Application Vulnerabilities in Risk Assessment and Management

Fabrizio Baiardi*, Federico Tonelli, and Lorenzo Isoni
Dipartimento di Informatica, Universita di Pisa
{baiardi, tonelli, isoni}@di.unipi.it

Abstract

The Haruspex suite is an integrated set of tools that adopts a scenario approach to automate ICT risk assessment and management. Each scenario includes an ICT infrastructure under attack by some intelligent attackers with some predefined goals. An attacker can reach its goals only by sequentially composing the attacks. This overcomes the infrastructure complexity and its large number of nodes. The suite applies a Monte Carlo method with multiple simulations of the attacker behavior to discover the sequences of each attacker. This simulation exploits a formal model of the target infrastructure that describes the infrastructure nodes, the vulnerabilities of the components these nodes run, and the logical topology. The multiple simulations of the Monte Carlo method support the discovering of alternative sequences. They also return a statistical sample of these sequences to compute statistics to assess and manage the risk. This paper extends the original model of the infrastructure to describe in a more accurate way how the implementation hierarchy and the interactions affect the attacks. After describing this extension, we show how it supports the modeling of web applications. In the end, we adopt the new model to assess a critical infrastructure that supervises and manages gas distribution.

Keywords: vulnerability, risk assessment, Monte Carlo method

1 Introduction

The Haruspex suite integrates tools to assess and manage the risk due to targeted attacks against an information and communication technology (ICT) infrastructure. The suite assesses ICT risk by considering a scenario with the infrastructure and some intelligent attackers. Each attacker implements targeted attacks to acquire some predefined sets of access rights or privileges. Each set is a distinct attacker goal.

Haruspex considers infrastructures with a large number of components and of nodes. This implies that an attacker can acquire all the rights in any of its goals only by sequentially implementing some attacks enabled by the component vulnerabilities. Each sequence escalates the attacker privileges in some nodes. An attacker uses the privileges it acquires through an attack to implement the following ones till it owns all the privileges in a goal. The attacker may also acquire privileges on a node by attacking a distinct one. As an example, an attacker can steal some cookies on a node to acquire privileges on a web application onto a distinct node.

The suite applies a Monte Carlo method by running an experiment that consists of a set of independent runs. Each run simulates how an attacker selects and implements its attacks. The simulation uses formal models of the target infrastructure and of the attackers. Some tools of the suite build a formal model of the target infrastructure and one model for each attacker. Further tools use these models to apply the Monte Carlo method. An experiment computes a statistical sample on the attacks the attackers have selected and the goals they have reached by collecting values in each run. The assessment uses this sample to compute statistics to evaluate and manage risk.

The infrastructure model is critical in the overall framework because an accurate description of the nodes and of the interconnection topology is a prerequisite for a realistic simulation of an attacker. In the

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 7:2 (June 2016), pp. 41-59
*Corresponding author: Dipartimento di Informatica, Università di Pisa, Lrgo B.Pontecorvo 3, 56127 Pisa, Italy, Tel: +390502212762

current version, the tools model an infrastructure as a set of components, e.g. applications, mapped onto nodes connected into a logical topology. This model does not describe in an accurate way the cascading effects of an attack even on nodes that have not been directly attacked. We extend the current model through the notion of *environment*. An *environment* is a set of components mapped onto the same node. Some environments may be related because either one supports the implementation of the other one or they interact. Relations may involve components mapped onto distinct nodes to describe those attacks against a component that grant rights on the same node or on a distinct one.

We structure the paper into seven sections. Sect. 2 briefly reviews works on vulnerabilities, attacks, and attack simulation. Sect. 3 briefly outlines the Haruspex suite. Sect. 4 describes the validation of the Haruspex suite by adopting it in the Locked Shield 2014 cyber defense exercise. Sect. 5 presents the model extensions to describe application environments. Sect. 6 applies the new model to describe web vulnerabilities to assess a critical infrastructure that supervises and controls gas distribution. Lastly, we present our conclusions.

This paper enhances the one presented at the 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP) 2016 [1]. Besides improving the presentation, a set of extensions has been made. A first extension to [1] describes how we have validated the tools that build the model of an attacker and the one of the target infrastructure by adopting the suite in a NATO Cyber Defence exercise. In this exercise we have used the Haruspex suite to compute the countermeasures to be deployed to minimize the success probability of attackers. Another extension concerns the definition of the *security stress*, a new measure we have introduced to evaluate the robustness of an ICT infrastructure. The case study described in this paper uses this new measure to evaluate the increase in the infrastructure robustness due to the countermeasures the suite selects. The last enhancement to [1] is a more in-depth discussion of the case study and of the methodology underlying the study.

2 Related Works

Related works of interest focus on the description and classification of attack sequences.

[2] discusses privilege escalation and the detection of attackers but it neglects attacker simulation. [3, 4] discuss the simulation of attacks against ICT infrastructures or critical one but neglect functional connections among applicative environments. [5] proposes a notion of look-ahead that differs from the one of Haruspex. [6, 7] analyzes intelligent and goal oriented attackers with reference to terrorism. The attack model in [8] is a similar to the proposed one it only considers information from IDS sensors without simulating the attacks.

Several papers have proposed alternative classifications of vulnerabilities and of the attacks they enable. The [9] maps each vulnerability into just one class. The attack analysis in [10] focuses on the compromised level of the target nodes. This minimizes execution overhead but it cannot discover all the attacks. [11] computes the success probability of a sequence of attacks but it neglects the alternative sequences an attacker may implement to reach the same goal.

Attack graphs and attack trees [12], [8], and [13] supports the investigation of the relations among attack sequences, attacks, and vulnerabilities. None of the proposals automatically builds an attack graphs starting from the output of a standard vulnerability scanner [14], and [15]. [16] automatically computes attack pre and post conditions through a vulnerability database but it neglects information about attack sequences.

[17] discusses the notion of dependency and relation among components of a critical infrastructure. [18] and [19] analyze cascading effects of attacks.

[20], and [21] survey agent-based simulation, whereas [22] and [23] focus on the discovery of dependencies among components of a critical infrastructure. [24] discusses the evaluation of ICT risk,

Table 1: Abbreviations used in the paper

S	the target of the assessment
c	a component of S
op	an operation that c defines
atk	an attack
v	a vulnerability
att	an attacker
g	a goal of att
$v(atk)$	the vulnerabilities enabling atk
$pre(atk)$	the rights to execute atk
$res(atk)$	the resources to execute atk
$post(atk)$	the rights granted by the success of atk
$succ(atk)$	the success probability of atk
$\lambda(att)$	the look-ahead of att

while [25], [26], and [27] review multiobjective optimization that underlies attacker selection strategies.

[28, 29] outline the tools of the Haruspex suite to describe a scenario and apply the Monte Carlo method.

3 The Haruspex suite

This section describes four tools of the Haruspex suite: the *builder*, the *descriptor*, the *engine*, and the *manager*. These tools are the kernel of the suite.

The *builder* and the *descriptor* build the formal models to describe, respectively, the target infrastructure and the attackers. These models drive the simulation of the attacker behaviors in the different runs during an *engine* experiment. Starting from the *engine* output, the *manager* selects a set of countermeasures and invokes again the *engine* to evaluate their effectiveness.

