Assessing and Managing the Information and Communication Risk of Power Generation

F.Baiardi[a], F.Tonelli[a], L.Guidi[b], D.Pestonesi[b], V.Angeletti[b]

[a] *Dipartimento di Informatica, Università di Pisa, Italy*
[b] *ENEL Ingegneria e Ricerca SpA, Pisa, Italy*

Abstract We describe a model-based assessment of information and communication technology (ICT) risk that produces statistical samples by simulating the attacks of intelligent agents. To support this assessment, we have developed an integrated set of tools, the Haruspex suite. Some of its tools build the models of the target system and those of the agents that other tools apply to simulate the agent attacks. Further tools analyze the output of the simulation. After outlining the proposed approach and the suite, we describe the assessments of two industrial control systems that supervise, respectively, a thermoelectric generation plan and a hydroelectric one. To simplify the presentation of the output of these assessments, we introduce the security stress, a synthetic measure of how a system resists to attacks.

Keywords: Risk Assessment; Metric; Intelligent Agent; Model Based Assessment; Monte Carlo Method, Industrial Control System.

1. Introduction

Intelligent agents are among the most dangerous threats of information and communication technology, ICT, systems because they escalate their privileges, i.e. access rights, through a sequence of attacks that uses the rights that an attack grants to execute the following ones.

To assess the risk due to these agents we propose a model based approach that collects statistical samples by applying a Monte Carlo method to a scenario where some agents attack the target system of the assessment. The method uses abstract models of an ICT system and of the agents in multiple step-by-step simulations of how each agent selects and executes its attacks. These simulations return a sample to compute the statistics to assess and manage the risk. Our approach does not need to collect historical data because it is model based.

Email addresses: baiardi@di.unipi.it (F.Baiardi), tonelli@di.unipi.it (F.Tonelli), luca.guidi@enel.com (L.Guidi), daniela.pestonesi@enel.com (D.Pestonesi), valentino.angeletti@enel.com (V.Angeletti)

The Haruspex suite is an integrated set of tools to support the proposed approach. These tools build the simulation models, apply the Monte Carlo method, and analyze the samples it returns. We describe how the suite assess two critical ICT systems, each acting as an Industrial Control System, ICS, of a power generation plan. Each assessment determines the probability that some attackers acquire the control of the plan and proposes cost effective countermeasures.

This paper is structured as follows. Sect. 2 briefly reviews works on security metrics, vulnerabilities, and attack simulation. Sect. 3 describes the Haruspex tools to build the models and simulate the agent attacks. After discussing the selection of countermeasures, Sect. 4, introduces synthetic measures to simplify an assessment. Lastly, it discusses the validation of the suite. Both sections refer to the same running example to simplify the suite description. Sect. 5 describes the adoption of the suite to assess and manage the ICT risk of two industrial control systems, ICSs, that supervise and manage, respectively, a hydroelectric power generation plant and a thermoelectric one. After briefly resuming the lesson learned in these assessments, we draw some conclusions.

This work integrates the results outlined in [1, 2] and presents them systematically. Furthermore, it applies the suite tools to assess and manage the ICT risk of two ICSs, each supervising a distinct power generation plant. Each assessment considers a scenario where attackers aim to control power generation by acquiring the proper access rights on some ICS components. Each assessment models both insider and external attackers. Lastly, the paper introduces the security stress, a synthetic measure to simplify the communication of the results of an assessment.

2. Related Works

This section reviews previous works on the description and the simulation of attacks against ICT system as well as on metrics of ICT robustness. We also review some works on the impact of ICT risk on power generation and smartgrids. While a large number of works has addressed attack simulation, ICT risk assessment and management, just a few works propose an integrated approach to these issues. This integrated approach is the main original contribution of our work on the Haruspex suite.

[3, 4, 5] analyze the simulation of attacks against ICT systems. [6] discuss intelligent, goal oriented agents with reference to terrorism. [7] describes attack pre conditions and pairs an attack with the proper countermeasure. [5] models agents with partial information. These papers do not exploit attack simulation to produce data to assess a system. Furthermore, most tools to analyze privilege escalation do not discover attack sequences. The taxonomy in [8] introduces a classification of vulnerabilities. [9, 10] discuss the modeling and the selection of countermeasures through attack graphs. [11] considers goal oriented attackers.

[12, 13] review security metrics. [14, 15, 16] propose metrics of the robustness of an ICT system under attack by intelligent agents but they do not integrate these metrics with alternative attacks. The metric in [17] focuses on zero-day

Table 1: List of Abbreviations

S	the target system
c	a component of S
ag	a threat agent
g	a goal of an agent
at	an elementary attack
v	a vulnerability
$v(at)$	the vulnerabilities enabling at
$pre(at)$	the set of rights an agent needs to implement at
$res(at)$	the resources to execute at
$post(at)$	the set of rights at grants if it succeeds
$succ(at)$	the success probability of at
$time(at)$	the execution time of at
$\lambda(ag)$	the look-ahead of ag

vulnerabilities. The one in [18] is similar to security stress as it considers the amount of work to attack a system. [19] computes the probability that an agent reaches a goal but it neglects alternative attacks for a goal.

[20, 21, 22] discuss the role and the assessment of ICT risk in power generation and smart grids.

3. The Haruspex Suite: Running Experiments

for the sake of brevity, we use *risk assessment* as a synonymous of *probabilistic risk assessment* while *right* and *privilege* are shorthands for *access right*.

The Haruspex suite supports risk assessment and management of an ICT system with reference a scenario where it is the target of some agents that aim to reach their predefined goals. Some tools of the suite build the models of the target system and of the agents. Other tools use these models to implement independent simulations of the agent attacks. These simulations return a statistical sample with information on, among other, the attacks the agents have executed, the goal they have reached and the time this takes. The resulting approach supports a security-by-design strategy to assess an ICT system during its design and before its deployment.

This section briefly describes the *builder* and the *descriptor*, the tools to build the models of, respectively, the system and an agent. Then, it introduces the *engine*, a tool that applies the Monte Carlo method and simulate the agent attacks. The following section describes the tools of the suite that analyze the samples the *engine* returns.

Table 1 defines some abbreviations and the main parameters of the two models. In the following, we use the system in Fig. 1a) as a running example to describe the tools and the information they return.

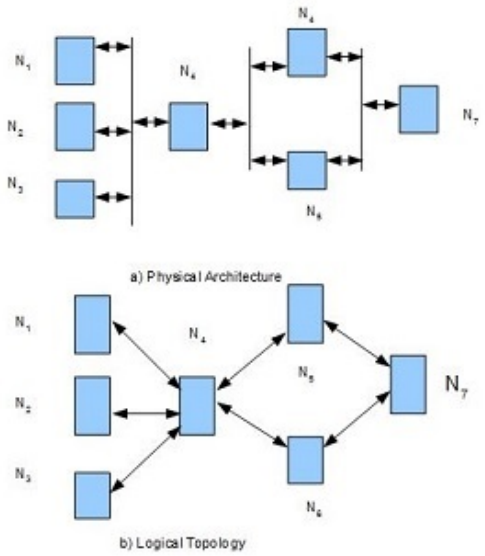

Figure 1: Running Example

3.1. Modeling an ICT System

The *builder* model decomposes S into components, i.e. hardware/software modules, that define some operations that the users of S and/or the other components invoke. The *security policy* of S defines the operations each user is entitled to invoke.

