Analysis of Exploitable Vulnerability Sequences in Industrial Networked Systems: A Proof of Concepts

Manuel Cheminod, Luca Durante, Lucia Seno, Adriano Valenzano
 National Research Council of Italy – IEIIT
 c.so Duca degli Abruzzi 24
 I-10129 Torino
 Italy
www.ieiit.cnr.it
manuel.cheminod, luca.durante, lucia.seno, adriano.valenzano@ieiit.cnr.it

Software vulnerabilities can affect the security of any computer and industrial networked systems are no exception. Information about known vulnerabilities and possible countermeasures is being collected and published since several years, however the methodical introduction of changes and/or software patches in many industrial networks is not always possible, so that some known flaws can be left untreated as they are not considered harmful in principle. Unfortunately, a suitable combination (sequence) of vulnerabilities which are not dangerous when considered as insulated, can provide undesired attack paths to malicious users. This paper deals with the automated discovery of such sequences of known vulnerabilities in industrial scenarios by leveraging an analysis framework already developed for the verification of access control policies in real-world systems.

Keywords: industrial distributed systems, software vulnerabilities, automated analysis

1. INTRODUCTION

Modern control systems distributed over a network (DCS) are pervading an increasing number of application areas including, for instance, factory automated plants, advanced manufacturing systems, intelligent transportation systems, smart grids and critical infrastructures. Because of the ever growing connectivity of DCS, which more and more frequently include interfaces to public networks and the Internet, and the widespread use of advanced communication technologies such as wireless networks, the adoption of methods and mechanisms to grant an adequate level of security is now perceived as a basic requirement in the design, deployment and operation of DCS Granzer et al. (2010); Lin et al. (2009); Shakshuki et al. (2013); Cheminod et al. (2013). Generally speaking, this need emerges whenever technical solutions borrowed from the information and communication technology (ICT) world are used in the supervision, monitoring and/or management of some physical system as in the case of chemical process control, gas, oil and energy production and distribution networks, robotic and highly automatized plants and so on.

As a matter of fact, real ICT software (s/w) and hardware (h/w) components are not perfect and, unavoidably, include design and implementation bugs that can be exploited by malicious users to jeopardize the security of specific equipment, devices or even whole networked systems Cheminod et al. (2013). Such kinds of flaws are better known as vulnerabilities and represent potential security weaknesses that have to be corrected or taken under strict control at least. It is worth remembering that vulnerabilities are not the same as attacks, but rather a means to make possible attacks successful. Moreover, in order to be actually dangerous, vulnerabilities must be exploitable, that is an attacker should be put in the condition of taking advantage of the vulnerability itself. Non-exploitable vulnerabilities, that are possibly present in a system, are not worrying in principle, because they cannot be leveraged to perform attacks and do not need to be fixed mandatory. This aspect can be of particular interest (and useful if dealt with carefully) in many kinds of DCS, where the introduction of changes in the h/w and or s/w is hardly possible or simply not convenient from an economic point of view. 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.

Software vulnerabilities are much more frequent and easy to spread than hardware flaws and have been receiving particular attention since several years. Several publicly-available databases MITRE (a); NIST (NIST); OSVDB (OSVDB); Sufatrio et al. (2004) have been set up to collect and share information about known vulnerabilities and are regularly updated as new security flaws are discovered. To the best of our knowledge, no repository is specifically tailored to either industrial control systems (ICS) or supervision, control, monitoring and data acquisition systems (SCADA), thus vulnerabilities concerning ICS and/or SCADA are mixed up with those affecting general-purpose ICT systems, and only keyword-based searches in these databases (e.g. SCADA, PLC and so on) can properly filter elements of interest.

Even when an existing vulnerability cannot be considered dangerous (not exploitable) on its own, it can become useful to the attacker because of the presence of other different flaws in the same system. This exposes the system itself to attacks than can be carried out by leveraging suitable sequences (or *chains* Cheminod et al. (2009); Maggi et al. (2008)) of vulnerabilities, that represent a sort of path the attacker can follow to achieve his/her malicious goal. Understanding whether a system includes some of these attack paths and finding suitable ways to break them are important issues for assuring adequate levels of security to DCS.

In the past, our research was oriented to the analysis of attacks to industrial distributed systems carried out through sequences of known vulnerabilities Cheminod et al. (2009); Maggi et al. (2008). The approach we adopted made use of a rough-grained abstract model of the system that had the advantage of enabling a fast analysis (as also recognized in Ma and Smith (2013) with a proposal inspired by Cheminod et al. (2009); Maggi et al. (2008)), but also exhibited some drawbacks. In particular, we found it was quite difficult keeping the model aligned with the actual implementation of the system itself, and maintaining adequate track of all those implementation details that could make the difference between a satisfactory and an oversimplified model from the analysis point of view. These aspects, in our experience at least, tended to become hard to manage as the system size and complexity grew.