The *manager* selects a distinct set of countermeasures anytime the experiment signals that the attackers can still reach their goals in the spite of countermeasures because they can select alternative sequences. Then, it invokes again the *engine* to evaluate the new set. A finite number of iterations suffices because each iteration considers a larger number of attacker sequences. This implies that after some iterations either there are no sequences the attacker can select or no countermeasures are available for some sequences. In both cases, the *manager* cannot further decrease ICT risk.

This section also defines the *security stress*. This is a measure of how an ICT infrastructure can resist to the attacks and we use it to evaluate the effectiveness of a set of countermeasures.

For the sake of brevity, in the following, *assessment* is a shorthand for *probabilistic risk assessment* and Haruspex denotes the whole suite. Table 1 lists the abbreviations the paper uses in the following.

3.1 Describing a scenario

The suite assesses the risk with reference to a scenario that consists of the target infrastructure S and a set of attackers that attacks S. The suite models S as a set of components, e.g. applications, mapped onto a set of nodes connected into a logical topology. In turn, the model of each component considers the operations it defines, its vulnerabilities, and the attacks they enable. The topology describes the connections among the nodes and the components mapped onto each node. The model of each attacker

att includes its initial rights, e.g. the operations *att* can legally invoke, its goals, the resources *att* has available for its attacks, and the strategy *att* applies to select the attacks to implement.

3.1.1 The Builder

This tool returns the formal model of S. An assessment applies this tool only once and then it can use its output to define even alternative scenarios. Each scenario involves S and a distinct set of attackers. The availability of the *builder* strongly reduces the complexity of the scenario building and it increases both the accuracy of the assessment and the size of the infrastructure that are the target of the assessment.

The input of the *builder* is a database that merges the outputs of the vulnerability scanning of each node of S. The information in this database describes the applications running on the nodes of S and their vulnerabilities. The model also describes the logical connections among the nodes of S. An assessment can use distinct scanners to produce the database. Furthermore, it can also apply more than one scanner to the same node of S to cover distinct classes of vulnerabilities. As an example, two distinct scanners can focus, respectively, on system vulnerabilities and web ones. Haruspex can also apply static analyzers to the source code to discover further vulnerabilities [30].

The database the *builder* receives describes a vulnerability through a tuple <id, description, complexity, isExploitAvailable> where:

- *id*: a not null, unique identifier of the vulnerability, e.g. the CVE-id;
- *description*: a semi-formal description of the vulnerability;
- *complexity*: the complexity of implementing the attacks the vulnerability enables. Possible values are {*easy*, *medium*, *hard*, *insane*};
- *isExploitAvailable*: this boolean indicates if an exploit is publicly available;

The *builder* returns a model that describes the vulnerabilities of S and the *attacks* they enable. This model pairs each attack atk with some attributes [31] that describe the main attack features:

- *vulnerabilities*, $v(atk)$, the component flaws that enable atk;
- *precondition*, $pre(atk)$, the rights to execute atk;
- *resources*, $res(atk)$, the resources to implement atk;
- *success probability*, $succ(atk)$, the probability that atk is successful;
- *postcondition*, $post(atk)$, the rights that atk grants if successful;
- *execution time*, $time(atk)$, the time to implement atk.

atk is enabled if any vulnerability in $v(atk)$ is public and it succeeds with probability $succ(atk)$. This probability models that atk may fail because of reasons outside the attacker control. If at least one vulnerability in $v(atk)$ is not public, atk always fails. [31] describes in more details attack attributes.

The *builder* computes most attributes by classifying the corresponding attack into a number of predefined classes. It determines the class of atk through a pattern matching process that searches for predefined patterns in the CVE *description* of each vulnerability in $v(atk)$. The *builder* accesses further information on vulnerabilities [32–34]. [29] describes in full details the classification algorithm and its implementation.

The Haruspex user can insert further vulnerabilities and the attacks they enable into the model of S the *builder* returns. Some of these vulnerabilities model users of S as further components. The vulnerabilities of these components enable social engineering attacks [35]. The user can also specify suspected vulnerabilities and pair each of them with the probability it is discovered at a given time.

In some infrastructures, some vulnerabilities are not related to the program of the component itself. As an example, some components of a cyber physical system, e.g. sensors, may be physically attacked to manipulate the inputs they produce for other components. Attackers can sequentially compose these attacks in new and unexpected ways. As an example, after acquiring the control of a SCADA component, an attacker can open a locked door, enter a room and physically attack the sensors in the room. The Haruspex model of the target infrastructure can describe the attacks in this sequence as it can properly describe attack pre and post conditions.

3.1.2 The Descriptor

This tool receives the attributes of attackers of interest and returns the corresponding models. Haruspex models any attacker *att* as a user of S that initially owns some rights and aims to illegally reach some *goals*. Each goal g is a set of rights that *att* reaches after acquiring any of its rights through a sequence of attacks. *att* can implements an attack atk in a sequence it owns both the resources in $res(atk)$ and the rights in $pre(atk)$. Alternative sequences exist for the same goal and *att* selects the one to implement according to its priorities and current privileges. *att* causes an impact, i.e. a loss for the owner of S anytime it reaches a goal. The loss increases per each unit of time *att* owns the rights in g.

The attributes of *att* that determines the sequences of attacks it can implement are the initial rights, the goals, the resource it can access. *att* invokes its selection strategy anytime it can implement alternative sequences. This strategy ranks the set *sas* of distinct sequences *att* can implement according to the priorities of *att*. An important parameter of any selection strategy is $\lambda(att)$, the look-ahead of *att*. This integer parameter bounds the length of the sequences in *sas*. If *sas* is empty, *att* is idle till the discovery of a vulnerability enables further attacks. Otherwise, the strategy returns any sequence of *sas* that leads to a goal. If no sequence leads to a goal, the strategy ranks all the sequences in *sas* according to the attributes of the attacks in each sequence. Then, it returns the first sequence in the rank.

att implements the first attack of the sequence the strategy return. It should be noticed that this attack grants some rights that are useless to reach a goal.

Some of the selection strategies the suite supports are:

1. *maxprob*: ranks sequences according to their success probabilities;
2. *maxincr*: ranks sequences according to the set of rights they grant;
3. $maxeff$: ranks sequences according to ratio between success probability and execution time of attacks;
4. $SmartSubnetFirst$: it privileges sequences where an attack grants some rights on a node in a distinct subnet.

We introduce and describe further parameters to model *att* in the following.

3.2 Assess ICT Risk in a Scenario

The *engine* applies the Monte Carlo method and implements an experiment that consists of multiple independent runs. Each run uses the outputs of the *builder* and of the *descriptor* to simulate the attacker

behavior for the same time interval. The *engine* returns a statistical database by collecting in each run a sample that describes, among others, the goals the attackers have reached, the time they have taken to reach these goals, the attack sequences they have implemented.

Each run simulates in some details the discovery of suspected vulnerabilities, the selection and the implementation of attack sequences by each attacker in the scenario. At each time step of a run, the *engine* considers each attacker *att* in the scenario of interest that still has to reach a goal. Then, the *engine* builds the set of sequences *att* can implement according to its current rights. If this set is empty then *att* is idle. Otherwise, the *engine* applies the selection strategy of *att* and it executes the first attack *atk* of the sequence the strategy returns, if any. *att* is busy for the next $time(atk)$ steps plus the time of the selection. The *engine* determines the success of *atk* according to $succ(atk)$. If *atk* succeeds, the *engine* checks if *att* has reached a goal *g*, and updates the impact due to *att*. Otherwise, *att* repeats *atk* for a predefined number of times before invoking again its selection strategy. The number of repetitions of a failed attack is another attribute in the model of *att*.

