Selecting Cost-Effective Countermeasures for Intelligent Threat Agents

F.Baiardi, F.Tonelli, A. Bertolini, R. Bertolotti

Dipartimento di Informatica, Università di Pisa, Italy
haruspex@di.unipi.it

Abstract. Haruspex is a suite of tools based upon a Monte Carlo method to manage and assess the risk posed by an ICT system. The tools apply a scenario-based method and simulate how some intelligent threat agents compose the attacks enabled by the system vulnerabilities to reach some predefined goals. Some tools of the suite build a description of the scenario of interest with the vulnerabilities, the attacks they enable and the various agents. Another tool applies a Monte Carlo method to simulate step by step the sequence of attacks each agent implements. This tool returns samples on the attacks of each agent, the goals it has reached and the resulting impact. Further tools analyze these samples to return information to support the assessment. This paper is focused on the *manager*, the risk management tool to select the countermeasures to deploy. To take into account that an intelligent agent may react to countermeasure by selecting distinct attacks, this tool adopts an iterative solution that alternates the selection of countermeasures and the application of the Monte Carlo method. Lastly, we consider three ICT systems with SCADA components and show how the *manager* reduces the risk due to each of these systems.

Keywords: Risk Assessment and Management; Countermeasures; Scenario; Monte Carlo Method.

1 Introduction

Intelligent threat agents, or agents, chain the attacks enabled by the system vulnerabilities into sequence of attacks, or sequences, to reach some predefined set of privileges. Each sequence implements a privilege escalation [1] where an agent uses the privileges by some attacks to implement the following ones in the sequence till collecting all the privileges in a goal. The privilege escalation of an agent may involve one or several computing nodes of an ICT network.

Haruspex is an integrated set of tools to support a probabilistic risk assessment of an ICT system with respect to some intelligent agents. The suite adopts a scenario based approach where each scenario includes the target ICT system and some agents. Some Haruspex tools build the description of a scenario starting from the output of a standard vulnerability scanner of each system node. Further tools receive this description and apply a Monte Carlo method that runs

multiple simulations of how an agent select and implement sequences. In each simulation, samples are collected to compute statistics to assess and manage the risk in the considered scenario.

This paper discusses the *manager* and the *planner*, the tools of the Haruspex suite that uses the outputs of the Monte Carlo method to mitigate the risk by selecting cost effective countermeasures to deploy.

The paper is structured as follows. Sect. 2 briefly reviews related works on vulnerabilities, attacks, agents and their simulation. Sect. 3 presents the main tools of the suite. Sect. 4 introduces the *manager*, the *planner* and the algorithm they apply to select cost effective countermeasures. Sect. 5 discuss the application of the two tools to three versions of an industrial control system to supervise a power generation plant. Lastly, sect. 6 draws some conclusions and outlines some future works.

Some tools of the Haruspex suite have been previously described in [2,3,4,5] that consider the building of a scenario description and the application of the Monte Carlo method. These papers focus on the simulation engine and the description builder rather than on the *planner* and the *manager*.

2 Related Works

We briefly review the main related works on attacks, plans, their description, and the evaluation of the corresponding risk.

[6,7,8,9,10,11] analyze the simulation of attacks against ICT or critical infrastructures. Intelligent, goal oriented agents have been analyzed with reference to terrorism [12,13]. [14] presents a formal model of plans similar to the one we have adopted. [15] describes the prerequisites and the effects of attacks and pairs each attack with the proper countermeasure. [12,16,17] describe how the deployment of countermeasures may affect the behavior of threat agents. [11] discusses the modeling of agents with partial information and [18] defines the notion of look-ahead but in a different perspective than the one of Haruspex.