More recently we have focused on the definition and development of a framework for the automated verification of correct implementation of access policies defined at a high level of abstraction,

which is particularly suited to industrial networked systems Bertolotti et al. (2015). One peculiar aspect of this approach is the construction of two different views of the system to be analyzed, which take into account respectively an abstract specification of policies and a detailed description of the implemented system. The aim of this paper is proposing a suitable extension of this analysis framework to enable the study of sequences of vulnerabilities in DCS and to prove its conceptual feasibility through a simple example.

The paper is then structured as follows: Section 2 briefly recalls the main characteristics of the analysis framework that are needed to understand the remaining part of the paper. Section 3 deals with the extensions and changes needed to enable the description of vulnerabilities in the system model and to allow the subsequent search for possible sequences of exploitable flaws. Section 4 presents a small example showing how the underlying analysis can be carried out in practice while some conclusions are drawn in Section 5.

2. ANALYSIS FRAMEWORK

Our analysis framework is based on the ability to model the system characteristics of interest from two different points of view. Fig. 1 shown the basic blocks building up our approach. In practice, the designer has to provide an high-level definition of the security policies to be analyzed (access policies in our previous studies) and a fine-grained description of the system implementation including its security mechanisms and settings. These two descriptions are then processed by an automated analyzer to produce two disjoint sets of (*user, object, operation*) triples taking into account all the actions that can be performed by all users on all objects in the system. Users, objects and operations are “bridging” elements between the two views and, roughly speaking, the related triples are used by the automated analyzer to make a comparison between the actual system configuration and its (expected) high abstraction level behavior, in order to highlight differences and inconsistencies representing design or implementation flaws.

The implementation view of the system consists of a data model $\mathcal{D}$ describing all objects and their physical and/or logical interconnections, the initial state for each user (i.e. physical location, owned credentials and so on) and a set of inference rules that interactions between the user and the system must obey. Inference rules are used by the automated analyzer to compute the set of all actions allowed to the user and to build the corresponding triples in the implementation view. An exhaustive and

Figure 1: Building blocks of the analysis framework

formal description of the model, its elements and relationships can be found in Bertolotti et al. (2015). To keep the computational complexity and the state explosion problem under control, we consider the system *static*, i.e. interactions between users and the system do not affect the latter. Effects of changes in the system can be evaluated with new runs of analysis after modifying the system model as needed.

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

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

where $\Omega ::= \{\omega^*\}$ is the set of objects, that is the set of all elements of the system on which operations can be performed. Objects include rooms and other containers (e.g. cabinets), host devices (e.g., PCs, PLCs and so on) and relevant software services (e.g. web and database servers, mail servers, s/w applications), and network devices for traffic control (e.g. firewalls, switches, routers).

$\Lambda ::= \{\lambda\}$ keeps track of all physical network connections.

Any object $\omega^* \in \Omega$ is then formally described as

$$\omega^* ::= \langle \omega_{id}, \{\pi^*\}, \omega_{path}, \{acc\}, \{pp\}, \{fr\}, \{sw\} \rangle \quad (2)$$

ω_{id} and ω_{path} in Eq. (2) are an object identifier and a pathname respectively. Since objects can be nested in the data model (an object can contain objects and can be contained in other objects, thus allowing, for instance, the description of virtualized

hosts nested within hypervisors) anyone of them is uniquely identified by both its identifier ω_{id} and its path $\omega_{path} = \omega_{id_1}, \omega_{id_2}, \dots, \omega_{id_k}$, that is the ordered sequence of identifiers of the k objects that need to be “crossed” to access the target one. The unique identifier of an object is obviously given by $\omega = \omega_{path}, \omega_{id}$. To keep the notation as simple as possible, in the following we will use the shortest suffix of ω allowing to uniquely identify the related object. This means that when ω_{id} is sufficient to identify an object with no ambiguity, it will be adopted as the object full name, that is in our example $\omega = \omega_{id}$. Moreover, to refer to a component or sub-part of a structured object, e.g. ω^* , we’ll use the “.” notation, e.g. $\omega^*.\{acc\}$ means the set $\{acc\}$ of ω^* .

Tab. 1 lists the other elements in Eq. (2) with their definitions (note that $[\dots]$ stands for *optional* and $\{\dots\}$ means *a possible empty set*). In summary, the meaning of items appearing in the table is the following:

- *acc* is a user or group *account*, possibly requiring remote authentication through ω_{aa} , defined for *object* ω^* .
- *pp* is a physical port (network interface). id_{pp} is the port unique identifier, while *dla* is a data-link address (e.g. 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 the interface supports

acc	$::= (n, g)[:(\omega_{aa}, n_{aa})]$ is an <i>account</i> where n and g respectively are a username and a group on ω^* . $[(\omega_{aa}, n_{aa})]$ is optional and, if present, it means that the user, in order to log in as user n belonging to group g must authenticate through a remote authentication authority ω_{aa} , with username n_{aa} . Otherwise, the user is authenticated by ω^* , without the need of third parties.		
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 na_d dest network addr na_s source network addr act action (<i>allow</i> or <i>deny</i>)	dla_d dest data link addr dla_s source data link addr pr protocol (TCP,UDP,...)	id_{pp_d} dest physical port pn_d dest port number pn_s source port number
sw	$::= \langle sw_name, ver \rangle$		
λ	$::= \{id_{pp}\}$		

Table 1: Implementation elements

wireless communications, $[c]$ is the possible credential required to connect to the interface (i.e., 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.