The *builder* returns a model that include any information on *S* that the owner has available. However, to accurately describe the time it takes *att* to reach a goal, Haruspex takes into account that *att* may have no information on *S* and acquires it during its attacks. To models in some detail this activity, the *engine* assumes that *att* discovers the vulnerabilities of a node *n* through a vulnerability scanning. The scanning occurs once, the first time *att* ranks a sequence with an attack enabled by a vulnerability in a component on *n*. The time of this scan depends upon *n*. This implies that larger values of $\lambda(att)$ result both in a better visibility and in a slower selection due to a larger number of scannings. We pair any attacker *att* that models an insider with the nodes of *S* it already knows. *att* can attack these nodes without scanning them as it already knows their vulnerabilities.

When a run ends, the *engine* inserts into the output database with information on the attacker behavior in the run. Then, to guarantee run independence, the *engine* re-initializes the state of *S* and of any attacker and it starts a new run. The number of runs determines the confidence level of the statistics computed through the database because each run contributes with one sample to the database. The *engine* can fully exploit any parallelism in the underlying architecture because the runs are fully independent.

3.3 Evaluating ICT Robustness

We have defined the *security stress*, a measure of ICT robustness, i.e. of the ability of an ICT infrastructure to resist its attacker.

$Str^S_{ag,g}(t)$ the security stress of *S* at *t* due to *att* to reach *g* is the probability that *att* reaches *g* within *t*. $Str^S_{ag,g}(t)$ is monotone non decreasing in *t* and it is a probability distribution. Furthermore, $Str^S_{ag,g}(0) = 0$. We define two times to explain the proposed definition:

1. t_0 is the lowest time where $Str^S_{ag,g}(t) > 0$,
2. t_1 is the lowest time where $Str^S_{ag,g}(t) = 1$.

We assume both times exist and consider *att* as a force that tries to change the shape of *S*. This force has no effect till t_0. After this time, the shape of *S* begins to change. *S* cracks after t_1, because *att* is always successful for larger times. In the interval $t_1 - t_0$ *S* can resist, at least partially, to the attacks of *att*.

$Str^S_{ag,g}$ is a synthetic evaluation of the robustness of *S* because critical attributes of *S* determines its shape. As an example, t_0 depends upon both the times to execute attacks against *S* and the length of the shortest attack sequence to reach *g*. t_1 is a function of the success probability of attacks in the sequence leading to *g* because each of these probability determines the average number of times *att* repeats *atk* before it succeeds. Furthermore, the standard deviation of the lengths of the sequences to reach *g* influences $t_1 - t_0$. As a consequence, an evaluation of the robustness of *S* that considers $Str^S_{ag,g}(t)$

is more accurate than when considering just one value, such as the average time or the average number of attacks to reach g.

$Str^S_{ag,g}$ is the inverse of a survival function [36] because it plots the probability that *att* is successful instead that the one that S resists the attacks of *att*.

An assessment can approximate $Str^S_{ag,g}(t)$ as the percentage of runs in an *engine* experiment where *att* reaches g before t. The confidence level of the approximation of $Str^S_{ag,g}$ is the one the experiment achieves on the time *att* takes to reach g.

The definition of stress when *att* aims to achieve a set of goals *sg* considers experiment where *att* stops its attacks after reaching any goal in *sg*. Under this assumption, $Str^S_{ag,sg}(t)$ is the probability that *att* is idle after t.

The stress $Str^S_{sa,sg}(t)$ due to a set of attackers *sa* each with the goals in *sg* is equal to the one of the most dangerous attacker in *sa*. If this attacker exists, it is the one with a stress curve higher than those of other attackers.

We define the stress of a set of attackers with distinct goals as the weighted average of the stress due to each attacker in *sa*. The weight of an attacker evaluates its contribution to the overall impact. In general, this definition is of little interest because attackers with distinct goals usually result in different impacts.

3.4 Managing ICT Risk in a Scenario

The *manager* is the suite tool that drives the selection of countermeasures to deploy. This tool iteratively invokes the *engine* to implement experiments to evaluate the effectiveness of the countermeasures it selects. The *manager* assumes that some attacks have a countermeasure, i.e. a change of the infsastructure to decrease their success probability. As an example, the patching of a vulnerability results in the failure of any attack it enables while longer passwords or longer encryption keys decrease the success probability of an exhaustive attack. For the sake of simplicity, we assume that a scenario includes one attacker *att* with one goal g.

First of all, the *manager* invokes the *engine* to implement an experiment that simulates the attacks of *att*. Then, the *manager* analyzes the *engine* output to map the sequences *att* implements to achieve g and computer their success probabilities. The mapping of a sequence remove from it removes useless attacks. This increases the cost effectiveness of countermeasures the *manager* selects as it avoids those for useless attacks. The *manager* computes the success probability of a plan p as the ratio between the number of runs where *att* reaches g through a sequence mapped into p and the number of runs in the experiment.

After discovering the plans of *att*, the *manager* determines the countermeasures to reduce the corresponding risk. The *manager* works in an iterative way to reduce the success probability of *att* to a value lower than *hisu*. This is an input of the suite user that defines the largest success probability of *att* the owner of S is willing to accept. After selecting, as described in the following, some countermeasures, it updates the description of S to model their deployment. Then, it runs an experiment with the new description. The goal of the experiment is the discovery of those sequences that *att* neglects when attacking S but that it can successfully exploit against the new version of S. Each of these sequences is successful because the deployed countermeasures do not affect its attacks. We denote these sequences and the corresponding plans as $dependent$ ones because *att* selects one of these sequences only when and if some countermeasures affect those it previously selected. If the new experiment shows that dependent plans have a success probability larger than *hisu*, the *manager*selects further countermeasures to deploy, runs another experiment and so on.

The selection of countermeasure for a set of plans focuses on the attacks these plans share. This exploits that countermeasure for *atk* affects all the plans where it appears and reduces both the number of

countermeasures and their overall cost. In other words, the *manager* computes a set of countermeasures that is a coverage of *Sp* the set of plans *att* implements. A coverage of *Sp* includes one countermeasure for at least one attack in each plan in *Sp*.

4 Validating the Suite

The ability of the tools to mimic in a realistic way how the attackers select and implement their sequences strongly determines the accuracy of outputs such as the success probability of an attacker. We have validated the suite in the Locked Shields 2014 exercise as described in the following. By taking part in this exercise we had the unique opportunity of validating our models against a real team of excellent attackers

4.1 Locked Shield 2014

Locked Shields is a real-time network defense exercise, organized by the NATO Cooperative Cyber Defence Centre of Excellence, which involved participants from 17 nations. In the exercise, 12 defending teams were put against one attacking team, the red team. The exercise scenario placed the teams in a fictional country of Berylia where the industry fell under increasing cyber attacks. Day 1 started with low level hacktivist campaigns and led to espionage and sabotage attacks against the networks of the defenders by the end of day 2. In addition to technical defense, the exercise includes a number of additional tasks such as legal assignments and forensics challenge, to make the exercise as lifelike as possible. The exercise is built up as a competitive game where the defending teams are scored based on their performance. Although the defending teams competes with each other, the exercise encourages information sharing and cooperation. The teams were participating from their home countries; exercise control was located in Tallinn, Estonia.

