Detection of attacks based on known vulnerabilities in industrial networked systems

Manuel Cheminod, Luca Durante, Lucia Seno, Adriano Valenzano
CNR-IEIIT, c.so Duca degli Abruzzi 24, I-10129 Torino, Italy
{manuel.cheminod, luca.durante, lucia.seno, adriano.valenzano}@ieiit.cnr.it

Authors' Accepted Manuscript (AAM): this is the *accepted* version of the paper

M. Cheminod, L. Durante, L. Seno, and A. Valenzano, "Detection of attacks based on known vulnerabilities in industrial networked systems", *Journal of Information Security and Applications*, Vol. 34, Part 2 (June 2017), pp. 153-165, Elsevier Science B. V., Amsterdam (The Netherlands). DOI: 10.1016/j.jisa.2016.06.003.

available at https://doi.org/10.1016/j.jisa.2016.06.003.

Detection of Attacks Based on Known Vulnerabilities in Industrial Networked Systems

Manuel Cheminod, Luca Durante, Lucia Seno, Adriano Valenzano

CNR-IEIIT, c.so Duca degli Abruzzi 24, I-10129 Torino, Italy
{*manuel.cheminod, luca.durante, lucia.seno, adriano.valenzano*}*@ieiit.cnr.it*

Abstract

Vulnerabilities in software and hardware components can be exploited by attackers to cause damages through the cyberspace. Nowadays, this problem also affects a large number of industrial networked systems (INS) and experts are well aware that suitable prevention/detection techniques and countermeasures have to be developed, taking into account INS characteristics and peculiarities. The exposure of a large and complex system to attacks carried out by exploiting well-selected sequences of vulnerabilities can be hard to evaluate, but this is a fundamental step to prevent potential menaces in both the system design and operation phases. This paper deals with an innovative technique, which is able to compute all attack patterns leveraging known vulnerabilities present in an industrial system. The proposed approach is based on the extension of a twofold model, which was successfully developed for verifying the implementation of access control policies in INS. Our solution enables the development of an automated software analyser that can help with the design and maintenance of INS when their security is considered.

Keywords: Industrial distributed systems, software vulnerabilities, attack paths, automated analysis

1. Introduction

Cyber-security has become a major issue of concern in most application areas which involve cyber-physical systems either directly or indirectly [1, 2, 3, 4, 5]. Among the others, industrial networked systems (INS), such as those used for automated production plants and process control, are now exposed to the same menaces and attacks, carried out by hackers and saboteurs, as

those experienced by many non-industrial Internet users every day. Major reasons for this are a higher degree of openness, which has progressively been introduced in INS with respect to the proprietary solutions widely adopted till some years ago, and the ever increasing use in industrial scenarios of advanced information and communication technologies (ICT), such as wireless communications and off-the-shelf hardware (h/w) and software (s/w) components, that were originally conceived for boosting general-purpose and/or consumer applications.

Electronic devices and s/w products are notoriously known for not being perfect, at least from the security point of view, and are frequently subject to patches and updates when bugs and weaknesses are discovered. Unfortunately, this good practice, which is now routine in many networked systems, can be rarely adopted in INS, because of their well-know peculiarities [4, 5]. Vulnerabilities, which unavoidably affect IT components, can be leveraged by malicious users to carry out attacks and cause damages to the target systems in the broader sense. Vulnerabilities are not the same as attacks, but rather a means to make possible attacks successful. This may occur when vulnerabilities are exploitable so that attackers can take advantage of them. However, a system can tolerate the presence of non-exploitable vulnerabilities because they cannot be leveraged to carry out attacks, and this aspect should also be considered in many INS where changes to the h/w and/or s/w are hardly possible or simply not financially convenient. In this case, in fact, the presence of vulnerabilities can be tolerated until they are proven not to be exploitable for causing harms/damages to the people, the system and the environment.

Unfortunately, weaknesses that are not exploitable on their own, can also give rise to serious security threats and enable attacks when they are suitably combined in the same network. In this case, in fact, attackers can leverage well-selected patterns of known vulnerabilities to carry out sequences of steps that eventually result in reaching their malicious goals. Therefore, detecting such a kind of attack paths before their possible exploitation is of utmost importance to prevent unwanted consequences for the system.

Software vulnerabilities, in particular, are of predominant practical interest because they are frequently included in products and are easily replicated, so that they are being collected and sorted in publicly-available databases [6, 7, 8, 9] since several years. Vulnerability databases are maintained and updated as new possible security breaches are discovered, and some information repositories, that are particularly oriented to industrial ICT components and

supervision, control and data acquisition (SCADA) systems [10, 11], have also been started and are regularly updated.

Our past investigations on the automated analysis and detection of attacks based on sequences ("chains" in [12, 13]) of known vulnerabilities have enabled us to pioneer researches in this area and to evaluate pros and cons of the original solution described in [12, 13].

More recently, we have presented a new technique [14, 15], based on a twofold innovative model, for computer-aided security analyses (and, in particular, for checking the implementation of access control policies as described in [16, 17]) by means of automatic software tools, that has been conceived bearing in mind the typical requirements and peculiarities of INS. The proposed approach and underlying formal model can be successfully extended to the study of attacks leveraging sequences of vulnerabilities as shown in [18]. This paper focuses on the extension of the technique introduced in [14, 15] to enable the semi-automated analysis of vulnerability patterns, while circumventing some practical limitations of our previous approach in [18].

The following benefits are obtained by putting together vulnerability analysis and verification of access control policies:

1. Just one model is needed to carry out both analyses. In fact, even if the effort for building the model is seldom stressed by authors in the literature, it represents a major obstacle to make a theoretical framework applicable in practice. Moreover, as far as we know, the two kinds of analysis have always been dealt with separately till now, thus requiring different models and system descriptions, even if many system elements in the model (e.g., the network topology), are needed for both. This is why, for instance, Section 2 discussing the current state of the art, takes into account each kind of analysis separately.
2. What-if analyses are enabled, for instance, to evaluate either the effects of an employee identity theft or the behaviour of a disgruntled employee. Actually, this becomes quite natural in our unified framework, where the attacker is managed as whatever user interacting with the system. In other words, it is easy to study how employees can exploit vulnerabilities to enlarge their set of feasible actions, despite their initial allowed capabilities.
3. Physical access and system topology are of utmost importance in INS and cyber-physical systems and our model takes into account users' movements inside the system. This enables the description of attacks

which leverage physical accesses to the system resources, and results in the security analysis of more realistic scenarios.

The reader should be warned that the proposed methodology, as the one in [14, 15], is not applicable only to INS but can be also profitably employed for detecting chain of exploitable vulnerabilities in general-purpose networked systems too. By contrast, while many tools, which rely on different techniques, are already available for the latter, some characteristics of INS (e.g., heterogeneity of h/w and s/w components, proprietary/special purpose communication protocols, peculiar application s/w, 24/7 availability requirements) often prevent them from borrowing existing solutions (unfortunately based on unacceptable assumptions) and justify the development of ad-hoc formalism and techniques [4].

The paper is structured as follows: Section 2 discusses related works on both vulnerability analysis and access control policy management, while Section 3 briefly recalls the main characteristics of the analysis framework that are needed to understand the remaining part of the paper. Section 4 deals with the extensions and changes that enable the description of vulnerabilities, their inclusion in the formal system model and the computation of all their possible exploitable sequences. Section 5 presents a small realistic example built to show how the analysis can be carried out in practice, while some conclusions are drawn in Section 6. Appendix A describes those differences with respect to the formalism presented in [14], that were specifically introduced to make the model more comprehensive and to deal with the exploitation of vulnerabilities.

2. Related works

Vulnerability analysis and access control policy management are major research topics for the security of networked computer systems. As far as we know, they have always been addressed separately, and this paper is an attempt to put them together. The main reason of such an approach can be found in the development of the system model. Actually, in our experience, this step is often a critical aspect for any analysis and assessment framework, since an adequate system description can be a cumbersome and complex task, and the same is true for keeping the model up to date and aligned with the real system. A single unified model able to satisfy requirements of different kinds of analysis, besides making things easier, can save effort

and time. Scientific papers, published in the past, focus on either policy management or vulnerability exploitation, so we deal with them separately in the following. More detailed discussions of the relevant literature can be found in the "Related Works" sections of [12] and [15] respectively.

2.1. Vulnerability analysis

Vulnerability analysis, also known as vulnerability assessment, was introduced in [19, 20, 21]. All those works were aimed at computing an attack graph in which each vertex represents a system state, whereas each edge describes a possible state transition determined by an attacker action. The attack graph generation in [19, 20, 21] has exponential computational complexity, thus leading to intractability when dealing with reasonably sized real systems. By means of the *monotonicity* hypothesis (i.e., the attacker never discards any of the acquired capabilities), introduced in [22] and, for instance, followed by [23, 12], the computational complexity becomes polynomial. Some tools such as TVA [23], NetSPA [24], MulVAL [25] and Skybox [26] rely on this assumption. Monotonicity is a reasonable hypothesis, as it is conservative, i.e., it guarantees that the worst case is never missed.

More recent works introduce enhancements to the above techniques along different axes. [27] suggests an *anytime* approach based on a set of algorithms that provide results with different timing and precision, and support the system manager in a near real-time fashion, paying particular attention to the (semi)automatic feed of (parts of) the system model (e.g. vulnerabilities) [6, 7]. [28] focuses on a subsequent step (i.e., the network hardening) by taking into account possible interdependencies of concurrent hardening mechanisms and their costs. Known vulnerabilities do not have the same exploitation probability: [29] proposes a technique, relying on bayesian attack graphs, able to quantify the chances of any network compromise, thus allowing managers to prioritize actions in a resource constrained environment. [30] computes the probability of successful attacks on the system. The main achievement, in that case, is that no special security expertise is needed to build the model because the related attributes are computed by probabilistic relational model. To validate the technique, results produced by the analysis tool were compared to those computed by a security professional. Difficulties related to the modeling phase preparatory to the analysis are also dealt with in [31, 32], where the attack graph is created in an incremental way, with the aid of an automated tool, also considering probabilities.

2.2. Access control policy management

Access control, in the broader sense, is one of the most important topics in computer and network security, so it has been debeated in a number of scientifc papers. In particular, RBAC [33] is one of the most known formalism to describe access control policies in a formal, easy and efficient way. It is based on the concept of *roles*, to which both *users* and *permissions* are assigned through many-to-many relationships.

Access control policy management and compliance check are important subtopics and have been studied along three main directions which differ in the degree of connection to the real world. In practice, works as [34, 35] only deal with policies and are aimed at checking that they satisfy predefined requirements without any explicit relationship with real implementations. Approaches such as [36, 37], instead, take into account the need of enforcing policies in the real system, by either assuming or proposing that the system provides suitable mechanisms to ensure such an enforcement. These mechanisms usually require sophisticated h/w and s/w support that today's INS are not able to offer, because of either their limited computational and communication resources, or the strict performance needs they typically must satisfy.

