incident meta-model is used to represent an incident instance, the incident entities created to characterize the activities should refer to specific system *Asset*s in a cyber-physical system. Thus, the *name* and *type* of incident entities should refer respectively to the name (e.g., *sl1*) and the class (e.g., *SmartLight*) of an asset in a cyber-physical system. Containment and connectivity relations between incident entities should also refer to containment and connectivity relations between the corresponding system assets or actors in the cyber-physical system.

The incident instance model of the *Research Center* incident is depicted in Fig. 6. We only show details of two of the incident activities ("*enter toilet*" and "*connect laptop to hvac*"). Activity "*enter toilet*" has a *visitor* as an *ActivityInitiator*, who has the *role* of an *offender*. This Activity happens in room *hallway* (*Location*) and targets room *toilet* (*Target*). As shown in the model of the research centre in Fig. 4, the *toilet* also contains a smart light (*sl1*), which is connected physically to the bus network. Activity "*connect laptop to hvac*" aims to establish digital connectivity between a laptop and an hvac. It is still performed by the *visitor* inside the *toilet*. It targets the *hvac* and uses the *laptop* as a resource to establish connectivity.

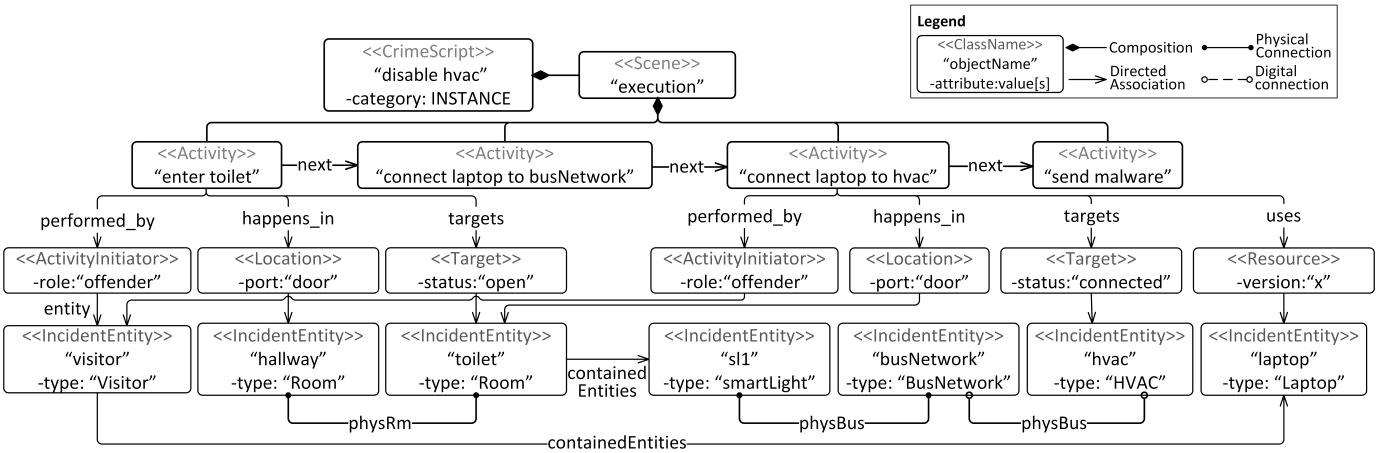

Fig. 6: Incident instance model of the Research Center incident.

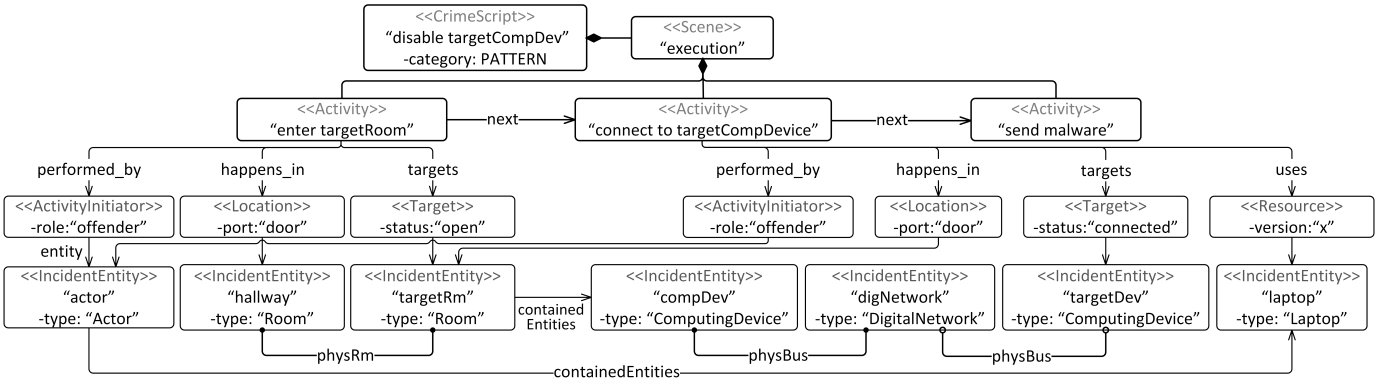

Fig. 7: A potential incident pattern model extracted from the incident instance model.

As shown in the model of the research centre in Fig. 4, the *hvac* is connected physically to the *bus network*.

An activity in an incident instance has a direct mapping to an action in the model of the cyber-physical system. Pre- and post-conditions of an activity are the same as pre-and post-conditions of the corresponding action but component types are replaced with concrete component instances from the system. Containment, physical and digital connectivity relations expressed in the pre- and post-conditions of an action are also replaced with concrete relations expressed in the system model. For example, as shown in the left side of Fig. 8, activity "*connect laptop to busNetwork*" is associated with action "*connect Laptop to BusNetwork physically*", which is specified in Table I. $Actor$, Lap, Dev and Bus in the action pre- and post-conditions are replaced by *visitor*, *laptop*, smart light *sl1*, and *busNetwork*, respectively. Physical connectivity $\{phys\}$ is replaced by *physBus*, i.e. the physical connectivity between the *busNetwork* and *sl1*. As shown in the right side of Fig. 8, activity "*connect laptop to hvac*" is created based on action "*connect Laptop to ComputingDevice via BusNetwork*" specified in Table I. $Actor$, Lap, Bus, and Dev are replaced by *visitor*, *laptop*, *busNetwork*, and *hvac*, respectively. Physical connectivity $\{phys\}$ is replaced by *physBus*, i.e. the physical connectivity between the *busNetwork* and the *hvac*, while $\{dig\}$ is replaced by a new digital connectivity between the laptop and the hvac, which is created as an effect of the action. If a security administrator needs to define an incident activity that does not have any corresponding action in the system model, s/he will first need to create a corresponding action in the system model. Then, s/he can associate the incident activity with this newly created action. Our approach assigns random names to incident activities in order to distinguish them from other activities in the same incident model. However, for reasons of clarity, in this paper we use intelligible names to identify incident activities.

2) Modeling Incident Patterns: When an incident pattern is represented, an incident entity can only refer to abstract system components that can match more than one concrete component in a cyber-physical system. Thus, in an incident pattern model the *name* of incident entities is just a random unique name (e.g., "*compDev*"), and the *type* refers to an entity type in the smart building meta-model (e.g., *ComputingDevice*).