4.2 Using the Suite in the Exercise

One of the teams involved in the exercise has adopted the Haruspex suite to analyze the infrastructure to defend, to select vulnerabilities to patch, and to schedule the deployment of these patches. The only countermeasure the defending team can apply in the exercise is the patching of vulnerabilities before the red team begins its attacks.The time to apply the patches is short. For this reason, the *manager* ranks alternative sets of countermeasures according to the time to apply the corresponding patches.

As previously said, the input of the suite is the output of the vulnerability scanning of the nodes to defend. This conflicts with a rule of the exercise that introduces unknown nodes, e.g. a defender team cannot scan some of the nodes it has to defend. In other words, the team has no information on some nodes of the target infrastructure. These nodes may even store some malware that attacks other nodes. This contradicts the basic Haruspex axioms that the suite user has complete information on the target infrastructure. We have handled this new constraint by assuming that the attacker team fully controls these nodes. This is the best situation for the attacker. The Haruspex suite considers a scenario with one attacker. This attacker aims to control the nodes more valuable to the defender. To fully exploits the information on the infrastructure the attacker acquires, we have assumed its look ahead is 3. We have applied the *manager* to the resulting scenario. This tool returns a list of the vulnerabilities to patch before the red team begins its attack. The defender team has manually applied the patches. The *manager* returns a list with about twenty patches, a small percentage of the more than one thousand vulnerabilities that affect the target infrastructure.

4.3 Results of the Exercise

The team that has applied the Haruspex suite has scored the best results as far as concerns network defense in the exercise. Obviously, this is the result of an integrated set of team abilities, such as the one to deploy the patches and to monitor the network status. Hence, we cannot attribute it to Haruspex only: however the good result in network defense confirms that Haruspex models attackers in an accurate and realistic way. As a consequence, the *manager* can decrease ICT risk in an effective way because the experiments it runs to select the countermeasures model the attackers in an accurate way. However, the overall effectiveness also depends upon the accuracy of the infrastructure model. In the following, we discuss how to improve this accuracy.

5 A More Accurate Model

This section discusses some limitations of the current model of an infrastructure. Then, it introduces a new model of S built around the idea of environments. Then, to outline some advantages of this model, it discusses the modeling of web vulnerabilities.

5.1 Current limitations

The accuracy of the model of S that the $builder$ returns strongly influences the quality of the sample that the *engine* returns. The current model cannot describe in an accurate way the postcondition of an attack and, in particular, the cascading effects of an attack. As an example, a SQL Injection attack against a Web Server *ws* also has a cascade effect on the node $n(d)$ hosting the database d that executes the query q. In other words, *att* illegally acquires rights on d through a malicious query q to *ws*. The current version of Haruspex can model the vulnerability enabling the SQL injection but it neglects the relation between the rights granted by the web server vulnerability and those on $n(d)$.

A further loss of accuracy is due to the assumption that *att* can attack from a node n_1 a distinct node $n_2$only after becoming an administrator of n_1. The next section proposes a solution to overcome most of these limitations.

5.2 Environments

As outlined in Sect.3, a Haruspex scenario models the target infrastructure as a set of interconnected nodes, each running some components. To increase the accuracy of attack simulation, we extend this model by clustering components into environments. Then, to describe more accurately attack postconditions, we introduce relations among environments. A relation exists when the rights *att* acquires on components in one environment grant further rights on components in a distinct environment. Each relation is paired with three attributes:

1. the *source* environment;
2. the *destination* environment;
3. a set of pairs. Each pair includes sets of rights on components in, respectively, the source environment and the destination one.

A relation $\langle E_s, E_d, \{\langle sr_1, dr_1\rangle, ..., \langle sr_n, dr_n\rangle\}\rangle$ denotes that each time an attacker acquires the rights in $sr_i, i \in 1..n$ on the components in E_s then the same attacker also acquires those in dr_i on those in E_d. If a component belongs to the source environment or to the destination one, then at least one right on its operations appears in a pair in the third element.

In the following, we distinguish **structural** relations from **functional** ones.

Structural relations arise because of the implementation hierarchy in the target infrastructure. As an example, there is a structural relation between a virtual machine and the applications it supports. In general, the relation exists between a layer in the implementation stack and those that are implemented through the services it offers. A structural relation denotes that the capability of invoking some operations of a layer L also grants some privileges on the operations the components implement through the services that L offers. In general, the source environment of a structural relation only includes the component that implements the services the destination environment invokes.

Both a structural relation from E_1 to E_2 and one in the reverse direction may exist. The reverse relation models a vulnerability in a component in E_2 that enables an attack granting some rights on components in E_1. In general, this attack manipulates some parameters of an invocation to a component in E_2. The attack is successful if the invoked component transmits the parameters to E_1 without a control.

Functional relations arise because of interactions among the components in the source environment and in the destination one. Even in this case a component accesses the services that another one offers but now the two components belongs to the same implementation layer. In general, both the source environment and the destination one only include one component. As an example, there is a functional relation among two environments onto distinct nodes anytime a success attack against the components running on a node grants some rights on components that interact with the attacked one. This happens even where the two sets of components run on distinct nodes. As shown in the following, after successfully attacking a web server, an attacker can transmit dangerous queries to a database server on another node.

Two environments cannot be involved simultaneously in a functional relation and in a structural one.

Fig. 1 shows the model of an infrastructure where the control of P_1 grants to *att* some rights on C_1, C_2, and C_3. The model includes three *structural* relations with the same source, P_1, and where the destinations are, respectively, C_1, C_2, and C_3. Fig. 1 also shows two *functional* relations. The first one is between C_1 and C_2, and the other one between C_3 and C_4. The first relation connects two environments structurally related to the same environment. Instead, each of C_4 and C_5 is structurally related to a distinct environment.

One environment may appear in distinct *structural* or *functional* relations. We introduce distinct relations among environments to specify in a more refined way, for each node of S, the proper dependencies among the rights *att* can acquire. Furthermore, we can describe the domino or cascade effects of an attack [18] due to implementation hierarchy or to interactions among components.

Fig. 1 shows the new model. It is worth noticing that both *structural* and *functional* relations can be freely composed. Consider an example where the destination of a *structural* relation is a component that executes a virtual machine. This virtual machine can produce further *structural* relations among the environments related to the applications it runs.

To discover structural relations, Haruspex pairs each component with the corresponding implementation layers of the target infrastructure. Instead, functional relations depend upon the flow of information among the components in the same or in distinct nodes.

To exemplify *structural* and *functional* relations, consider two nodes of an infrastructure that runs five applications. To model the nodes and the applications, we introduce seven environments:

1. P_1 and P_2, two instances of an operating system;
2. C_1, C_2, C_3, C_4, and C_5, five components that models the applications that either P_1 or P_2 support.