Few works [26, 38, 39, 40, 41, 42] explicitly address the implementation of policies in networked systems without any enforcement assumption. [26] focuses its analysis on firewalls and other traffic control devices, and cannot be applied to other INS components. NP-View [38, 39, 40] is a tool able to check the network topology and configuration against global access policies. To do this the tool considers network nodes running SE-Linux but, also in this case, other kinds of devices are not taken into account. [41] is able to establish *who can do what on what*, as in [14, 15], but only for systems where all hosts run SE-Linux, a very uncommon situation in INS. Finally, [42] deals with the automatic extraction of access control information by running the real system, but the technique is applicable only to web applications.

3. Analysis framework

Our analysis framework is based on the twofold model presented in [14, 15] which allows to describe the system characteristics from two different points of view. Fig. 1 illustrates the main building blocks of the analysis presented in [14, 15]. In practice, the designer has to provide both a high-level description of the security policies to be checked (access policies in

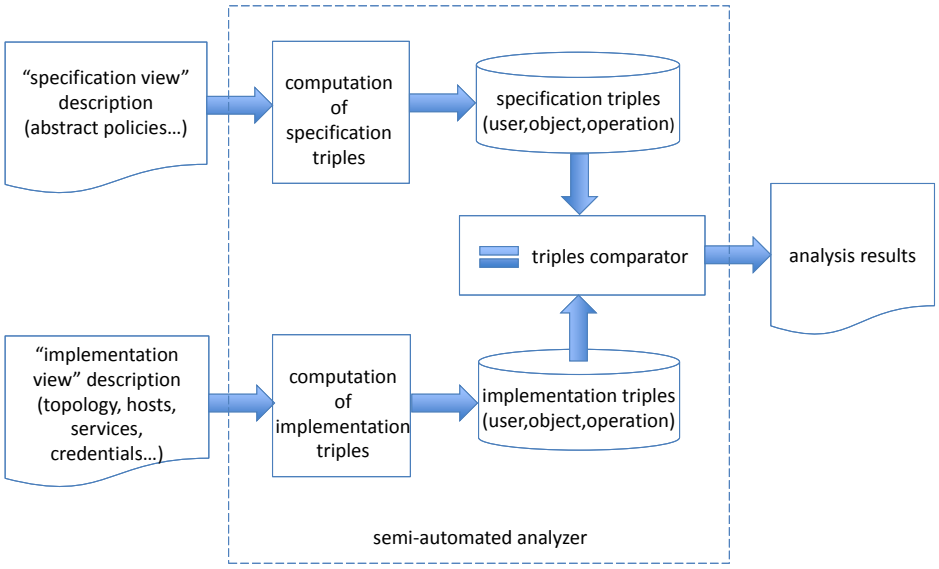

Figure 1: Building blocks of the analysis framework

our previous studies) and a fine-grained description of the system actual implementation, including its security mechanisms and settings. These two descriptions are then processed by an automated analyser to produce two sets of (*user, operation, object*) triples including all the actions that can be performed by all users on all objects in the system. Users, operations and objects are "bridging" elements between the two views and the computed triples are used by the automated analyser to make a comparison between the actual system configuration and its (expected) high abstraction level behaviour, in order to highlight differences and inconsistencies representing design and implementation flaws.

In the following we will briefly recall the basics of the language (model) used by the designer to describe the implementation view of the system, as this part of our approach will be exploited to describe known software vulnerabilities and to fed the automated analyser in order to highlight possible attack paths, exploiting chains of vulnerabilities, constituting a threat to system security. In particular, we will focus on the additional model components specifically introduced to describe vulnerabilities and, also, centralized authentication mechanisms often employed in modern ICT systems (e.g., Windows Active Directory [43], Apache Central Authentication Service (CAS) [44]). An exhaustive description of the analysis framework, and

in particular of the previous version of the twofold model as well as of the automated analyser, can be found in [14, 15].

The implementation view of the system consists of a data model $\mathcal{D}$ describing all objects of the system and their physical and/or logical interconnections, the initial state for each considered user (i.e., physical location, owned credentials and so on) and a set of inference rules that regulate interactions between users and system. All these components are used by the automated analyser to compute the set of all actions allowed for each user and to build the corresponding triples in the implementation set. It is worth noting that the data model $\mathcal{D}$ is system-specific and must be provided by the designer, while the set of inference rules is a predefined, fixed building block of our analysis framework. In other words, inference rules are "hard-wired" in the analysis tool and no designer's action is required for their definition. In our previous works humans interacting with the systems were generically called "users" or "players" and their state, employed for computing the sets of actions they could carry out given the set of their assigned credentials and the room they are initially in, was consequently referred to as user (or player) state. In this framework, however, as we are interested in detecting possible chains of vulnerabilities exploitable by a single malicious user, thus the agent interacting with the system is the "attacker" and his/her state is accordingly called "attacker state". Moreover, in the computation of triples (*user, operation, object*), we omit the user as it is implicitly assumed to be the attacker.

Note that to keep the computational complexity and the state explosion problem under control, we consider the system as *static*, i.e., we suppose interactions between users and the system do not affect the system description $\mathcal{D}$. This hypothesis is not restrictive as effects of system changes can be taken into account by off-line modifying the data model $\mathcal{D}$ accordingly, and by carrying out new analysis runs.

Formally, the data model $\mathcal{D}$ is a pair:

$$\mathcal{D} ::= (\Omega, \Lambda) \tag{1}$$

where $\Omega ::= \{\omega^\star\}$ is the set of system objects, i.e., entities on which operations can be performed, while $\Lambda ::= \{\lambda\}$ is the set of physical communication links. Objects include rooms and other physical containers (e.g., cabinets), host devices (e.g., PCs, PLCs, HMIs), software services (e.g., web and database servers, mail servers, s/w applications) and networking devices (e.g., firewalls,

Table 1: Object formal description

acc	::=	$(n, g)[:(\omega_{aa}, n_{aa})])[:adm]$			
pp	::=	$\langle id_{pp}, \{\langle dla, \{na\}, [w, [c], \{\omega\}]\rangle\}\rangle$			
fr	::=	$\langle id_{pp_s}, id_{pp_d}, dla_s, na_s, pn_s, dla_d, na_d, pn_d, pr, act\rangle$			
		id_{pp_s}	source physical port	id_{pp_d}	dest physical port
		dla_s	source data link addr	dla_d	dest data link addr
		na_s	source network addr	na_d	dest network addr
		pn_s	source port number	pn_d	dest port number
		pr	protocol (TCP,UDP,...)	act	action (*allow* or *deny*)
sw	::=	$\langle sw\_name, ver\rangle$			
λ	::=	$\{id_{pp}\}$			

switches, routers). Any object $\omega^\star \in \Omega$ is defined as

$$\omega^\star ::= \langle \omega_{id}, \{\pi^\star\}, \omega_{path}, \{acc\}, \{pp\}, \{fr\}, \{sw\}\rangle \tag{2}$$

where ω_{id} and ω_{path} are the object identifier and path-name, respectively. Indeed, in the data model objects can be nested (i.e., an object can contain and can be contained in other objects, so as to allow, for instance, the description of virtual hosts within hypervisors) and anyone of them is uniquely identified by both its identifier and its path $\omega_{path} = \omega_{id_1}, \omega_{id_2}, \ldots, \omega_{id_k}$, i.e., the ordered sequence of identifiers of the k objects that need to be "traversed" to access the considered one (note that ω_{id_1}, i.e. the most external container object, is always a room). Let $\omega = \omega_{path}, \omega_{id}$ be the unique identifier of any object $\omega^\star \in \Omega$, in the following to keep the notation as simple as possible we will identify any object $\omega^\star$ by using the shortest univocal suffix of ω. For example, if ω_{id} is uniquely defined in the data model it will be adopted as the $\omega^\star$ full name (i.e., $\omega = \omega_{id}$). Moreover, to address any element of a structured object we will use the "dot notation", so $\omega^\star.\{acc\}$ is the set $\{acc\}$ of object $\omega^\star$. The formal description of the other elements in Eq. (2) is provided in Tab. 1 (note that $[\cdots]$ stands for *optional* part and any set $\{\cdots\}$ is possibly empty). In summary:

- acc is an account where n and g are, respectively, a username and a group defined for $\omega^\star$. Part $[:(\omega_{aa}, n_{aa})]$, if present, means that the user, in order to login as user n belonging to group g must be authenticated by the remote authentication authority ω_{aa} with username n_{aa}. If part $[:(\omega_{aa}, n_{aa})]$ is omitted, the user is authenticated directly by $\omega^\star$ without

any need of third party intervention. Part $[:adm]$, if present, identifies an account with administrator privileges. These slight modifications of the model in [14, 15] have been specifically introduced to describe both vulnerability exploitation effects (i.e., gaining "root" privileges on some node) and centralized authentication mechanisms. Inference rules presented in [14], which involve *acc* elements, have been modified according to the new *acc* formal description and are detailed in Appendix A.

- *pp* is a physical port (i.e., network interface). id_{pp} is the port unique identifier, while *dla* is a data-link address (e.g., a MAC address: some network interfaces, such as in firewalls, are not assigned a data-link address), and $\{na\}$ is a set of network addresses bound to *dla*. $[w, [c], \{\omega\}]$ is meaningful only for wireless interfaces: w means that wireless communication is supported, $[c]$ is the possible credential required to connect to the interface (e.g., access point), and $\{\omega\}$ is the set of room objects where the interface is accessible.
  Many sub-tuples $\langle dla, \{na\}, [w, [c], \{\omega\}]\rangle$ are allowed for any network interface.

- fr is a filtering rule. The general form of fr enables the specification of typical rules for any kind of networking device. As not all parameters are always needed, the symbol "_" denotes unused values.

- *sw* describes an installed software package by means of its name and version.

- λ is a set of strongly connected physical port identifiers (e.g., a point-to-point link is described by means of two ports, whereas a bus is described by several ports).

Any object $\omega^\star$ includes a set of operations ($\{\pi^\star\}$ in Eq. (2)) representing all possible actions a user may, in principle, carry out on the object itself. Any operation has one of two possible forms:

$$\pi^\star \quad ::= \quad \langle \pi, \{\langle d, \{c\}\rangle\}\rangle \tag{3}$$

$$\pi^\star \quad ::= \quad \langle \pi, \{f\}\rangle \tag{4}$$

Form (3) is used when the relevant object is either a room or a physical container: π is the operation name, e.g. *enter*, whereas the set $\{\langle d, \{c\}\rangle\}$

describes all doors which allow entering the room. Set $\{c\}$ is the set of all credentials necessary to open door d. In analogy to form (3), form (4) describes operations a user can carry out on devices and their software resources. π is the operation name (e.g., *upload_part_program*, *start_part_program*, *admin*), while f specifies both the preconditions and effects of π on either the involved or other (software) resources.