The component *vulnerabilities* enable some *attacks* [23, 24]. Haruspex supports both *effective* vulnerabilities, those already known, and *potential* ones, those that the user suspects. Haruspex pairs each potential vulnerability with the probability the attackers discover it at a given time. To model social engineering attacks, vulnerabilities may affect even the users of S [25].

The model of S neglects the actions of at and pairs it with the parameters in Table 1. at is *enabled* if any vulnerability in $v(at)$ is effective and it succeeds with a probability $succ(at)$, otherwise it fails.

The *builder* receives the effective vulnerabilities of S through the output of a vulnerability scanning of each node of S [26]. The assessment can extend the vulnerabilities the scanning return by specifying potential vulnerabilities and/or those of the users.

As an example, to assess the system in Fig. 1 first of all we scan each of its seven nodes to discover its vulnerability. Then, for each node, the assessment can insert further, suspected, vulnerabilities.

The *builder* maps each vulnerability v into the attacks it enables. To this

purpose, it classifies v by matching some predefined patterns against the description of v in the Common Vulnerability Enumeration, CVE, database [23], a de facto standard [1]. The class of v determines the attributes of each attack at it enables such as $succ(at)$ and $time(at)$. The *builder* signals any vulnerability it cannot classify. Let us suppose that a vulnerability of a component c of a node n enables an attack at that is a buffer overflow [27]. Here, $pre(at)$ includes the right of invoking the proper function of c. $post(at)$ depends upon the role of c. As an example, if c is a component of the OS of n, then $post(at)$ may include any right of the administrator of n.

The builder also discovers if at is a remote attack that ag can launch against n from another node. For this reason, one input of the *builder* is the logical topology of S and the components that control it, e.g. firewalls. The *builder* assumes that only a node administrator can launch a remote attack. ag exploits remote attacks to acquire some rights on a node n where it is not entitled to any access right. If the logical topology prevents any interaction between n and any node that ag controls, first of all ag has to implement some attacks to become the administrator of a node that can interact with n. From this node, ag may implement remote attacks against n. Assume that Fig. 1 b) shows the logical topology of the system in Fig. 1 a) and that a vulnerability in N_2 enables a remote attack against this node. Even after becoming an administrator of N_1, ag cannot implement this remote attack because N_1 cannot directly interact with N_2. Given the logical topology, only the administrator of N_4 can launch a remote attack against N_2. If ag only owns some rights on a component on N_1 then it should at first attack and become an admistrator of N_4 to launch from this node a remote attack against N_2.

3.2. Modeling Agents

An agent ag is a user of S that legally owns some rights on some components of S and aims to illegally reach one or more *goals*, each a set of rights. No generality is lost by assuming that ag is a user of S because the security policy of S can grant no right to ag. ag reaches a goal after acquiring all its rights and this results in an *impact*, i.e. a loss the owner of S pays per each unit of time ag owns the rights in g. While the security policy forbids ag to acquire any right in g, there is an impact only if, and when, ag reaches g.

For each agent ag, the user specifies to the *descriptor* the initial rights, the resource it can access, and the goals. A further parameter we describe in the following is the ranking strategy of ag and its parameters.

ag can implement at only if it owns the resources in $res(at)$. Furthermore, ag also needs the rights in $pre(at)$ that it may have acquired through previous attacks. Being intelligent, ag selects a sequence of attacks that minimizes the time to reach a goal or maximizes the probability of reaching it. We model this selection as a *ranking strategy*, or *strategy*, that depends upon the goals and the preferences of ag, its current rights, the attacks it can implement, and the information on S it has available or it is willing to acquire. ag sequentially executes the attacks in the sequence the strategy returns and it invokes again

the strategy after $ns(ag)$ attacks. ag waits for the discovery of a potential vulnerability when its strategy cannot return a sequence.

$\lambda(ag)$, the look ahead of ag, is a parameter of ag that determines the longest sequence the ranking strategy of ag considers. If some sequences with, at most, $\lambda(ag)$attacks lead to a goal, then the strategy returns one of these sequences according to the preferences of ag and the attributes of attacks in each sequence. The strategy selects the sequence to execute according to attack attributes only anytime $\lambda(ag)$ is too low to discover sequences that leads to a goal. Here, the strategy may even return a sequence with useless attacks. at is useless if rights in $post(at)$ are useless to reach a goal. If $\lambda(ag) = 0$ then ag can only adopt the *random* strategy that considers any attack ag can implement given its current rights and that increases these rights. The strategy returns with the same probability any attack that satisfies both conditions, even if it is not enabled. Among the other strategies, we recall those used in Sect. 5:

1. $maxProb$: returns the sequence with the largest success probability,
2. $maxIncr$: returns the sequence granting the largest set of rights,
3. $maxEff$: returns the sequence with the largest ratio between success probability and execution time of attacks.

With reference to our running example, suppose that ag can interact with some components on N_1 and aims to acquire the right of reading a component on N_7. If $\lambda(ag) = 0$ then ag randomly selects one attack against N_1 and repeats it till the attack succeeds. Then, ag selects an attack against N_4. When controlling N_4, ag can launch a remote attack against any of N_2, N_3, N_5, and N_6. There is no guarantee ag attacks a component on N_5 or one on N_6, the nodes that can interact with N_7. If $\lambda(ag) = 1$, then ag selects one attack according to its success probability or the rights it grants. If $\lambda(ag) = 2$, ag ranks sequences with, at most, two attacks. As an example, a sequence may attack N_4 and then N_5 or N_2. Again, the strategy may favor the joint success probability of the two attacks or the rights they grant. Lastly, if $\lambda(ag) = 3$, ag discovers the sequence to attack the component on N_7 and it can avoid useless attacks. In the example, larger values of $\lambda(ag)$ are meaningful only if ag needs more than one attack to become the administrator of an intermediate node in a path.