A potential incident pattern model can be extracted from the incident instance model shown in Fig. 6. Such an incident pattern model is shown in Fig. 7. Each Incident Entity refers to a more abstract entity (e.g., in Level 2 in the cyber-physical system meta-model). For instance, incident instance *sl1* which has the type *SmartLight* is abstracted to an incident entity that has type *ComputingDevice*. Activities of an incindent instance can also be merged. For example, the two activities of

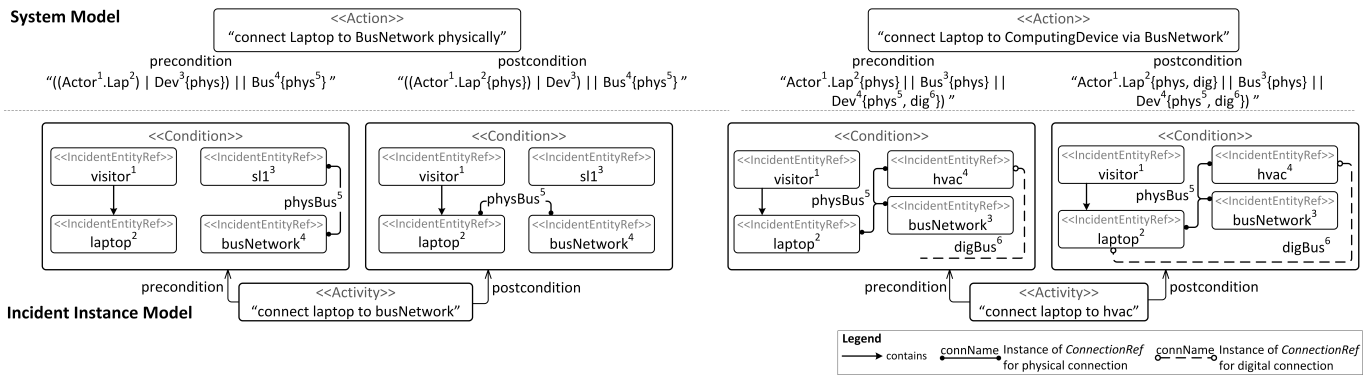

Fig. 8: Mapping the system actions *"connect Laptop to BusNetwork physically"* and *"connect ComputingDevice via BusNetwork"* to the incident instance activities *"connect laptop to busNetwork"* and *"connect laptop to hvac"* respectively.

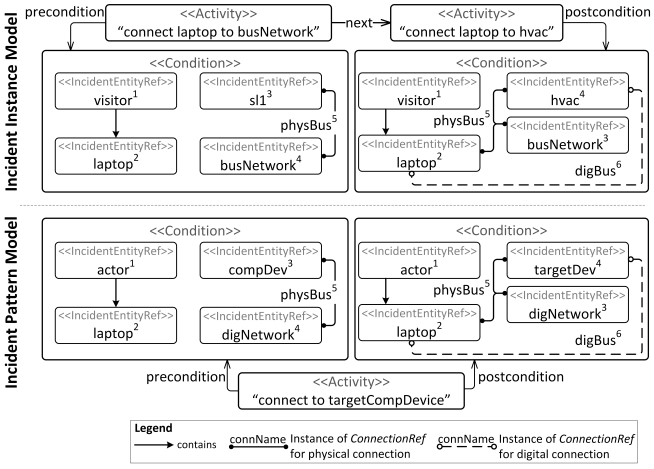

Fig. 9: Merging the incident instance activities *"connect laptop to busNetwork"* and *"connect laptop to hvac"* into the incident pattern activity *"connect to targetCompDevice"*.

the incident instance model, "*connect laptop to busNetwork*" and "*connect laptop to hvac*", are abstracted to one activity, "*connect to targetCompDevice*". In this case, the pre-condition is set to the pre-condition of the first activity ("*connect to busNetwork*") in the sequence. The post-condition is set to the post-condition of the last activity ("*connect to hvac*") in the sequence. Pre- and post-condition of the new activity, "*connect to targetCompDevice*" are shown in Fig. 9. The process we follow to extract an incident pattern is described in Section V.

V. Incident Pattern Extraction

In this Section we describe the technique we propose to extract an incident pattern from an incident instance, automatically. Our technique includes two steps: merging and abstraction.

A. Merging

We map a sequence of activities in the input incident instance to an activity pattern, which represents an activity that is usually performed by an offender. In our approach, we define manually a set of activity patterns based on the Common Attack Pattern Enumeration and Classification (CAPEC) catalog [17]. CAPEC provides more than 500 common attack patterns. An attack pattern describes, textually, the common attributes and approaches that an offender can exploit to harm or weaken a target system. To create our activity patterns, we analyze the CAPEC attack patterns and model them as activities of the incident meta-model. We only focus on attack patterns that can materialize in cyber-physical systems, particularly in smart buildings. Table II shows the name, category, and abstraction level of the attack patterns that we use to model activity patterns. CAPEC abstraction levels include *Standard*, which is a description of an attack technique (e.g., collect data from common resource locations [2]), and *Meta*, which is an abstract characterization of an attack technique (e.g., Excavation[3]).

TABLE II: Modeled CAPEC attack patterns.

Name	Category	Level
Collect Data from Common Resource Locations	Collect and Analyze Information	Standard
Sniffing Attacks	Collect and Analyze Information	Standard
Content Spoofing	Engage in Deceptive Interactions	Meta
Establish Rogue Location	Engage in Deceptive Interactions	Standard
Exploitation of Trusted Credentials	Subvert Access Control	Meta
Physical Theft	Subvert Access Control	Meta
Using Malicious Files	Subvert Access Control	Standard
Functionality Bypass	Abuse Existing Functionality	Meta
Email Injection	Inject Unexpected Items	Standard
Hardware Integrity Attack	Manipulate System Resources	Meta

A CAPEC attack pattern is characterized by a *description*,

[2]https://capec.mitre.org/data/definitions/150.html
[3]https://capec.mitre.org/data/definitions/116.html

an indication of its *severity*, a set of *resources* that offenders require to perform the attack, some *prerequisites* that should be met, and some *related weaknesses* that must be present in the target system —at least one of them— to perform the attack successfully. Weaknesses are usually represented using ids of relevant CWE (Common Weakness Enumeration) vulnerabilities. For example, Table III shows attack pattern "Collect Data from Common Resource Locations".

TABLE III: Collect Data from Common Resource Locations.

Description
An adversary exploits well-known locations for resources for the purposes of undermining the security of the target. Even when the precise location of a targeted resource may not be known, naming conventions may indicate a small area of the target machine's file tree where the resources are typically located. For example, configuration files are normally stored in the /etc directory on Unix systems. Adversaries can take advantage of this to commit other types of attacks.
Severity
Medium
Prerequisites
The targeted applications must either expect files to be located at a specific location or, if the location of the files can be configured by the user, the user either failed to move the files from the default location or placed them in a conventional location for files of the given type.
Resources Required
None
Related Weaknesses
CWE-ID: 552. Files or Directories Accessible to External Parties

The activity we create to represent attack pattern "Collect Data from Common Resource Locations" is shown in Fig. 10. We first read and examine the description of the attack pattern to identify some keywords that can be useful to define the activity initiator, location, target and/or resource. For example, keyword "adversary" indicates that the activity is performed by an actor playing the role of an *Offender*. Keyword "target" refers to the activity target, which in this case is a *File* inside a *SystemDirectory*. Resources can be identified from the resources required and/or the description of the attack pattern. In this example, although no resources are necessary, we still consider important for an attacker to use a computing device in order to access a file. Thus, in our activity pattern we define a resource to be a *ComputingDevice*. Relations between entities can also be identified from the description of the attack pattern. For example, we represent a containment relation between the *File* and the *SystemDirectory*.

