Detecting Chains of Vulnerabilities in Industrial Networks

Detecting Chains of Vulnerabilities in Industrial Networks

Manuel Cheminod, Ivan Cibrario Bertolotti, *Member, IEEE*, Luca Durante, Paolo Maggi, Davide Pozza, Riccardo Sisto, and Adriano Valenzano, *Senior Member, IEEE*

***Abstract*—In modern factories, personal computers are starting to replace traditional Programmable Logic Controllers, due to cost and flexibility reasons, and also because their operating systems now support programming environments even suitable for demanding real-time applications. These characteristics, as well as the ready availability of many software packages covering any kind of needs, have made the introduction of PC-based devices at the factory field level especially attractive.**

However, this approach has a profound influence on the extent of threats that a factory computing infrastructure shall be prepared to deal with. In fact, industrial personal computers share the same kinds of vulnerabilities with their office automation counterparts. Then, their introduction increases the risk of cyber-attacks.

As the complexity of the network grows, the problem rapidly becomes hard to tackle by hand, due to the subtle and unforeseen interactions that may occur among apparently unrelated vulnerabilities, thus bearing the focus on the full automation of the analysis. Going into this direction, this paper presents a software tool that, given an accurate and machine-readable description of vulnerabilities, detects whether or not they are of concern and evaluates consequences in the context of a factory network.

***Index Terms*—Industrial communication systems, computer network security, chains of vulnerabilities, automatic analysis tools.**

I. Introduction

NOWADAYS, personal computers are starting to replace more expensive and less flexible special purpose hardware such as, for example, Programmable Logic Controllers, at the field level of the factory environment.

This trend has been made possible because, recently, the most important limiting factor in this respect, that is, the limited ability of personal computers and their operating systems to support demanding real-time applications, has been overcome. In fact, several products, either commercial or open-source, now tightly integrate real-time control applications with a general purpose operating system [1]–[6]. The ever increasing viability and interest of this approach are corroborated by the importance and reputation of its supporters.

However, besides advantages, this approach contributes to bring to the factory field level the typical risks experienced daily in office automation, such as bugs, system vulnerabilities, and cyber-attacks, also because the tighter integration among different management levels makes the factory field level more sensitive to faults and errors propagating from other layers. Indeed, the widespread usage of commercial off-the-shelf software products and the increased connectivity among the field level and the corporate network — and even the Internet — is a trend widely aknowledged [7]–[9].

The increased and widespread interest in the security of factory field infrastructures is highlighted in [10], where a broad overview of cyber security and risk assessment for Supervisory Control and Data Acquisition (SCADA) and Distributed Control Systems (DCS) is provided, together with references to public and private groups which are active in this area. Moreover, the National Institute of Standards & Technology (NIST) has recently provided a comprehensive set of best practices and design criteria of Industrial Control Systems (ICS) [11] to meet their (new) security requirements and needs in the framework of a standardization effort aimed at both rising awareness and setting guidelines for enhancing the security of control networks and SCADA systems in general.

In turn, this means that, nowadays, the design and management of the factory field infrastructure have to deal with a whole new set of risks which were generally neglected in the past.

This process is speeded up by the coexistence of real-time tasks with a general-purpose operating system on the same host that makes the former susceptible, at least to some extent, to the same vulnerabilities affecting the latter. In particular, any vulnerability of the general-purpose operating system, which grants the attacker the ability to run code in the most privileged execution mode of the CPU (for example, privilege ring 0 in Intel processors), also enables the attacker to halt or disrupt the system as a whole, including real-time tasks.

As a consequence, the paramount problem is reasoning about consequences of a vulnerability in the context of a factory network, often part of a multi-layered corporate network, but performing this task by hand is very tedious, error-prone, and thus impractical for large networks. Taking into due account the subtle interactions among apparently unrelated vulnerabilities only makes things worse, also because it is often possible for the attacker to take advantage of a vulnerability

Manuscript received —; revised —.

P. Maggi, D. Pozza, and R. Sisto are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, I-10129 Torino, Italy (e-mail: {paolo.maggi, davide.pozza, riccardo.sisto}@polito.it).

M. Cheminod, I. Cibrario Bertolotti, L. Durante, and A. Valenzano are with IEIIT, National Research Council, I-10129 Torino, Italy (e-mail: {manuel.cheminod, ivan.cibrario, luca.durante, adriano.valenzano}@polito.it).

This work has been partly funded by the EU FP6/IST Project DESEREC: Dependability and Security by Enhanced Reconfigurability (IST-2004-026600) and the CNR project “Metodi e strumenti per la progettazione di sistemi software-intensive ad elevata complessità”.

in order to make another network component vulnerable as well, giving rise to a so-called *chain* of vulnerabilities.

Currently, the job of discovering the hosts in a network that are potentially prone to known vulnerabilities can be automated by using vulnerability scanners [12]–[14], but these tools give little or no information about attacks that can be conducted through *sequences* of vulnerabilities encompassing *multiple* hosts.

System vulnerability analysis is not a new topic by itself, at least from the theoretical point of view [15]–[26], but, as far as the authors know, no attempts have been made in the field of factory infrastructures: [10], [11], [27] only address best practice design criteria and risk assessment based on probability of malicious intrusions, without taking into account attacks that can be conducted through sequences of vulnerabilities, and automatic tools for their detection, while [28] proposes an automatic technique, based on a stochastic attack model too, to provide risk assessment of control networks and of the controlled power plants.

More recently, [29] attempted to integrate system vulnerability and fault propagation analysis [30]–[32] — another important source of concern in industrial networks — in order to address both of them within a single framework, comprising both the system description and the analysis method.

Since several preliminary and more qualitative results presented in [33] look promising from the feasibility point of view, the goal of this paper is to show how the analysis technique outlined in [29] can be profitably implemented as a usable tool and then applied to real-world factory information systems with an acceptable computational effort. However, this paper mainly focuses on vulnerability analysis, so fault propagation is not going to be considered in the following.

The extent and quality of the information provided by any tool of this kind is strongly influenced by its input data, hence another important issue to be considered is the availability of accurate, machine-readable information about the new vulnerabilities that are continuously being discovered, in order to consider them and determine what vulnerabilities can represent a real threat for a given system.

Moreover, the existing vulnerability databases [34]–[39] are mainly designed for human consumption, and the information they contain is not easy to read and process automatically. In order to tackle this issue, this paper adopts an extended XML-based language to describe vulnerabilities, based on the existing *Movtraq* [40] and *OVAL* [41] projects and described in [42].

Both the analysis tool and the vulnerability description language have been developed within, and integrated with, the formal analysis workflow of the European Union FP6/IST Integrated Project "Dependability and Security by Enhanced Reconfigurability" (DESEREC) [43]. As far as we know, no practical experience about the application of these techniques to real factory information systems has ever been reported elsewhere in the literature.

The paper is organized as follows: Section II presents the overall architecture of the analysis tool, as well as its typical workflow. Then, Section III summarizes the formalism being introduced for vulnerability description and Section IV describes in detail the underlying Prolog-based analysis engine. Section V draws an application example in the context of factory networks and Section VI gives some information about the performance and scalability of the tool. Section VII reviews some of the related works and draws some conclusions.

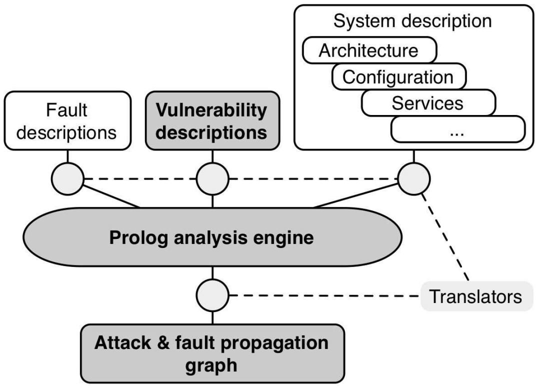

Fig. 1. Overall architecture of the analysis tool, with the elements presented in detail in this paper highlighted.

A demonstration version of the tool is also available online for further experimentation, at `http://www.dai-arc.polito.it/enginframe/analyze`.

II. Analysis Architecture

The overall architecture of the analysis tool is shown in Fig. 1. The analysis engine takes its input from several sources of information by means of appropriate translators:

- The *vulnerability* descriptions convey information about known vulnerabilities and their effects, as better described in Section III.
- The *fault* descriptions contain information about the possible faults that can affect the network components.
- The *system* description describes the architecture of the network being analyzed, as well as the configuration of each host of the network, services running in the network, and relationships and dependencies among hardware and software components of the network.