Fig. 1 shows five structural relations between environments in the same node and a functional one between environments in distinct nodes. In particular, P_2, C_4, and C_5, in Fig. 1, model one node where P_2 acts as the operating system. There are two *structural* relations to P_2 from C_4 and C_5, two environments

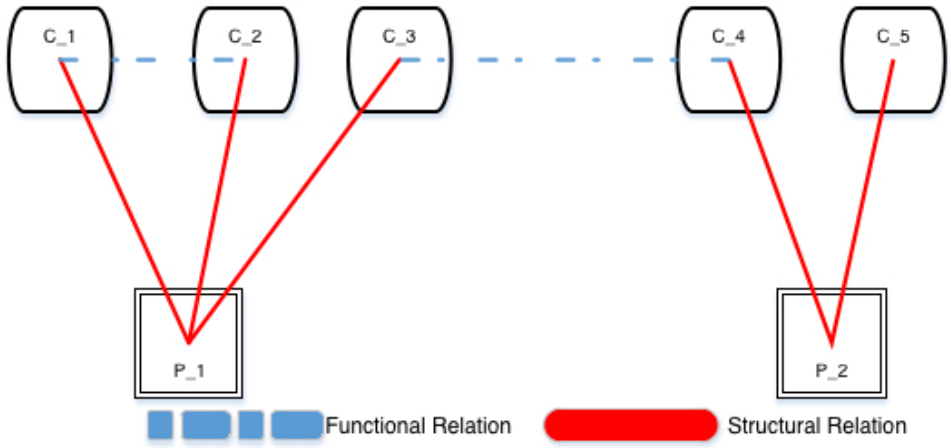

Figure 1: Relations among components

that model, respectively, a database and a FTP server. These relations are *structural* because C_4 and C_5 model applications that run on P_2 and *att* acquires distinct sets of rights on these applications when controlling P_2. In particular, by controlling P_2, *att* also acquires some privileges on the FTP server in C_4, and it may transmit information to C_3, an application environment on another node. In this example, further relations may exist from C_1 and C_2 to P_1 because, as an example, if *att* acquires some rights on any of these applications it may also execute some codes on P_1 or manipulate some information in its file system.

5.3 Generating Environment Relations

The Haruspex suite maps the components that a scanning returns into a set of predefined environments interconnected through some standard structural relations. The suite defines no functional relation. The suite user can freely specify the relations of interest for an assessment.

First of all, we consider a node of S running an operating system, OS, and some applications. The applications belong to the following environments:

1. OSE is an environment that only includes OS and that is the source of a distinct relation for each of the following environments;
2. E_1 that includes any application the os fully controls;
3. E_2 that includes any application the os partially controls but that is not the source of a relation having OSE as its destination;
4. B_1, ..., B_n where each includes one application that is the source of a relation having OSE as its destination.

An application A belongs to the first environment if an OS administrator fully control A while no rights on A grants a right on OS. As an example, this class includes a DBMS where data is not encrypted because an attacker that control OS can read and update information in the database. The control of OS only grants some rights on the database if its data is encrypted. As an example, an attacker that controls OS can erase some files but cannot steal or update any information. Here, A belongs to E_2 if no privilege on A grants the attacker some rights on OS. Otherwise, A belongs to an environment that only includes A itself and that is the source of a relation targeting OS.

The Haruspex suite maps into a predefined environment each application recognized by the vulnerability scanning of the nodes of S. Then, it produces the proper set of pairs of rights for each relation. As an example, the suite maps any DBMS into the E_1 environment but the suite user can change this mapping and define a new relation involving the DBMS.

Anytime the scanning detects a node of S that runs a hypervisor that manages some virtual machines Haruspex introduces one environment for each of these machines and a distinct one for the hypervisor. Then, it defines all the pairs to describe how some rights on the hypervisor result in distinct rights on a virtual machine. According to the particular hypervisor that the scanning detects, we may introduce further relations among the hypervisor and the *OS* and the applications running on each virtual machine.

5.4 Modeling Web Environments

To show the modeling of web vulnerabilities, we consider the two nodes in Fig. 2 where n_1 runs Windows XP and executes an Apache Tomcat, while n_2 runs a MySQL Server on top of Windows. Haruspex models this infrastructure through the following environments:

- two **os-environments**, ose_1 and ose_2, each including a XP instance;
- a **web-environment**, wbe_1;
- a **database-environment**, dbe_1.

5.4.1 Environments and Relations

ose_1 and ose_2 model the two instances of Windows XPs running on, respectively, n_1 and n_2. The components in each environment include all the vulnerabilities that affect the corresponding operating system.

wbe_1, executed by ose_1, models the web server application, i.e. the Apache Tomcat. It collects the components with all the vulnerabilities affecting the web application, the java-servlet, the related website, and the web-users.

dbe_1, executed by ose_2, models the database application, i.e. the MySQL Server. It collects all the vulnerabilities related to the service provided by the database server and the users of the database.

These environments define the following relations, see Sect. 5.2:

1. S_1: a *structural* relation from ose_1 to wbe_1;
2. S_2: a *structural* relation from wbe_1 to ose_1;
3. S_3: a *structural* relation from ose_2 to dbe_1;
4. S_4: a *structural* relation from dbe_1 to ose_2;
5. F_1: a *functional* relation from wbe_1 to dbe_1.

The component pairs of S_1 denotes that when *att* controls ose_1 then it also owns the rights of reading, writing and denying the service of wbe_1. S_2 denotes that some vulnerabilities in wbe_1 grant to *att* administration rights on ose_1. S_3 denotes that, by controlling ose_2, *att* acquires the rights of writing and denying the service on dbe_1. S_4 models that any attack of *att* that exploits some vulnerabilities in dbe_1 also returns the right of executing code on ose_2. Lastly, the *functional* relation between wbe_1 and dbe_1 models that if *att* can transmit SQL Queries to wbe_1, then it can also read and write information on dbe_1.

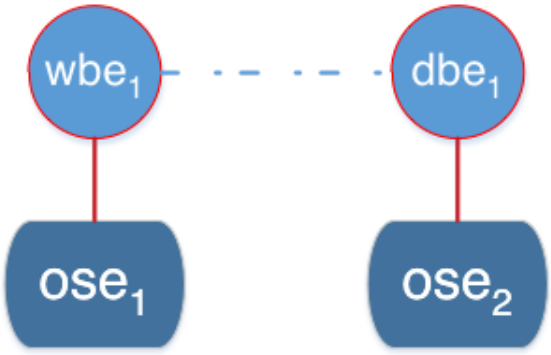

Figure 2: A model of Application Environments to Describe Web Vulnerabilities

5.4.2 Attack Modeling

S_2 can model attacks such as buffer overflow and path traversal. In particular, the former enables *att* both to execute arbitrary code from wbe_1 and to acquire administration rights on ose_1. Instead, path traversal enables *att* to access files and directories in wbe_1. *att* can also gains the rights to write and read in the file-system ose_1.

F_1 is relation to describe a SQL injection attack [37]. *att* exploits the lack of controls on the input of wbe_1 to acquire both read and write accesses on dbe_1 that belongs to the destination environment of the relation.

Lastly, S_4 supports the modeling of an advanced SQL injection attack [38] because the grant of submitting a query to dbe_1 enables *att* to write and execute code on ose_2.

6 Case Study

We applied Haruspex to assess an ICT infrastructure that acts as the control system of an infrastructure for gas distribution and smart metering.

6.1 Structure of the ICT

Fig.3 shows the target infrastructure of the assessment. It consists of 3 main parts: $Net1$, $Net2$, $Net3$. Each subnet is flat as any two of its nodes can interact. A VPN-Connection exists between $Net1$ and $Net2$. Only $Net1$ has internet access. $Net3$ runs on a cloud architecture. A direct VPN-Connection exists between $Net1$ and $Net3$. Two firewalls separate $Net1$ and $Net2$. A further firewall connects $Net1$ and $Net3$. All these firewalls also act as routers.