Second, pre- and post-condition of the activity pattern are identified by examining the prerequisites, weaknesses, and description of the attack pattern. The prerequisites and weaknesses allow us to identify the pre-condition. In this example, from the prerequisites, we identify that the targeted *File* should be *contained* in the *SystemDirectory*. From the weaknesses we identify, as part of the pre-condition, that the offender should have accessibility to the location, which we translate as connectivity between the offender's *ComputingDevice* and the *SystemDirectory*. The post-condition is inferred from the description. In this example, we identify that the offender's *ComputingDevice* should contain the targeted *File*. Finally, the severity of the attack pattern is assigned directly to the severity attribute of the activity pattern. In this case, the severity of the activity pattern is set to *MEDIUM*.

In addition to CAPEC attack patterns, we create pattern activities based on common actions that occur in a smart building, such as movement between rooms, and connectivity to a network. For example, we create an activity pattern that expresses the activity of digitally connecting one's computing device (e.g., laptop) to another computing device (e.g., HVAC) through physical connectivity to a network. The created activity pattern is shown in Fig. 11. This activity pattern allows the merging of the two activities shown in Fig. 9.

During merging we identify sequences of activities in an incident instance that satisfy the pre- and post-condition of an activity pattern. The first activity in the sequence should satisfy the pre-condition of the activity pattern, while the last activity in the sequence should satisfy the post-condition of the activity pattern. A (pre- or post-)condition in the activity pattern matches a (pre- or post-)condition of an activity in the incident instance if the entity types (classes) referred in the condition of the activity pattern are types or super-types of the concrete system components referred in the incident instance. Also, the containment and connectivity relations between incident entities —referred in the condition of the activity pattern— should be included in the condition of the matching activity of the incident instance. To implement the matching we convert the pre- and post-conditions of an activity pattern and the activities in the incident instance as Bigraphs and we re-use the matching functionality implemented in LibBig [18], a publicly available library supporting various manipulation functionalities for Bigraphical Reactive Systems.

To perform the merging we look for an activity in the incident instance whose pre-condition matches the pre-condition of the activity pattern. If a match is identified, we look for a subsequent activity in the incident instance whose post-condition matches the post-condition of the activity pattern. If a match for the post-condition of the activity pattern is found, then the activity sequence, which begins from the activity that matches the pre-condition of the activity pattern and ends with the activity that matches the post-condition of the pattern, is kept.

Because different activity patterns can match an overlapping set of activities in an incident instance, we propose a strategy to prioritize the activity patterns to be selected for the matching. More precisely, we aim to maximize the number of activity patterns that match a non overlapping sequence of activities in an incident instance, and that have a maximum severity. We implement the selection of the activity patterns as a constraint solving problem, using Choco [19].

After the matching between each selected activity pattern and a sequence of activities in the incident instance is completed, each sequence is replaced by a new activity. The pre-condition of the new activity corresponds to the pre-condition of the first activity in the matching sequence. While the post-condition of the new activity corresponds to the post-condition of the last activity in the matching sequence. Activities in the incident instance that are not part of any sequence of activities matching an activity pattern remain unmodified.

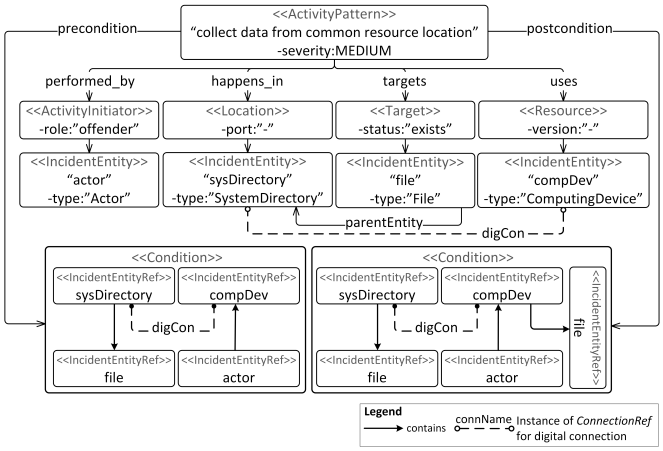

Fig. 10: "Collect data from common resource location" activity pattern.

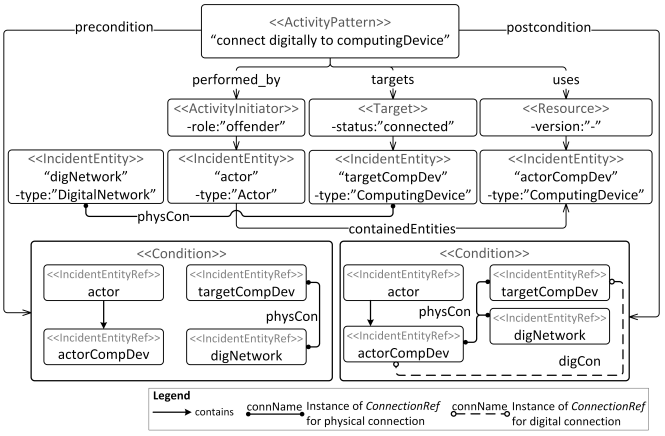

Fig. 11: "connect digitally to computingDevice" activity pattern.

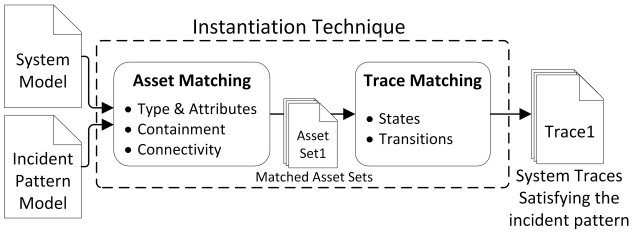

Fig. 12: The inputs, steps, and output of the instantiation technique.

B. Abstraction

After merging, we replace the system components (assets or actors) referred by each incident entity in the incident instance with a more abstract representation. We define heuristics to decide the level of abstraction for the system components. Generally, we abstract a component's type to one level up (i.e. to a more general type) in the system meta-model. For example, a component of type *SmartLight*, level-3, is abstracted to *ComputingDevice*, level-2. Similarly, we abstract a component's *Connection*s. For example, a *BusConnection* can be abstracted to a *DigitalConnection*. In other cases, instead, we keep the same level of abstraction. For example, a specific room (e.g., *hallway*) will be simply referred to as a *Room*. Determining the appropriate level of abstraction for different component types is a challenging task, and there is no silver bullet solution. Our rules are derived from our experience in using our incident meta-model to represent various incidents in smart buildings. However, incident patterns can also be reviewed by a security administrator, who can adjust the level of abstraction for selected system components, if needed. During abstraction, the *name* of a component is obfuscated. For example, *toilet* becomes *room1*. Moreover, only selected *properties* of a component are kept. Currently, property selection is performed manually by a security administrator. Finally, the *category* of the *CrimeScript* is changed from INSTANCE to PATTERN.

VI. Incident Pattern Instantiation

In this section we present the technique we propose to instantiate an incident pattern onto a specific cyber-physical system, in order to assess whether and how such pattern can manifest again. We exemplify our technique by instantiating the incident pattern represented in Fig. 7 to the research center smart building shown in Fig. 4. The inputs, steps, and output of the technique are shown in Fig. 12. It includes two steps: asset matching and trace matching.

A. Asset Matching

During asset matching we identify the concrete assets in the cyber-physical system that match the abstract assets referred by the incident entities in the incident pattern. To achieve this aim, we define a matching criteria that takes into account the type and other attributes of an incident entity, and also its containment and connectivity relations.