The system description can, in principle, be very detailed and include the list of installed software with their configurations for each network node, as well as fine-grained details on running processes and the list of available services. The more precise these models are, the more accurate the analysis is. However, modelling precisely the configuration of every host in the network is not always feasible, because some details can be unavailable or simply unknown when the analysis is being performed. Therefore, the description formalism must admit that details might be missing and the analysis tool must be prepared to degrade its performance gracefully when coping with such a potential lack of details.

This means that the tool makes some conservative assumptions in order to provide a "worst case" analysis: missing or unknown details in the configurations are assumed to be the worst, from the point of view of the system administrator (who yearns for a secure network), but, at the same time, to be the best from the point of view of a malicious agent (who tries to compromise the network). Of course, the drawback of these conservative assumptions is that if details are missing the probability of false alarms increases. In our case, a false positive is a weakness situation checked by analyzing the model, which simply does not exist in the real system. However, because of the conservative nature of our assumptions, it is not possible that in the absence of certain details some attacks go undetected while being instead detectable on the fully detailed model.

Working on its input models, the analysis engine performs an exhaustive search for every action an attacker can perform in order to reach his malicious goals, considering both vulnerabilities and faults. The output of the tool is an *attack and fault propagation graph*, in which every node is a step of an attack or of the propagation of a fault and arcs connect steps together to build a causal and temporal relationship between them. That is, two nodes are connected by an arc if the attack step described by the latter node is enabled as a consequence of the exploitation of the attack step described by the former. Root nodes represent the "access points" of the attacker into the network, that is, they represent the attack steps that can be carried out on the network from the very beginning of the attack itself.

Attack steps can also have different meanings, depending on the kind of attacker's action associated with them. Namely, the attacker can try to exploit some vulnerabilities on hosts he has access to, or he can try to perform some actions allowed by the configuration of the system. For instance, a host can be configured in order to allow a password-less login from a trusted remote computer, but if such a remote host is already compromised, then the attacker can perform the password-less login without actually "attacking" the system, rather by "leveraging" its configuration.

Every time the malicious agent can perform more than one action, the tool has to consider all of them, thus leading to more than one arc coming out from a node and to a tree-like structure in the graph.

III. Vulnerability Model

The vulnerability definition language has been defined as an extension of the OVAL language [41], aimed at overcoming its main limitations and at making it more suitable as a modeling language for automatic analysis. First of all, the test clauses of OVAL determine if a vulnerability is *present* on a given host in terms of installed software and its configuration, but they do not assess if the vulnerability can be *exploited*, that is, if an attacker has the capabilities — in terms of required privilege level, connectivity, file access rights, and so on — to leverage a vulnerability known to be present on a given host. Furthermore, the *effects* of the exploitation of a vulnerability cannot be described.

Instead, both these concepts are well defined in *Movtraq* [40] and are very useful, because they allow to verify if the exploitation of a vulnerability can induce the conditions needed for the exploitation of other vulnerabilities in the system, thus creating a *chain* of vulnerabilities. On the other hand, the choice of OVAL as the starting point is strongly motivated by the existence of a comprehensive OVAL vulnerability repository, whose contents can be readily converted into the enhanced modeling language once complemented with the missing information; the same is not true for Movtraq [40].

In more detail, the main changes revolve around the extension of the XML element that defines a vulnerability (the `definition` element) to incorporate two new sub-elements, `pre-conditions` and `post-conditions`:

- The `pre-conditions` section extends the information provided by the `criteria` section of standard OVAL (that describes the tests needed to establish whether a vulnerability exists or not) by supporting additional tests aimed at checking whether a vulnerability can be exploited or not. The extended grammar provides support for logic statements like, for example, "the vulnerability can be exploited only if the attacker is a local user with root access" or "the vulnerability can be exploited only if there is no free space on the disk".
- The `post-conditions` section contains the conditions that will hold *after* the vulnerability is exploited, that is, it contains definitions of effects of the vulnerability exploitation. Like the `pre-conditions` section, the `post-conditions` one is made up of one or more `criteria` elements joined together with `AND` and `OR` operators. In turn, each `criteria` element is composed of `post-criterion` elements and/or other nested `criteria`. Unlike the `criterion` statements in the `pre-conditions` case, `post-criterion` elements do not refer to tests, but to effects.

It is worth noting that tests and effects do not have the same expressive power. In general, effects describe higher level conditions than tests, whereas tests express more precise and detailed conditions than effects. This situation arises because vulnerability post-conditions usually imply a larger set of new attacker capabilities and such conditions are also more difficult to be determined in a precise way.

Hence, the automatic reasoning tool will be responsible for mapping post-conditions (i.e. a small set of wide conditions) into preconditions (i.e. a larger set of detailed and precise conditions). This mapping can be performed by means of rules that correlate post-conditions with preconditions. For example, consider a vulnerability where the effect is "the attacker can write on file X". Now, consider the following vulnerability precondition: "File X must contain the text Y". In that case, a mapping rule could infer that if an attacker can write into a certain file, then he can change the file contents so that the file includes the required text and consequently exploit the vulnerability.

Another extension, not further discussed in this paper, provides the ability to indicate, for each vulnerability, a set of metrics that describe its characteristics and impact such as, for example, the effects severity, the exploitation complexity, whether authentication is required or not to exploit the vulnerability and the kind of impact in terms of confidentiality, integrity, availability, and privilege escalation. These metrics are based on the Common Vulnerability Scoring System (CVSS v2) [44] and can be useful to perform risk analysis, in order to determine which vulnerabilities are more likely to be exploited and which ones can cause more harm.

Besides the major changes and additions presented above, a minor but very important enhancement has also been introduced. It consists of the addition of a new boolean attribute to `criterion` elements. This attribute, called `mutable`, has been designed as a hint to improve the performance of the analysis tool. If `mutable` is false, it is assumed that the result of the test the `criterion` refers to cannot change over time. Hence, the analysis tool is allowed to evaluate the test only once and cache the result. If `mutable` is true instead, the outcome of the test may change during analysis, as an effect of the exploitation of some other vulnerability.

For example, a criterion that checks whether the contents of a certain configuration file match some requirements will likely be `mutable`, because the contents may change if the attacker gains write rights on that file. On the other hand, a test that checks the version of the operating system running on a host is unlikely `mutable`, because the installation of a new operating system version as part of an attack is usually considered an extremely unlikely event.

As an example, the listing of Fig. 2 shows a machine-readable description of the vulnerability “CVE-2006-0058”. Albeit several parts of the description, not relevant to the discussion, have been either simplified or omitted completely, the main elements described in this section are still evident.

In particular, the vulnerability preconditions criteria (rows 12–24 of the listing) are expressed as the logical and/or combination of several tests, from `tst:1` to `tst:8`. In turn, these tests are defined in their own `<tests>` section: for example, `tst:1` (rows 33–36) is a software existence test, and is true if at least one instance of the software object `obj:1` is found on the victim host. The details about this software object are found in their own section (rows 54–57), which specifies to look for `Sendmail` version `8.13.0`.

Tests `tst:2` through `tst:6` have been omitted because are quite similar, and check for `Sendmail` versions `8.13.1` through `8.13.5`, respectively. Hence, the first part of the preconditions criteria (rows 14–18) is met if either one of these $sendmail$ versions are running on the victim host.

The second part of the criteria (rows 19–22) is met if either `tst:7` or `tst:8` is true, that is, if the attacker is either a remote (rows 58–60) or a local (rows 61–63) user as specified by the corresponding user objects `obj:7` and `obj:8`. These two tests are neither mutually exclusive, nor exhaustive, because being a remote user requires the ability of reaching the target node. Hence, for instance, a user can both have a local account on a host and be able to reach it through the network (thus being both a local and a remote user). On the other hand, it is also possible for a user not to have a local account and being unable to reach the target node (in this case, that user is neither local nor remote).

The overall criteria are then met if both parts are satisfied, because they are tied by a logical AND relation (row 13).

The preconditions of the above-mentioned vulnerability can also be expressed in natural language as:

1) a $sendmail$ server version 8.13.x, with $0 \leq x \leq 5$, is running on the victim host, and
2) the attacker is either a local user on the victim host, or can reach the server as a remote user.

Moreover, by looking at the $sendmail$ server characteristics, found in the system description, the tool can also determine that the `smtp` remote access must come through port 25 that, in this case, corresponds to the default port.

The vulnerability post-conditions (rows 25–29) describe the vulnerability effects, only `eff:1` (rows 48-52) in this case. The effect specifies that the attacker gains user-level privileges (`ste:1`, rows 66–68) as a local user (`obj:8`, rows 61–63).