The time to reach a goal also includes the one to acquire the information to select a sequence. To model this time, we assume that ag runs a vulnerability scanning of n the first time its strategy ranks a sequence with an attack enabled by a vulnerability of a component executed by n. The scanning takes a time depending upon n. To model insiders, we pair ag with the nodes it does not scan as it already knows their vulnerabilities. ag scans a node only once because the scanning returns any vulnerability in the node components. As an example, if $\lambda(ag) = 2$ then ag scans N_1, N_2, N_3, N_4, N_5, and N_6 before selecting its first sequence. If, instead, $\lambda(ag) = 1$, then ag will initially scan N_1. It will scan N_4 only after successfully attacking N_1. Furthermore, it scans N_2, N_3, N_5, and N_6 after successfully attacking N_4 only. Hence, larger values of $\lambda(ag)$ increase the accuracy of the strategy and avoid useless attacks at the expense of a larger number of scannings before each selection.

3.3. Simulation Engine

The inputs of the *engine* include the model of the system and those of the agents in a scenario. The *engine* applies the Monte Carlo method to implement an experiment with independent *runs* that simulate, for the same time interval, the agent attacks and the discovery of potential vulnerabilities. In each run, the *engine* collects the samples to return.

In a time step of a run, the *engine* considers any idle agent that still has to reach a goal and it applies the ranking strategy of ag. If the strategy cannot return a sequence, then ag is busy for the ranking time only. Otherwise, the *engine* simulates the first $ns(ag)$ attacks of the sequence and ag is busy for the ranking time plus the sum of the times of these attacks. ag retries a failed attack for $nr(ag)$ times and then it invokes again its ranking strategy. $nr(ag)$ is a further attribute of ag. If ag executes the whole sequence and reaches a goal, the *engine* updates the corresponding impact.

At the end of a run, the *engine* inserts in its output database a sample with the sequence of each agent, the goals it has reached and the corresponding time. Then, it initializes the state of S and those of the agents and starts a new run.

An assessment uses the *engine* output database to compute statistics of interest. The confidence level of these statistics depends upon the number of runs because the *engine* collects one sample for each run. An experiment ends either after executing the specified number of runs or when a predefined statistic reaches the required confidence level.

It is worth noticing that no tool computes in advance alternative attack sequences of an agent because this result in an intolerable complexity. The suite discovers the sequences an agent executes by simulating its behavior in the runs of an experiment.

4. Analyzing the Output of Experiments

This section describes the selection of a cost effective set of countermeasures and then introduces measures to evaluate system robustness. Lastly, it discusses the validation of the overall suite.

4.1. Selecting Countermeasures

We describe the *planner* and the *manager*, the tools that cooperate to select countermeasures.

4.1.1. Discovering the Agent Plans

The *planner* analyzes the *engine* output database to remove useless attacks from the sequences ag has executed to reach g. As previously discussed, ag may select useless attacks because of a low value of $\lambda(ag)$. In the running example, if $\lambda(ag) = 1$ then ag may reach its goal through a sequence that attacks N_1, N_4, N_3, N_6 and, lastly, N_7. Obviously, any attack against N_3 is useless. By removing useless attack, we increase cost effectiveness as we only select countermeasures for attacks that contribute to reach g. In the following, we denote as a *plan* any

sequence without useless attacks. The *planner* maps each sequence s to reach g into a plan $p(s, g)$ through a backward scanning of s. The scanning inserts into $p(s, g)$ any attack of s that grants rights that belong neither to g nor to the post condition of the current attacks in $p(s, g)$. This algorithm is correct if ag only executes attacks that increase its rights and s does not interleave distinct plans. To solve the latter problem, we also map any permutation of s that is a sequence, e.g. its first $j - 1$ attacks grants a set of rights that includes the precondition of the $j - th$ one.

To take into account that distinct sequences may implement the same plan, the *planner* computes the success probability of a plan as the percentage of runs that have successfully implemented it.

4.1.2. Iterative Selection Countermeasures for a Set of Plans

In the following, we assume that any attack at has a countermeasure with a finite, known cost that decreases $succ(at)$. As an example, the patching of a vulnerability in $vuln(at)$ results in the failure of at, while we decrease the probability of discovering passwords or encryption keys by increasing their lengths. We discuss attacks with no countermeasures at the end of the section. For the sake of simplicity, we also assume that a scenario includes one agent ag with one goal g. Extensions to more general scenarios are straightforward.

The *manager* receives both the *engine* output database and $lowrisk$, the highest success probability of ag the assessment accepts, and it runs a number of iterations. In each iteration:

1. it applies the *planner* to the output database to discover the agent plans,
2. it selects some plans and determines one countermeasure for each plan,
3. it updates the model of S to mimic the deployment of countermeasures,
4. it runs a new experiment.

The new experiment discovers how ag reacts to the deployed countermeasures, i.e. if ag can select other sequences to replace those affected by the countermeasures. We denote these sequences as dependent ones and their discovery requires a new experiment because agents implement them only after deploying some countermeasures. The *manager* starts a new iteration as long as the risk due to dependent sequences is larger than $lowrisk$.

With reference to the system in Fig. 1, suppose that ag reaches its goal through sequences that attack N_5 while it neglects attacks against N_6 due to their low success probability. However, if attacks against N_5 fail because of some countermeasures, ag can select the attacks against N_6. The *manager* runs a new iteration if the resulting risk is larger than $lowrisk$.

The countermeasures the *manager* returns are strongly related to the plans it considers at each iteration. In the global approach, the $i - th$ iteration selects countermeasures for all the plans of ag previously discovered, independently of the iteration that discovers a plan. Hence, an iteration may select countermeasures that differ from those previously selected. Instead, in the incremental approach, each iteration extends the countermeasures previously selected with those for the plans discovered in the current experiment.

To compare the two approaches, we recall that the selection of countermeasures for some plans favors the attacks they share to minimize the number of countermeasures. Obviously the global approach exploits shared attacks at best. Instead the incremental one cannot anticipate which attacks the current plans share with the dependent ones the following iterations will discover.

The current version of the *manager* adopts a global approach where the $i-th$ iteration considers all the plans previously considered and a subset, Cp_i, of those discovered in the current iteration. We insert plans into Cp_i starting from those with the largest success probability and stop as soon as the overall success probability of the remaining plans is lower than $lowrisk$. This strategy reduces the computational overhead but it neglects that countermeasures can change the success probability of a plan. The user can bound the size of Cp_i as a fixed percentage of the plans discovered in the $i-th$ iteration.

To select countermeasures, the *manager* maps each attack at in the plans in Sp into $Sp(at) = \{i_1, ..., i_k\}$, the indexes of the plans that share at. A set of countermeasures affects all the plans in Sp if it affects all the attacks in a set Sa that covers Sp, i.e. if all the plans in Sp belong to $\bigcup(Sp(at))\forall at \in Sa$. The coverage problem is NP-Complete and the *manager* analyzes any coverage to determine the optimal one. To reduce the execution time, we abort the building of a coverage as soon as its cost exceeds the current optimum. Furthermore, the *manager* removes redundant attacks from the plans. at_1 is redundant if all the plans that share at_1 also share a distinct attack at_2 with a cheaper countermeasure. We have experimentally verified that, given the number of plans and the attacks they share, the *manager* execution time is acceptable provided that $Sp(at)$ includes at most 100 plans. If agents have available a larger number of plans, then S requires a new, more robust design rather than a more efficient selection of countermeasures.