First, an incident entity "type-matches" an asset in a cyber-physical system if the *type* of the incident entity is a superclass or the same class of the asset. For example, incident entity *compDev* of type *ComputingDevice* in Fig. 7 can potentially match *sl1*, *fireAlarm*, and *server* in Fig. 4. These assets are of classes *SmartLight*, *FireAlarm*, and *Server*, respectively, which extend *ComputingDevice* (see Fig. 3). If an attribute is retained in an incident entity during extraction of the incident pattern, that attribute should be assigned the same value in the matched asset. For example, if attribute *status* is retained in *compDev* and is assigned value *OPEN*, then all matched assets should have their status set to *OPEN*.

Second, for each containment relation specified for a given incident entity, it is necessary to verify whether the assets that type-match the incident entity, in turn, contain assets that type-match the contained incident entities. The matching can be exact or partial. An exact matching implies that the asset type-matching a given incident entity should only contain assets type-matching exactly those contained by the incident entity. For example, if exact matching is required for incident entity *targetRm* that contains *compDev*, the asset

(e.g., *toilet*) matching *targetRm* should also contain exactly an asset type-matching *compDev* (e.g., *sl1*). A partial matching implies that an asset type-matching a given incident entity can contain additional assets, other than those type-matching the ones contained by the incident entity. For example, if partial matching is required for the incident entity *targetRm* that contains *compDev*, it is sufficient that the assets matching *targetRm* (e.g., *toilet*, *serverRoom*, *controlRoom*) contain at least one asset that type-matches *compDev*.

Finally, similar to containment relations, also connectivity relations should be matched; matching can be exact or partial. For example, if exact matching of connectivity relations is required for *compDev*, which has one *PhysicalConnection physBus*, then all matched assets (e.g., *sl1*, *sl2*, *hvac*) should have exactly one connectivity relation that type-matches *physBus*. If partial matching is required, then all matched assets (e.g., *sl1*, *sl2*, *hvac*) should have at least one connectivity relation that type-matches *physBus*. In this case, as shown in Fig. 4, *sl1*, *sl2*, *hvac* have one connectivity relation that type-matches (partially and exactly) *physBus*.

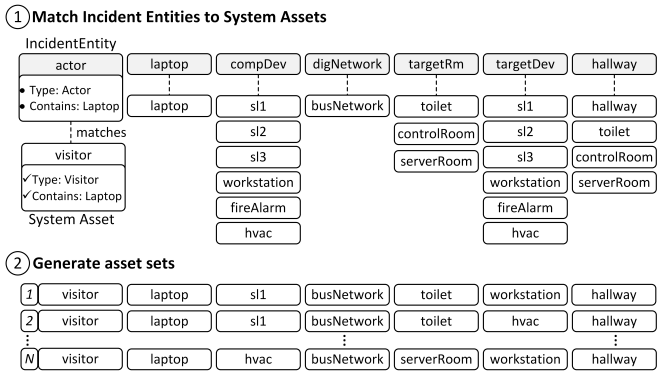

Fig. 13: Asset matching performed between the incident pattern, shown in Fig. 7, and the system model, shown in Fig. 4.

Fig. 13 (1) shows for each incident entity in the incident pattern the matching assets in the research center, assuming that partial matching is required for containment and connectivity relations. If at least one incident entity in the incident pattern cannot be matched to any asset, the incident pattern cannot be instantiated onto a given cyber-physical system. Otherwise, as shown in Fig. 13 (2), we generate sets of assets ($1..N$) matching each incident entity in the incident pattern. For example, [*visitor*, *laptop*, *sl1*, *busNetwork*, *toilet*, *workstation*, *hallway*] is one of the set of assets matching, respectively, the following incident entities [*actor*, *laptop*, *compDev*, *digNetwork*, *targetRm*, *targetDev*, *hallway*].

B. Trace Matching

During trace matching we identify a set of behavior traces (sequence of actions) of the cyber-physical system that satisfy the pre- and post-conditions of the activities in the incident pattern. Pre- and post-conditions need to be instantiated using the set of assets matching the incident entities, identified during asset matching.

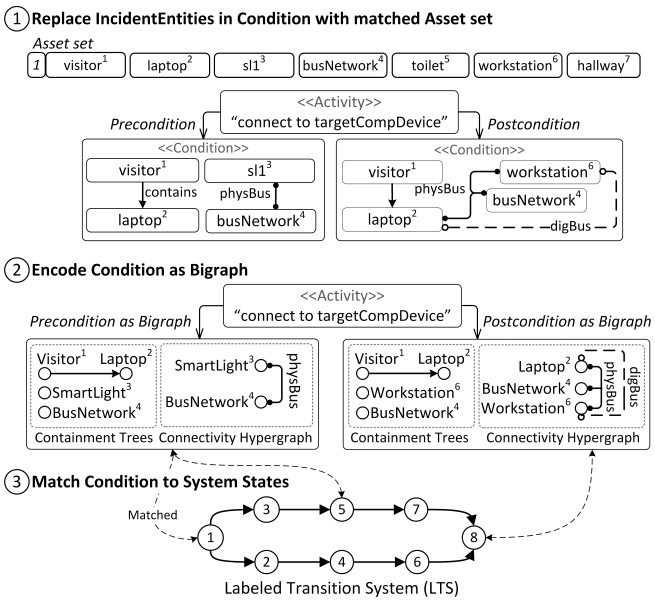

Fig. 14: Matching an incident pattern activity conditions (pre & post) against a LTS.

To achieve this aim, for each activity of an incident pattern, we identify the system states that satisfy their pre- and post-conditions. To do so, for each set of assets matching the incident entities, we replace each incident entity with its matched asset in the set, as shown in Fig. 14 (1). Then we encode pre- and post-conditions of activities in an incident pattern into a Bigraph, as shown in Fig. 14 (2). Each node of the Bigraph corresponds to an incident entity referred to in the pre- or post-condition. More precisely, the node control (i.e. type) corresponds to the class of the asset matching the incident entity referred to in the pre- or post-condition. For example, incident entity *compDev*, which is referred in the pre- and post- condition of activity "*connect to targetCompDevice*" matches asset *sl1* among others in Fig. 13; thus, it is replaced with a Bigraph node having control set to *SmartLight*. After that, we create containment and connectivity relations between these nodes based on existing relations in the incident pattern activity. For example, containment between the *visitor* and the *laptop* and connectivity between *sl1* and the *busNetwork* are maintained in the Bigraph.

Finally, we match the pre- and post-conditions of the incident pattern activities encoded as Bigraphs to system states. This is shown in Fig. 14 (3). We represent system states and transitions (action execution) using a Labeled Transition System (LTS), which is generated from the smart building model. To do this, we transform the structure of the smart building (components and their connectivity and containment relations) represented using the system model to a Bigraph [20]. We also encode the actions defined for the smart building to a set of reaction rules. We then use BigraphER [21], a software tool that implements bigraphs and their dynamics, to generate the LTS automatically.

Once the LTS is generated, we identify possible traces (sequences of states) matching the pre-and post-condition of an

incident activity, also preserving their order. The first state of the sequence satisfies the pre-condition of the matched incident activity, while the last state of the sequence satisfies the post-condition of the activity. We use *LibBig* [18] library to perform Bigraph matching.