IV. System Description and Analysis Engine

The analysis tool relies and works on the abstract concept of *node*, into which all aspects of the input descriptions are mapped. In turn, each node is given a set of *attributes* that convey various kinds of information about the node itself. Unlike node and attribute names, which are restricted to be Prolog atoms, an attribute value can be any valid Prolog term and thus hold structured, complex data items and even functions.

Special attributes, containing references to other nodes, are used to arrange the nodes into a layered, hierarchical architecture that reflects the structure of the system and, according to an object-oriented paradigm, the organization of nodes into classes. In order to give the representation expressive power enough to cover all requirements of the analysis without sacrificing its efficiency, the binding between an attribute and its value is established on-demand, that is, only when the value is actually required.

Besides a static assignment, the value itself may be derived from several dynamic sources, namely, the knowledge base built during the analysis or the lazy evaluation of a Prolog predicate. In logic programming, this technique delays a computation until it becomes known that its results are actually needed. As a consequence, performance increases because unnecessary calculations are avoided, especially when dealing with complex control and data structures.

Moreover, an inheritance mechanism allows nodes to get attribute values from their ancestors in the class hierarchy. A value accumulation mechanism can also be enabled for an attribute; it allows an attribute to get multiple values, from both a given node and its ancestors. In this way, attributes are able to represent both information taken directly from an input description and information deduced by the tool itself during the course of the analysis in a convenient and unified way.

As an example, Fig. 3 shows the internal description of several components of a simple network. In the figure, solid ovals represent components, dashed ovals denote classes, and dotted arrows are a graphical representation of the special attribute conveying the instance/class and class/class relationship. Other attributes are listed beside the nodes themselves.

The host node is a generic super class with two attributes: type and vulnerabilities_list. The first one simply states what kind of node is represented by the class, a generic computer in this case. The second attribute, on the contrary, is one of the most important attributes in the analysis process, because it explicitly lists all the vulnerabilities modeled in the system.

```
<?xml version="1.0" encoding="utf-8"?>
<deserec_definitions>
 <definitions>
  <definition id="..." class="vulnerability">
   <metadata>
    <affected family="unix">
     <platform>Any</platform>
     <product>Sendmail</product>
    </affected>
    <reference source="CVE" ref_id="CVE-2006-0058"/>
   </metadata>
   <preconditions>
    <criteria operator="AND">
     <criteria operator="OR">
      <criterion test_ref="tst:1"/>
      ...
      <criterion test_ref="tst:6"/>
     </criteria>
     <criteria operator="OR">
      <criterion mutable="true" test_ref="tst:7"/>
      <criterion mutable="true" test_ref="tst:8"/>
     </criteria>
    </criteria>
   </preconditions>
   <postconditions>
    <criteria operator="AND">
     <postcriterion effect_ref="eff:1"/>
    </criteria>
   </postconditions>
  </definition>
 </definitions>
 <tests>
  <software_existance_test
     check="at least one" id="tst:1">
   <object object_ref="obj:1"/>
  </software_existance_test>
  ...
  <user_test
     check="at least one" id="tst:7">
   <object object_ref="obj:7"/>
  </user_test>
  <user_test
     check="at least one" id="tst:8">
   <object object_ref="obj:8"/>
  </user_test>
 </tests>
 <effects>
  <gain_effect id="eff:1">
   <object object_ref="obj:8" />
   <state state_ref="ste:1" />
  </gain_effect>
 </effects>
 <objects>
  <software_object id="obj:1">
   <name>Sendmail</name>
   <version>8.13.0</version>
  </software_object>
  <user_object id="obj:7">
   <type>remote</type>
  </user_object>
  <user_object id="obj:8" version="1">
   <type>local</type>
  </user_object>
 </objects>
 <states>
  <user_state id="ste:1">
   <privilege>user</privilege>
  </user_state>
 </states>
</deserec_definitions>
```

Fig. 2. A machine-readable description of vulnerability “CVE-2006-0058”.

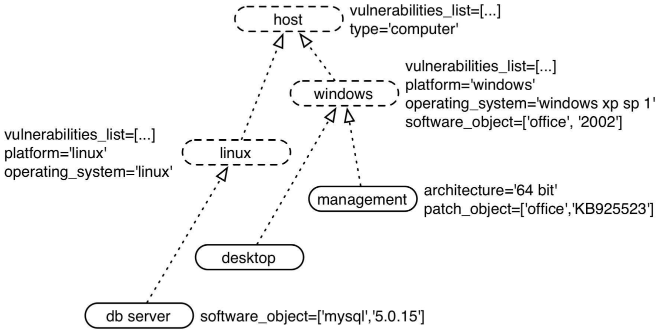
host
vulnerabilities_list=[...]
type='computer'
vulnerabilities_list=[...]
platform='windows'
operating_system='windows xp sp 1'
software_object=['office', '2002']
windows
vulnerabilities_list=[...]
platform='linux'
operating_system='linux'
linux
management
architecture='64 bit'
patch_object=['office','KB925523']
desktop
db server
software_object=['mysql','5.0.15']

Fig. 3. A simplified example of system description.

Unless otherwise specified, they all are supposed to affect the components derived from this class.

Two different classes are derived from this base class: windows and linux, each one represented by a node. These classes represent two distinct families of computers with different potential vulnerabilities, hence the vulnerabilities_list attribute is overridden in order to distinguish vulnerabilities affecting the Windows systems from those concerning Linux.

At this level of abstraction, classes have additional attributes like, for example, operating_system. These attributes can just be placeholders without other details, or can behave as shared attributes with a default value. In the example, the linux class points out a *generic* linux operating system, whereas the windows class precisely indicates that the running operating system is windows xp sp1. Moreover, the software_object attribute specifies that the office 2002 suite is installed. Every derived subclass or instance will inherit such attributes.

The software_object attribute also shows how the value accumulation mechanism already described can be useful. In this case, it allows the system description to specify which software has been installed on each host in an incremental way starting, for example, from software common to a group of hosts (to be described in software_object attributes associated with the corresponding host class) and then delving into the fine details about the software configuration specific to one single host in the software_object attributes associated with that specific host.

Finally, the db server, desktop and management nodes are declared as instances of two derived classes. More precisely, db server describes a server of the linux family that runs the mysql database software version 5.0.15, desktop is an instance of the windows class (and inherits all the attribute values specified for that class). Finally, management also inherits the attribute values of the windows class, but further specifies the architecture as being 64 bit and adds a patch_object to model the installation of a patch for the Office suite.

With this kind of configuration, if the operating_system attribute of the management node is needed, the Prolog engine will step through the class inheritance mechanism, eventually retrieving the attribute value of the windows class. On the contrary, the value of the architecture attribute will be retrieved directly from the management node.

The Prolog analysis tool is layered on top of the factual information just described. In particular, an intermediate layer contains information about the meta-model, that is, it defines which classes and attributes are meaningful and available for use in the analysis. This layer also gives semantics to complex attributes, whose value cannot be found in the system description, but comes from a possibly elaborated computation. Barring the fact that these attributes are defined at the class level through lazily evaluated Prolog functions, they can nonetheless be used like any other.

Finally, the topmost layer is the Prolog engine. It reasons about the input descriptions (bottom layer), by means of the meta-model (intermediate layer). As done by other authors [18], the most important assumption taken for granted at this level is the so-called *monotonicity* assumption, which states that if a new knowledge item is gained, then it is never thrown away. Since it implies that further reasoning can never retract an already stated fact, this assumption has the advantage of mitigating the complexity of the analysis but, at the same time, does not loose its realism.

In the prototype of the tool described in [29], the concept of vulnerability and its associated model were both quite abstract. That is, vulnerabilities preconditions and post-conditions were represented in a straightforward way, as Prolog facts concerning the node affected by them, like other authors did [22].

Unfortunately, this approach lacks the necessary expressive power when confronted with the way real-world vulnerabilities are described in the existing vulnerability databases because, as discussed in Section III, the vulnerability model used in this case must be much more sophisticated. In particular, the possibility of exploiting a certain vulnerability is related to the evaluation of a set of *tests*, and not to merely considering whether a certain *fact* about a node is true or not.

The core of the analysis, however, still resides in the *unification* between the preconditions of a vulnerability and the current state of the system. For instance, if a precondition requires the presence of a specific software_object in the system, like a software named sendmail with version number 8.13.1, the tool has to navigate through the system description in order to find a compatible object, otherwise the precondition is evaluated as not valid. This search mechanism enables the analyzer to automatically identify any possible initial attack point in the modelled network as well as any subsequent attack node.

The state of the system, though, can change when a vulnerability is exploited and, therefore, the effects of the vulnerability have to be applied to the system in order to bring it into the new state. This action is performed by updating or adding the attributes associated with the affected nodes.