The *manager* pairs attacks with no countermeasure with a countermeasure with an infinite cost. If the *manager* returns a coverage with an infinite cost, at least one plan only includes attacks with no countermeasure. The success probability of this plan is a lower bound on the success probability of ag.

4.2. Security Measures

We briefly describe some measures to synthesize the analysis of the samples returned by an experiment that adopts the Monte Carlo method.

4.2.1. Metrics based upon Countermeasures

A first measure considers the number, or the cost, of countermeasures to reduce the risk to a predefined, user defined, value. Besides simplifying the evaluation of ICT robustness, this metrics also supply information on the weakest components of the target system. These components are those affected by the vulnerabilities targeted by the deployed countermeasures.

4.2.2. Security Stress

We define $Str^S_{ag,g}(t)$, the security stress at t due to ag that aims to achieve g, as the cumulative probability distribution that ag reaches g within t.

$Str^S_{ag,g}(t)$ is a synthetic measure of the robustness of S that is monotone non decreasing in t and $Str^S_{ag,g}(0) = 0$. To explain its definition, we denote by t_0 the lowest time where $Str^S_{ag,g}(t)$ is larger than zero. If t_0 does not exist, $Str^S_{ag,g}(t)$ is of no interest because ag cannot successfully attack S. Furthermore, let t_1 be the lowest time, if it exists, where $Str^S_{ag,g}(t)$ is 1. If we see the attacks of ag as a force trying to change the shape of S, then this force is ineffective till t_0 when the shape of S begins to change. S cracks after t_1, the ultimate time, because ag is always successful if $t \geq t_1$. If both t_0 and t_1 exist, $t_1 - t_0$ evaluates how long S, partially, resists to the force of ag to achieve g. $Str^S_{ag,g}(t)$ is the inverse of a survival function [28] as it plots the success probability of ag as a function of t instead than the one that S survives ag attacks.

$Str^S_{ag,g}(t)$ is a synthetic and accurate evaluation of the robustness of S because several attributes of S and of ag contribute to determine the value of this function. As an example, distinct selections strategies of ag result in distinct stress values due to distinct numbers of useless attacks. Other attributes that influence $Str^S_{ag,g}(t)$ includes the length of attack sequences and the success probabilities of attacks.

To generalize $Str^S_{ag,g}(t)$ to a set of goal Sg, we assume that ag stops its attacks after reaching any of the goals in Sg. Under this assumption, $Str^S_{ag,Sg}(t)$ is the probability that ag is idle after t. To define $Str^S_{Sag,g}$ where Sag is a set of agents, we define the most dangerous agent in Sag as the one that results in the largest stress value at any time.

Starting from the *engine* output database, we approximate $Str^S_{ag,g}(t)$ as the percentage of runs that reach g before t. The confidence level of the approximation depends upon the one of the experiment.

A further synthetic measure of robustness is $AttS^S_{ag,g}(n)$, the attack stress. Its definition is similar to the one of security stress but it considers the number of attacks that an agent as available to reach a goal and that includes both successful and failed attacks. $AttS^S_{ag,g}(n)$ evaluates in a more detailed way the amount of work of ag to reach g.

4.3. Validation of the Suite

We have designed and implemented consistency checks to validate the correctness of the suite. However, these checks cannot guarantee that the simulations mimic in a realistic way the behavior of the agents. We have tackled this problem in two steps. The first one has assessed and managed the risk of an ICT system consisting of a set of virtual components with a large number of vulnerabilities. Then, we have produced an improved system that adopts the countermeasures selected by the *manager*. When white hat attackers have attacked both systems, they have not been able to reach their goals in the improved system. We believe this confirms, at least partially, that our tools simulate in a realistic way intelligent attackers.

5. Risk Assessment and Management of Two ICSs

This section describes two assessments that have adopted the Haruspex suite. The targets of the assessments are two ICSs that supervise, respectively, a thermoelectric power generation plant and a hydroelectric one. The two assessments show how the availability of data resulting from the simulation of attacks strongly simplifies ICT risk assessment and management to achieve a better level of assurance on the risk due to the target system. This characterizes the Haruspex approach with respect to methodologies that cannot access data on possible attacks against the target system.

5.1. Overall Structure of the Assessments

We have discovered the vulnerabilities in both ICSs through a Nessus vulnerability scan [29]. We have developed some dedicated plug in modules to scans the programmable logical components, PLCs, in both ICSs. PLCs interface the ICS with the industrial plant it has to control. We assume the assessment has no information on the attackers. As a consequence, any experiment covers any combinations of the agent parameters. As an example, there is a distinct agent for each value in the Cartesian product of initial rights, goals and ranking strategies. This does not imply that there is a distinct attackers for each value in the Cartesian product but only that an experiment analyzes each combination to discover the most dangerous agent. An assessment can consider a lower number of agents, e.g. neglect some ranking strategy, provided that further information is available. We neglect values of λ larger than 2 after experimentally verifying they are not effective. Each experiment consists of 60.000 runs. This results in a 95% confidence level on the components an agent attacks to reach a goal. The time limit of each run is three days or 72 hours because both ICSs can detect the attacks after this limit. To simplify the presentation, for each assessment we only discuss the security and the attack stresses due to the agents of interest. In both assessments, a countermeasure for an attack patches one of its enabling vulnerabilities.

5.2. Thermoelectric ICS: Risk Assessment and Management

After briefly describing this ICS, we discuss its assessment and the management of the corresponding risk.

5.2.1. Structure of the ICS

This experimental ICS consists of three subnets: the *intranet* network, the *process* network and the *control* one. Each subnet is flat as any two of its nodes can interact. A switch and some firewalls define the perimeters of each subnet and filter communications to/from subnets. This ICS adopts a defense in depth strategy where only the nodes in the *process* network can connect to those in the *control* one. For the same reason, *intranet*nodes can only interact with those in the *process* one. The six nodes in the *intranet* network interface the nodes in the generation plant to nodes in the *control* network. The main nodes are a Windows Domain Server and two VPN Clients that remotely access the

process network. The 17 nodes in the *process* network run SCADA servers and clients that supervise and control power generation. Through these nodes, the operators can control the whole production plant. Some nodes are redundant for safety reasons. Lastly, the 7 nodes in the *control* network simulate the power generation plant through hydraulic circuits and two PLCs. Any agent aims to control the PLCs to control of a subset of the plant.

More than 2700 vulnerabilities affect the ICS nodes and they enable 1900 attacks. We have manually verified that agent can implement more than 700 attack sequences. The Windows Domain Server is the *intranet* node with the largest number of vulnerabilities, 61. The *process* network node with the largest number of local vulnerabilities, 634, is the ASC server. Finally, the PLCs are the *control* network nodes with the largest number of vulnerabilities, 10.