We implemented a Breadth First Search (BFS) algorithm to identify a trace. We use BFS since it allows us to identify all unique and non-cyclic traces between two states in an LTS. By unique traces we mean that none of the traces identified from an initial state covers the same system states, i.e. a state cannot be visited twice. This avoids identifying repetitions of certain sequences of actions (partial-traces), hence, filtering out traces that are redundant. An example is shown in Fig. 15 (a), in which the trace (1→2→4→6→8) is skipped since state (6) is already visited via the trace (1→3→6→8). In other words, considering two alternative ways to reach state 6 (from state 3 or from states 2→4), we only consider the shortest one to keep the number of traces generated low. Similarly, to reduce the number of irrelevant traces, we modified the BFS algorithm to skip traces that contain an intermediate state, which satisfies the initial pre-condition of the incident. We do this since the intermediate state will be used (or was used) as an initial state by the BFS algorithm. This situation is depicted in Fig. 15 (b), in which trace (1→3→5→7→8) is skipped since state (5) is still satisfying the pre-condition of the incident pattern activity to be matched but not its post-condition. Thus, identified traces from different initial states are unique. We also bound trace's length to a maximum number of states.

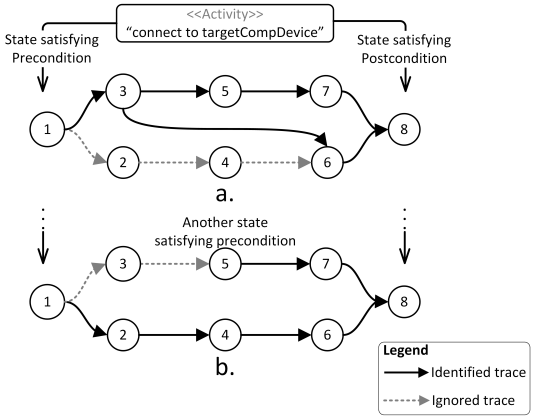

Fig. 15: a. Skipping traces containing states that are already visited. b. Skipping traces containing states that are satisfying the pre-condition.

Fig. 16 shows some of the traces generated during the instantiation of the incident pattern in Fig. 7. For each state, we indicate the number of the incident pattern activity that has its pre- or post-condition satisfied. For example, *2.post* indicates that the state satisfies the post-condition of activity 2; while *3.pre* indicates that the state satisfies the pre-condition of activity 3. The instantiation identifies two traces that match the activities in the incident pattern. The other traces are not identified as valid traces for the following reasons. First, a trace may not satisfy all the activities pre- and post-conditions. For example, one of the traces (A) in Fig. 16 does not satisfy the post-condition of activity 3 ("send malware"). Second, a trace (B) forms a cycle (i.e. "connect to workstation" & "disconnect from workstation" actions). Third, a trace (C) exceeds the maximum number of actions allowed —6 in this example.

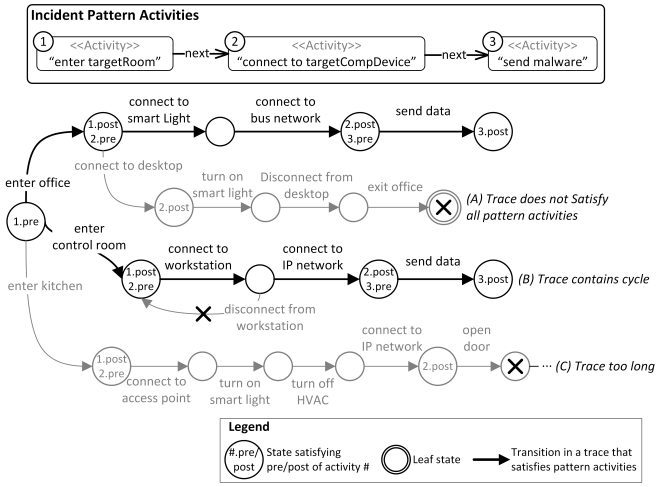

Fig. 16: An instantiation of incident pattern activities in a LTS.

We implemented filters that can be applied over the generated traces to identify those that can be more relevant. The shortest traces filter provides some insight about the traces requiring the minimum number of actions to cause an incident in a cyber-physical system. For example, the top trace in Fig. 16 is one among the shortest traces. We also provide a filter to identify traces containing the highest number of frequent actions (i.e. actions occurring in many traces) or the least frequent actions.

VII. Evaluation

We evaluated correctness, scalability and performance of our approach using two case studies inspired by real-world systems and incidents.

Correctness. We assessed correctness by verifying that both the extraction and instantiation techniques are sound. In other words, the extraction technique should generate an incident pattern that is "compatible" with the input incident instance and the model of the cyber-physical system where the incident supposedly occurred. While, the instantiation technique should generate a set of incident instances that are "compatible" with the incident pattern and a model of the cyber-physical system where the pattern is instantiated.

Compatibility between incident pattern and incident instance(s) was verified using the following criteria:

- Each activity of the incident pattern should map to one or a sequence of different activities of the incident instance.
- The activities of the incident pattern should map to all the activities of the incident instance in the same order.

An activity of the incident pattern maps to one or a sequence of activities of the incident instance if:

- Each incident entity referred to in the pre- and post-condition of the incident pattern activity can be mapped to an asset, respectively, in the pre-condition of the first

activity and the post-condition of the last activity of the sequence of activities in the incident instance. The type of the incident entity should correspond to the class or a superclass of the associated asset. Different incident entities cannot refer to the same asset.
- The assets mapped to the incident entities in the pre- and post-condition of the incident pattern activity have the same structure (containment and connectivity relations) of the corresponding incident entities.

If an incident pattern is instantiated in the same system in which the incident instance occurred, it should at least generate the same incident instance from which it was extracted. Moreover, the generated incident instances should include traces that exist in the LTS representing the system behavior.

We only assessed scalability and performance for the instantiation technique, because it uses the system LTS that can have a large number of states and transitions.

Scalability. We assessed whether our instantiation technique can scale for large systems, up to 100K states.

Performance. We assessed the time necessary to instantiate an incident pattern. To improve performance we parallelized our instantiation technique using multi-threading by allocating state matching tasks of different incident pattern activities to different threads.

A. Setup

We used the floor layout of two real physical buildings we have access to. Here, for reasons of confidentiality, we refer to the two buildings as RC1 and RC2. We modeled one of the floors of RC1 and RC2 using our smart building meta-model. For both buildings, we modified the floor layout by adding various smart devices (e.g., smart lights, motion sensors, air conditioning units) to mimic a realistic smart building setup. We also represented the dynamic behavior of the two building floors by generating their corresponding LTSs, also varying their size, using BigraphER. Note that BigraphER allows specifying a maximum number of states when generating the LTS. For RC1 we generated LTSs ranging from 10K [34K] to 43K [207K] states [transitions]; for RC2 we generated LTSs ranging from 10K [34K] to 100K [445K] states [transitions].

We modeled an incident instance that occurred in RC1 and extracted an incident pattern that we instantiated on RC1 and RC2. The incident instance is about an offender that collects data transmitted over the bus network. It consists of the following activities: enter RC1, move to a room containing a fire alarm, connect to bus network through the fire alarm, collect data transmitted on the bus, and analyze collected data to identify location of critical assets. The incident pattern consists of three abstract activities: reach a location in smart building that contains a device connected to internal network, gain access to the internal network via device, and collect transmitted data. The model of the floors of RC1 and RC2, the incident patterns and incident instances used in our evaluation are available online[4]. The interested reader can replay the extraction and instantiation techniques on smaller versions of the LTSs generated for RC1 and RC2. We evaluated our extraction and instantiation techniques on a virtual machine that has Ubuntu 18.04.1 LTS 64bits as operating system, Intel® Xeon® CPU E5-2660 2.2GHz (32 CPUs), and 64GB of memory.