For example, a vulnerability that allows an attacker to gain user level privileges on the victim host, once exploited, associates the new attribute privileges(user) with the victim. This new information can then be matched with a precondition like "the attacker needs to have user level privileges on victim host", thus potentially starting a new step in an attack path.

It should also be noted that the unification process is not always trivial like in the previous case, because preconditions and effects can contain and depend on variables and their relationships. For instance, a vulnerability can have an effect like "the attacker can modify any file in directory X", whereas another vulnerability can have the precondition "the configuration file F must contain a certain string S", with S known to, or computable by, the attacker.

When the tool checks whether the second vulnerability can be exploited or not, first of all it has to determine whether file F resides in directory X or not. Then, it can infer that if the attacker can modify the contents of file F, it can also change its contents so that it contains the required string S and come to a conclusion.

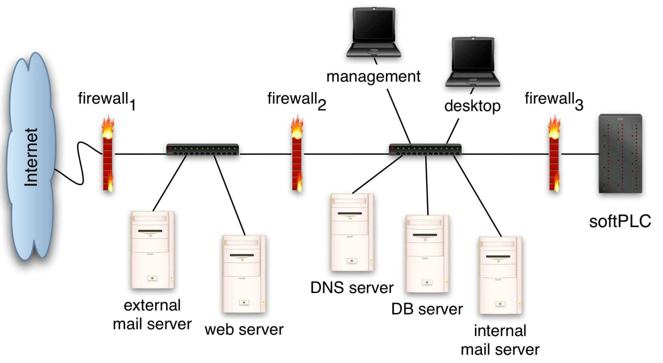

Fig. 4. Structure of the industrial network considered in the example.

Hence, the fact that a vulnerability can be exploited does not depend only on the presence of a vulnerable agent running on a given host, but may also be affected by the details of the host configuration. This additional information may not be captured by other tools that only use the results of a scanner as their source of vulnerability data [21]–[24]. For example, a remote scan executed at the sub-network level could not detect the presence of vulnerabilities affecting a given software component if that component were not running at the time the scan is performed.

The same framework can also be used to model complex post-conditions such as, for example, arbitrary code execution in a simple and efficient way. Since the match between vulnerability preconditions and system state is carried out by unification, arbitrary code execution can be represented by means of a process object containing a unification wild card as the process name.

V. An Example

A. Network Architecture and Configuration

The architecture of the network being analyzed is depicted in Fig. 4. Albeit simplified to keep the example short, its overall structure has been inspired by a real factory network that was designed to control a (small) production system. It still bears a strong resemblance with the network from which it has been derived. The network is subdivided into three main areas:

- A DeMilitarized Zone (DMZ), where publicly accessible servers are hosted. All hosts in this area have public IP addresses, in order to accept connections from the outside world.
- A private LAN, with internal corporate hosts and internal services. The hosts in this area have private IP addresses since firewall$_2$ also performs Network Address Translation (NAT). Albeit a typical corporate network comprises many computers for administrative and office personnel, such richness of nodes has been here abstracted away into two classes: desktop and management nodes. This abstraction is realistic because it is likely that nodes with similar purposes have almost identical configurations.
- The field level of the network resides behind firewall$_3$ and is made up of softPLC computers that run both a general

TABLE I
SOFTWARE PACKAGES INSTALLED ON THE EXAMPLE NETWORK

Host	Software	Version
web server	apache	2.0.59
internal mail server	MS Exchange	2007
external mail server	sendmail	8.13.1
DNS server	MS Windows Server	2003
DB server	mysql	5.1.15
desktop	MS Office	2002
management	MS windows XP	64bit SP1
soft PLC	Linux kernel	2.6.10

TABLE II
MAIN FIREWALL RULES FOR CONNECTION REQUESTS

Firewall	Direction	Protocol	Host
1	in	http	web server
	in	smtp	external mail server
	out	*any*	*any*
2	in	smtp	internal mail server
	out	*any*	*any*
3	in	ssh	management
	out	*any*	*any*

purpose operating system and a real-time application. As before, it is assumed that the single softPLC node depicted in the example actually models a whole class of hosts with the same configuration.

The network exposes, in the DMZ area, two public servers: the web server and the external mail server. The former hosts an *apache* web server, whereas the latter is a *sendmail* server configured as a relay for the internal mail server. This is a commonly used practice to avoid direct exposure of the corporate mail server to the Internet. In the internal network, the dns server and the db server provide DNS features and general DB storage capabilities, respectively.

Last, desktop represents a set of generic office computers and management stands for a set of administrative computers. The main difference between these two classes, besides the installed software packages, is that only an administrator of the network can have an account on the administrative computer. Moreover, the administrative computer allows incoming Server Message Block (SMB) connections from the desktop machine, and an SSH trust relationship between the management and the softPLC hosts allows management people to log into the softPLC host without typing any password once they have logged into the management computer.

Table I summarizes the software packages installed on the various hosts of the network, which are relevant for the example.

The network is partitioned into different areas by means of three *stateful* firewalls that, besides examining the protocol and port fields of incoming packets as stateless firewalls do, also keep track of the state of network connections going through it and, in particular, are aware of and act upon TCP connection requests. More precisely:

- The first firewall, $\mathsf{firewall}_1$, protects the factory network from the outside world of Internet. It only allows incoming `http` and `smtp` connections directed to the hosts in charge of them (within the DMZ zone) and does not provide any kind of routing.
- The second firewall, $\mathsf{firewall}_2$, acts as a filter to isolate the private, internal LAN and routes traffic from the private LAN to the Internet through a NAT service. The only incoming connections allowed by this firewall are `smtp` connections directed to the internal mail server. This is necessary in order to accept the relay of the incoming mail from the external mail server.
- The third and last firewall, $\mathsf{firewall}_3$, protects hosts at the field level from unauthorized access. The only incoming connections allowed by this firewall are `ssh` connections. This kind of connection is used to supervise and manage the field level equipment from the management node.

The actual firewall rules are shown in Table II: from left to right, the columns list the firewall number, traffic direction, protocol and destination host. With respect to Fig. 4, the "in" traffic direction refers to traffic from the outside (left side of the picture) to the inside (right side of the picture), whereas "out" refers to traffic from inside to outside. In Table II, the host represents either the source or the target of the connection request, depending on the direction.

The topology described here prevents the field level from being directly accessed by untrusted users. However, this small example also shows how the extent of connectivity in a real industrial network has significantly increased from the scenario in which human operators initially worked within the process control system's "blast zone" with its stand-alone network [9].

B. Analysis Results

At a first glance, the network described in the previous section does not allow any kind of direct access from the Internet to the field area. As expected, the only servers accessible from outside the corporate network are the web server and the external mail server. In a perfect world, this configuration would be secure and the field area would be effectively isolated from the outside.

However, the analysis of the network produces an attack graph with 35 nodes and 11 distinct attack paths. For the sake of brevity, only one of the attack paths, shown in Fig. 5, will be discussed in detail. In the picture each node, representing an attack step, has been labelled with the name of the network element being attacked. Solid nodes correspond to the exploit of a vulnerability, whereas dashed nodes represent attack steps, in which the attacker used an intended quality of the network to his own advantage. A more detailed description of the attack step, including a reference to a vulnerability database entry if appropriate, appears beside each node.

In this example, the attacker initially acts from the Internet, that is, from outside the corporate network. From there, his first malicious action is directed to the DMZ zone and, in particular, to the external mail server. As specified in Table I, this host runs version 8.13.1 of *sendmail*, but for this software version the tool is aware of vulnerability "CVE-2006-0058". This vulnerability is exactly the one discussed in Section III and Fig. 2. As all other vulnerabilities discussed in the paper, it is a real-world vulnerability taken from the CVE database [34] and its description is publicly available on the CVE web server.

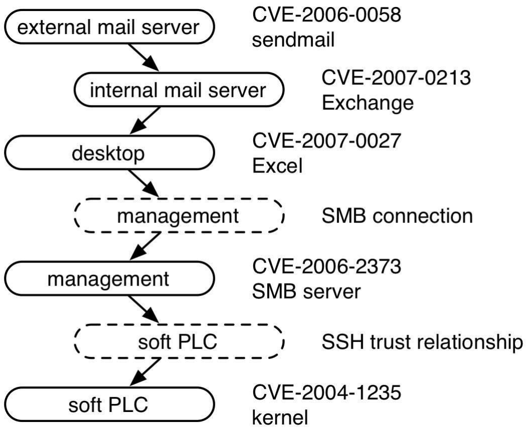

Fig. 5. An attack path from the Internet to the factory field level.