- fr is a filtering rule, whose general form is based on the constitutive elements listed in the lower part of Tab. 1. A single rule for a specific device may not include all elements in the table, and wildcards are also allowed. We will not discuss filtering rules in more details in this paper, since we rely on Liu and Khakpour (2013) for the network reachability computation. The general format for fr contains the union of fields needed to specify rules for firewalls, switches and so on. Of course, given a particular device and/or rule, not all fields are either meaningful or needed and, in this case, we use the symbol “_” to denote unused fields.
- sw describes an installed sw package by means of name and version.
- λ is a set of fully interconnected physical port identifiers. A point-to-point link is described by means of two ports, whereas buses can have several ports.

Object descriptions in Eq. (2) include operations. In general, any operation π^* available on object ω^* can take one of the two possible forms:

$$\pi^* ::= \langle \pi, \{\langle d, \{c\}\rangle\} \rangle \quad (3)$$

$$\pi^* ::= \langle \pi, \{f\} \rangle \quad (4)$$

Form (3) is used when the object, to which the operation is bound, is a room or a physical container: π is the operation name, e.g. *enter*, whereas the

f	$::= pre, post$
pre	$::= \langle phy_acc [c] \rangle$ $ \quad \langle loc_acc \omega': n'[c] \rangle$ $ \quad \langle loc_acc \omega': g'[c] \rangle$ $ \quad \langle rem_acc port [c] \rangle$ $ \quad \langle rem_auth lp [c] \rangle$ $ \quad \langle phy_acc \wedge auth \omega': n' \rangle$ $ \quad \langle rem_acc lp \wedge auth \omega': n' \rangle$
$port$	$::= dla \mid lp$
lp	$::= \langle id_{lp}, [pn], na, [pr] \rangle$
$post$	$::= [\omega'': n'']$

Table 2: Preconditions and effects f

set $\{\langle d, \{c\}\rangle\}$ describes all the doors which allow entering the room. $\{c\}$ is the set of all credentials that are needed to open door d . The adoption of a set is useful, in this case, to model situations where each user has his/her own credential to open a given door, besides circumstances where just one credential is shared among all users. Moreover, different credentials can also be assigned to the same door, depending on the direction it is actually crossed.

In analogy with form (3), form (4) describes operations a user can carry out on devices and their hosted 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 (possibly) other resources. 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 a user must be in the same room as ω , i.e. have physical access to

ω , and own credential c (if specified) in order to be able to perform π .

- *loc_acc* $\omega' : n' [c]$ and *local_access* $\omega' : g' [c]$ mean that, in order to perform π on ω , a user must already be active (e.g., logged) on some object ω' , by means of either the user's username n' or the group g' he/she belongs to. Note that preconditions may involve different objects. Moreover, when specified, credential c must also be owned by the user.
- *rem_acc port* $[c]$ means that the operation 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 communications at the data-link level. In the definition of *port*, lp is a logical port whose unique identifier is id_{lp} . 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* $[c]$ means that a remote (possibly centralized) authentication is needed. This precondition is used when π is a *remote authentication operation* provided by an *authentication authority* ω_{aa} listening on logical port lp .
- *phy_acc* $\wedge$ *auth* $\omega' : n'$: this logical and of two preconditions enables the corresponding operation if the user is in the same room as the object, and provided he/she has already been authenticated as user n' by the remote authentication authority ω' , or he/she is able to authenticate with ω' .
- *rem_acc lp* $\wedge$ *auth* $\omega' : n'$ this precondition is similar to the previous one, but in this case the object is remotely accessible through the logical port lp .

The element *post* in Tab. 2 takes into account the possible effect of the operation: in particular $\omega'' : n''$, if 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 investigating sequences of vulnerabilities, we assume that $\mathcal{D}$ describes the static attacker's environment, and denote with $\mathcal{T}_p$ the attacker's state. Effects of actions performed by the attacker (i.e. access gained to a room, acquisition of a logged status and so on) are then recorder in $\mathcal{T}_p$. Consequently, $(\mathcal{D}, \mathcal{T}_p)$ is a representation of the *system state*.

$\mathcal{T}_p$	$::= (LA, \omega, C)$
LA	$::= \{(\omega, n) [: (\omega_{aa}, n_{aa})]\}$ set of all user's local accesses
ω	user current room
C	$::= \{c\}$ user's credentials

Table 3: User's state $\mathcal{T}_p$

As mentioned above, the computation of all actions (and steps) carried out by any user in our model has to take into account the reachability of objects in the network. However, as the way adopted for the description of the network and its devices is compatible with Liu and Khakpour (2013), we assume that the reachability of hosts and devices was computed in a previous step and stored in a suitable database, as the *static* view of the system enables us to do so. The following general query returns true or false as a result, depending on whether or not a logical path exists and is actually enabled by the configuration of the network infrastructure:

$$\exists_path(dla_s, pn_s, na_s, dla_d, pn_d, na_d, pr, \langle \omega, C \rangle) \quad (5)$$

The meaning of dla_s , pn_s , na_s , dla_d , pn_d , na_d has already been introduced and shown in Tab. 1, whereas ω is the identifier of the room where the relevant object is placed and C are the (user's) credentials exhibited to access existing wireless networks (C).