Table 2: Preconditions and postconditions f

f	::=	$pre,\ post$
pre	::=	$\langle phy\_acc\ [c] \rangle$
		$\mid \langle loc\_acc\ \omega' {:}\ n' [c] \rangle$
		$\mid \langle loc\_acc\ \omega' {:} g' [c] \rangle$
		$\mid \langle rem\_acc\ port\ [c] \rangle$
		$\mid \langle rem\_auth\ lp\ [c] \rangle$
		$\mid \langle phy\_acc\ \wedge\ auth\ \omega_{aa} {:} n_{aa} \rangle$
		$\mid \langle rem\_acc\ lp\ \wedge\ auth\ \omega_{aa} {:} n_{aa} \rangle$
$port$	::=	$dla \mid\ lp$
lp	::=	$\langle id_{lp}, [pn], na, [pr] \rangle$
$post$	::=	$[\omega'' {:} n'']$

Tab. 2 shows the f syntax, where seven different and mutually exclusive kinds of preconditions can be specified. Their semantics is, informally, the following:

- *phy_acc* $[c]$ means that, in order to perform operation π on object ω, a user must be in the same room as ω and own credential c (if specified).

- *loc_acc* $\omega' {:} n'$ $[c]$ (or *loc_acc* $\omega' {:} g'$ $[c]$) means that, in order to perform π on ω, a user must already be active (e.g., logged) on some object ω', by means of username n' (or any username n' belonging to group g'). In addition, when specified, credential c must also be owned by the user, unless n' has administrator privileges.

- *rem_acc port* $[c]$ means that operation π can be carried out by a user through a remote connection at either the network or data link level. The user must own credential c when specified. It is worth noting that TCP/IP connections are always established between logical ports, but industrial systems often include special-purpose devices and software that adopt communication at the data-link level, for this reason *port*

can be either a data link address or a logical port lp. Formally, lp is uniquely identified by id_{lp}, while pn is an optional port number (e.g., 8080), na is the network address (e.g., IP address), while pr is an optional protocol (e.g. Modbus).

- $rem\_auth\ lp\ [c]$ is the precondition associated to any remote authentication operation provided by a central authentication authority ω_{aa} listening on logical port lp. Operations characterized by this kind of precondition are not recorded as a triple in the implementation view (i.e., they cannot be performed by the user) and are used to describe remote authentication. To be remotely authenticated as n_{aa} by ω_{aa} a user should own credential c when specified.

- $phy\_acc \wedge auth\ \omega_{aa}{:}n_{aa}$ means that, in order to perform operation π on object ω, a user must be in the same room as ω and, at the same time, he/she should be able to be authenticated as user n_{aa} by the remote authentication authority ω_{aa}.

- $rem\_acc\ lp\ \wedge\ auth\ \omega_{aa}{:}n_{aa}$ resembles the previous precondition, but, in this case, π is remotely accessible through the logical port lp.

The element *post* in Tab. 2 describes the possible effects of the operation execution: in particular $\omega''{:}\,n''$, when specified, means that, by performing the operation, the user gains (local) access with username n'' to object ω'' (e.g., the effect of the UNIX *su* operation on the login status).

For the purpose of detecting sequences of vulnerabilities, we assume that $\mathcal{D}$ describes the static attacker's environment. The second element of the implementation model is the attacker state $\mathcal{T}_a$, whose form is shown in Tab. 3 and consists of the set LA of all local accesses (i.e. active logged-in conditions) already obtained by the attacker, the current room he/she is in and the set of credentials he/she owns. Efects of any action performed by the attacker (i.e., access gained for a given room, acquisition of a logged-in status and so on) are recorded in $\mathcal{T}_a$ and, consequently the pair $(\mathcal{D}, \mathcal{T}_a)$ represents the comprehensive *system state*.

The computation of all actions the attacker can carry out in our model depends on the reachability of objects in the network. Since our description of the network and its devices is static and compatible with [45], reachability can be (and actually is) precomputed according to the technique described in [45] and stored in a suitable database. The following general query returns

true or false as a result, depending on whether or not a logical path exists between a source and a destination node and is actually enabled by the configuration of the network infrastructure:

$$\exists\_path(dla_s, pn_s, na_s, dla_d, pn_d, na_d, pr, \langle \omega, C \rangle) \tag{5}$$

The meaning of dla_s, pn_s, na_s, dla_d, pn_d, na_d is that in Tab. 1, whereas ω and C refer to the use of possible wireless connections. In practice, given a *source host* (either accessible to the attacker or where he/she is already logged in) whose networking details are the data link address dla_s, network address na_s and port number pn_s, a *destination host* with networking details dla_d, na_d, and pn_d respectively and a *protocol* pr, (5) checks whether an allowed network path exists between source and destination, matching the specified networking details. The rightmost pair $\langle \omega, C \rangle$ specifies the location of the source host (room id ω), and the set C of attacker's credentials. This also enables the computation of network paths exploiting wireless connections reachable from the attacker's room ω provided he/she owns the suitable credentials stored in C. Actually also wireless access points are described by means of attributes listed in Tab. 1, and the description of a wireless physical port *pp* includes the set of rooms where the interface is accessible.

Given a system model, described by means of the formalism introduced above, and the attacker initial state (i.e. the initial composition of $\mathcal{T}_a$) our goal is computing all sequences of actions the attacker can perform in the system in order to highlight possible unwanted or harmful behaviours. To do this, one additional element is still needed, that is a suitable set of rules $\mathcal{R}$ defining and controlling interactions between the considered user and the system. Set $\mathcal{R}$ consists of the inference rules formally described in [14] (some of them have been modified according to the new form of element *acc* and are reported in Appendix A). Each rule has an associated set of preconditions, that is, logical predicates acting on both the data model $\mathcal{D}$ and the considered user state $\mathcal{T}_a$. Whenever predicates are satisfied by both the data model and

Table 3: User state $\mathcal{T}_a$

$\mathcal{T}_a$	::=	(LA, ω_r, C)	
		LA	::= $\{(\omega, n)\,[:(\omega_a, n_a)][:adm]\}$ user local accesses
		ω_r	user current room
		C	::= $\{c\}$ user credentials

the user state, the corresponding pair (*operation, object*) (i.e., the pair that makes the predicate true) and the user identifier are stored as a triple (*user, operation, object*) in the implementation set and postconditions of the rule are applied, i.e., the user state is updated accordingly. This procedure is then repeatedly executed until the whole user state space (i.e., the set of all states reachable by the user) is built. It is worth reminding that all inference rules are integrated in the automated analyser which, in turn, applies them to a designer-defined system description $\mathcal{D}$ any time an analysis run has to be carried out. As a consequence, the user of the tool is not requested to be aware of the formalism adopted for the rule specification.

The inference rule set $\mathcal{R}$ in [14] was extended to take into account remote authentication mechanisms and vulnerability exploitations. Roughly speaking, besides rules dealing with any possible form of precondition f in Tab. 2, two additional rules were designed that allow the user to move between adjacent rooms and exploit possible vulnerabilities in the system. As an example, let us consider the rules needed to describe remote authentication mechanisms (i.e, operation with $phy\_acc \ \wedge \ auth \ \omega_{aa} : n_{aa}$ or $rem\_acc \ lp \ \wedge \ auth \ \omega_{aa} : n_{aa}$ as a precondition). The symbol $A \vdash B$ means that an object A (belonging to the pertaining set in the system) exists such that B is a subset of the elements of A (e.g., $\omega^{\star} \vdash (dla, na)$ means that there exists object $\omega^{\star} \in \Omega$ with a physical port bound to the associated data link address dla and network address na). Rule (6) states that, whenever the user is in the same room of an object ω ($\mathcal{T}_a.\omega_r = \omega^{\star}.\omega_{id_1}$) having as associated operation π characterized by precondition $f = phy\_acc \ \wedge \ auth \ \omega_{aa} : n_{aa}$ and optional postcondition $\omega'' : n''$, then if there is a remote authentication server ω_{aa} with an associated operation π_{aa} having precondition $rem\_auth \ lp_{aa}$ $[c]$ and authenticating the user as $\omega_{aa} : n_{aa}$ and the user owns the necessary c credential ($c \in \mathcal{T}_a.C$) and there exists a suitable connection between ω_{aa} and ω through the system network, then the user can perform operation π on object ω. When all preconditions are satisfied the corresponding triple (*user, operation, object*) is stored in the set of allowed actions. Moreover, after performing π on ω the user gains local access $(\omega'', n'') : (\omega_{aa}, n_{aa})$, that is, he/she becomes active as n'' on ω'' through remote authentication as n_{aa} by ω_{aa}. As in real situations the remote authentication does not provide the user with any local access on the remote authentication server.

All logical predicates in the formal definition of rule (6) are implicitly combined through "and" operators and, in this case, symbol "_" is used to replace parameters whose value is not relevant for the described logic

predicate.

$$\frac{\begin{array}{ccc} \omega^\star \vdash (\omega, \pi, f) & \omega^\star \vdash (dla, na) & f = phy\_acc \ \wedge \ auth\ \omega_{aa}{:}\, n_{aa}, [\omega''{:}\, n''] \\ \omega^\star_{aa} \vdash (\omega_{aa}, \pi_{aa}, f_{aa}) \quad \omega^\star_{aa} \vdash (dla_{aa}, lp_{aa}.na) & & f_{aa} = rem\_auth\ lp_{aa}\ [c],\ \omega_{aa}{:}\, n_{aa} \\ \mathcal{T}_a.\omega_r = \omega^\star.\omega_{id_1} & & c \in \mathcal{T}_a.C \\ \exists\_path(dla, \_, na, dla_{aa}, lp_{aa}.pn, lp_{aa}.na, lp_{aa}.pr, \langle \omega^\star.\omega_{id_1}, \mathcal{T}_a.C \rangle) & & \end{array}}{(\mathcal{D}, \mathcal{T}_a) \xmapsto{(\pi,\omega)} (\mathcal{D}, (\mathcal{T}_a.LA \cup \{(\omega'', n''){:}\, (\omega_{aa}, n_{aa})\}, \mathcal{T}_a.\omega_r, \mathcal{T}_a.C))} \tag{6}$$

For instance, rule (6) allows to model the practical situation where a user has to login on a PC whose access is controlled by a centralized authentication mechanism (e.g., Windows Active Directory). Clearly, in this case the user shall own adequate credentials and a suitable connection must exist between the PC (physically accessed by the user) and the central server providing authentication.