In this case, both preconditions of the vulnerability are satisfied and the attacker can exploit it. The exploitation leverages a signal handler race condition in *sendmail* and allows the remote attacker to execute arbitrary code on the server, thus *acquiring privileges* on the victim host. Since this host is within the DMZ, the attacker actually gains access to a server behind the first firewall.

The external mail server has been configured to act as a relay for the internal mail server, hence $\mathsf{firewall}_2$ has been configured to allow `smtp` connections from the DMZ to the corporate network. Due to the previous attack step, which gave to the attacker user level privileges on the external mail server, he can now access the internal mail server and exploit one of its vulnerabilities, namely, the "CVE-2007-0213" vulnerability concerning the Exchange product family.

In more detail, this flaw in the email decoding process allows an attacker to gain administrator level privileges on the victim host by sending it an appropriately forged email message. This is a very important accomplishment for the attacker, because it can now use a host within the corporate network to further pursue its goals.

Indeed, from the internal mail server it can exploit the "CVE-2007-0027" vulnerability, affecting the Microsoft Office suite. More precisely, he can gain administrator level privileges on the desktop node by sending a specially crafted file to it and tricking the victim user into opening it with Microsoft Excel. This attack step is indeed possible, because the attacker can now send email messages to the corporate hosts from the internal mail server, and the desktop user is supposed to trust emails coming from this source (and modeled accordingly).

It is worth noting that the tool does not report the management node as susceptible to the same vulnerability. This is correct because the vulnerability has among its preconditions the absence of software patch "KB925523" and, as shown in Fig. 3, the system description states that the patch has indeed been installed only on the management node. This example remarks the importance of a detailed system description in order to have accurate results and also shows how the hierarchical description method being used supports a compact description of wide node classes without hindering the specification of fine-grained details about a single node when appropriate.

From the compromised desktop host the attacker can establish an SMB connection with, and gain user level privileges on, the management host. Then, the tool found that the SMB component of Windows XP 64bit SP1 installed on the management host has a flaw that allows a local user to *elevate* his privileges.

After acquiring administrator level privileges on the management node, the attacker uses the SSH trust relationship between the management node and the softPLC to his own advantage. Finally, the last step involves another local vulnerability, "CVE-2004-1235", which affects the Linux kernel version 2.6.10 running on the softPLC host. This vulnerability is quite severe and very relevant in this example, since it permits execution of arbitrary code in privilege ring 0 and the consequent possible disruption of any real-time application running on that host.

After examining the results of the analysis, the system administrator can now look for countermeasures to apply in order to avoid such an attack scenario. For instance, in this case, a good choice would be to patch the Office suite on the desktop host (like it has already been done on the management node), given that a patch is readily available. After having applied this modification, another run of the tool can then be performed in order to show how many problems have been solved.

VI. Performance and Scalability

In general, the performance and scalability of a software tool can be assessed in two very different ways. On the one hand, its design documents and source code can be inspected in order to determine the computational complexity of the algorithms being used in an *analytic* way. On the other hand, it is also possible to perform a set of experiments, in which the tool is actually executed on a set of test problems believed to be representative of real-world usage patterns, and *measure* its performance in some way.

In this particular case, due to some intrinsic difficulties of the Prolog language, like the lack of an explicit control flow, an approximate assessment of the complexity was carried out by analysis, and then double-checked with a set of experiments.

For what concerns measurements, although the *execution time* is with no doubt a very useful performance index because it provides an immediate idea of the practical usefulness of the tool, when using the Prolog language another common way of measuring performance is to count the number of logical *inferences* executed by the software during its execution. This approach has the advantage of providing a platform-independent indicator, which measures the actual complexity of the algorithms being executed rather than the level of sophistication of the Prolog execution engine or the raw speed of the computing platform being used. Both these factors are in fact likely to rapidly change in the future.

For this reason, the results presented in this paper use the inference count as the primary performance index. To correlate this information with the real behavior of the tool on

TABLE III
NOTATION USED IN THE COMPLEXITY ANALYSIS.

Symbol	Meaning
n	Number of nodes
v	Number of known vulnerabilities
p	Max. number of preconditions
q	Max. number of post-conditions
v'	Max. number of vulnerabilities affecting a node
r	Number of reachability rulesets

a consumer PC platform, its execution times — using the SWI Prolog engine version 5.6.14 on a Linux PC with a 1.73 GHz Intel Pentium M CPU and 1.5 GByte of RAM — are also given in a few cases.

A. Analytic Assessment

The tool consists of two sequential phases whose complexity has been evaluated separately:

1) In the first phase, for each node in the network and for each known vulnerability, the tool evaluates its preconditions to check whether it can be exploited or not on the node under consideration. If it is, its post-conditions are put into effect, thus possibly introducing new items into the knowledge base (and enabling the exploitation of other vulnerabilities). The whole process is repeated until a fixed point in the knowledge base is reached, that is, until a whole iteration of this procedure does not add any new item to the knowledge base.

   The analysis starts from a finite set of locations in the network, assumed to represent the attacker's initial positions. These locations are specified by the network administrator and the tool considers them all in the analysis. In most cases, unless the administrator wants to consider attacks that originate from inside his own network, a sensible choice is to choose these locations so that the attacker is located anywhere in the Internet.

   This phase determines the *maximal* set of vulnerable nodes (that is, the set of all nodes on which a vulnerability can possibly be exploited), a piece of information which can indeed provide valuable insights to the designer, because it can point out the risky nodes in the network within a short amount of analysis time. Then, the designer can fix the most important vulnerabilities and then possibly target the second phase of the analysis only on the problematic areas, instead of using it on the whole network at once.

   In addition, this phase also provides several important opportunities for optimization. In particular, during the precondition evaluation, the truth value of the preconditions flagged as not `mutable` is calculated once for all, so that the same calculation will never be repeated in the future. Moreover, for each node, the *maximal* set of vulnerabilities that can possibly be exploited on that node is also determined. This information greatly restricts the number of vulnerabilities to be considered during the second phase.

   The evaluation of a vulnerability requires a number of inferences that is linear in the number of its preconditions. Once a vulnerability has been detected as exploitable, applying its post-conditions to the knowledge base also requires a number of inferences that is linear in the number of post-conditions.

   Assuming that all vulnerabilities are exploitable on all nodes, the best situation for the complexity happens when all of them are discovered in a single iteration. In this case, using the notation of Table III, the complexity is linear with respect to n, p, q, and the number of vulnerabilities v because the whole set of vulnerabilities, with their preconditions and post-conditions, must be evaluated once for each node. On the other hand, the worst case implies discovering the vulnerabilities one at a time, in v iterations. The complexity of a single iteration is still the same as before, but the overall complexity now depends on v^2 instead of v. The complexity C_1 of the first phase can therefore be estimated as:

$$C_1 = \mathrm{O}(npqv^k), \text{with } 1 \leq k \leq 2 \tag{1}$$

   It should be noted that this formula represents a relatively pessimistic upper bound also because it does not take into account that the post-conditions of a vulnerability are evaluated only when the vulnerability can be exploited, and further assumes that all preconditions are `mutable`.

2) The second phase of the analysis refines and completes the information provided by the first one, by determining the *causality* relationship among attacks, thus leading to the construction of the actual attack graph.

   As in the previous phase, the analysis starts from a finite set of locations in the network, assumed to represent the initial attacker's positions, and considers them all. From there, the set of nodes the attacker can reach is computed from the network topology and logical reachability relations among nodes. The calculation requires a number of inferences linear with respect to the number of nodes and reachability *rulesets*, that is, the possible ways of inferring reachability rules. Currently, the reachability rulesets include physical reachability, trust relationships and client/server relationships.

   Given this set, the analysis proceeds to check whether, for each node, the attacker can exploit one of the remote vulnerabilities that may possibly affect that node according to the results of phase one. Each successfully exploited vulnerability opens a new path in the attack graph, leading to the attacked node. When the analysis follows each one of these paths, it applies the post-conditions of the corresponding vulnerability.

   Then, for each new node added to the attack graph, the tool examines each local vulnerability possibly affecting that node in order to determine whether it can now be exploited. For what concerns local vulnerabilities, the tool does not determine all possible sequences of exploitation, in order to achieve a better efficiency. Instead, only one of the maximal sequences is put into the attack graph. This approach does not imply any loss of information, because the sequence taken into account comprises all the locally-exploitable vulnerabilities, and thus gives to the attacker the best position for his next step.

At this point, the whole process is repeated until the attacker is unable to gain further privileges or access rights in the network. Again, this condition happens when a fixed point in the knowledge base is reached.