Given a system model, described by means of the formalism introduced above, and a user who is assigned a set of credentials and is located in a certain room, we would like to compute all actions (and sequences of actions) that such a user can perform in the system. However, two elements are still needed to make this goal successful, that is the user's state $\mathcal{T}_p$ and a suitable set of rules $\mathcal{R}$. $\mathcal{T}_p$ is shown in Tab. 3: it includes the set LA of all local accesses (i.e. active logged-in conditions) already obtained by the user, the current room he/she has reached and the set of credentials he/she owns. The set of rules $\mathcal{R}$ defines and controls interactions between the user and the system. Indeed, elements of $\mathcal{R}$ are *inference rules*: their formal specification can be found in Bertolotti et al. (2015) and is not reported here. Roughly speaking, a rule exists for any possible form of precondition f in Tab. 2. Moreover, two additional rules allow the user respectively to move from room to room and to manage the exploitation of vulnerabilities in the system (note that this latter rule is not defined in Bertolotti et al. (2015) but is detailed in the following).

Each rule has an associated set of preconditions, that is logical predicates acting on both the system model and the user's state. When predicates are

satisfied for some triple (user, object, operation)¹, the operation and object elements are recorded in a suitable way, and the postconditions of the rule are applied, i.e. the user's state is updated accordingly. In practice, the new user's state can change all preconditions of other rules to true and so on, so that sequences of actions can be easily computed. This is why both the system model $\mathcal{D}$ and the user's state $\mathcal{T}_p$ appear as arguments in all inference rules.

3. VULNERABILITIES

Several databases, which are publicly accessible online, contain updated descriptions of known vulnerabilities MITRE (a); NIST (NIST); OSVDB (OSVDB); Sufatrio et al. (2004) but, unfortunately, this information is not machine-readable, that is ready to feed an automated software tool, because it is provided in textual, informal language. The need of a formal model suitable for this purpose was stressed in the past Maggi et al. (2008), and a solution was proposed to capture all meaningful information for automated processing (the approach in Maggi et al. (2008) merged outcomes from the *Movtraq* Sufatrio et al. (2004) and *OVAl* MITRE (b) international projects). By applying some ideas borrowed from Maggi et al. (2008) to the formalism presented in Bertolotti et al. (2015), we can say that the *preconditions* of any vulnerability v can be translated into a corresponding set of logical predicates. When all predicates hold for some object ω^* in the current system state $(\mathcal{D}, \mathcal{T}_p)$, then the *postconditions* of v affect the system state by changing it to $(\mathcal{D}, \mathcal{T}_p')$. This means that v postconditions can lead to a new system state where other preconditions are now enabled, thus allowing to analyze *sequences* of vulnerabilities. It is worth reminding that such an analysis enables the study of effects caused by exploiting chains of *known* vulnerabilities, while it is not able to discover new (unknown) flaws, i.e. *zero day attacks*.

As an example, let us consider the following preconditions, which are met very frequently in vulnerability descriptions:

- *remotely_reachable* $(\omega^*, (\mathcal{D}, \mathcal{T}_p))$ is true if the system state $(\mathcal{D}, \mathcal{T}_p)$ allows the attacker to reach object ω^* through a network connection;
- *has_program* $(sw_name, ver, \omega^*, (\mathcal{D}, \mathcal{T}_p))$ is true if object ω^* runs version ver of software sw_name ;
- *usage_link_compromised* $(\omega^*, (\mathcal{D}, \mathcal{T}_p))$ is true if the attacker is authenticated and logged on

some object (host), and such an authentication allows him/her to do some action / exploit some service on the reachable object ω^* ;

- *locally_exploitable* $(\omega^*, (\mathcal{D}, \mathcal{T}_p))$ is true if the attacker is logged on object ω^* .

The main *postcondition* of interest for our purpose is also the most common: *gain_privilege* $(root, \omega^*, (\mathcal{D}, \mathcal{T}_p))$, states that the attacker gains root (unlimited) privileges on object ω^* .

The simple elements discussed above enables the description of real vulnerabilities such as the following (vulnerabilities are identified by means of their CVE MITRE (a) id):

CVE-2006-0058: vulnerability affecting the sendmail program. The affected versions are: 8.13.0, 8.13.1, ..., 8.13.5.

An attacker able to exploit this vulnerability can *execute arbitrary code* on the affected node, thus he gains some kind of *privilege* on the node.

Formal precondition set:

$\{remotely_reachable(\omega^*, (\mathcal{D}, \mathcal{T}_p)),$
 $has_program(sendmail, 8.13.1, \omega^*, (\mathcal{D}, \mathcal{T}_p))\}$

Formal postcondition set:

$\{gain_privilege(root, \omega^*, (\mathcal{D}, \mathcal{T}_p))\}$

CVE-2007-0213: vulnerability affecting the MS Exchange program. The affected versions are: 2000 sp3, 2003 sp1, 2003 sp2, 2007. An attacker able to exploit this vulnerability can *execute arbitrary code* on the affected node, thus he/she gains some kind of *privilege* on the node itself.