B. Results

We evaluated *correctness* of the *extraction* technique by verifying that the incident pattern generated from the incident instance that occurred in RC1 is compatible with the incident instance. In other words, we verified that each activity in the generated incident pattern maps to at least one action in the incident instance that occurred in RC1, and that the generated activities map to all the actions of the instance. We evaluated *correctness* of the *instantiation* technique by checking that all generated incident instances instantiated in RC2 are compatible with the incident pattern. We also verified that all generated traces belong to the LTS generated for RC2. Our results suggest that our techniques produce sound output for the incident scenarios and systems on which they were evaluated.

We assessed *scalability* of the *instantiation* technique by instantiating the incident pattern in RC2 using LTSs of increasing sizes (10K-100K states). Table IV shows the number of generated incident instances for different LTS sizes. As can be noticed from Table IV, the number of generated instances increases as the size of the LTS increases, because, intuitively, a bigger LTS provides additional ways in which an incident can occur.

Moreover, although the traces generated are compatible with the activities of the incident pattern, they may contain some actions that are not necessary to match the activities of the incident pattern. To filter out the traces that contain those unnecessary actions, we mine frequent sequential patterns of actions, i.e. subsequent actions that occur frequently inside the generated traces. We only consider the shortest traces that share a frequent sequential pattern of actions that has the maximum length. This allows security administrators to focus their attention on traces that only contain sequences of actions that are necessary to cause a security incident. Frequent sequential patterns of actions are identified using the ClaSP Algorithm [22], which is implemented in an open source data mining library called SPMF[5]. The number of filtered traces found during instantiation of the incident patterns on RC2 are shown in Table IV. We can notice a significant reduction in the number of traces between *Generated* and *Filtered*. However, other filtering criteria can also be adopted. For example, considering only those traces containing actions occurring more or less frequently among all the traces.

We evaluated *performance* of the *instantiation* technique by measuring the time that the instantiation technique requires to generate all the traces using an LTS of increasing size, representing the dynamic behaviour of RC2. The execution times for instantiating the incident pattern activities in RC2 are depicted in Fig. 17. To enhance performance, we applied parallelization in various parts of the instantiation technique.

[4]https://github.com/FaeqAlrimawi/SharingIncidentKnowledge

[5]http://www.philippe-fournier-viger.com/spmf/

TABLE IV: Output of the instantiation technique applied over different LTS sizes of RC2. Output shown is all generated traces and relevant ones.

LTS		Instantiation Output	
States	**Transitions**	**Generated traces**	**Filtered traces**
10,000	33,850	720	40
20,000	73,734	2,845	90
30,000	110,569	5,403	120
40,000	158,477	10,648	160
50,000	198,771	14,777	200
60,000	252,897	23,848	240
70,000	295,160	98,720	265
80,000	349,517	143,186	305
90,000	399,319	184,269	310
100,000	445,028	216,561	340

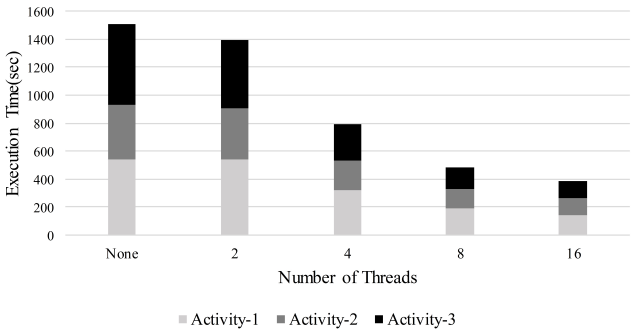

Fig. 18: Execution time of instantiating incident pattern activities using different number of threads in RC2 with LTS of size 10,000.

For example, matching system states to activity conditions was carried out by dividing all states into subsets that are then handled by different threads. Fig. 18 shows performance improvement when increasing the number of threads. Execution time is almost cut by half using 4 threads, and reaches about a quarter the original time (i.e. without parallelism) using 16 threads.

C. Threats to Validity

There are several threats that may affect the validity of our approach.

Some threats are related to the internal validity of our approach. Implementation of the abstraction and instantiation techniques can affect correctness of our results. Matching incident pattern activities pre- and post-conditions to LTS states can be error-prone. We use *BigraphER* to generate Bigraph states, but use *LibBig* to match generated Bigraph states to Bigraph conditions. *BigraphER* and *LibBig* may adopt a different interpretation of BRS. To eliminate this threat, we verified that the matching results obtained using *BigraphER* are consistent with those obtained using the *LibBig* library. We also used widely adopted tools (e.g., Choco constraint solver) and algorithms (e.g., shortest path and breadth first search) to implement our abstraction and instantiation techniques.

Another threat to the correctness of our instantiation technique can arise from bounding. In particular, to reduce instantiation complexity we limit the size of the LTS to a given bound and we also limit the length of the generated traces. This can lead to missing potential traces that satisfy an incident pattern. This threat can be mitigated by defining a bounding of the LTS, depending on the number of actions in a cyber-physical system, and a maximum trace length, depending on the number of activities in an incident pattern. In future work we will investigate techniques to identify an optimal bound for the LTS and a maximum length for the generated traces.

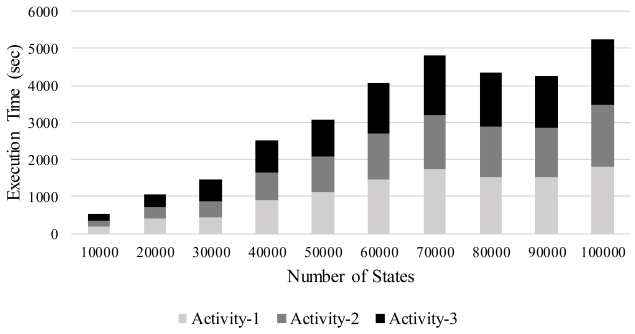

Fig. 17: Execution time of incident pattern activities measured over various LTS sizes (10K-100K states) of RC2.

Other threats are related to the external validity of the approach and, more precisely, to the generalizability of the results to other smart buildings and cyber-physical systems. To alleviate this threat we created the meta-model of the smart building in collaboration with practitioners working on physical access control. The meta-model was validated by using it to represent various smart buildings. Note that our meta-model is extensible and can incorporate additional components (e.g., digital devices) if necessary using polymorphism. In future work we will extend our system meta-model to represent other cyber-physical systems, such as smart cities and transport. We are confident this is feasible because these types of cyber-physical systems can still be described in terms of containment and connectivity relations between their components [23].

Although our work is motivated by practical problems that organizations may face in representing and sharing security incident knowledge, threats to validity can arise from the practicality of our approach. In particular, modeling smart buildings and incidents could be tedious. It requires security administrators to be familiar with the meta-model that we propose to represent smart buildings and incidents. To facilitate modeling of smart buildings and incidents, we provide a graphical designer [24]. This allows extraction of the physical structure of a building directly from its BIM (Building Information Modeling) representation. It also allows re-using actions that have already been defined to represent other smart buildings. Use of this tool can greatly reduce modeling time and effort. We are currently working on improving the implementation of this prototype to foster its wider adoption.

Applying our approach to systems more complex than those considered in our evaluation can still be problematic and bring scalability issues. This limitation can be addressed by exploit-

ing connectivity and containment relations to decompose a smart building into several parts (or floors), which can be analyzed separately.

VIII. Related Work

Previous work has focused on representing and analyzing purely cyber incidents and only few approaches represent and analyze incidents in CPSs. Current attack modeling techniques (e.g., attack graphs [25], cyber kill chain [26], and Diamond model [27]) focus on representing and analyzing how a traditional cyber attack (e.g., denial of service) can occur. As these techniques do not account for the interactions between cyber and physical components, they may not be suitable to represent and analyze cyber-physical incidents [28]. Existing resources for incident information are also focused on cyber incidents. For example, the Common Vulnerabilities & Exposures (CVE) [17] is a publicly available dictionary of known cybersecurity vulnerabilities in software and devices.