The overall complexity of the second analysis step depends on the relationships between the attacked nodes, namely:

- If the attacked nodes are *uncoupled*, that is, they are unable to attack each other, the complexity of each iteration of the analysis is cubic with respect to the number of nodes in the network, and the method converges in the worst case after adding all nodes to the attack graph. Following the same line of reasoning adopted to derive (1), and using again the notation of Table III, the complexity can then be expressed as:

$$C_2 = \mathrm{O}(n^m rpqv'), \text{with } 3 \le m \le 4 \quad \text{(uncoupled)} \qquad (2)$$

- Instead, if k of the n nodes are *coupled*, that is, they can attack each other, the analysis enumerates all possible 2^k subsets of these nodes in the attack graph. In turn, since $0 < k \le n$, this necessarily leads in the worst case to an exponential complexity with respect to the number of nodes:

$$C_2 = \mathrm{O}(\mathrm{e}^n rpqv') \quad \text{(coupled nodes)} \qquad (3)$$

B. Experimental Results

The set of measurements presented in this section was carried out on several concrete instantiations of the general network structure described in Section V. Since the analysis discussed in SectionVI-A points out that the number of nodes in the network is the most critical parameter for complexity, the test cases being examined share the same baseline discussed in Section V but differ from one another in the numbers of hosts.

More specifically, each test case includes different numbers of replicas of the desktop, internal mail server, and external mail server nodes in the network. These nodes were chosen for parameterization because they are the closest to the Internet, the attacker's access point. Hence, they are also the closest to the root of the attack graph (as it can be seen by looking at the attack path of Fig. 5) and their replication has the worst possible effect on the growth of the attack graph itself.

For the internal mail server nodes, two different configurations were considered. In the first case, the nodes have been kept *uncoupled*, whereas they have been *coupled* in the second one. The latter situation has been considered because, as explained in Section VI-A, it affects performance most heavily.

The analysis efforts in the scenarios just discussed are compared in Fig. 6, where the number of Prolog inferences is plotted versus the number of replicated nodes of each kind, the remainder of the network being the same. It can be seen that, when the desktop, external mail server, and the *uncoupled* internal mail server nodes are replicated, even in the worst case the experimental data are bounded in a very good way by a power function $y = nx^m$ with an exponent $m = 2.66$, leading to a polynomial complexity. The same trend is confirmed by looking at the actual execution times of the analysis, presented in Fig. 7.

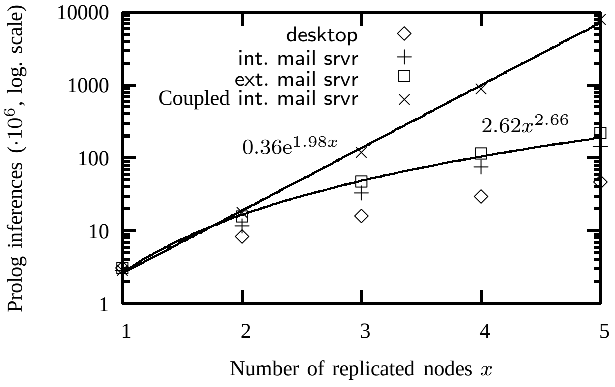

Fig. 6. Complexity of the full-fledged analysis process, as a function of the number of replicated network nodes, their kind, and their relationships.

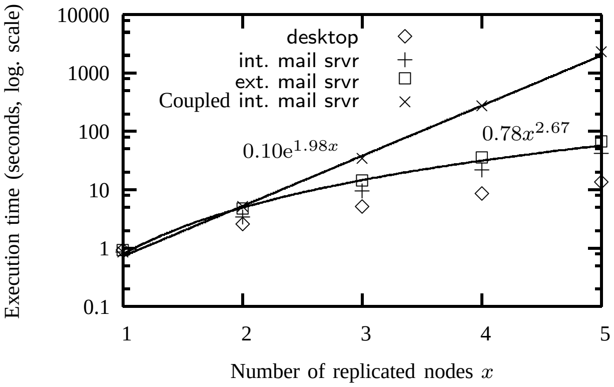

Fig. 7. Execution time of the full-fledged analysis process, as a function of the number of replicated network nodes, their kind, and their relationships.

It should be noted that the correlation between number of inferences and execution time is not necessarily linear because several parameters, for instance the size of the knowledge base, may certainly have an impact on the time required to perform an inference, even if they do not affect the inference count directly.

In order to further inspect the behavior of the tool and show that it is suited to analyze larger networks, too, the example was further extended to comprise up to about 300 nodes. In this case, the focus was put only on the desktop nodes because, among the kinds of node considered in the previous experiments, they are the only ones that can be replicated without hindering the realism of the resulting network. In fact, it is unlikely for an industrial network to have more than a few mail servers, whereas desktop computers usually abound.

The results, in terms of Prolog inferences versus the number of replicated desktop nodes are presented in Fig. 8. The experimental data fit very well a power function $y = nx^m$ with an exponent $m = 3.36$, hence they substantially agree

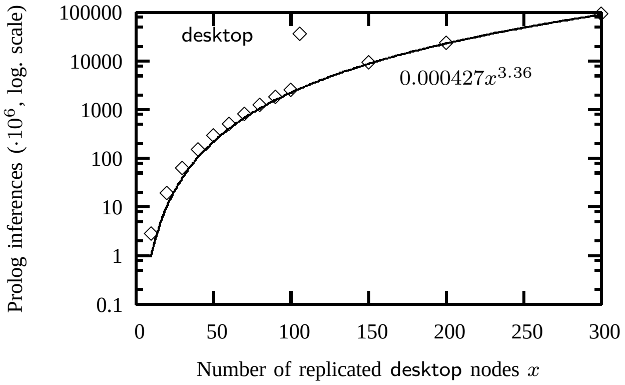

Fig. 8. Complexity of the full-fledged analysis process, as a function of the number of replicated desktop nodes.

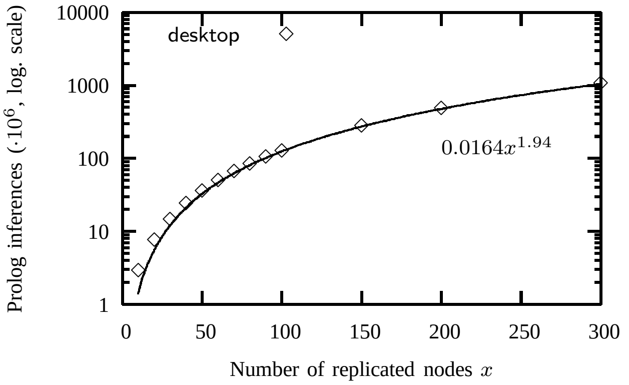

Fig. 10. Complexity of the first step of the analysis process, as a function of the number of replicated desktop nodes.

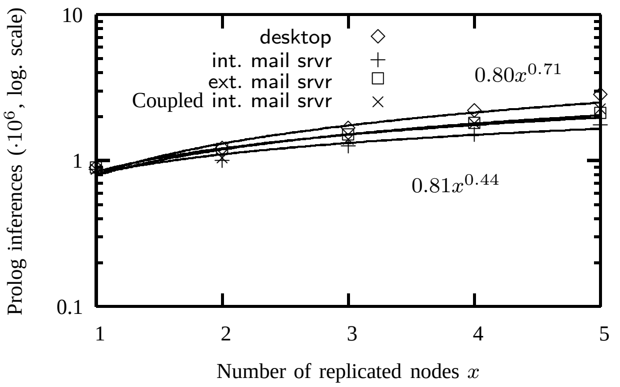

Fig. 9. Complexity of the first step of the analysis process, as a function of the number of replicated network nodes, their kind, and their relationships.

with the complexity trend (2) determined in Section VI-A. With respect to Fig. 6, the exponent of the fitting function increased because, with a bigger number of nodes, the relative weight of the higher powers of x in the complexity formula increased. It should also be remarked that, even in the most challenging case, the total run time of the tool did not exceed several hours.

By contrast, referring again to Fig. 6, the case of the *coupled* internal mail server nodes is very different. As estimated in (3), it correlates very well with the exponential function $y = \mathrm{e}^{1.98x}$, where x is the number of replicated nodes. In this case, one method to alleviate the complexity can be to forgo the attacks *order* and only look at the *set* of attacked nodes. This corresponds to the execution of the first analysis step described in Section VI-A and, as estimated in (1), it has a much lower, polynomial complexity in all situations, as it can be seen in Figs. 9 and 10, for a small and large number of nodes, respectively.

Actually, it can also be noted that the measured complexity shown in Figs. 6–10 is lower than predicted. This is due to the fact that (1) and (2) are worst-case, asymptotic bounds that are reached only for a large number of nodes and have been calculated neglecting several optimizations performed by the tool.

By contrast, the measurements have been taken for a relatively small number of nodes, leading to an optimistic approximation. This point of view is also corroborated by observing that the results better match the theoretical expectations as the number of nodes gets larger.