Formal precondition set:

$\{remotely_reachable(\omega^*, (\mathcal{D}, \mathcal{T}_p)),$
 $has_program(MS\ Exchange, 2007, \omega^*, (\mathcal{D}, \mathcal{T}_p))\}$

Formal postcondition set:

$\{gain_privilege(root, \omega^*, (\mathcal{D}, \mathcal{T}_p))\}$

CVE-2007-0027: vulnerability affecting the MS Office Suite (2000, 2002, 2003 versions). In particular the Excel software is flawed: when a file containing a malformed IMDATA record is *opened* in Excel, system memory can be corrupted in a way that may allow an attacker to execute arbitrary code.

Formal precondition set:

$\{usage_link_compromised(\omega^*, (\mathcal{D}, \mathcal{T}_p)),$
 $has_program(MS\ Office, 2002, \omega^*, (\mathcal{D}, \mathcal{T}_p))\}$

Formal postcondition set:

$\{gain_privilege(root, \omega^*, (\mathcal{D}, \mathcal{T}_p))\}$

CVE-2004-1235: vulnerability affecting the Linux kernel 2.4 through 2.6.10. It allows *local* users to execute arbitrary code by manipulating the VMA descriptor.

¹in this case *some* means that these entities are considered as variables to be bound to instances able to make the relevant predicates true.

Figure 2: Structure of the industrial network

Formal precondition set:

$\{locally_exploitable(\omega^*, (\mathcal{D}, \mathcal{T}_p)),$
 $has_program(Linux\ kernel, 2.6.10, \omega^*, (\mathcal{D}, \mathcal{T}_p))\}$

Formal postcondition set:

$\{gain_privilege(root, \omega^*, (\mathcal{D}, \mathcal{T}_p))\}$

The exploitation of any vulnerability mentioned above can be managed through a suitable inference rule whose predicates and postconditions are the same as the vulnerability formal sets.

The extension of the analysis framework Bertolotti et al. (2015) to include the study of vulnerabilities requires a suitable adaptation of the syntax derived from Maggi et al. (2008) to fit in well with the model. Let us informally define the following action

$$\omega^* \vdash (\overline{dla}, \overline{na}) \quad (6)$$

which returns a pair consisting of a datalink address $\overline{dla}$ and a network address $\overline{na}$ such that $\overline{na}$ is bound to $\overline{dla}$ for some physical port of ω^* .

Formal definitions for the vulnerability preconditions are then the following:

$remotely_reachable(\omega^*, (\mathcal{D}, \mathcal{T}_p)) ::=$

$$\begin{aligned} \exists \omega_s^* \in \mathcal{D}.\Omega \mid & (\omega_s, n) : [(\omega_{aa}, n_{aa})] \in \omega_s^*.\{acc\} \wedge \wedge \\ & (\omega_s, n) : [(\omega_{aa}, n_{aa})] \in \mathcal{T}_p.LA \\ \omega_s^* \vdash & (\overline{dla}_s, \overline{na}_s) \wedge \omega^* \vdash (\overline{dla}, \overline{na}) \wedge \\ \exists path & (dla_s, -, na_s, dla, -, na, -, \langle \mathcal{T}_p.\omega, \mathcal{T}_p.C \rangle) \end{aligned}$$

$has_program(sw_name, ver, \omega^*, (\mathcal{D}, \mathcal{T}_p)) ::=$

$$(sw_name, ver) \in \omega^*.\{sw\}$$

$locally_exploitable(\omega^*, (\mathcal{D}, \mathcal{T}_p)) ::=$

$$\begin{aligned} (\omega, n) : & [(\omega_{aa}, n_{aa})] \in \omega^*.\{acc\} \wedge \\ (\omega, n) : & [(\omega_{aa}, n_{aa})] \in \mathcal{T}_p.LA \end{aligned}$$

$$\begin{aligned} gain_privilege(root, \omega^*, (\mathcal{D}, \mathcal{T}_p)) ::= \\ (\omega, root) \text{ added to } \mathcal{T}_p.LA \end{aligned}$$

The $usage_link_compromised(\omega^*, (\mathcal{D}, \mathcal{T}_p))$ is similar to $remotely_reachable(\omega^*, (\mathcal{D}, \mathcal{T}_p))$ but also requires that the user be remotely logged with the same account which enables the execution of some operation on ω^* .