$Net1$ includes four subnets:

- DMZ Server: it includes 5 web servers. Two of them host a public website;
- DMZ VideoConf: it includes 4 nodes that run video conferencing software;
- LAN VOIP: it includes 7 nodes to support VOIP communications;
- LAN: it includes 61 nodes. Its main components are development PCs and databases.

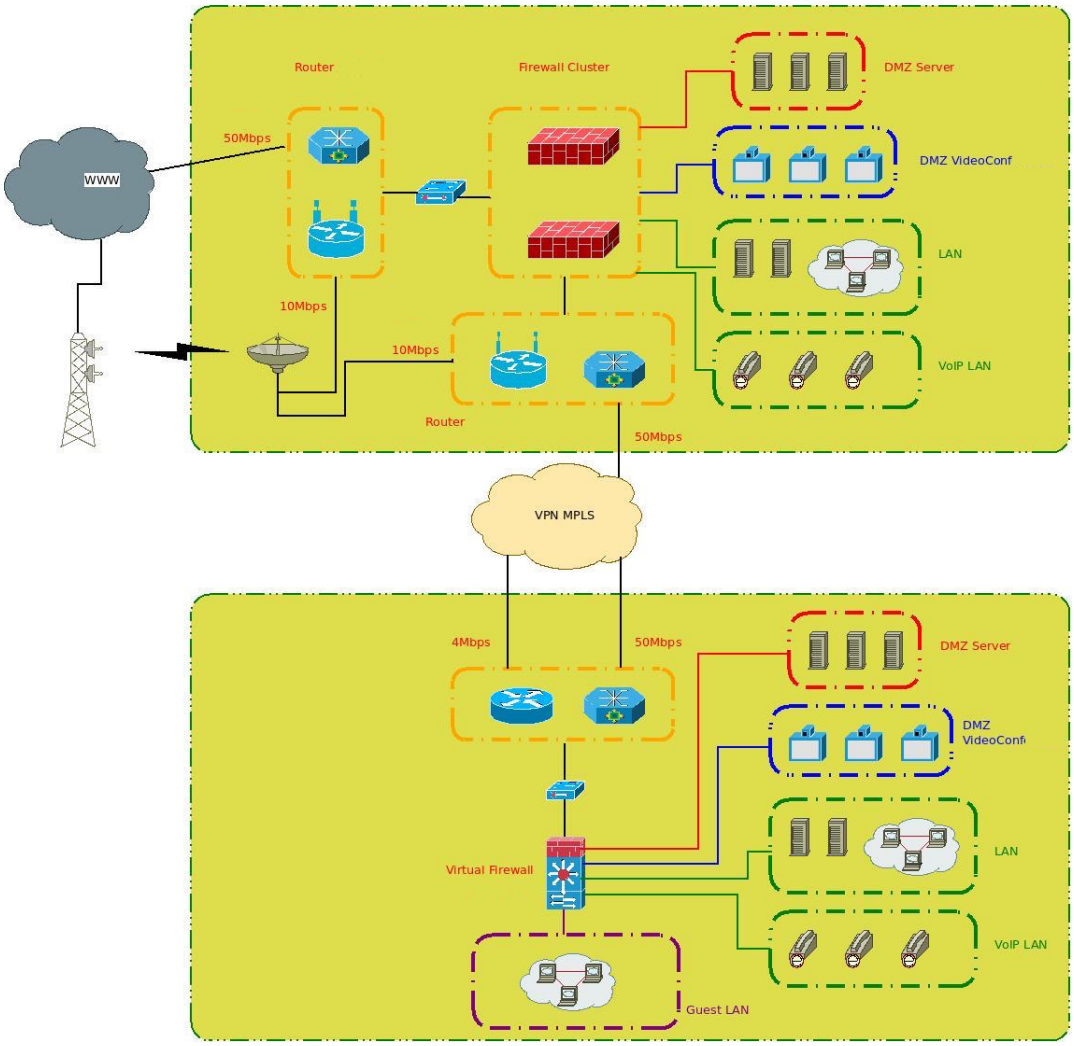

Figure 3: The topology of the gas distribution plant

*Net*2 and *Net*1 have the same structure but *Net*2 includes a larger number of nodes. Instead, *Net*3 consists of a single subnet with 15 nodes. The main component running on this network is the Meter Data Reader (MDR), a central data acquisition system that manages and validates data on gas consumption in two databases, DB_1 and DB_2. *Net*3 also runs a website that customers of the gas company access to check their bills the amount of gas they have consumed. Operators can access all users data.

6.2 The Attackers in the Scenario

The attackers we consider model customers that initially can access the nodes that offer Internet services, e.g. website located in the DMZ Server and in *Net*3 subnets. Each attacker has one of three alternative goals:

1. the control of *MDR*;
2. the control of DB_1 and DB_2;
3. the control of *MDR*, DB_1, and DB_2.

Table 2: Results of *att* simulations with the support of web-environment

class of attacker	*number of simulations*	*n. of sims using web vuln.*	*percentage of web attacks*
1	300000	18982	6%
2	300000	61849	21%
3	300000	137897	46%

The owner has no information about the attackers and in particular, about their selection strategy. We cover this lack of information by considering six selection strategies: *maxprob*, *maxincr* with λ equals to 1 and 2, $maxeff$, and $SmartSubnetFirst$. Hence, a scenario includes 18 attackers, one for each possible combination of goal and strategy. Then, we consider the most powerful attacker in the scenario. This is the attacker that reaches its goal in the shortest time. The *manager* considers this attacker when selecting countermeasures to deploy. The ability of handle uncertainty on the attackers by introducing a set of attackers to discover the most dangerous one is an important advantage of the model based approach of the Haruspex suite. Each experiment consists of 50000 runs. This results in a confidence level larger than 99% on the success probability of each attacker and a level larger than 95% on the sequences that an attacker selects.

6.3 Differences between old and new Haruspex model

We compare the output produced by the old version of Haruspex with the new one we propose in this paper.

The input of the $builder$ is the output of the Nessus Vulnerability Scanner [39] applied to whole infrastructure nodes and the output of Owasp ZAP [40] applied to each website.

The previous model describes the infrastructure as a set of 155 nodes. The new scenario model and the implementation of *web-environment*, described in Sect. 5 and in Sect. 5.4, introduce 9 further components. These components include the web servers that belong to a *web-environment* and the database that belongs to a *database-environment*. The Table 2 shows the results of our new model as attack sequences that concern web vulnerabilities.

The new model enables Haruspex to consider new attack sequences *att* may implement to its goal. As an example, *att* may acquire the rights to read on DB_1 or DB_2 by exploiting a cross-site scripting vulnerability to steal the session of an operator. Another attack exploits a SQL Injection vulnerability. This attack grants *att* write privileges on some databases, even if *att* owns no administrative rights on the node running the database server. This shows how relations can model attacks with cascading effects that enable an attacker to control some components on a node even without attacking this node. This is a noticeable difference with respect to the previous model because this model can only describe infrastructures where an attacker can acquire the control of node only through an attack that targets a component running on the node.

Table 2 shows the number of runs where attackers exploit web vulnerabilities to reach a goal. The percentage refers to the overall number of attack sequences for all the attackers with the same goal. This table shows that nearly half of these sequences exploit web vulnerabilities. Hence, no model of S can neglect these vulnerabilities because they appear in most attacker sequences.