VII. Related Works and Concluding Remarks

The vulnerability analysis problem was already addressed in pioneering works such as [15], [16], and [19]. In particular, [15] makes use of ad-hoc solutions, whereas [16] and [19] rely on more standard model checking techniques and tools. All these works compute an attack graph where each vertex is a system state, and each edge takes into account a system state transition, triggered by some attacker's action. An exponential computational complexity is the price paid for generating all the possible attack paths. [17] introduces a technique aimed at reducing the graph size of [15], although the problem of the graph generation is not completely addressed.

The computational complexity of the attack graph generation can be reduced in most practical cases under the hypothesis of monotonicity introduced in [18]. Assuming that any exploit doesn't involve more than three hosts, [18] shows that the computational complexity can be $O(n^6)$, where n is the number of hosts in the network. Some hypothesis of monotonicity has been adopted in most of subsequent works.

The Topological Vulnerability Analysis (TVA) tool presented in [20] follows the same approach as [18], but the authors mainly focus on the post-processing of attack graphs, i.e. how the information provided by the attack graph can be profitably presented to the network manager in order he can make the network stronger against potential attacks [45], [46].

An alternative approach to network vulnerability analysis is based on the perspective that penetration testers can have about the maximal level of possible penetration on a given host and was proposed in [21]. The underlying data structure in that case is an access graph, where vertices correspond to hosts, and edges show how an attacker can reach a destination host starting from a given source. In order to keep the computational complexity as low as possible, the authors propose a greedy algorithm, without backtracking. Instead of taking into account all the ways an attacker can reach the target host starting from a given source, the way giving the attacker the greatest power on the target is selected. This approach leads to a sub-optimal solution, where only the worst-case attack paths to all compromisable hosts are computed, with a computational complexity $O(n^3)$.

The authors of [23] compute a logical attack-graph with a complexity between $O(n^2)$ and $O(n^3)$ by using Datalog and its inference engine to model the system and perform the analysis. A computation complexity of $O(n^2)$ is obtained under the hypothesis of constant table look-up time which does not always hold, unfortunately. [26] extends the rules of MulVAL [22], [23] to include the security policies of popular operating systems such as Windows XP® and SELinux™. Moreover, its authors take an incremental approach to update the attack graph, in order to keep the computational complexity low when a what-if analysis is required.

The commercial tool Sky Box View [47] carries out attack graph analysis and the company's patent [48] claims that the algorithm is $O(n^3)$, and $O(n^2)$ is possible.

Finally, by aggregating hosts with respect to their reachability in the network, NetSPA [24] computes multiple-prerequisite attack graphs of the provided example nearly in a linear time. Multiple-prerequisite attack graphs can be expanded in the corresponding, full attack graph: it means that details are not lost in the compact representation. Moreover, NetSPA is also able to deal with credentials, such as passwords, besides usual preconditions.

The automatic tool presented in this paper checks the combined effects of a chain of software vulnerabilities. Also thanks to the development of a sophisticated, XML-based description language stemming from recent research and non-profit organization efforts, notably Movtraq [40] and OVAL [41], the tool is able to reason about real-world threats and can hence be applied in the context of a realistic industrial network.

Even if the raw performance of this tool appears to be worse with respect to some of the competing works [23], [24], at least on the given examples, it should also be noted that its way of modeling vulnerabilities is more powerful due, for example, to the support of `mutable` attributes and to the use of unification in matching vulnerability preconditions with the system state. Both capabilities can be useful in the analysis of real-world networks.

The practical relevance of this problem will likely grow in the near future, due to the ever increasing adoption of hardware and software derived from the office automation environment at every level of the factory computing infrastructure, and even at the field level.

The operating principle and practical applicability of the tool were first described by means of a small example. Then, most importantly, encouraging results about the performance and scalability of the tool with respect to the network size and structure were also derived on realistic networks with up to about 300 nodes. A simplified variant of the analysis was also shown to be suited, at least in principle, to check even much larger networks.

As a future work, we will investigate the possibility of extending the tool to perform risk analysis based on appropriate metrics associated with each vulnerability. Another area of further research is the automatic acquisition of the input data needed by the analysis tool such as, for example, relevant information about vulnerabilities, system structure and configuration. Of course, automation becomes more and more important as the size of the system under analysis grows.

References

[1] TwinCAT. Beckhoff Automation GmbH. [Online]. Available: http://www.beckhoff.com/

[2] Simatic WinAC. Siemens AG. [Online]. Available: http://www.siemens.com/simatic-winac/

[3] Real-Time Linux. MontaVista Software Inc. [Online]. Available: http://www.mvista.com/real_time_linux.php

[4] RTLinux. [Online]. Available: http://www.rtlinux-gpl.org/

[5] RTAI. [Online]. Available: https://www.rtai.org/

[6] CoDeSys. 3S - Smart Software Solutions GmbH. [Online]. Available: http://www.3s-software.com/

[7] A. N. Bessani, P. Sousa, M. Correia, N. Neves, and P. Verissimo, “The crutial way of critical infrastructure protection,” *IEEE Security & Privacy*, vol. 6, no. 6, pp. 44–51, 2008.

[8] M. Brundle and M. Naedele, “Security for process control systems,” *IEEE Security & Privacy*, vol. 6, no. 6, pp. 24–29, 2008.

[9] A. Miller, “Trends in process control systems security,” *IEEE Security & Privacy*, vol. 3, no. 5, pp. 57–60, 2005.

[10] P. A. S. Ralston, J. H. Graham, and J. L. Hieb, “Cyber security risk assessment for SCADA and DCS networks,” *ISA Transactions*, vol. 46, no. 4, pp. 583–594, October 2007.

[11] K. Stouffer, J. Falco, and K. Scarfone, *Guide to Industrial Control Systems (ICS) Security*, NIST - National Institute of Standard and Technology, 2008, Final Public Draft. [Online]. Available: http://csrc.nist.gov/publications/drafts/800-82/draft_sp800-82-fpd.pdf

[12] Security auditor's research assistant (SARA). Advanced Research Corp. [Online]. Available: http://www-arc.com/sara/

[13] Security administrator's integrated network tool (SAINT). SAINT Corp. [Online]. Available: http://www.saintcorporation.com/

[14] Nessus. [Online]. Available: http://www.nessus.org/

[15] C. Phillips and L. P. Swiler, “A graph-based system for network-vulnerability analysis,” in *Proc. ACM workshop on New security paradigms (NPSW)*, 1998, pp. 71–79, DOI: 10.1145/310889.310919.

[16] R. W. Ritchey and P. Ammann, “Using model checking to analyze network vulnerabilities,” in *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2000, pp. 156–165, DOI: 10.1109/SECPRI.2000.848453.

[17] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, “Computer-attack graph generation tool,” in *Proc. DARPA Information Survivability Conference & Exposition (DISCEX)*. IEEE Press, 2001, pp. 307–321, DOI: 10.1109/DISCEX.2001.932181.

[18] P. Ammann, D. Wijesekera, and S. Kaushik, “Scalable, graph-based network vulnerability analysis,” in *Proc. 9th ACM conference on Computer and communications security (CCS)*, 2002, pp. 217–224, DOI: 10.1145/586110.586140.

[19] O. M. Sheyner, J. W. Haines, S. Jha, R. Lippmann, and J. M. Wing, “Automated generation and analysis of attack graphs,” in *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2002, pp. 273–284, DOI: 10.1109/SECPRI.2002.1004377.

[20] S. Jajodia, S. Noel, and B. O'Berry, *Topological Analysis of Network Attack Vulnerability*, ser. Massive Computing. Springer US, 2005, vol. 5, ch. 9, pp. 247–266, DOI: 10.1007/0-387-24230-9_9.

[21] P. Ammann, J. Pamula, J. Street, and R. Ritchey, “A host-based approach to network attack chaining analysis,” in *Proc. 21st IEEE Annual Computer Security Applications Conference (ACSAC)*, 2005, pp. 72–84, DOI: 10.1109/CSAC.2005.6.

[22] X. Ou, S. Govindavajhala, and A. W. Appel, “MulVAL: A logic-based network security analyzer,” in *Proc. 14th USENIX Security Symposium (SECURITY)*, 2005, pp. 113–128.

[23] X. Ou, W. Boyer, and M. A. McQueen, “A scalable approach to attack graph generation,” in *Proc. 13th ACM conference on Computer and communications security (CCS)*, 2006, pp. 336–345, DOI: 10.1145/1180405.1180446.