The *builder* has not classified three vulnerabilities only. We have manually verified the correctness of the classification.

5.2.2. Agents in the Scenario

As previously discussed, the agents in each experiment cover four ranking strategies and, where appropriate, two λ values, 1 and 2. We define now the other parameters that characterize an agent: the goal and the initial rights. An $o-agent$ aims to control at least one PLC while a $b-agent$ aims to control both PLCs. Furthermore, initially an *insider* agent controls a *process* network node, while an *external* attacker controls an *intranet* node. To cover all the combinations, each experiment simulates the attack of 28 distinct agents.

5.2.3. Output of the Assessment

The most dangerous *insider*, $o-agents$ adopt the $MaxProb$ and $MaxIncr$ strategies with $\lambda = 2$. After 1 hour and 50 minutes, they reach their goal with a probability equal to 0.75. The probability is 1 after 3 hours. The attack stress is 0.75 after 3 attacks and it is 1 after 8 attacks. The least dangerous $o-agent$ adopts the *random* strategy. The security stress positive after 7 hours and 45 minutes and it reaches 1 after more than 20hours. The attack stress is positive after 12 attacks and becomes 1 after 36 attacks.

The most dangerous $o-agent$ adopts $MaxIncr$ with $\lambda = 2$. This agent reaches a 0.7 success probability after 1 hour and 50 minutes and it is always successful after 4 hours. The agent reaches these probability values after, respectively, 3 and 8 attacks. The agent that adopts the $MaxProb$ strategy with $\lambda = 2$ results in a lower security stress but similar to this one. The agents with the worst performances adopt, respectively, the $MaxIncr$ strategy with $\lambda = 1$ and the *random* one that have a non-zero success probability after, respectively, 12 hours and 45 minutes and 4 hours. Both agents are always successful after 24 hours. The attack stress of a $MaxIncr$, $\lambda = 1$ agent is positive after 21 attacks and it reaches 1 after 41 attacks. The two corresponding values for the *random* agent are 5 and 41.

The most dangerous *insider* $b-agent$ adopts the *maxIncr* strategy with $\lambda = 2$. It controls both PLCs after 2 hours and 25 minutes or 2 attacks with a 0.65 probability and it is always successful after 5 hours and 30 minutes or

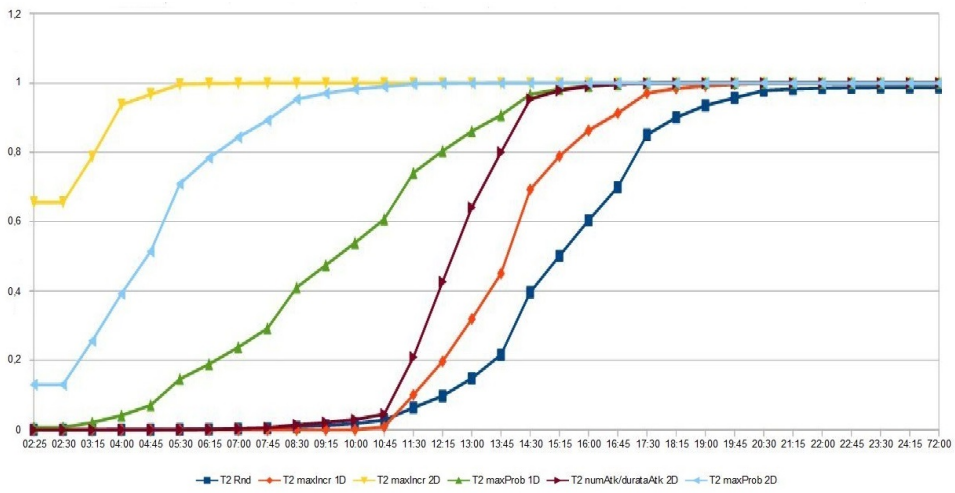

Figure 2: Thermoelectric ICS: Security Stress of Insider $b-agents$

9 attacks. The worst strategy is the *random* one and an insider $b-agent$ that adopts it reaches its goal after more than 23 hours and 30 attacks. With reference to the $maxIncr$ strategy, an increase of λ from 1 to 2 reduces the time to a goal of more than 60% and the number of attacks of about 75%.

The most dangerous external $b-agent$ adopts the $maxIncr$ strategy with $\lambda = 2$. The agent control both PLCs after 2 hours and 25 minutes or 3 attacks with a 0.65 probability and it is always successful after 6 hours and 30 minutes or 16 attacks. Even for external $b-agent$, the worst ranking strategy is the random one that results in a 300% increase in the number of attacks and in a similar increase in the time to a goal. If the agent adopts the $maxIncr$ strategy, an increase of λ from 1 to 2 halves the number of attacks and results in a 60% reduction in the time to a goal.

Fig. 2-3-4-5 show some stress curves computed through these experiments.

5.2.4. Countermeasures

In just one iteration, the *manager* computes a set with 10 countermeasures that affect 309 plans and patch any vulnerability in the nodes that connect the *process* network and *control* one. By patching less than 1% of the ICS vulnerabilities, we stop all the plans.

5.3. Hydroelectric ICS: Risk Assessment and Management

5.3.1. Structure of the ICS

This ICS, see Fig. 6, consists of seven subnets: the *intranet* network, the *central* network, the *antivirus* network, the *DMZ* network, the *PLC* network, and two *process* networks, PCN_1 and PCN_2. Three firewalls, fw_m, fw_{pt} and fw_{agp} route and filter the communications among the networks.