Fig. 4 and 5 show the stress curves due to some of the attackers. In particular, The figure 4 depicts the difference between insider and outsider attackers, the second one represents a set of attackers that aims to reach the control of MDR, DB_1, and DB_2.

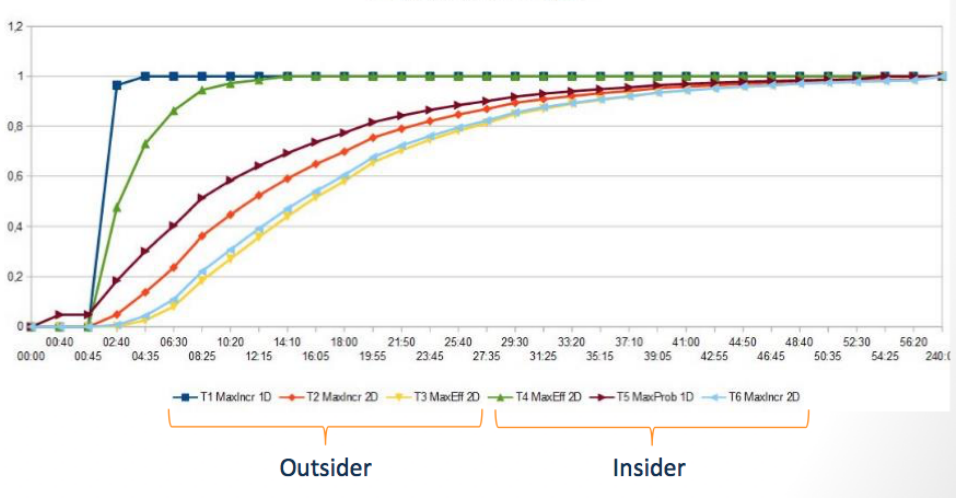

Figure 4: Stress curves according to the time

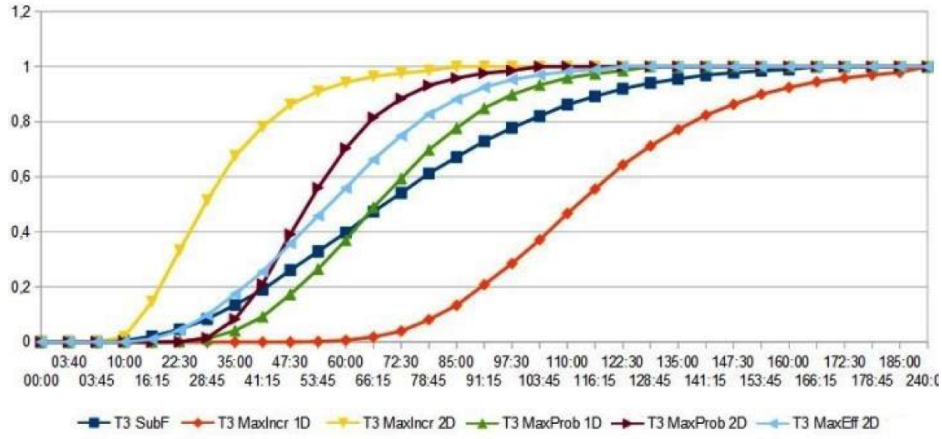

Figure 5: Stress curves according to the time

6.4 Countermeasures

The *manager* computes a set with 34 countermeasures to deploy in 9 iterations. Also in this case, the number of countermeasures is a small percentage of the overall number of vulnerabilities, 1700. Some countermeasures focus on the certificates of the host involved in a secure communication. Other ones avoid weak or unsupported encryption algorithms. The countermeasures for applicative vulnerabilities concern the invocations to predefined functions to validate some inputs and the deployment of a web firewall. Among others, these countermeasures stop both cross-site scripting and SQL injection against the company website.

The number of hosts involved in the countermeasures is very low.

An alternative countermeasure the *manager* proposes is a distinct segmentation of the overall network. The *manager* only suggests this countermeasure because of its large cost.

7 Conclusion

This paper has introduced the suite Haruspex. The tools in the suite support the automation of assessment and management of ICT risk in complex infrastructures. The tools use a model based approach. We have discussed how the adoption of the suite in a NATO cyber defence exercise has validated both the model of the infrastructure and those of the attacker. We have also defined an original measure of robustness that fully exploit the output of the suite tools

We have extended the infrastructure model to describe in more details the cascading effects of attacks. In the new model, each node runs a set of environments interconnected by some relations. Each environment includes a set of components and the relations among these environments describe either the implementation hierarchy or the interactions among components. Lastly, we have adopted the new model to assess and manage the risk of an ICT infrastructure to control gas distribution and metering. This exploits the model ability of describing web vulnerabilities and the resulting attacks to web applications. A comparison of the results returned by the old model against those of the new one confirms that an assessment that adopts the proposed extensions can achieve more detailed and accurate results. This increases the effectiveness of the countermeasures the suite returns to manage the risk.

References

[1] F. Baiardi, F. Tonelli, and L. Isoni, "An extension of haruspex to cover vulnerabilities in application environments," in *Proc. of the 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'16), Heraklion, Crete, Greece*. IEEE, February 2016, pp. 574–580.

[2] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Network Security*, vol. 2011, no. 8, pp. 16–19, August 2011.

[3] I. Kotenko, "Active vulnerability assessment of computer networks by simulation of complex remote attacks," in *Poc. of the 2003 International Conference on Computer Networks and Mobile Computing (ICCNMC'03), Shanghai, China*. IEEE, October 2003, pp. 40–47.

[4] D. Helbing and S. Balietti, "How to do Agent Based Simulations in the Future," Santa Fe Institue, Tech. Rep. 11-06-024, June 2011.

[5] E. LeMay, W. Unkenholz, D. Parks, C. Muehrcke, K. Keefe, and W. Sanders, "Model-based Security Metrics using ADversary VIew Security Evaluation (ADVISE)," in *Proc. of the 8th International Conference on Quantitative Evaluation of SysTems (QEST 2011), Aachen, Germany*. IEEE, September 2011, pp. 191–200.

[6] D. Rios Insua, J. Rios, and D. Banks, "Adversarial risk analysis," *Journal of the American Statistical Association*, vol. 104, no. 486, pp. 841–854, 2009.

[7] D. M. Buede, S. Mahoney, B. Ezell, and J. Lathrop, "Using plural modeling for predicting decisions made by adaptive adversaries," *Reliability Engineering & System Safety*, vol. 108, pp. 77–89, December 2012.

[8] S. Cheung, U. Lindqvist, and M. Fong, "Modeling multistep cyber attacks for scenario recognition," in *Proc. of the 2003 DARPA Information Survivability Conference and Exposition (DISCEX'03), Washington, DC, USA*, vol. 1. IEEE, April 2003, pp. 284–292.

[9] S. Engle, S. Whalen, D. Howard, and M. Bishop, "Tree approach to vulnerability classification," May 2005, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.6848&rep=rep1&type=pdf [Online; Accessed on June 10, 2016].

[10] P. Ammann, J. Pamula, R. Ritchey, and J. Street, "A host-based approach to network attack chaining analysis," in *Proc. of the 21st Annual Computer Security Applications Conference (ACSAC'05), Tucson, AZ, USA*. IEEE, December 2005.