Liu et al. [29] focus their work on analyzing specific attacks, i.e. switching attacks, that can occur in the domain of smart grids. Do et al. [30] propose an adversary model to analyze attacks in smart homes, which exploit weaknesses (e.g., unencrypted communications) in common devices (e.g., smart lights). Their adversary model consists of several capabilities: listen (e.g., eavesdrop on local communication between a smart light and a smartphone), transmit (e.g., transmit messages to detect the presence of some devices), modify (e.g., send spoofing messages), and intercept (e.g., drop messages). Such approaches focus on providing analysis of a specific attack in a specific system. Hence, techniques developed cannot be applied to analyze different types of attacks that can also happen in other application domains. Additionally, they do not represent incident entities (e.g., motivation) that can be useful in case an investigation is required. Hawrylak et al. [31] propose a Hybrid Attack Graph (HAG) to model cyber-physical attacks. Their approach produces a graph of all possible ways a set of exploit patterns can be applied to a system. However, the approach focuses on representing malicious actions that exploit vulnerabilities found in some devices and does not consider other non-malicious interactions between cyber and physical components that can lead to undesired state in a system.

Few approaches tackle security incident representation and sharing. Bollé and Casey [32] propose an approach to identify and share linkages between cyber crime cases in order to support digital investigations. The approach employs exact and near string similarity algorithms (e.g., Levenshtein algorithm) to identify similar digital traces between different cyber crimes. Their current implementation focuses on identifying similarities in email addresses. While it can be useful to identify and share similar digital traces, the approach focuses on cyber interactions, which may not be sufficient for investigating incidents in CPSs. STIX (Structured Threat Information Expression) [33] has been proposed as a language for representing Cyber Threat Intelligence (CTI). CTI is shared using TAXII (Trusted Automated Exchange of Intelligence Information) [34]. STIX provides only a representation for CTI without any analysis of how CTI should be extracted from one organization or how it should be instantiated in another. Finally, Fani and Bagheri [35] provide a light-weight security incident ontology to represent security events. The ontology consists of temporal, spatial, and event entities. However, their ontology is centralized around events and do not represent other characteristics of the space, such as containment and connectivity relations or offender's motive, which can be relevant to the incident.

IX. Conclusions & Future Work

In this paper we proposed a novel approach for representing and sharing security incident knowledge across different organizations. We suggested that knowledge of security incidents can be represented as *incident patterns*. Incident patterns are a more abstract representation of specific incident instances. Thus they can be applied over different systems. They can also avoid disclosing potentially sensitive information about an organization's assets and resources (e.g., physical structure of a building or vulnerable devices). We also provided a representation of the system where an incident occurs. We provided two meta-models to represent incidents and cyber-physical systems, respectively. To obtain incident patterns, we proposed an automated technique to *extract* an incident pattern from a specific incident instance. To understand how an incident pattern can manifest again in other systems, we proposed an automated technique to *instantiate* incident patterns to different systems. We demonstrated the feasibility of our approach in the application domain of smart buildings. We evaluated correctness, scalability, and performance using two substantive scenarios inspired by real-world systems and incidents.

In future work we plan to apply our approach to design security controls that can prevent or mitigate potential security incidents obtained from the traces generated during incident pattern instantiation. We also aim to use our approach to improve system accountability and support forensic readiness [36], [37], which is the ability of a system to proactively identify and collect data that can be used in possible future investigations. In particular, we intend to develop an automated technique to design monitors that can collect data necessary to detect and/or investigate the incidents represented by the traces generated during pattern instantiation. We will assess applicability of our approach to a wider set of scenarios, for example, when an incident happens across different floors, and across smart buildings. We will also extend our approach to other types of CPS such as smart cities and transportation which can still be characterized using containment and connectivity relationships between their components. Finally, we are planning to increase automation of incident reporting activities. In particular, we plan to use natural language processing techniques to generate attack patterns from descriptions of CAPEC patterns, and more generally to automate incident modeling from incident reports.

Acknowledgment

This work was partially supported by ERC Advanced Grant no. 291652 (ASAP), EPSRC, and Science Foundation Ireland grants 13/RC/2094 and 15/SIRG/3501.

REFERENCES

[1] E. A. Lee, "CPS Foundations," in *Proc. of 47th ACM/IEEE Design Automation Conference*, 2010, pp. 737–742.

[2] G. Loukas, *Cyber-physical Attacks: A Growing Invisible Threat*. Butterworth-Heinemann, 2015.

[3] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the Cyber Attack on the Ukrainian Power Grid," *Electricity Information Sharing and Analysis Center (E-ISAC)*, 2016.

[4] ——, "German Steel Mill Cyber Attack," *Industrial Control Systems*, vol. 30, p. 62, 2014.

[5] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, S. Sastry *et al.*, "Challenges for Securing Cyber Physical Systems," in *Workshop on Future Directions in Cyber-Physical Systems Security*, vol. 5, 2009.

[6] K. A. Scarfone, T. Grance, and K. Masone, "SP 800-61 rev. 1. Computer Security Incident Handling Guide," 2012.

[7] T. Schreck, "IT Security Incident Response: Current State, Emerging Problems, and New Approaches," Ph.D. dissertation, Friedrich-Alexander-University Erlangen-Nuremberg (FAU), 2018.

[8] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, "Communication Systems for Building Automation and Control," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1178–1203, 2005.

[9] T. Mundt and P. Wickboldt, "Security in Building Automation Systems - A First Analysis," in *Proc. of the 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security)*, 2016, pp. 1–8.

[10] B. Krebs, "IoT Reality: Smart Devices, Dumb Defaults," https://krebsonsecurity.com/2016/02/iot-reality-smart-devices-dumb-defaults, 2016.

[11] A. Ahmad, J. Hadgkiss, and A. B. Ruighaver, "Incident Response Teams–Challenges in Supporting the Organisational Security Function," *Computers & Security*, vol. 31, no. 5, pp. 643–652, 2012.

[12] M. J. West-Brown, D. Stikvoort, K.-P. Kossakowski, G. Killcrece, and R. Ruefle, "Handbook for Computer Security Incident Response Teams (csirts)," Carnegie-mellon univ pittsburgh pa software engineering inst, Tech. Rep., 2003.

[13] N. Tulechki, "Natural Language Processing of Incident and Accident Reports: Application to Risk Management in Civil Aviation," Ph.D. dissertation, Université Toulouse le Mirail-Toulouse II, 2015.

[14] F. Alrimawi, L. Pasquale, D. Mehta, and B. Nuseibeh, "I've Seen This Before: Sharing Cyber-Physical Incident Knowledge," in *Proc. of the 1st International Workshop on Security Awareness from Design to Deployment, SEAD@ICSE 2018, Gothenburg, Sweden, May 27, 2018*, 2018, pp. 33–40.

[15] R. Milner, "Pure bigraphs: Structure and dynamics," *Information and computation*, vol. 204, no. 1, pp. 60–122, 2006.

[16] D. Cornish, "Crimes as scripts," in *Proceedings of the inter. seminar on env. criminology and crime analysis*. Tallahassee: Florida Criminal Justice Executive Institute, 1994, pp. 30–45.

[17] MITRE Corporation, "Common Attack Pattern Enumeration & Classification." [Online]. Available: http://capec.mitre.org/

[18] M. Miculan and M. Peressotti, "A CSP implementation of the bigraph embedding problem," dec 2014.