4. EXAMPLE

Fig. 2 shows the simple network used to show how our technique works. Since we are interested in proving the feasibility of the proposed approach we selected the same example presented in Cheminod et al. (2009), irrespective of whether it might not represent a real threat today because of the presence of very old software versions and the existence of widespread updates. The main motivation is that the adoption of the same example enables comparisons of the two approaches also in terms of computational complexity, besides the precision of results. For sake of conciseness only nodes and devices actually needed in the discussion are explicitly taken into account in Fig. 2. The subnetwork protected by firewalls fw_1 and fw_2 represents a demilitarized zone (DMZ) containing company servers that are directly accessible from the outside world, i.e. with public IP addresses. Company internal hosts, either servers or desktop computers, are located between firewalls fw_2 and fw_3 . These devices have private IP addresses and fw_2 performs suitable address translations (NAT). A rather common architectural solution adopts two different mail servers: access to the corporate (internal) server ims is more protected with respect to the external server ems , which is simply used to relay emails to ims . The field (control) network is

$$\begin{aligned} & \langle i, \{ \langle login, \{ \langle phy_acc \ i: root \rangle \} \}, \langle W \rangle, \{ (root, adm) \}, \{ \langle pp_i, \{ \langle mac_i, \{ ip_i \} \} \rangle \}, \emptyset, \emptyset \rangle \\ & \langle aa, \{ \langle auth, \{ \langle rem_auth \langle lp_{aa}, pn_{lp_{aa}}, ip_{aa}, pr_{lp_{aa}} \rangle \ c \ aa: n \} \} \}, \langle W \rangle, \left\{ \begin{array}{c} (root, adm), \\ (n, g) \end{array} \right\}, \{ \langle pp_{aa}, \{ \langle mac_{aa}, \{ ip_{aa} \} \} \rangle \}, \emptyset, \emptyset \rangle \\ & \langle ems, \{ \langle mail, \{ \langle rem_acc \langle lp_{ems}, pn_{lp_{ems}}, ip_{ems}, pr_{lp_{ems}} \rangle \} \} \}, \langle W \rangle, \{ (root, adm) \}, \{ \langle pp_{ems}, \{ \langle mac_{ems}, \{ ip_{ems} \} \} \rangle \}, \emptyset, \{ (sendmail, 8.13.1) \} \rangle \\ & \left\langle ims, \left\{ \left\langle \begin{array}{c} \langle mail, \{ \langle rem_acc \langle lp_{ims1}, pn_{lp_{ims1}}, ip_{ims1}, pr_{lp_{ims1}} \rangle \} \rangle, \\ login, \left\{ \left\langle \begin{array}{c} rem_acc \langle lp_{ims2}, pn_{lp_{ims2}}, ip_{ims1}, pr_{lp_{ims2}} \rangle \\ \wedge \\ auth \ aa: n \end{array} \right\rangle \right\rangle \right\rangle \right\rangle, \langle W \rangle, \left\{ \begin{array}{c} (root, adm), \\ (n, g): (aa, n) \end{array} \right\}, \{ \langle pp_{ims}, \{ \langle mac_{ims}, \{ ip_{ims} \} \} \rangle \}, \emptyset, \{ (MS \ Exchange, 2007) \} \right\rangle \\ & \left\langle m, \left\{ \left\langle \begin{array}{c} login, \left\{ \left\langle \begin{array}{c} rem_acc \langle lp_{m1}, pn_{lp_{m1}}, ip_{m1}, pr_{lp_{m1}} \rangle \\ \wedge \\ auth \ aa: n \end{array} \right\rangle \right\rangle \right\rangle, \\ runOffice, \left\{ \left\langle \begin{array}{c} rem_acc \langle lp_{m2}, pn_{lp_{m2}}, ip_{m1}, pr_{lp_{m2}} \rangle \\ \wedge \\ auth \ aa: n \end{array} \right\rangle \right\rangle \right\rangle \right\rangle, \langle W \rangle, \left\{ \begin{array}{c} (root, adm), \\ (n, g): (aa, n) \end{array} \right\}, \{ \langle pp_m, \{ \langle mac_m, \{ ip_m \} \} \rangle \}, \emptyset, \{ (MS \ Office, 2002) \} \right\rangle \\ & \left\langle plc, \left\{ \left\langle \begin{array}{c} login, \left\{ \left\langle \begin{array}{c} rem_acc \langle lp_{plc}, pn_{lp_{plc}}, ip_{plc}, pr_{lp_{plc}} \rangle \\ \wedge \\ auth \ aa: n \end{array} \right\rangle \right\rangle \right\rangle, plc: n \right\rangle \right\rangle, \langle W \rangle, \left\{ \begin{array}{c} (root, adm), \\ (n, g): (aa, n) \end{array} \right\}, \{ \langle pp_{plc}, \{ \langle mac_{plc}, \{ ip_{plc} \} \} \rangle \}, \emptyset, \{ (Linux \ Kernel, 2.6.10) \} \right\rangle \end{aligned}$$

Figure 3: Description of hosts

protected by firewall f_{w_3} and host computers plc in this area are soft programmable logic controllers (PLCs) running real-time applications under control of a conventional operating system. A server aa acting as a central authority authenticates domain users to access domain services. The management host m is needed to interact with soft PLCs and provides some services, including network-wide productivity tools for other hosts (not visible in the picture).