[24] K. Ingols, R. Lippmann, and K. Piwowarski, “Practical attack graph generation for network defense,” in *Proc. 22nd IEEE Annual Computer Security Applications Conference (ACSAC)*, 2006, pp. 121–130, DOI: 10.1109/ACSAC.2006.39.

[25] R. Rieke, “Abstraction-based analysis of known and unknown vulnerabilities of critical information infrastructures,” *International Journal of System of Systems Engineering*, vol. 1, no. 1–2, pp. 59–77, 2008, DOI: 10.1504/IJSSE.2008.018129.

[26] D. Saha, "Extending logical attack graphs for efficient vulnerability analysis," in *Proc. 15th ACM conference on Computer and communications security (CCS)*, 2008, pp. 63–74, DOI: 10.1145/1455770.1455780.

[27] Plant floor security. Rockwell Automation. [Online]. Available: http://www.isd.mel.nist.gov/projects/processcontrol/members/documents/RA-Workshop-Agenda.pdf

[28] C.-W. Ten, C.-C. Liu, and M. Govindarasu, "Vulnerability Assessment of Cybersecurity for SCADA systems," *IEEE Trans. Power Systems*, vol. 23, no. 4, pp. 1836–1846, 2008.

[29] M. Cheminod, I. Cibrario Bertolotti, L. Durante, R. Sisto, and A. Valenzano, "Evaluating the combined effect of vulnerabilities and faults on large distributed systems," in *Proc. 2nd IEEE International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX)*, 2007, pp. 11–18.

[30] M. L. Shooman, *Probabilistic Reliability*, 2nd ed. Krieger Publishing Company, 1990.

[31] M. Malhotra and K. S. Trivedi, "Power-hierarchy of dependability-model types," *IEEE Trans. Reliability*, vol. 43, no. 3, pp. 493–502, 1994, DOI 10.1109/24.326452.

[32] A. Zarras, P. Vassiliadis, and V. Issarny, "Model-driven dependability analysis of webservices," in *Proc. OTM Confederated International Conferences, CoopIS, DOA, and ODBASE*, ser. Lecture Notes in Computer Science, vol. 3291. Springer-Verlag, 2004, pp. 1608–1625, DOI 10.1007/b102176.

[33] M. Cheminod, I. Cibrario Bertolotti, L. Durante, and A. Valenzano, "On the analysis of vulnerability chains in industrial networks," in *Proc. 7th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2008, pp. 215–224.

[34] Common vulnerabilities and exposures (CVE). [Online]. Available: http://cve.mitre.org/

[35] National vulnerability database (NVD). National Institute of Standards and Technlogy (NIST). [Online]. Available: http://nvd.nist.gov/

[36] Open Source Vulnerability Database (OSVDB). [Online]. Available: http://osvdb.org/

[37] SecurityFocus vulnerability database. Symantec Corp. [Online]. Available: http://www.securityfocus.com/vulnerabilities/

[38] Public cooperative vulnerability database. [Online]. Available: https://cirdb.cerias.purdue.edu/coopvdb/public/

[39] Securebase. SPI Dynamics Inc. [Online]. Available: http://www.spidynamics.com/spilabs/assessment/vulnerability.html

[40] Sufatrio, R. H. C. Yap, and L. Zhong, "A Machine-Oriented Integrated Vulnerability Database for Automated Vulnerability Detection and Processing," in *Proc. LISA '04: Eighteenth Systems Administration Conference*, 2004, pp. 47–58.

[41] Open Vulnerability and Assessment Language (OVAL). [Online]. Available: http://oval.mitre.org/index.html

[42] P. Maggi, D. Pozza, and R. Sisto, "Vulnerability modelling for the analysis of network attacks," in *Proc. 3rd IEEE International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX)*, 2008, pp. 15–22.

[43] FP6/IST Integrated Project "Dependability and Security by Enhanced Reconfigurability" (DESEREC) web site. [Online]. Available: http://www.deserec.eu/

[44] P. Mell, K. Scarfone, and S. Romanosky. Common vulnerability scoring system (CVSS). [Online]. Available: http://www.first.org/cvss/cvss-guide.html

[45] S. Noel and S. Jajodia, "Understanding complex network attack graphs through clustered adjacency matrices," in *Proc. 21st IEEE Annual Computer Security Applications Conference (ACSAC)*, 2005, pp. 160–169, DOI: 10.1109/CSAC.2005.58.

[46] ——, "Optimal ids sensor placement and alert prioritization using attack graphs," *Journal of Network and Systems Management*, vol. 16, no. 3, pp. 259–275, 2008, DOI: 10.1007/s10922-008-9109-x.

[47] Skybox. Skybox Security Inc. [Online]. Available: http://www.skyboxsecurity.com/

[48] G. Cohen, M. Meiseles, and E. Reshef, System and method for risk detection and analysis in a computer network. United States Patent 6,952,779, October 2005.

Manuel Cheminod graduated in Computer Engineering in 2005 at Politecnico di Torino, Torino, Italy. He is now a PhD student at Politecnico di Torino and he is also working with the Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT). His current research interests include formal verification of cryptographic protocols and formal methods applied to vulnerability and dependability analysis in distributed networks.

Ivan Cibrario Bertolotti (M'06) received his Laurea degree (summa cum laude) in computer science from the University of Turin, Italy, in 1996. Since then, he has been a researcher with the Italian National Research Council (CNR). Currently, he is with the Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT), Politecnico di Torino, Turin, Italy. His current research interests include real-time operating system design and implementation, industrial communication systems and protocols, and formal methods for cryptographic protocol analysis and verification. Dr. Cibrario also teaches several courses on real-time operating systems at Politecnico di Torino and has served as a technical referee for several international conferences and journals.

Luca Durante is Senior Researcher with the Italian National Research Council (CNR). He is currently with Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT). He graduated in Electronic Engineering in 1992, and received a PhD degree in Computer Engineering in 1996, both from the Politecnico di Torino. He has co-authored about 40 scientific journal, conference papers and technical reports in the area of industrial communication protocols and formal techniques for distributed systems. He also served as a technical referee for several international conferences and journals. Currently his research interests include formal verification of cryptographic protocols, source-level model checking of software and network vulnerability and dependability analysis.

Paolo Maggi graduated in Computer Engineering in 1998, and received a Ph.D degree in Computer Engineering in 2002, both from Politecnico di Torino, Torino, Italy. He works at Politecnico di Torino where he teaches courses on computer networks and distributed programming. His research activities have been focused on distributed systems and formal methods applied to computer security. Since 2004, he also acts as a consultant for Nice s.r.l.

Davide Pozza graduated in Computer Engineering in 2002, and received a Ph.D degree in Computer Engineering in 2006, both from Politecnico di Torino, Torino, Italy. He is currently a post doctoral researcher at the Department of Computer Engineeer-ing at that institution. His current research interests include: processes, methodologies, and techniques that address software security, reliability, and safety, static analysis techniques to detect software vulnerabilities, formal methods for modelling and analyzing network vulnerability and cryptographic protocols, and automatic code generation of cryptographic protocols by starting from their formal specifications. He teaches courses on network and distributed programming, and on secure software engineering. He also provides consultancies in the area of reliable and secure software.

Riccardo Sisto received the MS degree in electronic engineering in 1987, and the Ph.D degree in computer engineering in 1992, both from Politecnico di Torino, Torino, Italy. Since 1991 he has been working at Politecnico di Torino, in the Computer Engineering Department, first as a researcher, then as an associate professor and, since 2004, as a full professor of computer engineering. He teaches introductory courses on programming and undergraduate and graduate courses on network and distributed programming. Since the beginning of his scientific activity, his main research interests have been in the area of formal methods, applied to software engineering, communication protocol engineering, distributed systems, and computer security. On this and related topics he has authored and co-authored more than 60 scientific papers. Dr. Sisto has been a member of the Association for Computing Machinery (ACM) since 1999.

Adriano Valenzano (SM'09) is Director of Research with the Italian National Research Council (CNR). He is currently with Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT), Politecnico di Torino, Turin, Italy, where he is responsible for researches concerning distributed computer systems, local area networks, and communication protocols. Since 1983, he has been involved in many national and international research projects and has led a number of research teams in the information and communication technology area. He has co-authored about 200 journal and conference papers in the area of computer engineering and the books *Advanced Microprocessor Architectures* (Reading, MA: Addison-Wesley, 1987) and *MAP and TOP Communications: Standards and Applications* (Reading, MA: Addison-Wesley, 1992). He has served as a technical referee for several international journals and conferences, also taking part in the program committees of international conferences of primary importance. Since 2007 he has been serving as an Associate Editor for the Transactions on Industrial Informatics. His current research interests are in the areas of industrial information and communication technologies and formal methods for the verification and analysis of networks and cryptographic protocols.