Similarly, rule (7) regulates the execution of operations requiring remote authentication, when they are carried out by the user through a remote connection. Object ω_d should support the execution of operation π_d, requiring the remote authentication of users as n_{aa} by authority ω_{aa}, through a network connection to its logical port lp_d. The user must be already logged in some source host ω_s ($(\omega', n')[{:}\, (\_, \_)] \in \mathcal{T}_a.LA$, with the (ω', n') account defined on ω_s). ω_s must have a physical port bound to a data-link address dla_s, which, in turn, has to be bound to a network address na_s and a network path should exist between (dla_s, na_s) and lp_d. Moreover, for the user to be able to be remotely authenticated as in rule (6), a network path should also exist between (dla_d, na_d) and lp_{aa}, that is between the object on which the operation is defined and the remote authentication authority ω_{aa}. For instance, rule (7) is useful to describe the user access to remote shared resources (e.g., shared documents) managed by Windows Active Directory. In this situation, the user must be able to authenticate with the central server and be recognised as one of the enabled users logged on a network node connected to the PC hosting the resource. This requires two accessible network paths: the first connecting the attacker host to the remote shared resource, and the second

linking the shared service to the authentication authority.

$$\frac{\begin{array}{c}\omega_s^\star \vdash (\omega_s, \pi_s, f_s) \quad \omega_d^\star \vdash (\omega_d, \pi_d, f_d) \qquad \omega_{aa}^\star \vdash (\omega_{aa}, \pi_{aa}, f_{aa}) \\ \omega_s^\star \vdash (dla_s, na_s) \quad \omega_d^\star \vdash (dla_d, lp_d.na) \qquad \omega_{aa}^\star \vdash (dla_{aa}, lp_{aa}.na) \\ f_d = rem\_acc\ lp_d \ \wedge \ auth\ \omega_{aa}{:}n_{aa}, [\omega''{:}n''] \\ f_{aa} = rem\_auth\ lp_{aa}\ [c],\ \omega_{aa}{:}n_{aa} \\ (\omega', n')[:(\_,\_)] \in \mathcal{T}_a.LA \mid \omega_s^\star \vdash (\omega', \_, \_) \qquad c \in \mathcal{T}_a.C \\ \exists\_path(dla_s, \_, na_s, dla_d, lp_d.pn, lp_d.na, lp_d.pr, \langle \omega_s^\star.\omega_{id_1}, \mathcal{T}_a.C\rangle) \\ \exists\_path(dla_d, lp_d.na, na_d, dla_{aa}, lp_{aa}.pn, lp_{aa}.na, lp_{aa}.pr, \langle \omega_d^\star.\omega_{id_1}, \mathcal{T}_a.C\rangle)\end{array}}{(\mathcal{D}, \mathcal{T}_a) \xmapsto{(\pi_d, \omega_d)} (\mathcal{D}, (\mathcal{T}_a.LA \cup \{(\omega'', n''){:}(\omega_{aa}, n_{aa})\}, \mathcal{T}_a.\omega_r, \mathcal{T}_a.C))} \quad (7)$$

The data model $\mathcal{D}$ and the initial state of the attacker are fed to the automated analyser, which uses the inference rules in $\mathcal{R}$ for computing all possible sequences of actions on objects the user can perform in the system. Basically, starting with the initial system state $(\mathcal{D}, \mathcal{T}_a)$, the analyser applies all inference rules in $\mathcal{R}$ and builds a state transition (edge) for any pair $(\pi,\ \omega)$ the user can perform. The edge, labeled with $(\pi,\ \omega)$ connects the current system state to a possibly new state characterised by the operation execution. The procedure is iteratively repeated for any new state and the result of the process is the construction of a state machine recording, on its edges, all possible sequences (chains) of actions the user can perform. Sect. 4 describes how vulnerabilities can be added to the implementation model and how their exploitation can be recorded and stored in the machine state in the same way as all other actions.

4. Vulnerability model

Several databases [6, 7, 8, 9], which are publicly accessible online, store the continuously updated description of known vulnerabilities. Unfortunately, a significant part of this information is still not recorded in machine-readable formats, i.e. it is not ready to feed an automated software tool, as it is provided in natural language. The need for formal models effectively characterizing vulnerabilities and suitable to the purpose of automated processing has been stressed several times in the past and a proposal was described [13] merging outcomes from the Movtraq [46] and OVAL [47] international projects.

By applying some ideas borrowed from [13] to the formalism presented in [14, 15], we can model any vulnerability v as a set of preconditions, that need to be all satisfied (preconditions are combined by means of logical "and" operators) by the current system state $(\mathcal{D}, \mathcal{T}_a)$ for the vulnerability to be exploitable by the attacker, and a postcondition, describing the effects of the vulnerability exploitation on the system state. Formally any vulnerability is defined as

$$v ::= \langle v_{id}, \{pre_v\}, post_v \rangle \tag{8}$$

where v_{id} is the vulnerability unique identifier, while $\{pre_v\}$ and $post_v$ are, respectively, the set of logical predicates representing associated preconditions and postcondition. Basically, when all predicates in pre_v hold for some object $\omega^\star \in \Omega$ and for the current system state $(\mathcal{D}, \mathcal{T}_a)$, then the attacker is able to exploit vulnerability v and postcondition $post_v$ affects the system state by changing it to $(\mathcal{D}, \mathcal{T}'_a)$. This means that the exploitation of v may lead to a new system state where other preconditions may be enabled, thus allowing to analyse sequences of vulnerabilities. It is worth reminding that such an analysis enables the study of effects caused by the exploitation of chains of *known* vulnerabilities, while it is not able to discover new (unknown) flaws, i.e. "zero day attacks".

The study of a number of real world examples led to conclude that any precondition pre_v may assume one of the following forms, that are enough to flexibly describe vulnerabilities of interest:

- $remotely\_reachable(\omega^\star, (\mathcal{D}, \mathcal{T}_a))$: precondition is satisfied if the system state $(\mathcal{D}, \mathcal{T}_a)$ allows the attacker to reach object $\omega^\star$ through a network connection starting from some ω_s where the attacker is already logged on;

- $has\_program(sw\_name, \{ver\}, \omega^\star, (\mathcal{D}, \mathcal{T}_a))$: precondition is true if object $\omega^\star$ runs any version ver in $\{ver\}$ of software sw_name;

- $locally\_exploitable(\omega^\star, (\mathcal{D}, \mathcal{T}_a))$: precondition is true if the attacker is logged on object $\omega^\star$;

- $compromised\_interaction(\omega^\star, (\mathcal{D}, \mathcal{T}_a))$: precondition is satisfied if object $\omega^\star$ is able to interact with some remote host that has been already compromised by the attacker. Note that this precondition describes the dual situation with respect to $remotely\_reachable(\omega^\star, (\mathcal{D}, \mathcal{T}_a))$. Indeed, $compromised\_interaction(\omega^\star, (\mathcal{D}, \mathcal{T}_a))$ is used to describe those

vulnerabilities where the potential victim has to access a remote service whose behaviour has been maliciously modified by an attacker. The most common example of this scenario is the case of "cross-site scripting" vulnerabilities frequently affecting web services. Another example is when a potential victim downloads an infected document from a compromised server or simply browses a legitimate website including malicious advertisement banners.

Formally, preconditions above are described as follows

$$
\begin{aligned}
&remotely\_reachable(\omega^\star, (\mathcal{D}, \mathcal{T}_a)) ::= \\
&\quad \exists \omega_s^\star \in \mathcal{D}.\Omega \mid \begin{array}{l}(\omega_s, n)[: (\omega_{aa}, n_{aa})] \in \omega_s^\star.\{acc\} \wedge \\ (\omega_s, n)[: (\omega_{aa}, n_{aa})] \in \mathcal{T}_a.LA\end{array} \wedge \\
&\quad \omega_s^\star \vdash (dla_s, na_s) \wedge \omega^\star \vdash (dla, na) \wedge \\
&\quad \exists\_path(dla_s, \_, na_s, dla, \_, na, \_, \langle \mathcal{T}_a.\omega_r, \mathcal{T}_a.C \rangle)
\end{aligned}
$$

$$
\begin{aligned}
&has\_program(sw\_name, \{ver\}, \omega^\star, (\mathcal{D}, \mathcal{T}_a)) ::= \\
&\quad \exists ver \in \{ver\} | (sw\_name, ver) \in \omega^\star.\{sw\}
\end{aligned}
$$

$$
\begin{aligned}
&locally\_exploitable(\omega^\star, (\mathcal{D}, \mathcal{T}_a)) ::= \\
&\quad (\omega, n)[: (\omega_{aa}, n_{aa})] \in \omega^\star.\{acc\} \wedge (\omega, n)[: (\omega_{aa}, n_{aa})] \in \mathcal{T}_a.LA
\end{aligned}
$$

$$
\begin{aligned}
&compromised\_interaction(\omega^\star, (\mathcal{D}, \mathcal{T}_a)) ::= \\
&\quad \exists \omega_t^\star \in \mathcal{D}.\Omega \mid \begin{array}{l}(\omega_t, n)[: (\omega_{aa}, n_{aa})] \in \omega_t^\star.\{acc\} \wedge \\ (\omega_t, n)[: (\omega_{aa}, n_{aa})] \in \mathcal{T}_a.LA\end{array} \wedge \\
&\quad \omega_t^\star \vdash (dla_t, na_t) \wedge \omega^\star \vdash (dla, na) \wedge \\
&\quad \exists\_path(dla, \_, na, dla_t, \_, na_t, \_, \langle \mathcal{T}_a.\omega_r, \mathcal{T}_a.C \rangle)
\end{aligned}
$$

Possible postconditions $post_v$ of interest for detecting chains of vulnerabilities are

- $gain\_privilege(\omega^\star, (\mathcal{D}, \mathcal{T}_a))$, which means that the attacker gains administrator (unlimited) privileges on object $\omega^\star$.
- $Denial\_of\_Service(\omega^\star)$ which indicates the disruption of the affected service or node $\omega^\star$. As we deal with a static scenario (neither services

nor nodes are added to/removed from the system during the same analysis run), whenever either a service or a node is affected by denial of service we assume that the system state $(\mathcal{D}, \mathcal{T}_a)$ becomes, symbolically, $(DoS, \mathcal{T}_a)$. In other words, the space of all possible "values" the data model $\mathcal{D}$ can acquire is augmented with an additional value (DoS) indicating a denial of service occurred at some system service/node. Moreover, whenever the system state becomes $(DoS, \mathcal{T}_a)$ (because of any sequence of actions), the analysis stops (i.e., any $(DoS, \mathcal{T}_a)$ state is a dead state).

Formally, vulnerability postconditions are described as follows

$$gain\_privilege(\omega^\star, (\mathcal{D}, \mathcal{T}_a)) ::= (\mathcal{D}, (\mathcal{T}_a.LA \cup (\omega, \_)[:(\_, \_)]:adm, \mathcal{T}_a.\omega_r, \mathcal{T}_a.C))$$
$$denial\_of\_service(\omega^\star, (\mathcal{D}, \mathcal{T}_a)) ::= (DoS, \mathcal{T}_a)$$

Once defined the implementation view of the system (i.e., the data model and the attacker initial state) and the set of known vulnerabilities $\mathcal{V} = \{v\}$, possible sequences of exploitable vulnerabilities can be computed by applying an extended set $\mathcal{R}$, which includes a new rule to explicitly manage vulnerability pre and postconditions. This additional inference rule basically states that, given any current system state $(\mathcal{D}, \mathcal{T}_a)$, if the system state is such that preconditions of any vulnerability v are satisfied for some object $\omega^\star$ in the system, than v is exploited, that is action $exploit(v_{id})$ on object ω is added to the set of operations allowed for the attacker and the system state is modified according to $v.\{post_v\}$. Formally, the vulnerability exploitation rule can be specified as

$$\frac{\begin{array}{cc}\omega^\star \vdash (\omega, \_, \_) & v \vdash (v_{id}, \{pre_v\}, post_v) \\ \forall pre_v \in \{pre_v\}, & (\omega^\star, (\mathcal{D}, \mathcal{T}_a)) \models pre_v\end{array}}{(\mathcal{D}, \mathcal{T}_a) \xmapsto{(exploit(v_{id}), \omega)} post_v(\omega^\star, (\mathcal{D}, \mathcal{T}_a))} \quad (9)$$

where $(\omega^\star, (\mathcal{D}, \mathcal{T}_a)) \models pre_v$ means that pre_v is satisfied by object $\omega^\star$ and the current system status $(\mathcal{D}, \mathcal{T}_a)$.