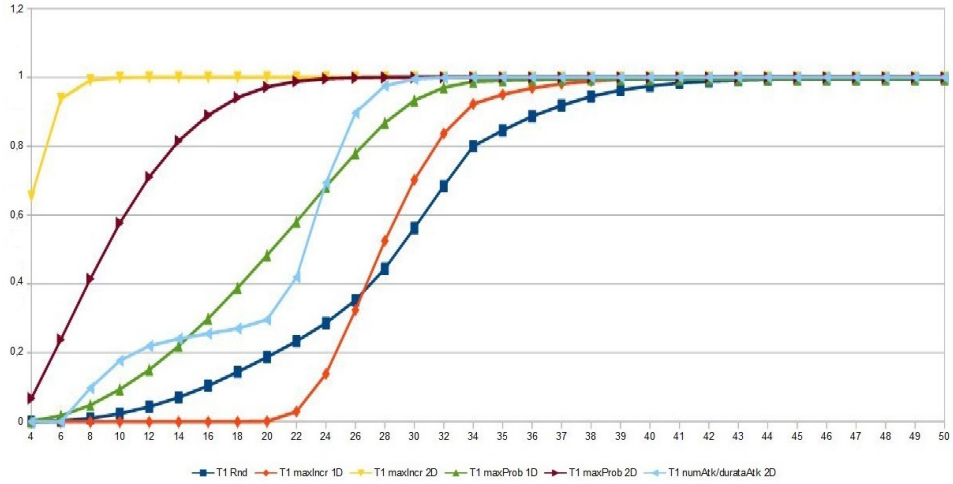

Figure 3: Thermoelectric ICS: Attack Stress of External $b - agents$

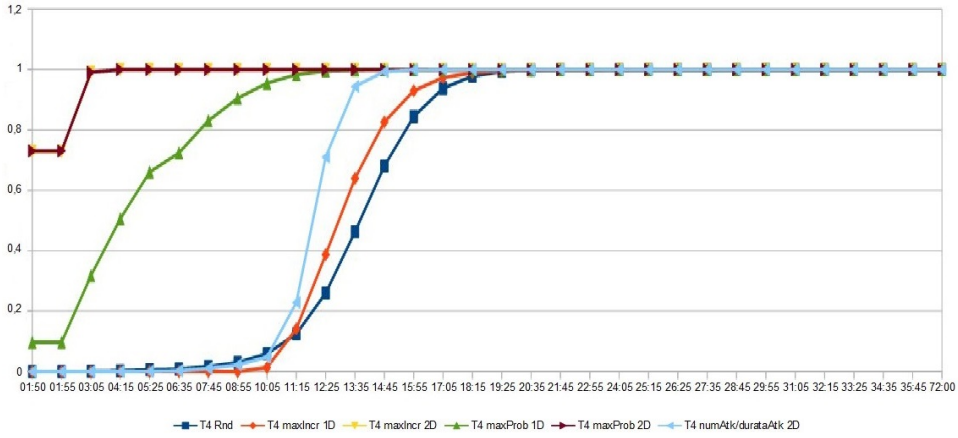

Figure 4: Thermoelectric ICS: Security Stress of Insider $o - agents$

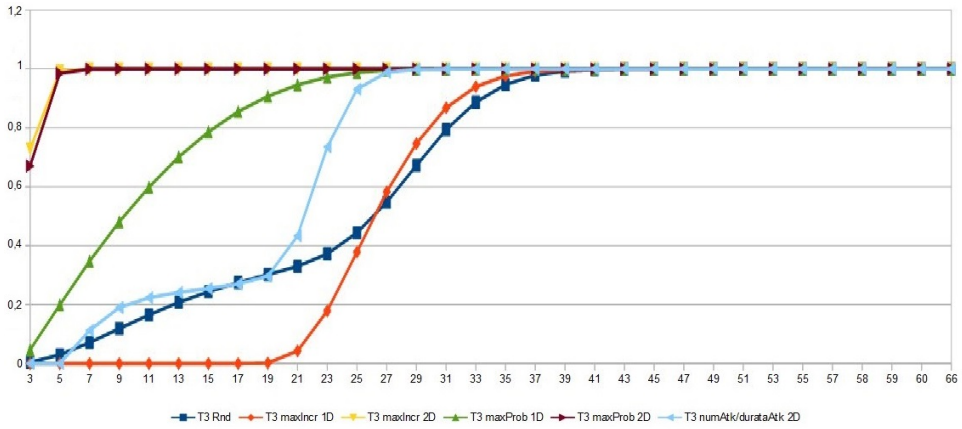

Figure 5: Thermoelectric ICS: Attack Stress of External $o - agents$

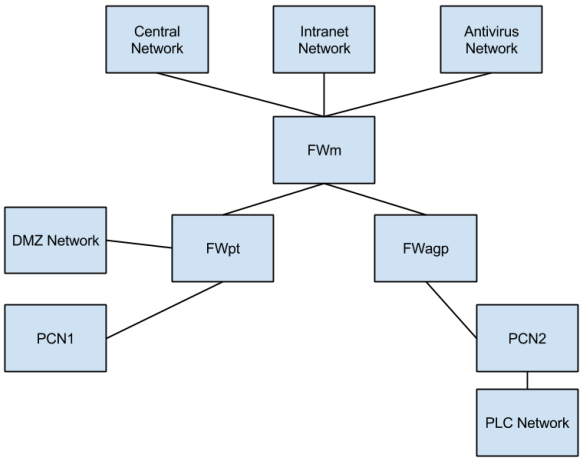

Figure 6: Hydroelectric ICS architecture

fw_m interconnects the *intranet*, the *central* and the *antivirus* networks and route messages among these networks. The *intranet* acts as a bridge that securely connects the SCADA operators and the *process* network. The *central* network has 12 nodes, one of them runs 5 virtual machines that we consider as further network nodes. Operators manage power generation through SCADA clients and workstations in this network. The *antivirus* network consists of one server to update the virus definitions in the overall ICS.

fw_{pt} interfaces the DMZ and the PCN_1 networks. The DMZ network consists of two Web Servers that collect statistical information on the processes involving the SCADA components. The PCN_1 network consists of 5 servers, one

is a Domain Controller and two are SCADA servers to manage power generation.

fw_{agp} interfaces the PCN_2 network that includes a SCADA Server with two network interfaces connected to, respectively, the PCN_2 network and the PLC network. The latter includes some PLCs that manage hydroelectric power production.

The vulnerabilities returned by the scanning of all the nodes, including the PLCs, enable 764 attacks.

5.3.2. Agents in the Scenario

Also in this assessment, the agents in an experiment cover all possible combinations of agent parameters. However, in this ICS any two PLCs are equivalent and all the agents aim to control any PLC. Hence, any experiment considers 14 agents only. Any external attacker initially controls a node in the *intranet* network. Instead, each insider initially control a node in the PCN_1 network.

5.3.3. Security and Attack Stress

Even in this ICS, the most dangerous *insider* adopts the $maxIncr$ ranking strategy with $\lambda = 2$. Its security stress is positive after 2 hours and 30 minutes to reach its goal and it is 1 after 11 hours and 15 minutes. The worst ranking strategy is $maxIncr$ with $\lambda = 1$. The security stress of this agent is 0.8 after 72 hours. The attack stress reaches the same value after 99 attacks.

The most dangerous *external* agent adopts a $maxIncr$, $\lambda = 2$ strategy and its stress is 1 after at least 12 hours or 23 attacks. The least dangerous external agent adopts $maxIncr$ and $\lambda = 1$. Its stress is 0.8 after 72 hours or 99 attacks.

It is worth noticing that this is the only assessment we have run till now where the *random* strategy is not the worst one. It is also the first one where an agent stress does not reach 1 in a run.

Fig. 7-8 show some stress curves for this ICS.