Let us assume that all firewalls and switches are properly configured in terms of addresses, ports and protocols, so as to enable the expected legitimate traffic flows, and neglect their corresponding formal models. We only focus on those elements of the host models that are needed to describe a possible attack. To keep the example small, all objects are located in the same room w : this simplifying assumption does not offer too much power to an *external* attacker, since no operation is allowed in the system requiring a *physical access* precondition, except for the node where the attacker is actually logged on. Moreover, let us take into account any installed software package only if it can play some role in the attack, and assume that a remote node i exists where an attacker can log on and start his/her offensive against some company critical device, that is a soft PLC in particular.

Fig. 3 shows the formal description of all hosts in Fig. 2 by using the syntax introduced in equations (2) and (4) and Tab. 1 and 2. As all hosts are placed in room W ($\omega_{path} = \langle W \rangle$), prefix $W :$ has

been omitted everywhere in the picture, and path names are shrunk to host names without any risk of ambiguity. The meaning of the specification in Fig. 3 is informally the following:

- *i* is the host where the attacker can log in to start his/her malicious activity. After physically accessing *i*, he/she carries out the *login* operation and acquires *root* privileges since *i* is under full control of the attacker. The host is equipped with a network interface and has MAC and IP addresses.
- *aa* is the domain central authority for authentication. It support the *auth* operation to authenticate remote users (*rem_auth* precondition), i.e. users who either log in on remote hosts, or connect to servers belonging to the domain. The authentication server accepts requests received through logical port *lp_{aa}* configured with proper attributes and bound to network interface *pp_{aa}*. If the connecting user knows credential *c* (that is he/she has got *c* in his/her credential bag $\mathcal{T}_p.C$) he/she is successfully authenticated by *aa* with username *n* and group *g*.
- *ems* is the external mail server which supports the *mail* operation. *mail* is remotely exploitable (*rem_acc* precondition) by connecting to logical port *lp_{ems}*, bound to the *pp_{ems}* interface. Usually, email relaying services do not require credentials (the server accepts all incoming messages, possibly discarding malformed

packets, and dispatches them to proper recipients). An administrative account ($root, adm$) is defined for ems , but attacks based on its direct exploitation are not considered in this example. By contrast, the existence of such an account is leveraged by the attacker by means of vulnerability $CVE - 2006 - 0058$ affecting the software *sendmail*, version 8.13.1 installed on ems .

- ims offers conventional mail services (*MS Exchange* version 2007) by receiving requests through the proper logical port lp_{ims} , bound to the pp_{ims} interface. The service is not protected by any kind of credential/authentication mechanism: firewalls fw_1 and fw_2 ensure that only messages coming from legitimate hosts, i.e. ems and internal clients, can reach it. A *login* operation for general purpose activities is allowed through domain authentication: the user has to connect to logical port lp_{m_1} from a remote host. The login operation is successful when either the user has already been authenticated with account n on the remote host by the aa authority or he/she performs a preliminary authentication step with aa . If everything proceeds smoothly, the user is logged in on ims , and $(ims, n) : (aa, n)$ is added to his/her bag of gained accesses $T_p.LA$.
- m supports a *login* operation similar to ims , and a *runOffice* service (package *MS Office*, version 2002 installed), which can be invoked by users (already) authenticated in the domain. The user does not gain any login status on m through the service invocation.
- plc runs a *Linux* distribution with kernel version 2.6.10 and offers a conventional *login* service to the domain users. Also in this case, the user gains a logged in status $(plc, n) : (aa, n)$ in case of successful operation.

4.1. A possible attack sequence

Let us assume that the attacker is in room W at the beginning, and both his/her local access (logged in status) and credential bags are empty, i.e. $T_p.C = T_p.LA = \emptyset$. With the exception of i , which is under the attacker's control, all hosts enable operations that can be exploited only remotely. In practice, this means that neither actions can be immediately invoked by the attacker, nor he/she is logged in on any node. Then the only initial step allowed is the *login* operation on i , whose successful completion changes $T_p.LA$ to $\{(i, root)\}$.

The attacker is now able to exploit remote connections, but firewall configurations only allow

him/her to reach the ems server. However, this is enough since ems runs *sendmail* version 8.13.1 affected by vulnerability $CVE - 2006 - 0058$. Vulnerability preconditions are unfortunately met and let the attacker gain root access on ems and bypass any authentication mechanism. $T_p.LA$ becomes $\{(i, root), (ems, root)\}$ and the attacker can get control over a host internal to the company network.

Remote network connections from ems to ims are allowed by firewall fw_2 , and this is what the attacker needs to exploit vulnerability $CVE - 2007 - 0213$ affecting a remotely reachable host running *MS Exchange* version 2007. In brief he/she gets unlimited rights on such a host. Root privileges allow the attacker to impersonate any user (account) defined there, and inherit all rights and permissions on the same node. For this reason, $T_p.LA$ is now $\{(i, root), (ems, root), (ims, root), (ims, n) : (aa, n)\}$.