The basic elements discussed above enable the description of real vulnerabilities such as those involved in the example of Sect. 5 (in the following, vulnerabilities are identified by means of their CVE [6] id):

- **CVE-2014-5424**: vulnerability affecting the `Connected Components Workbench` (CCW) program, version 6.01.00. This software suite is

used for programming and configuring Rockwell Automation devices. A flawed ActiveX component allows a remote attacker to crash the application or possibly execute arbitrary code, thus letting he/she to gain some kind of privilege on the affected node.

- **CVE-2014-8551**: vulnerability affecting the `WinCC` software server, used in Siemens SCADA products. The same vulnerability also affects the `TIA Portal` application used on engineering workstations for automation products in Siemens environments. A flawed component in the application allows an attacker to remotely execute arbitrary code by sending specially crafted packets to the application. A successful attack allows the attacker to gain privileges on the node. The software versions affected are `WinCC` 7.0,7.2,7.3 and `TIA Portal` 13.0. This vulnerability, affecting two different software products, is modelled as separate vulnerabilities CVE-2014-8551a and CVE-2014-8551b.

- **CVE-2011-4034**: vulnerability affecting the `Vijeo Historian` product, by Schneider Electric, version 4.20. A flawed ActiveX component in this application allows a remote attacker to cause a denial of service in the affected node, provided that the victim interacts with an object compromised by the attacker.

- **CVE-2015-2177**: vulnerability affecting the `SIMATIC S7-300` CPU included in Siemens PLCs. This vulnerability allows a remote attacker to remotely set the affected device into defect mode, causing a complete denial of service on that node.

- **CVE-2015-6490**: vulnerability affecting Rockwell Automation `Micrologix 1400` PLC. This vulnerability allows a remote attacker to cause a buffer overflow in the system and to execute arbitrary code if the firmware version is 15.002.

- **CVE-2014-0754**: vulnerability affecting the `SchneiderWEB` application included in some of the Modicon PLC Ethernet modules by Schneider Electric. This vulnerability allows a remote attacker to bypass the basic authentication system on the web server, which would allow unauthenticated administrative access (i.e., control) over the device.

- **CVE-2014-6332**: vulnerability affecting several `Microsoft Windows` operating system versions. This vulnerability allows a remote attacker

Table 4: Vulnerabilities preconditions and postconditions

	precondition			*postcondition*	
CVE-ID	Access	Affected software	Versions	execute any code	denial of service
2014-5424	remote	CCW	6.01.00	yes	
2014-8551a	remote	TIA Portal	13.0	yes	
2014-8551b	remote	WinCC	7.0,7.2,7.3	yes	
2011-4034	victim's	Vijeo Historian	4.20		yes
2015-2177	remote	SIMATIC S7-300 CPU	any		yes
2015-6490	remote	Micrologix 1400	15.002	yes	
2014-0754	remote	SchneiderWEB	any	yes	
2014-6332	victim's	MS Windows	7, 8, ...	yes	

to execute arbitrary code on the victim node via a crafted web site. The affected versions of the Windows operating system are 7, 8, 8.1, Server 2012 and others. To trigger this vulnerability, the user should access a web site compromised by the attacker.

Main information about these vulnerabilities is summarized in Table 4, where values of the "Access" column, that is "remote", "physical", "victim's" and "local", respectively mean that the vulnerability can be exploited remotely through a network connection, physically on the node itself, that the victim must interact with a compromised object or that the attacker must have already gained some local access on the victim node. Informal (textual) vulnerability descriptions can then be translated in the formal combination of predicates building up preconditions and postconditions. For instance, vulnerability CVE-2015-2177 is described formally as:

$$\langle \left\{ \begin{array}{l} remotely\_reachable(\omega^\star, (\mathcal{D}, \mathcal{T}_a)), \\ has\_program(S7-300CPU, \{\_\}, \omega^\star, (\mathcal{D}, \mathcal{T}_a)) \end{array} \right\}, Denial\_of\_Service(\omega^\star) \rangle$$

while CVE-2014-6332 is modelled as (only two affected versions are included):

$$\langle \left\{ \begin{array}{l} compromised\_interaction(\omega^\star, (\mathcal{D}, \mathcal{T}_a)), \\ has\_program(MSWindows, \{7, 8\}, \omega^\star, (\mathcal{D}, \mathcal{T}_a)) \end{array} \right\}, gain\_privilege(\omega^\star, (\mathcal{D}, \mathcal{T}_a)) \rangle$$

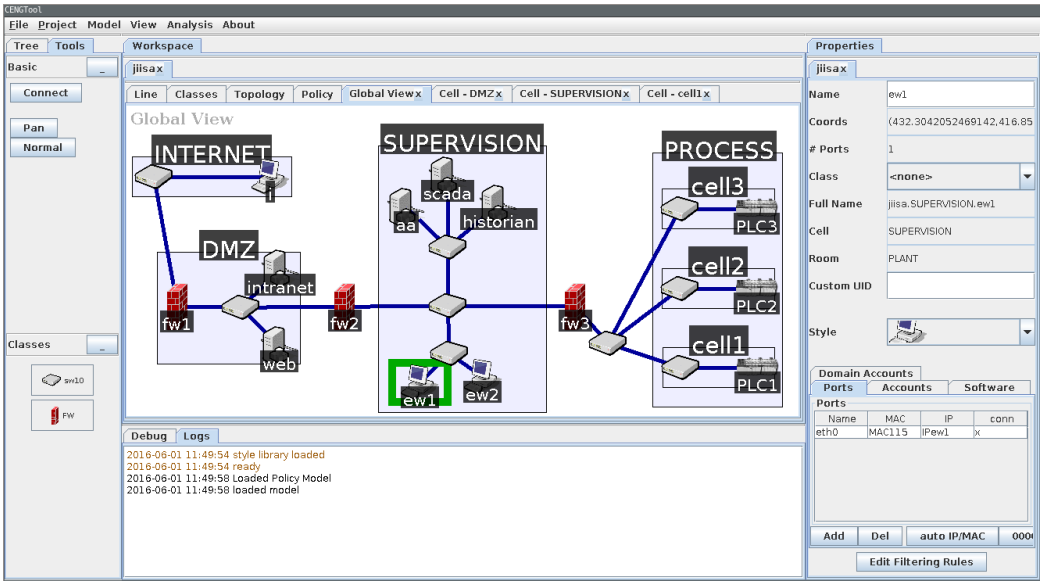

Figure 2: Example industrial networked system

5. Example

5.1. Prototype analyser implementation

To get more information about the feasibility of the proposed approach and its achievable performance, a prototype automated software tool was developed, which heavily relies on changes and extensions to the analyser presented in [15]. Also in this case, the model processing engine was written in Prolog language, because its logic rules and inferences fit in well with the design of our framework and enable rapid prototyping and fast preliminary verifications. Actually, the data model $\mathcal{D}$ and vulnerability descriptions are translated into a set of static Prolog *facts*. Moreover, the set of inference rules defined in [14], and extended with those needed for remote authentication and vulnerability analysis, is encoded once and for all as Prolog rules. These elements are then used by the engine to build the complete attack graph for the considered malicious user(s) interacting with the system. It is worth noting that, because of the adoption of the enlarged set of inference rules, the resulting graph includes both (unwanted) vulnerability exploitations and legitimate actions allowed by the system configuration.

Vulnerability descriptions were necessarily encoded by hand, due to the already mentioned lack of machine-readable sources, by gathering and interpreting information from well-known vulnerability databases [7].

Finally, the designer was aided with the definition of the system model $\mathcal{D}$ thanks to the introduction of a simple graphical interface (GUI), written in Java. The GUI is able to hide several details of the tool modelling language and helps to specify the system in a quite intuitive way, also performing several static syntax checks in a transparent way. A sample screenshot of the graphical interface is shown in Fig. 2, for the example system presented in the next section.

5.2. Case study

Fig. 2 shows a simple industrial networked system we use as an example to show how our technique works. The structure of the system has been derived from our experience with real world applications but its size has been limited for clarity. Moreover, the system has been populated, with respect to typical industrial configurations of same complexity, with more heterogeneous (i.e., from many different vendors) devices, so as to introduce many different known vulnerabilities (that in real-world situations usually coexist in much larger plants) and make it a meaningful case study.

The system in Fig. 2 consists of four logical zones namely (from left to right) external, demilitarized (DMZ), supervision and process areas.

The external world is represented (in a simplified way) by the Internet cloud. We assume that a remote node `i` exists there, where an adversary can log in and start his/her attack against some company critical device.

The Internet connection to DMZ is logically protected by firewalls $\texttt{fw}_1$, while firewall $\texttt{fw}_2$ separates it from the supervision area. DMZ is usually the first line of defense to prevent direct access from the outside world to the core areas of the system. In our example, we suppose that DMZ includes two servers respectively hosting a web site and enabling accesses to the company intranet (both servers are directly accessible from the outside world as they have public IP addresses). Firewall $\texttt{fw}_1$ is configured to filter out (incoming) connections from the Internet to services not provided by machines in DMZ, while outgoing connections are always allowed. In the following we use the terms “incoming” and “outgoing” to distinguish connections originating from devices that are located respectively on the left and right side of a firewall in Fig. 2.

The supervision area, connected to the DMZ through firewall $\texttt{fw}_2$, includes servers and workstations for managing data exchanged with devices in the process area (i.e., field devices). In particular, engineering workstations $\texttt{ew}_1$ and $\texttt{ew}_2$ are equipped with those software suites needed for field communications and concern devices from two specific different vendors. In detail, $\texttt{ew}_2$ runs the "Connected Components Workbench" by Rockwell Automation, while $\texttt{ew}_1$ is equipped with the "TIA Portal" software, used for managing Siemens devices.

The `scada` server in the supervision area is responsible for the execution of control tasks in the process area, while the `historian` collects statistical data from the field and makes them available to the higher levels in the plant hierarchical infrastructure. Server `aa` is the domain central authentication authority for users accessing the two workstations. The `scada` is assumed to run the Siemens SIMATIC WinCC and the `historian` is equipped with the Schneider Electric Vijeo Historian software. Firewall $\texttt{fw}_2$ is configured to block any incoming connections from both the Internet and DMZ to devices in the supervision area, while it enables any outgoing connection.