5.3.4. Countermeasures

The agents execute 9 distinct plans that we can stop by patching the same vulnerability in each PLC component. If we neglect this, trivial, countermeasure, the *manager* determines in two iterations a set of 7 vulnerabilities to patch. The first iteration discovers 7 plans and returns a set with 7 vulnerabilities to patch. The second iteration discovers 2 further, dependent, plans and it selects a distinct set that also includes 7 vulnerabilities shared among the 9 plans. The third experiment does not discover further dependent plans. This shows how a global approach minimizes the number of countermeasure because the second iteration can select distinct countermeasures for the same plans. Instead, in an incremental approach each iteration can only extend the countermeasures previously selected.

5.4. Lessons Learned

The differences between the two ICSs strongly influence the complexities of the two assessments. In fact, one ICS is an experimental, built and developed

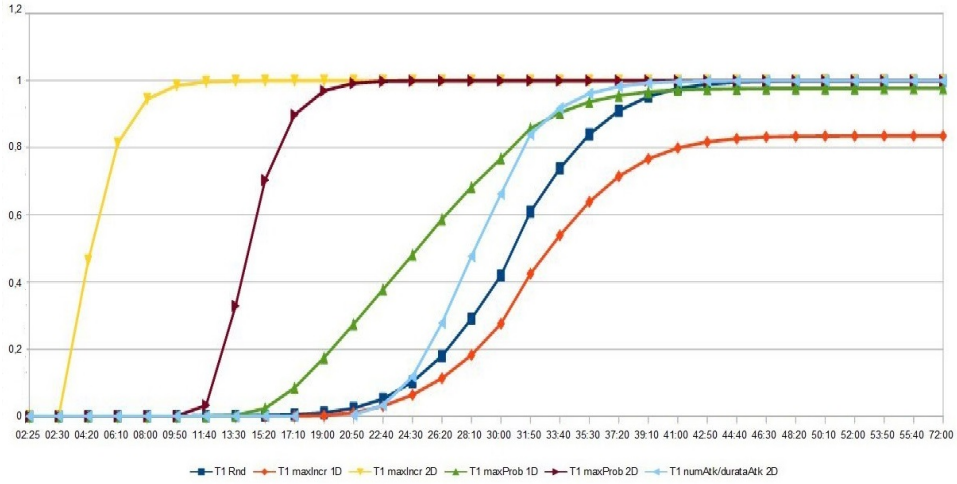

Figure 7: Hydroelectric ICS: Security Stress of an Insider Agent

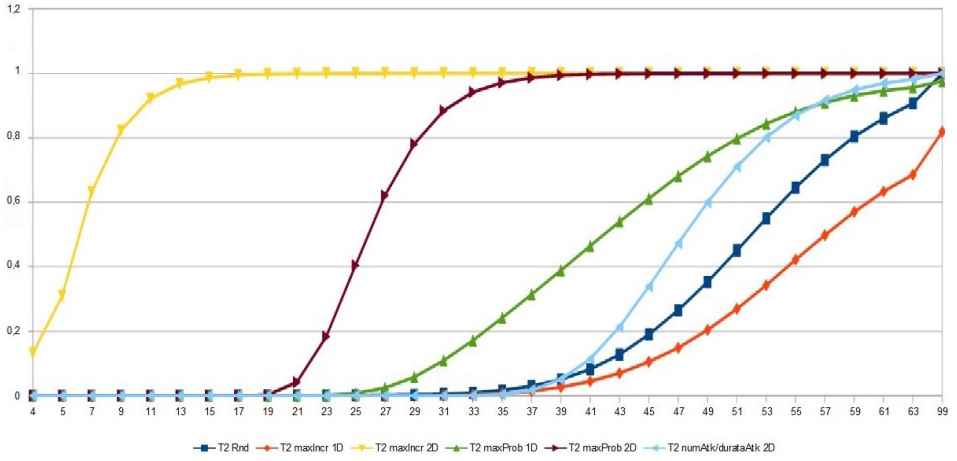

Figure 8: Hydroelectric ICS: Attack Stress of an External Agent

to investigate security issues while the other ICS controls a real plant. The complexity of a real ICS has influenced the time to build the system and the agent models and the one to run the experiments. The time to build the system model has doubled, from less than a day to a couple of days. The time to run each experiment increases from less than one hour to a few hours. This is due to the confidence level we require, 95% on the components that an agent attacks. Because of the complexity of a real system, this level strongly increases the number of runs in the experiments. However, Haruspex strongly reduces the time to assess and manage ICT risk with respect to traditional assessments. We have also verified that a much lower number of runs results in the same confidence level on the success probability of each agent.

The stress functions of the two ICSs confirm that the adoption of a larger number of subnets and of firewalls strongly increases the overall robustness. A further, expected, result is the influence of the agent parameters. All the scenarios considers more agents than those strictly required to recover the lack of information on the agents parameters. The availability of the large amount of data produced by attack simulation simplifies the assessment but has suggested the adoption of a synthetic measure, the security stress, to simplify results presentation. It also worth noticing, the similarities between the outputs of the assessment of an ICS and those of other ICT systems with high security requirements. As an example, even in ICSs, a low number of countermeasures suffices to guarantee that the attacker cannot reach their goal.

6. Conclusion

The Haruspex suite is an integrated set of tools to assess and manage ICT risk through multiple simulations of intelligent, goal oriented agents. These agents escalate their access rights by composing attacks enabled by the target system vulnerabilities. Lack of information on the threat agents may recovered by considering a set of agents, one for each distinct combinations of the parameters that determine the agent behavior. The adoption of the Haruspex suite enables the system architect to discover weakness and the proper countermeasures in the design phase before deploying the system. We have applied the suite to two ICSs that supervise power generation in a scenario where some attackers aim to control the generation plant. This results in the discovery of the most dangerous attackers, the probability they reach their goals and the time this takes. Both assessments have improved the overall resilience of ICT components and, hence, of the whole plant by selecting a small set of countermeasures that prevents an attacker from controlling power generation.

Future developments of our work concern the development of tools to model a larger class of threat agents, in particular those that heavily exploit malware or worms. These agents aim control a large number of nodes to launch further attacks. An even more challenging development concerns the ability of managing a larger number of countermeasures such as those that update the system topology. These countermeasures dynamically update the overall system structure and this has a deep impact on the system models and, hence, on attack

simulation. However, these developments are fundamental to apply the suite to highly complex systems such as smartgrids.