Host m supports remote executions of its *MS Office* version 2202 application by domain authenticated users. The attacker, thanks to the domain logged in status gained by conquering ims , is so able to run *MS Office* via the connection between ims and m . Preconditions of vulnerability $CVE - 2007 - 0027$ are true and he/she acquires root privileges on m too. $T_p.LA$ becomes $\{(i, root), (ems, root), (ims, root), (ims, n) : (aa, n), (m, root), (m, n) : (aa, n)\}$.

Firewall fw_3 is configured to let domain users login in on plc remotely from m , and the attacker is able to do so. It is worth noting that this is not a kind of flaw, but rather the use of a legal system mechanism that the attacker can access thanks to a sequence of exploited vulnerabilities. Sophisticated attack paths can often leverage legal system mechanisms to pursue the attacker's malicious goal. When he/she gets logged in on plc as domain user n $T_p.LA$ becomes $\{(i, root), (ems, root), (ims, root), (ims, n) : (aa, n), (m, root), (m, n) : (aa, n), (plc, n) : (aa, n)\}$.

The final step leverages vulnerability $CVE - 2004 - 1235$, which is now exploitable since the attacker is logged on plc as domain user n . $T_p.LA$ increases again: $\{(i, root), (ems, root), (ims, root), (ims, n) : (aa, n), (m, root), (m, n) : (aa, n), (plc, n) : (aa, n), (plc, root)\}$, and the attacker reaches its goal.

5. CONCLUSIONS AND FUTURE WORK

Analysing the combined effects of known vulnerabilities in large and complex networks is an important issue to grant the whole system an adequate level of security against attacks. This aspect is even crucial

when industrial networked systems are considered, because in many situations changes and patches cannot be applied systematically to cope with software and hardware flaws as they are discovered. Indeed, peculiarities and requirements of many industrial scenarios such as 24/7 availability or non-interruptibility of critical process control systems, prevent the methodic adoption of countermeasures and lead to tolerate the presence of vulnerabilities when they are recognized not to be exploitable for the security of the actual system. Unfortunately, the combined effects of vulnerabilities, which are not considered harmful when isolated, can give rise to attack paths that can be leveraged by malicious users to undermine the system.

This paper has dealt with the automated detection of such dangerous sequences of known vulnerabilities. In particular we have shown how an analysis framework we previously developed for the verification of access control policies in industrial networked systems can be suitably extended to study vulnerability sequences. In particular, we have discussed required changes to the formal model adopted in the analysis and introduced some additional elements in terms of preconditions and object attributes that are needed to manage vulnerabilities in the implementation view of the modelled system. Future work will focus on the complete formalization of the proposed approach and on the development of an automated analysis software tool derived from the analyzer already available for access control policies.

REFERENCES

- Cheminod, M. et al. (2009) Detecting chains of vulnerabilities in industrial networks. *IEEE Trans. Ind. Inform.*, 5 (2), 181–193.
- Cheminod, M., Durante, L., and Valenzano, A. (2013) Review of security issues in industrial networks. *IEEE Trans. Ind. Inform.*, 9 (1), 277–293.
- Bertolotti, I. C. et al. (2015) A twofold model for the analysis of access control policies in industrial networked systems. *Comput. Stand. Interfaces*, To appear.
- Granzer, W., Praus, F., and Kastner, W. (2010) Security in building automation systems. *IEEE Trans. Ind. Electron.*, 57 (11), 3622–3630.
- Lin, M. et al. (2009) Static security optimization for real-time systems. *IEEE Trans. Ind. Inform.*, 5 (1), 22–37.
- Liu, A. and Khakpour, A. (2013) Quantifying and verifying reachability for access controlled networks. *IEEE/ACM Trans. Netw.*, 21 (2), 551–565.
- Ma, Z. and Smith, P. (2013) Determining risks from advanced multi-step attacks to critical information infrastructures. In: *Proc. 8th International Workshop on Critical Information Infrastructures Security (CRITIS)*, 142–154.
- Maggi, P., Pozza, D., and Sisto, R. (2008) Vulnerability modelling for the analysis of network attacks. In: *Proc. 3rd International Conference on Dependability of Computer Systems (DepCoSRELCOMEX)*, 15–22.
- MITRE. *Common vulnerabilities and exposures (CVE)*. Bedford, MA: The MITRE Corporation Available from <http://cve.mitre.org/>
- MITRE. *Open vulnerability and assessment language (OVAL)*. Bedford, MA: The MITRE Corporation Available from <http://oval.mitre.org/index.html>
- National Institute of Standards and Technology. *National vulnerability database (NVD)*. Available from <http://nvd.nist.gov/>
- OSVDB. *Open source vulnerability database (OSVDB)*. Available from <http://osvdb.org/>
- Shakshuki, E., Kang, N., and Sheltami, T. (2013) EAACK - A secure intrusion-detection system for MANETs. *IEEE Trans. Ind. Electron.*, 60 (3), 1089–1098.
- Sufatrio, R., Yap, H. C., and Zhong, L. (2004) A machine-oriented integrated vulnerability database for automated vulnerability detection and processing. In: *Proc. 18th USENIX Systems Administration Conference (LISA)*, Symantec. SecurityFocus Vulnerability Database, 47–58. Available from <http://www.securityfocus.com/vulnerabilities/>