The additional firewall, $\texttt{fw}_3$ in Fig. 2, separates the supervision area from the process area. In real-world industrial networked systems, the process area typically consists of several automation cells, each one devoted to supervising a specific phase of the controlled process. A cell includes controlling devices (e.g., remote terminal units, programmable logic controllers-PLC) and field equipment (e.g., I/O, sensors and actuators). Our example system in Fig. 2 is based on three cells ($\texttt{cell}_1$, $\texttt{cell}_2$, $\texttt{cell}_3$), and each cell is built around a PLC from a different vendor (specifically, Siemens, Rockwell Automation and Schneider-Electric).

Firewall $\texttt{fw}_3$ is the last line of defense for the most inner part of the system and is configured to block every incoming communication except for those that are strictly required for correct interaction between the supervision and process areas. Specifically, workstations $\texttt{ew}_1$ and $\texttt{ew}_2$ are allowed to exchange data with cells $\texttt{cell}_1$ and $\texttt{cell}_2$ respectively, while `scada` and `historian` can access all system cells. Outgoing connections are always enabled.

5.3. Data model

Fig. 3 shows the formal description of four of the nodes in the example network in Fig. 2. Note that node `i` is supposed to be located in an "external" room (i.e., $\omega_{path} = \langle E \rangle$), while all other nodes have been placed in the same physical space (i.e., $\omega_{path} = \langle W \rangle$). Moreover, in the formal description

$$\langle i, \{\langle login, \{\langle phy\_acc\ i{:}root\rangle\}\rangle\}, \langle E\rangle, \{(root, adm){:}adm\}, \{\langle pp_i, \{\langle mac_i, \{ip_i\}\rangle\}\rangle\}, \varnothing, \varnothing\rangle$$

$$\langle ew_1, \left\{ \begin{array}{l} \langle login, \{\langle phy\_acc \wedge auth\ aa{:}n_1\rangle, ew_1{:}n\}\rangle, \\ \\ \langle designer, \left\{ \begin{array}{l} \langle loc\_acc\ ew_1{:}n\rangle \\ \left\langle \begin{array}{c} rem\_acc\langle lp_{ew_1}, pn_{ew_1}, na_{ew_1}, pr_{ew_1}\rangle \\ \wedge \\ auth\ aa{:}n_1 \end{array} \right\rangle \end{array} \right\}\rangle \end{array} \right\}, \langle W\rangle,$$

$$\left\{ \begin{array}{c} (root, adm){:}adm, \\ (n, g){:}(aa, n_1) \end{array} \right\}, \{\langle pp_{ew_1}, \{\langle mac_{ew_1}, \{ip_{ew_1}\}\rangle\}\rangle\}, \varnothing, \left\{ \begin{array}{c} (MS\ Windows, 7), \\ (TIA\ Portal, 13.0) \end{array} \right\}\rangle$$

$$\langle aa, \{\langle auth, \left\{ \begin{array}{l} \langle rem\_auth\ \langle lp_{aa}, pn_{lp_{aa}}, ip_{aa}, pr_{lp_{aa}}\rangle\ c_1\ aa{:}n_1\rangle \\ \langle rem\_auth\ \langle lp_{aa}, pn_{lp_{aa}}, ip_{aa}, pr_{lp_{aa}}\rangle\ c_2\ aa{:}n_2\rangle \end{array} \right\}\rangle\}, \langle W\rangle, \left\{ \begin{array}{c} (root, adm){:}adm, \\ (n_1, g_1), (n_2, g_2) \end{array} \right\},$$
$$\{\langle pp_{aa}, \{\langle mac_{aa}, \{ip_{aa}\}\rangle\}\rangle\}, \varnothing, \varnothing\rangle$$

$$\langle plc_2, \{\ \langle manage, \{\ \langle rem\_acc\langle lp_{plc_2}, pn_{plc_2}, na_{plc_2}, pr_{plc_2}\rangle\rangle\ \}\rangle\ \}, \langle W\rangle, \{(root, adm){:}adm\},$$
$$\{\langle pp_{plc_2}, \{\langle mac_{plc_2}, \{ip_{plc_2}\}\rangle\}\rangle\}, \varnothing, \{(MicroLogix1400, 15.002)\}\rangle$$

Figure 3: Data model fragment

we have considered only installed software packages playing some role in possible attack paths. The meaning of specification in Fig. 3 is, informally, the following:

- Node `i` represents a generic personal computer from which the attacker starts his/her action. It supports a *login* operation for any user having physical access to it.
- Node $\texttt{ew}_1$ enables a *login* operation when the user has physical access to the workstation and is authenticated by the centralized authentication authority `aa` as user n_1. Workstation $\texttt{ew}_1$ also hosts the software environment required to interact with the devices in $\texttt{cell}_1$. This service is modelled as a *design* operation which can be executed either locally or remotely provided that the user is already logged as n on $\texttt{ew}_1$ or authenticated by `aa` as n_1 respectively. Software packages MS Windows 7 and TIA Portal 13.0 are installed on $\texttt{ew}_1$.
- Node `aa` supports the centralized authentication mechanism needed

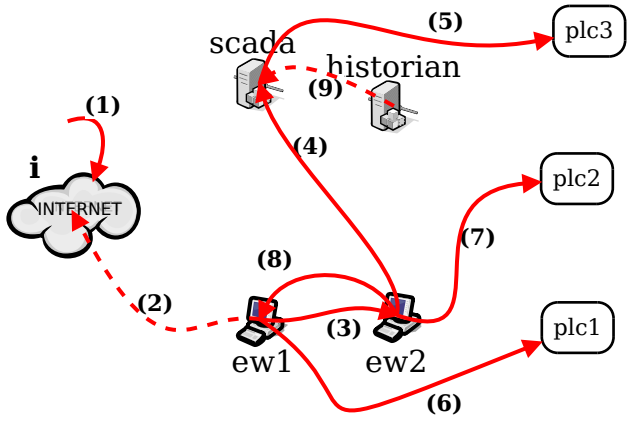

Figure 4: Attack sequence

to access the workstations. In particular, the remote authentication server is listening on logical port lp_{aa} and enables a user, by means of the *auth* operation, to authenticate as n_1 or n_2 provided that he/she knows either password c_1 or c_2, respectively.

- Node $\texttt{plc}_2$ models the "MicroLogix 1400" PLC in $\texttt{cell}_2$ that runs the 15.002 version of MicroLogix firmware. A simple *manage* operation is remotely accessible through logical port lp_{plc_2} bound to the pp_{plc_2} interface.

All nodes include a $(root, adm)$:adm account in their account set, representing a default user with administrator privileges.

5.4. A possible attack sequence

Fig. 4 shows a possible attack sequence in the example network, that can be detected by using our approach. In the figure, a solid arrow between a source and a destination node represents an action performed by the attacker directly from the source node under his/her control towards the target destination or victim. A dashed arrow, instead, represents a vulnerability that is triggered by the victim through his/her interaction with a compromised node. The sequential steps carried out by the attacker (enumerated from (1) to (9) in the figure) are the following. Let us assume the attacker to be in the external room E at the beginning, and both his/her local access (logged in status) and credential sets to be empty, i.e. $\mathcal{T}_a.C = \mathcal{T}_a.LA = \varnothing$. As a

consequence, the attacker is able to execute the *login* operation on node i (1) gaining administrator privileges on that node (i.e. $T_a.LA = \{(i, root)\}$).

The second step in the attack sequence exploits vulnerability CVE-2014-6332 affecting the $\texttt{ew}_1$ workstation. Since $\texttt{ew}_1$ runs the CVE-2014-6332 affected Windows version and connections are possible between $\texttt{ew}_1$ and node `i` (already compromised), then CVE-2014-6332 vulnerability preconditions are met and the attacker can exploit the vulnerability (2) gaining administrator privileges on node $\texttt{ew}_1$ (i.e. $T_a.LA = \{(i, root), (ew_1, root)\}$). Note that a direct access to the workstation would not be possible for the attacker as firewall $\texttt{fw}_2$ blocks any incoming connection to the supervision area. However, the exploited vulnerability leverages the fact that, as it usually happens in real systems, outgoing connections are not subjected to restrictions.

Once gained control on $\texttt{ew}_1$, the attacker is able to exploit remote connections to nodes in the supervision area. In particular the attacker can leverage vulnerability CVE-2014-5424 affecting $\texttt{ew}_2$. Node $\texttt{ew}_2$, in facts, runs version 6.01.00 of the CCW software and is remotely reachable from already compromised $\texttt{ew}_1$. The attacker exploits the vulnerability CVE-2014-5424 (3) and gains control over $\texttt{ew}_2$ node (i.e. $T_a.LA = \{(i, root), (ew_1, root), (ew_2, root)\}$).

Moreover, the attacker can remotely reach the `scada` (from either $\texttt{ew}_1$ or $\texttt{ew}_2$ on which he/she has complete control). The `scada` runs version 7.0 of the WinCC software, affected by CVE-2014-8551 vulnerability. Preconditions of CVE-2014-8551 are then met and the attacker can exploit it (4) gaining administrator privileges on the server (i.e. $T_a.LA = \{(i, root), (ew_1, root), (ew_2, root), (scada, root)\}$).

The configuration of $\texttt{fw}_3$ allows `scada` (and thus the attacker) to establish connections with PLCs in all cells. In particular, $\texttt{plc}_3$ runs the ScheinerWEB application affected by the CVE-2014-0754 vulnerability and the attacker can exploit it (5) gaining administrator privileges on the PLC and basically control over the entire cell (i.e. $T_a.LA = \{(i, root), (ew_1, root), (ew_2, root), (scada, root), (plc_3, root)\}$).

Steps described above are only a suitable selection of elements extracted from the exhaustive set of possible sequences of actions the attacker can perform in the system. The union of all these sequences builds up the complete state machine for the attacker, as mentioned in Sec. 3.