References

[1] Baiardi, F., Corò, F., Tonelli, F., Sgandurra, D.: Automating the assessment of ict risk. Journal of Information Security and Applications **19**(3), 182–193 (2014)

[2] Baiardi, F., Corò, F., Tonelli, F., Sgandurra, D.: A scenario method to automatically assess ict risk. In: Proc. of the 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'14), Turin, Italy, pp. 544–551 (2014). IEEE

[3] Kotenko, I., Konovalov, A., Shorov, A.: Agent-based modeling and simulation of botnets and botnet defense. In: Conference on Cyber Conflict. CCD COE Publications. Tallinn, Estonia, pp. 21–44 (2010)

[4] Brown, T., Beyeler, W., Barton, D.: Assessing infrastructure interdependencies: the challenge of risk analysis for complex adaptive systems. International Journal of Critical Infrastructures **1**(1), 108–117 (2004)

[5] LeMay, E., Unkenholz, W., Parks, D., Muehrcke, C., Keefe, K., Sanders, W.H.: Adversary-driven state-based system security evaluation. In: Proc. of the 6th International Workshop on Security Measurements and Metrics (MetriSec'10), Bolzano, Italy, p. 5 (2010). ACM

[6] Buede, D.M., Mahoney, S., Ezell, B., Lathrop, J.: Using plural modeling for predicting decisions made by adaptive adversaries. Reliability Engineering & System Safety **108**, 77–89 (2012)

[7] Barnum, S.: CAPEC schema description. http://capec.mitre. org, Accessed May 2015

[8] Engle, S., Whalen, S., Howard, D., Bishop, M.: Tree approach to vulnerability classification. Technical Report CSE-2006-10, Department of Computer Science, University of California, Davis (May 2005)

[9] Ingols, K., Chu, M., Lippmann, R., Webster, S., Boyer, S.: Modeling modern network attacks and countermeasures using attack graphs. In: Proc. of the 25th Annual Computer Security Applications Conference (ACSAC'09), Honolulu, Hawaii, pp. 117–126 (2009). IEEE

[10] Baca, D., Petersen, K.: Countermeasure graphs for software security risk assessment: An action research. Journal of Systems and Software **86**(9), 2411–2428 (2013)

[11] Evans, S., Heinbuch, D., Kyle, E., Piorkowski, J., Wallner, J.: Risk-based systems security engineering: Stopping attacks with intention. Security & Privacy, IEEE **2**(6), 59–62 (2004)

[12] Klaus, T.: Security metrics-replacing fear, uncertainty, and doubt. Journal of Information Privacy and Security **4**(2), 62–63 (2008)

[13] Salini, P., Kanmani, S.: Survey and analysis on security requirements engineering. Computers & Electrical Engineering **38**(6), 1785–1797 (2012)

[14] Vaughn Jr, R.B., Henning, R., Siraj, A.: Information assurance measures and metrics-state of practice and proposed taxonomy. In: System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference On, p. 10 (2003). IEEE

[15] Schudel, G., Wood, B.: Adversary work factor as a metric for information assurance. In: Proceedings of the 2000 Workshop on New Security Paradigms, pp. 23–30 (2001). ACM

[16] Langweg, H.: Framework for malware resistance metrics. In: Proceedings of the 2nd ACM Workshop on Quality of Protection, pp. 39–44 (2006). ACM

[17] Wang, L., Jajodia, S., Singhal, A., Cheng, P., Noel, S.: k-zero day safety: A network security metric for measuring the risk of unknown vulnerabilities. Dependable and Secure Computing, IEEE Transactions on **11**(1), 30–44 (2014)

[18] Pamula, J., Jajodia, S., Ammann, P., Swarup, V.: A weakest-adversary security metric for network configuration security analysis. In: Proceedings of the 2nd ACM Workshop on Quality of Protection, pp. 31–38 (2006). ACM

[19] Howard, J.D.: An analysis of security incidents on the internet 1989-1995. Technical report, DTIC Document (1997)

[20] Habash, R.W., Groza, V., Krewski, D., Paoli, G.: A risk assessment framework for the smart grid. In: Electrical Power & Energy Conference (EPEC), 2013 IEEE, pp. 1–6 (2013). IEEE

[21] Wang, X., Yan, Z., Li, L.: A grid computing based approach for the power system dynamic security assessment. Computers & Electrical Engineering **36**(3), 553–564 (2010)

[22] Genge, B., Siaterlis, C., Fovino, I.N., Masera, M.: A cyber-physical experimentation environment for the security analysis of networked industrial control systems. Computers & Electrical Engineering **38**(5), 1146–1161 (2012)

[23] NIST: National vulnerability database. http://nvd.nist.gov/, Accessed May 2015

[24] Scarfone, K., Mell, P.: An analysis of cvss version 2 vulnerability scoring. In: Empirical Software Eng. and Measurement, 2009, pp. 516–525 (2009)

[25] Jagatic, T.N., Johnson, N.A., Jakobsson, M., Menczer, F.: Social phishing. Commun. ACM **50**(10), 94–100 (2007)

[26] Holm, H., Sommestad, T., Almroth, J., Persson, M.: A quantitative evaluation of vulnerability scanning. Information Management and Computer Security **19**(4), 231–247 (2006)

[27] Chen, L.-H., Hsu, F.-H., Hwang, Y., Su, M.-C., Ku, W.-S., Chang, C.-H.: Armory: An automatic security testing tool for buffer overflow defect detection. Computers and Electrical Engineering **39**(7), 2233–2242 (2013)

[28] La Corte, A., Scatà, M.: Failure analysis and threats statistic to assess risk and security strategy in a communication system. In: Sixth Int. Conf. on Systems and Networks Communications, pp. 149–154 (2011)

[29] Beale, J., Deraison, R., Meer, H., Temmingh, R., Walt, C.V.D.: Nessus Network Auditing, (2004). Syngress Publishing

Authors' Biographies

Valentino Angeletti

He graduated cum laude in Electronic and Telecommunication Engineering in 2009 at Universitá di Bologna. He joined the Research Department of Enel in 2010. He has been involved in several projects on security and safety of energy environments, including conventional power generation, renewable generation and smart-grids. Currently, he works within the Cybersecurity Assurance Team.

Fabrizio Baiardi

He graduated cum laude in Computer Science at Universitá di Pisa where he currently is a full professor in Computer Science and leads the ICT Risk Assessment and Management research group. He has been involved in several research projects with both public and private partner to evaluate and manage the ICT risk of several critical ICT infrastructures.

Luca Guidi

He graduated in Nuclear Engineering at the University of Pisa in 1979. In 1981 he joined the Research Department of ENEL in Pisa. From 2000 he was responsible of the Diagnostics and Automation group. Now he is in the Generation Technology Strategy Unit of Global Generation - Research and Innovation Department and is the ENEL representative in EXERA.

Daniela Pestonesi

She graduated in Electronic Engineering at Politecnico di Milano in 1992. She is currently responsible of Automation and Diagnostics in the Global Generation - Research and Innovation Department. She was project manager of research projects about ICT for industrial process, diagnostics, robotics and cybersecurity of Industrial Control Systems. She is the ENEL scientific contact in the commission board of IEC 65-E standardization.

Federico Tonelli

He graduated in ICT security at Universitá di Pisa. He currently is a Ph.D. student at Dipartimento di Informatica, Universitá di Pisa. His main research interest is the definition of formal methodologies and tools to automate the assessment and the management of the risk posed by highly complex ICT infrastructures.