[11] J. D. Howard, "An analysis of security incidents on the internet 1989 - 1995," Ph.D. dissertation, Carnegie Mellon University, April 1997.

[12] F. Cuppens, F. Autrel, A. Miege, and S. Benferhat, "Correlation in an intrusion detection process," in *Proc. of the 2002 Workshop on Security of Communications on the Internet (SECI'02), Tunis, Tunisia*, September 2002, pp. 153–172.

[13] S. Wang, Z. Zhang, and Y. Kadobayashi, "Exploring attack graph for cost-benefit security hardening: A probabilistic approach," *Computers and Security*, vol. 32, pp. 158–169, 2013.

[14] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proc. of the 2002 IEEE Symposium on Security and Privacy (S&P'02), The Claremont Resort Oakland, CA, USA*. IEEE, May 2002, pp. 273–284.

[15] N. Ghosh and S. K. Ghosh, "A planner-based approach to generate and analyze minimal attack graph," *Applied Intelligence*, vol. 36, no. 2, pp. 369–390, March 2012.

[16] K. Kaynar and F. Sivrikaya, "Distributed attack graph generation," *IEEE Transactions on Dependable and Secure Computing*, vol. online publication, doi: 10.1109/TDSC.2015.2423682, 2015.

[17] G. Stergiopoulos, "Securing critical infrastructures at software and interdependency levels," Ph.D. dissertation, Athens University of Economics & Business, November 2015.

[18] L. Franchina, M. Carbonelli, L. Gratta, M. Crisci, and D. Perucchini, "An impact-based approach for the analysis of cascading effects in critical infrastructures," *International Journal of Critical Infrastructures*, vol. 7, no. 1, pp. 73–90, 2011.

[19] M. Ouyang, "Review on modeling and simulation of interdependent critical infrastructure systems," *Reliability engineering & System safety*, vol. 121, pp. 43–60, 2014.

[20] C. M. Macal and M. North, "Tutorial on agent-based modelling and simulation," *Journal of Simulation*, vol. 4, no. 3, pp. 151–162, 2010.

[21] A. Rob, "A Survey of Agent Based Modelling and Simulation Tools," Science and Technology Facilities Council, Tech. Rep. DL-TR-2010-07, 2010.

[22] A. Ghorbani, E. Bagheri, Onut, R. Zafarani, H. Baghi, and G. Noye, "Agent-based Interdependencies Modeling and Simulation (AIMS)," Intelligent and Adaptive Systems Research Group, Faculty of Computer Science, UNB, Tech. Rep. IAS-TR01-06, September 2006.

[23] E. Casalicchio, E. Galli, and S. Tucci, "Federated Agent-based Modeling and Simulation Approach to Study Interdependencies in IT Critical Infrastructures," in *Proc. of the 11th IEEE Int. Symp. on Distributed Simulation and Real-Time Applications (DS-RT'07), Chania, Greece*. IEEE, October 2007, pp. 182–189.

[24] A. Arora, D. Hall, C. Piato, D. Ramsey, and R. Telang, "Measuring the risk-based value of it security solutions," *IT Professional*, vol. 6, no. 6, pp. 35–42, 2004.

[25] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[26] K. Deb and H. Gupta, "Introducing robustness in multi-objective optimization," *Evolutionary Computation*, vol. 14, no. 4, pp. 463–494, 2006.

[27] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, pp. 369–395, 2004.

[28] F. Baiardi, F. Corò, F. Tonelli, and L. Guidi, "Qsec: Supporting security decisions on an it infrastructure," in *The 8th CRITIS Conference on Critical Information Infrastructures Security (CRITIS'13), Revised Selected Papers, Amsterdam, The Netherlands*, ser. Lecture Notes in Computer Science, vol. 8328. Springer International Publishing, September 2013, pp. 108–119.

[29] F. Baiardi and F. Corò and F. Tonelli and L. Guidi, "Gvscan: Scanning networks for global vulnerabilities," in *Proc. of the 8th International Conference on Availability, Reliability and Security (ARES'13), Regensburg, Germany*. IEEE, September 2013, pp. 670–677.

[30] J. Zhu, B. Chu, H. Lipford, and T. Thomas, "Mitigating access control vulnerabilities through interactive static analysis," in *Proc. of the 20th ACM Symposium on Access Control Models and Technologies (SACMAT'15), Vienna, Austria*. ACM, June 2015, pp. 199–209.

[31] F. Baiardi and D. Sgandurra, "Assessing ict risk through a monte carlo method," *Environment Systems and Decisions*, vol. 33, no. 4, pp. 486–499, 2013.

[32] NIST, "National Vulnerability Database," http://nvd.nist.gov/ [Online; Accessed on June 10, 2016].

[33] MITRE, "CWE - common weakness enumeration," https://cwe.mitre.org/ [Online; Accessed on June 10, 2016].

[34] K. Scarfone and P. Mell, "An analysis of CVSS version 2 vulnerability scoring," in *Proc. of the 3rd Interna-*

tional Symposium on Empirical Software Engineering and Measurement (ESEM'09), Lake Buena Vista, FL, USA. IEEE, October 2009, pp. 516–525.

[35] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer, "Social phishing," *Communications of the ACM*, vol. 50, no. 10, pp. 94–100, October 2007.

[36] A. La Corte and M. Scatà, "Failure analysis and threats statistic to assess risk and security strategy in a communication system," in *Proc. of the 6th International Conference on Systems and Networks Communications (ICSNC'11), Barcelona, Spain*, October 2011, pp. 149–154.

[37] W. Halfond, J. Viegas, and A. Orso, "A classification of SQL-injection attacks and countermeasures," in *Proc. of the 2006 IEEE International Symposium on Secure Software Engineering (ISSSE'06), Washington, DC, USA*. IEEE, March 2006, pp. 65–81.

[38] B. Damele and A. Guimaraes, "Advanced SQL injection to operating system full control," in *Black Hat Europe 2009 Briefings, Amsterdam, The Netherlands*, April 2009. [Online]. Available: https://www.blackhat.com/presentations/bh-europe-09/Guimaraes/Blackhat-europe-09-Damele-SQLInjection-whitepaper.pdf

[39] R. Rogers, *Nessus network auditing, 2 edition*. Syngress, 2008.

[40] S. Bennetts, "OWASP Zed Attack Proxy Project," https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project [Online; Accessed on June 10, 2016].

Author Biography

Fabrizio Baiardi graduated in Computer Science at Università di Pisa where is a Full Professor with Dipartimento di Informatica where he has chaired the degree on ICT security. His main research interests are formal approaches to risk assessment and management of complex ICT infrastructures. Fabrizio Baiardi has been involved in the risk assessment and management of several systems and of industrial control systems with SCADA components. He has authored several papers on ICT security and currently teaches several university courses on security.

Federico Tonelli graduated cum laude in Information Security at Università di Pisa. He is currently, Ph.D candidate at Dipartimento di Informatica, Università di Pisa. His research interest are focused on tools to assess and manage ICT risk with a particular interest on complex ICT infrastructures and industrial control systems with SCADA components. He has authored several papers on ICT security.

Lorenzo Isoni graduated in Computer Science at Università di Pisa. He holds a scholarship funded by Terranova Software on security of smart meters. He is currently involved in the design and evaluation of tools for ICT security assessments.