Fig. 5 shows a subgraph of the complete attack graph, built for the attacker in our example and including, among others, the attack sequence described above. In the subgraph every node represents a specific state of the system (i.e., $(\mathcal{D}, \mathcal{T}_a)$) and each edge is labeled with a pair $(exploit(v), \omega)$

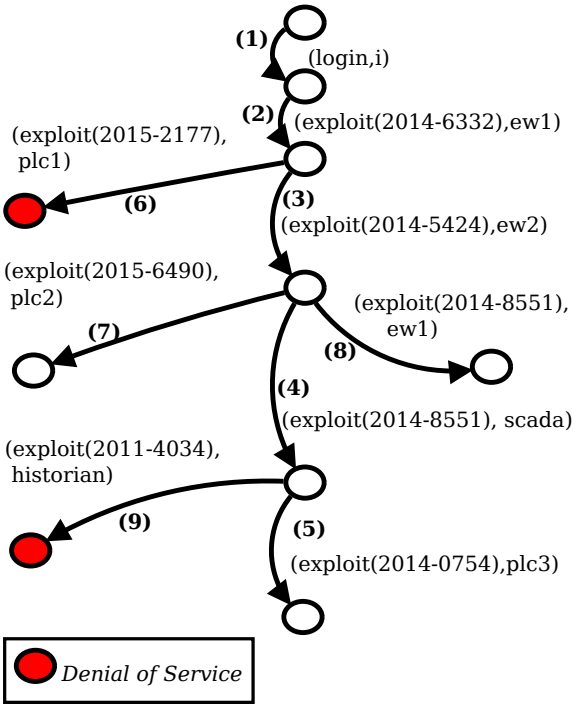

Figure 5: Partial attack graph

identifying the exploited vulnerability v and the affected node ω. The evolution of the system state depends on the effects of the exploited vulnerabilities and, in particular, empty circles indicate that the attacker gained some privileges on some node while filled circles identify states in which a node in the system is affected by *Denial of Service*. Edges (1)-(5) in Fig. 5 show the attack sequence described above and the way any action performed by the attacker (oriented edges) leads the system to a new state (i.e, the attacker gains some new local access at any attack step). In addition, Fig. 5 also shows a portion of other possible attack paths, as our approach allows to automatically determine all possible exploitable sequences of actions the attacker can perform on the system (see edges (6)-(9)). As an example, after performing the first two steps of the previously described attack ($(login, i)$ and $(exploit(2014-6332), ew_1)$), the attacker can exploit vulnerability CVE-2015-2177 on $\texttt{plc}_1$, as firewall $\texttt{fw}_3$ allows the compromised workstation to exchange data with the PLC. By performing action $(exploit(2015-2177), plc_1)$ (edge (6) in the figure) the attacker does not gain any other access to re-

Devices	Operations	Links	Vertexes	Edges	Inferences	Time [s]
23	9	22	6	57	3329	0.001
54	23	53	9	236	99863	0.023
84	37	83	11	476	540882	0.154
114	51	113	13	788	1704413	0.446
159	72	158	16	1391	6261921	1.901
249	114	248	22	3083	33918791	14.931

Table 5: Performances of the prototype implementation against networks with an increasing size of data model $\mathcal{D}$.

sources of the system but is able to cause a Denial of Service on node $\texttt{plc}_1$. An analogous behaviour can be easily deduced for edges (7)-(9).

The state machine structure, recording all possible sequences of actions and their effect on the systems state, is particularly useful for investigating the reasons leading to a specific unwanted state in the system and provides meaningful information about possible fixings and their implementation priorities.

5.5. Preliminary performance evaluation

To get some preliminary information about the tool performance, example systems with increasing size and complexity were taken into account and analysed. This was simply obtained by replicating several times those plant parts included in the supervision and process boxes of Fig. 2, besides adding a suitable number of physical rooms for containing replicas.

Performance tests were carried out with a SWI-Prolog interpreter (version 7.2.3), running under control of the Linux operating system on a personal computer equipped wih an Intel I7 CPU at 3.6 GHz and 8 GB RAM. Results are shown in Tab. 5. The three leftmost columns concern the size of the analysed systems (i.e., total number of devices, operations and network links included in the data model $\mathcal{D}$) while the remaining columns respectively contain the number of vertexes and edges of the resulting attack graph, the number of inferences reported by the Prolog engine and the total time needed to carry out the analysis. It is worth noting that each device (with the exception of switches) includes one vulnerability at least, thus making our experiments a set of quite pessimistic cases since, in real-world scenarios, usually only a part of the system is likely affected by vulnerabilities.

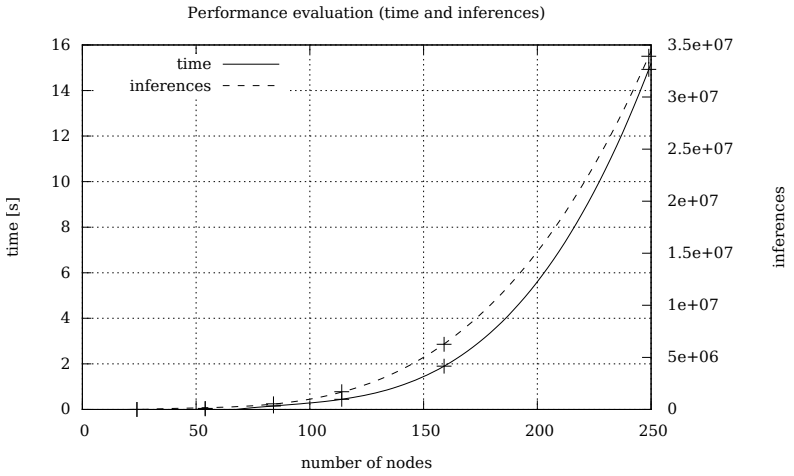

Figure 6: Time and inferences vs total number of nodes. Best fitting curve for time ($R^2 = 0.9998$): $f(n) = -0.608 + 0.034 * n - 5.140 * 10^{-4} * n^2 + 2.518 * 10^{-6} * n^3$, for inferences ($R^2 = 1$): $f(n) = -489732.2 + 30410.93 * n - 551.7273 * n^2 + 3.95385 * n^3$

The number of inferences provides some meaningful indication of the theoretical complexity of the analysis, as it represents the aggregated number of calls to internal Prolog functions. The total execution time, instead, gives a more practical and user-oriented indication of the actual tool performance. Fig. 6 shows how the analysis time and the number of inferences increase with the size of the considered system. Experimental results, in both cases, fit in quite well with a third-order polynomial (coefficient of determination R^2 close to 1). In general, processing times for systems with hundreds of nodes fell below 1 minute.

6. Conclusions and future work

Detecting potential attacks that can be carried out by leveraging sequences of known vulnerabilities is of utmost importance for INS security. In fact, the difficulty or even impossibility in some cases of applying patches and updates on a regular basis to INS h/w and s/w components makes the detection more critical for INS than for other general-purpose networked systems, and unidentified attack paths exploiting combinations of weaknesses affecting these systems are a serious threat to their security.

This paper has dealt with the automatic computation of vulnerabilities

that may be potentially exploited in sequence to attack INS. The proposed approach overcomes some limitations of our previous studies in this area, in particular with respect to the difficulty of keeping the high abstraction level description of the system correctly aligned to its real configuration. In particular, we showed how a suitable extension of the twofold formal model we developed for verifying the correct implementation of access control policies in INS may also be profitably employed for detecting exploitable vulnerability sequences. The presented methodology can then be the basis for developing automatic software analysers helping INS designers and maintainers with the task of granting satisfactory security levels in INS.

To show its effectiveness, a prototype implementation of the proposed methodology has also been developed and used to assess the viability of our approach. Experiments on prototype performance scalability are promising and confirm its applicability to reasonably sized INS.

Future work will deal with the inclusion of metrics (e.g., describing the probability that some existing vulnerability is actually exploited by an attacker or the criticality of some system state) in the exhaustive attack graph generated by our analysis. Note that the methodology presented in the paper is aimed at exhaustively computing all possible sequences of actions an attacker can carry out on system entities, so as to ensure that no threat to its security is left undiscovered. An optimised implementation of the automated analyser can provide results for medium-sized typical INS in a reasonable amount of time, so the adoption of metrics for trimming the generated attack graph does not look mandatory in this case. However, complexity and vulnerability exploitation probability metrics can be useful for very large systems and in the post-analysis phase, in particular for suggesting priorities for interventions. At present, the realistic computation of these metrics is a difficult task, but promising starting points have started to appear such as, for instance, the Common Vulnerability Scoring System used by the NVD database.

Future work will also focus on the full development and optimization of the analyser, in particular to make the vulnerability description automatically updatable through information retrieved from public databases.

[1] W. Granzer, F. Praus, W. Kastner, Security in Building Automation Systems, IEEE Trans. Ind. Electron. 57 (11) (2010) 3622–3630. doi:10.1109/TIE.2009.2036033.

[2] M. Lin, L. Xu, L. T. Yang, X. Qin, N. Zheng, Z. Wu, M. Qiu, Static Se-

curity Optimization for Real-Time Systems, IEEE Trans. Ind. Informat. 5 (1) (2009) 22–37. doi:10.1109/TII.2009.2014055.

[3] E. M. Shakshuki, N. Kang, T. R. Sheltami, EAACK - A Secure Intrusion-Detection System for MANETs, IEEE Trans. Ind. Electron. 60 (3) (2013) 1089–1098. doi:10.1109/TIE.2012.2196010.

[4] M. Cheminod, L. Durante, A. Valenzano, Review of Security Issues in Industrial Networks, IEEE Trans. Ind. Informat. 9 (1) (2013) 277–293. doi:10.1109/TII.2012.2198666.

[5] W. Knowles, D. Prince, D. Hutchison, J. F. Pagna Disso, K. Jones, A survey of cyber security management in industrial control systems, Int. J. Crit. Infrastruct. Prot. 9 (2015) 52–80. doi:10.1016/j.ijcip.2015.02.002.

[6] MITRE, Common Vulnerabilities and Exposures (CVE), the MITRE Corporation. https://cve.mitre.org/.

[7] NIST, National Vulnerability Database (NVD), national Institute of Standards and Technology. https://nvd.nist.gov/.

[8] OSVDB, Open Source Vulnerability Database (OSVDB), http://osvdb.org/.

[9] Symantec, SecurityFocus Vulnerability Database, http://www.securityfocus.com/bid/.

[10] U.S. Depaterment of Homeland Security, Industrial Control Systems Cyber Emergency Response Team (ICS-CERT), https://ics-cert.us-cert.gov/.

[11] Siemens AG, Siemens Product CERT, http://www.siemens.com/cert/en/cert-security-advisories.htm.

[12] M. Cheminod, I. Cibrario Bertolotti, L. Durante, P. Maggi, D. Pozza, R. Sisto, A. Valenzano, Detecting Chains of Vulnerabilities in Industrial Networks, IEEE Trans. Ind. Informat. 5 (2) (2009) 181–193. doi:10.1109/TII.2009.2018627.

[13] P. Maggi, D. Pozza, R. Sisto, Vulnerability Modelling for the Analysis of Network Attacks, in: Proc. of the 3rd Int. Conf. on Dependability of Computer Systems (DepCoS-RELCOMEX), 2008, pp. 15–22. doi:10.1109/DepCoS-RELCOMEX.2008.49.

[14] I. Cibrario Bertolotti, L. Durante, L. Seno, A. Valenzano, A twofold model for the analysis of access control policies in industrial networked systems, Comp. Stand. Inter. 42 (2015) 171–181. doi:10.1016/j.csi.2015.05.002.

[15] M. Cheminod, L. Durante, L. Seno, A. Valenzano, Semiautomated Verification of Access Control Implementation in Industrial Networked Systems, IEEE Trans. Ind. Informat. 11 (6) (2015) 1388–1399. doi:10.1109/TII.2015.2489181.

[16] M. Cheminod, L. Durante, L. Seno, A. Valenzano, On the Description of Access Control Policies in Networked Industrial Systems, in: Proc. of the 10th IEEE Wksp. on Factory Communication Systems (WFCS), 2014, pp. 1–10. doi:10.1109/WFCS.2014.6837594.