[19] C. Prud'homme and J.-G. Fages, "Choco Solver." [Online]. Available: http://www.choco-solver.org/

[20] R. Milner, *The Space and Motion of Communicating Agents*, 1st ed. New York, NY, USA: Cambridge University Press, 2009.

[21] M. Sevegnani and M. Calder, "BigraphER: Rewriting and Analysis Engine for Bigraphs," in *International Conference on Computer Aided Verification*. Springer International Publishing, 2016, pp. 494–501.

[22] A. Gomariz, M. Campos, R. Marin, and B. Goethals, "ClaSP: An efficient algorithm for mining frequent closed sequences," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer Berlin Heidelberg, 2013, pp. 50–61.

[23] L. Pasquale, C. Ghezzi, C. Menghi, C. Tsigkanos, and B. Nuseibeh, "Topology Aware Adaptive Security," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2014, pp. 43–48.

[24] F. Alrimawi, L. Pasquale, and B. Nuseibeh, "On the Automated Management of Security Incidents in Smart Spaces," *IEEE Access*, vol. 7, pp. 111 513–111 527, 2019.

[25] H. S. Lallie, K. Debattista, and J. Bal, "An Empirical Evaluation of the Effectiveness of Attack Graphs and Fault Trees in Cyber-Attack Perception," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2017.

[26] T. Yadav and A. M. Rao, "Technical aspects of cyber kill chain," in *Communications in Computer and Information Science*, vol. 536. Springer Verlag, 2015, pp. 438–452.

[27] S. Caltagirone, A. Pendergast, and C. Betz, "The Diamond Model of Intrusion Analysis," Tech. Rep., 2013. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a586960.pdf

[28] M. Yampolskiy, P. Horváth, X. D. Koutsoukos, Y. Xue, and J. Sztipanovits, "A language for describing attacks on cyber-physical systems," *International Journal of Critical Infrastructure Protection*, vol. 8, pp. 40–52, jan 2015.

[29] S. Liu, S. Mashayekh, D. Kundur, T. Zourntos, and K. Butler-Purry, "A framework for modeling cyber-physical switching attacks in smart grid," *IEEE Transactions on Emerging Topics in Computing*, vol. 1, no. 2, pp. 273–285, 2013.

[30] Q. Do, B. Martini, and K. K. R. Choo, "Cyber-physical systems information gathering: A smart home case study," *Computer Networks*, vol. 138, pp. 1–12, mar 2018. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128618301440

[31] P. J. Hawrylak, M. Haney, M. Papa, and J. Hale, "Using hybrid attack graphs to model cyber-physical attacks in the Smart Grid," in *5th ISRCS*, 2012, pp. 161–164.

[32] T. Bollé and E. Casey, "Using computed similarity of distinctive digital traces to evaluate non-obvious links and repetitions in cyber-investigations," *Digital Investigation*, vol. 24, pp. S2–S9, mar 2018.

[33] Mitre, "Standardizing Cyber Threat Intelligence Information with the Structured Threat Information eXpression (STIX ™)," *MITRE Corporation*, vol. 11, pp. 1–22, 2012.

[34] OASIS Open, "Introduction to TAXII." [Online]. Available: https://oasis-open.github.io/cti-documentation/taxii/intro

[35] H. Fani and E. Bagheri, "An Ontology for Describing Security Events," in *The 27th International Conference on Software Engineering and Knowledge Engineering, {SEKE}*, 2015, pp. 455–460.

[36] R. Rowlingson, "A ten step process for forensic readiness," *International Journal of Digital Evidence*, vol. 2, no. 3, pp. 1–28, 2004.

[37] L. Pasquale, D. Alrajeh, C. Peersman, T. Tun, B. Nuseibeh, and A. Rashid, "Towards Forensic-ready Software Systems," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. ACM, 2018, pp. 9–12.

Faeq Alrimawi received the B.Sc. degree in Computer System Engineering from Birzeit University, Palestine in 2010. In 2012, he received the M.Sc. degree in mobile & high speed telecommunication networks from Oxford Brookes University, UK.

He is pursuing a PhD degree in Computer Science at Lero – The Irish Software Research Centre, University of Limerick, Ireland. His research interests include software engineering, digital forensics, and security for cyber-physical systems.

Liliana Pasquale received the PhD degree from Politecnico di Milano (Italy), in 2011. She is a lecturer at University College Dublin (Ireland) and a researcher at Lero - the Irish Software Research Centre.

Her research interests include requirements engineering and adaptive systems, with particular focus on security, privacy, and digital forensics. She has served in the Program and Organizing Committee of prestigious software engineering conferences, such as ICSE, FSE, ASE, RE. She is also part of the review committee of the IEEE TSE journal and the TOSEM journal.

Deepak Mehta is a principal research engineer in Huawei Research Centre, Paris. Before joining Huawei, he was a principal research scientist in United Technologies Research Centre in Cork, Ireland. His expertise is in the design and development of solutions for very large-scale discrete combinatorial problems arising in different industries, e.g., aerospace, data centre, telecommunications, bioinformatics via implementing decision/optimisation models and custom solutions. He has also worked on machine learning-based solutions for cyber-physical security, bioinformatics, manufacturing and generic reasoning engines. Before moving to Industry, he was a senior research scientist in the Insight Centre for Data Analytics, University College Cork, Ireland. During that time he also worked on Google-ROADEF/EURO Challenge where his team was runner up in two categories (opensource and senior). He is a co-author of 50+ technical articles. He received his PhD in computer science from University College Cork in 2009 for his work in constraint programming. During his PhD, he also won the first constraint programming solver competition (in binary category).

Nobukazu Yoshioka is an associate professor at the National Institute of Informatics, Japan. Dr. Nobukazu Yoshioka received his M.E. and Ph.D. degrees in School of Information Science from Japan Advanced Institute of Science and Technology in 1995 and 1998, respectively.

His research interests include Security and Privacy Software Engineering and Software Engineering for Machine Learning-based Systems. He is a member of the Information Processing Society of Japan (IPSJ), the Institute of Electronics, information and Communication Engineers (IEICE) and Japan Society for Software Science and Technology (JSSST), the Japanese Society for Artificial Intelligence (JSAI) and IEEE CS. He has been a board member of a SIG of Machine Learning Systems Engineering since 2018, a board member of JSSST from 2011 to 2015 and an auditor of JSSST since 2017. He is a chair of IEEE CS Japan Chapter in 2020.

Bashar Nuseibeh is Professor of Computing at The Open University and a Professor of Software Engineering and Chief Scientist at Lero - The Irish Software Research Centre. He is also a Visiting Professor at University College London (UCL) and the National Institute of Informatics (NII), Tokyo, Japan. He serves as Editor-in-Chief of ACM Transactions on Autonomous and Adaptive Systems and Associate Editor of IEEE Security & Privacy Magazine.

He chaired the Steering Committee of the International Conference on Software Engineering (ICSE) and received an ICSE Most Influential Paper Award, a Philip Leverhulme Prize, an Automated Software Engineering Fellowship, and a Royal Academy of Engineering Senior Research Fellowship. He received an IFIP Outstanding Service Award (2009) and an ACM SIGSOFT Distinguished Service Award (2015). He is the recipient of a Royal Society-Wolfson Merit Award and two European Research Council (ERC) awards, including an ERC Advanced Grant on 'Adaptive Security and Privacy'. He is a Fellow of the British and Irish Computer Societies, a Fellow of the Institution of Engineering & Technology, and a Member of Academia Europaea.