[17] M. Cheminod, L. Durante, L. Seno, A. Valenzano, Analysis of access control policies in networked embedded systems: a case study, in: Proc. of the 10th IEEE Int. Symp. on Industrial Embedded Systems (SIES), 2015, pp. 1–10. doi:10.1109/SIES.2015.7185042.

[18] M. Cheminod, L. Durante, L. Seno, A. Valenzano, Analysis of Exploitable Vulnerability Sequences in Industrial Networked Systems: A Proof of Concepts, in: Proc. of the 3rd Int. Symp. for ICS & SCADA Cyber Security Research (ICS-CSR), 2015, pp. 63–72. doi:10.14236/ewic/ICS2015.7.

[19] C. Phillips, L. Painton Swiler, A Graph-Based System for Network-Vulnerability Analysis, in: Proc. of the Wksp. on New Security Paradigms (NPSW), 1998, pp. 71–79. doi:10.1145/310889.310919.

[20] R. W. Ritchey, P. Ammann, Using Model Checking to Analyze Network Vulnerabilities, in: Proc. of the IEEE Symp. on Security and Privacy (S&P), 2000, pp. 156–165. doi:10.1109/SECPRI.2000.848453.

[21] O. M. Sheyner, J. W. Haines, S. Jha, R. Lippmann, J. M. Wing, Automated Generation and Analysis of Attack Graphs, in: Proc. of

the IEEE Symp. on Security and Privacy (S&P), 2002, pp. 273–284. doi:10.1109/SECPRI.2002.1004377.

[22] P. Ammann, D. Wijesekera, S. Kaushik, Scalable, Graph-Based Network Vulnerability Analysis, in: Proc. of the 9th ACM conference on Computer and Communications Security (CCS), 2002, pp. 217–224. doi:10.1145/586110.586140.

[23] S. Jajodia, S. Noel, B. O'Berry, Managing Cyber Threats: Issues, Approaches and Challenges, Vol. 5 of Massive Computing, Springer US, 2005, Ch. Topological Analysis of Network Attack Vulnerability, pp. 247–266. doi:10.1007/0-387-24230-9_9.

[24] R. Lippmann, K. Ingols, C. Scott, K. Piwowarski, K. J. Kratkiewicz, M. Artz, R. K. Cunningham, Validating and restoring defense in depth using attack graphs, in: Proc. of the IEEE Military Communications Conf. (MILCOM), 2006, pp. 1–10. doi:10.1109/MILCOM.2006.302434.

[25] X. Ou, S. Govindavajhala, A. W. Appel, MulVAL: A Logic-based Network Security Analyzer, in: Proc. of the 14th USENIX Security Symp. (SSYM), 2005, pp. 113–128.

[26] Skybox, `http://www.skyboxsecurity.com`.

[27] I. Kotenko, A. Chechulin, A Cyber Attack Modeling and Impact Assessment Framework, in: Proc. of the 5th Int. Conf. on Cyber Conflict (CyCon), 2013, pp. 1–24.

[28] M. Albanese, S. Jajodia, S. Noel, Time-Efficient and Cost-Effective Network Hardening Using Attack Graphs, in: Proc. of the 42nd IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN), 2012, pp. 1–12. doi:10.1109/DSN.2012.6263942.

[29] N. Poolsappasit, R. Dewri, I. Ray, Dynamic Security Risk Management Using Bayesian Attack Graphs, IEEE Trans. Dependable Secure Comput. 9 (1) (2012) 61–74. doi:10.1109/TDSC.2011.34.

[30] T. Sommestad, M. Ekstedt, H. Holm, The Cyber Security Modeling Language: A Tool for Assessing the Vulnerability of Enterprise System Architectures, IEEE Syst. J. 7 (3) (2013) 363–373. doi:10.1109/JSYST.2012.2221853.

[31] B. Chen, Z. Kalbarczyk, D. M. Nicol, W. H. Sanders, R. Tan, W. G. Temple, N. O. Tippenhauer, A. H. Vu, D. K. Yau, Go with the Flow: Toward Workflow-oriented Security Assessment, in: Proc. of the Wksp. on New Security Paradigms (NPSW), 2013, pp. 65–76. doi:10.1145/2535813.2535821.

[32] N. O. Tippenhauer, W. G. Temple, A. H. Vu, B. Chen, D. M. Nicol, Z. Kalbarczyk, W. H. Sanders, Automatic Generation of Security Argument Graphs, in: Proc. of the IEEE 20th Pacific Rim Int. Symp. on Dependable Computing (PRDC), 2014, pp. 33–42. doi:10.1109/PRDC.2014.13.

[33] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, Role-Based Access Control Models, IEEE Computer 29 (2) (1996) 38–47. doi:10.1109/2.485845.

[34] K. Jayaraman, V. Ganesh, M. Tripunitara, M. Rinard, S. Chapin, Automatic Error Finding in Access-Control Policies, in: Proc. of the 18th ACM Conf. on Computer and Communications Security (CCS), 2011, pp. 163–174. doi:10.1145/2046707.2046727.

[35] Y. Sun, Q. Wang, N. Li, E. Bertino, M. Atallah, On the Complexity of Authorization in RBAC under Qualification and Security Constraints, IEEE Trans. Dependable Secure Comput. 8 (6) (2011) 883–897. doi:10.1109/TDSC.2010.55.

[36] A. Cau, H. Janicke, B. Moszkowski, Verification and enforcement of access control policies, Formal Methods in System Design (2013) 1–43doi:10.1007/s10703-013-0187-3.

[37] T. L. Hinrichs, D. Martinoia, W. C. Garrison, A. J. Lee, A. Panebianco, L. Zuck, Application-Sensitive Access Control Evaluation using Parameterized Expressiveness, in: Proc. of the 26th IEEE Symp. on Computer Security Foundations (CSF), 2013, pp. 145–160. doi:10.1109/CSF.2013.17.

[38] D. M. Nicol, W. H. Sanders, S. Singh, M. Seri, Usable Global Network Access Policy for Process Control Systems, IEEE Security Privacy 6 (6) (2008) 30–36. doi:10.1109/MSP.2008.159.

[39] D. M. Nicol, W. H. Sanders, M. Seri, S. Singh, Experiences Validating the Access Policy Tool in Industrial Settings, in: Proc. of the 43rd IEEE Hawaii Int. Conf. on System Sciences (HICSS), 2010, pp. 1–8. doi:10.1109/HICSS.2010.194.

[40] Np-view, `http://http://www.network-perception.com/`.

[41] H. Okhravi, R. H. Kagin, D. M. Nicol, PolicyGlobe: A Framework for Integrating Network and Operating System Security Policies, in: Proc. of the 2nd ACM Wksp. on Assurable and usable security configuration (SafeConfig), 2009, pp. 53–62. doi:10.1145/1655062.1655074.

[42] H. T. Le, C. D. Nguyen, L. Briand, B. Hourte, Automated Inference of Access Control Policies for Web Applications, in: Proc. of the 20th ACM Symp. on Access Control Models and Technologies (SACMAT), 2015, pp. 27–37. doi:10.1145/2752952.2752969.

[43] Microsoft, Active Directory Architecture, https://technet.microsoft.com/en-us/library/bb727030.aspx.

[44] A. S. Foundation, CAS (Central Authentication Service), https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=66427.

[45] A. X. Liu, A. R. Khakpour, Quantifying and Verifying Reachability for Access Controlled Networks, IEEE/ACM Trans. Netw. 21 (2) (2013) 551–565. doi:10.1109/TNET.2012.2203144.

[46] Sufatrio, R. H. C. Yap, L. Zhong, A Machine-Oriented Integrated Vulnerability Database for Automated Vulnerability Detection and Processing, in: Proc. of the 18th USENIX Conf. on System Administration (LISA), 2004, pp. 47–58.

[47] MITRE, Open Vulnerability and Assessment Language (OVAL), the MITRE Corporation. https://oval.mitre.org/language/.

Appendix A.

In the following, we report the modified version of a subset of inference rules presented in [14]. Changes are determined by the new definition of *acc* elements in Eq. (2) (necessary to manage remote authentication and privilege

acquisition by an attacker exploiting node and service vulnerabilities) and involve rules in [14] having *loc_acc* as a precondition.

$$\frac{\begin{array}{c}\omega^\star \vdash (\omega, \pi, f) \ \wedge \ f = loc\_acc\ \omega'{:}n'\ [c],\ [\omega''{:}n''] \\ \wedge \\ \left(\begin{array}{c}\Big((\omega', n')[:(\omega_{aa}, n_{aa})] \in \mathcal{T}_a.LA \ \wedge \ c \in \mathcal{T}_a.C\Big) \\ \vee \\ ((\omega, n)[:(\omega_{aa}, n_{aa})]{:}\,adm \in \mathcal{T}_a.LA\end{array}\right)\end{array}}{(\mathcal{D}, \mathcal{T}_a) \xmapsto{(\pi,\omega)} (\mathcal{D}, (\mathcal{T}_a.LA \cup \{(\omega'', n'')\}, \mathcal{T}_a.\omega_r, \mathcal{T}_a.C))} \tag{A.1}$$

Rule (A.1) enables an attacker to execute operation π, made available by object ω, with *loc_acc* precondition and requiring the user to own a specific account on object ω' defined by username n'. The rule requires the attacker either to be already logged as n' on ω' and to own the possibly needed credential c, or to have administrator privileges on object ω.

Basically, owning a local access with non-empty *adm* field on an object ω allows the attacker to perform any ω-defined *loc_acc* operation bypassing its precondition.

$$\frac{\begin{array}{c}\omega^\star \vdash (\omega, \pi, f) \ \wedge \ f = loc\_acc\ \omega'{:}g'\ [c],\ [\omega''{:}n''] \\ \wedge \\ \left(\begin{array}{c}\Big((n', g') \in \omega'.\{acc\} \wedge (\omega', n')[:(\omega_{aa}, n_{aa})] \in \mathcal{T}_a.LA \wedge c \in \mathcal{T}_a.C\Big) \\ \vee \\ (\omega, n)[:(\omega_{aa}, n_{aa})]{:}\,adm \in \mathcal{T}_a.LA\end{array}\right)\end{array}}{(\mathcal{D}, \mathcal{T}_a) \xmapsto{(\pi,\omega)} (\mathcal{D}, (\mathcal{T}_a.LA \cup \{(\omega'', n'')\}, \mathcal{T}_a.\omega_r, \mathcal{T}_a.C))} \tag{A.2}$$

Analogously to rule (A.1), rule (A.2) enables an attacker to execute operation π made available by object ω with *loc_acc* precondition and requiring the user to be active on object ω' with any account belonging to group g'. The only difference between the two rules is that rule (A.2) requires first looking for an account on object ω' whose username is bound to group g', and then checking whether the attacker has already gained a local access to the object by exploiting that username.

Even in this case owning administrator privileges on object ω allows the attacker to perform π independently of the satisfaction of its precondition.