Most of the tools that analyze attacks or attack chains do not support their automated discovery and the automatic selection of countermeasures. [19] proposes a taxonomy focused on a series of use case events. The vulnerability classification in [20] maps each vulnerability into just one class. The theoretical approach in [21] analyzes attack plans involving distinct computing nodes of an infrastructure and it is focused on the compromised level of a node. The resulting approach sacrifices completeness as it does not enumerate all sequences. [22,23] define, respectively, a language to model attack scenarios and a run time algorithm to detect stepping stones. [24] specifies M2D2 a formal data model for IDS alert correlation. [19] supports the discovery of attack plans but computes their success probability by analyzing each plan in isolation. [25] discusses the modeling of countermeasures through attack graphs. [26] considers goal oriented attackers.

The simulation of agent attacks has been analyzed in the framework of game theory [27,28]. With respect to this approach, the Haruspex suite assumes that

an assessment is interested in the probability that some agents reach their goals in some scenarios of interest. Hence, we are interested in reducing this probability rather than in diverting the agent to a distinct target [29,30].

[31,32] survey agent-based simulation, whereas [33,34] model the components of a critical infrastructure as agents to simulate it and discover dependencies among components. [35] discusses the measurement of the risk posed by ICT systems, while [36] reviews the problems posed by a Delphi approach. [37] describes alternative approaches to evaluate the risk due to software systems. [38,39,40] review multi objective optimization that underlies agent selection strategies.

3 The Haruspex suite

This section details the core of the Haruspex suite, namely the tools that, respectively, builds the description of a scenario and applies the Monte Carlo method to this description. The next section is focused on the tools to manage the risk. For the sake of brevity, we assume the risk posed by a system may be quantified through the loss for the owner of the system and the probability that the event occurs.

3.1 Describing a Scenario

After introducing some definitions, we outline the most important information in a scenario description. In the following, we show how this information is produced by mapping the output of a vulnerability scanner. Table 1 defines the acronyms used in this paper.

A scenario includes S, the system to be assessed, the agents that attack S. Haruspex describes S in a modular way by decomposing into a set of components that are described in terms of operations, vulnerabilities and attacks. Vulnerabilities enable the attacks in the sequences the agents implement to illegally acquire some privileges, e.g. access rights or rights. Each vulnerability may be known or suspected. Each suspected vulnerability is paired with a probability distribution of being discovered at time t. The pre and the post conditions of an attack at describe, respectively, the rights required to implement at and those that at grants if it is successful. A threat agent, or agent (ag), is one instance of a threat, e.g. an attacker of S, with the resources and the capability to violate the security policy of S to reach some goal g. g is a set of rights and it may be paired with a loss for the owner of S that occurs when, and if, ag owns the rights in g. The description of ag includes its goals, the resources it can access and the initial rights, e.g. the operations ag is entitled to invoke. Agents are rational and adaptive as they minimize their efforts to reach a goal and selects the attack to implement according to the goal of interest. Rational agents are the worst case of attackers because, besides minimizing their efforts, they can change the attacks they implement if some countermeasures are deployed.

Table 1. List of Components and Attributes

S	the target of the assessment
c	a component of S
op	an operation defined by a component
ag	a threat agent
$res(ag)$	the resources ag can access
g	a goal of an agent
at	an attack
v	a vulnerability
$v(at)$	the vulnerabilities enabling at
$pre(at)$	the rights to execute at
$res(at)$	the resources to execute at
$post(at)$	the rights acquired if at is successful
$succ(at)$	the success probability of at
$AttGr(S, ag)$	the attack graph with the plans of ag against S
n	a node of $AttGr(S, ag)$
$r(n)$	the rights paired with the node n
$\lambda(ag)$	the look-ahead of ag
$na(ag)$	the number of attacks ag executes before a new selection

3.2 Building the Scenario Description

We outline the design of the *builder* [4], the Haruspex tool that builds the description of S, of its components, their vulnerabilities and the corresponding attacks. Since most of the scenarios that an assessment considers differ because of the agents only, the *builder* strongly reduces the complexity of the assessment and, as a further advantage, it increases both the accuracy of the description and the complexity of the system that can be described.

The input of the *builder* is a MySQL database with all the vulnerabilities in the various components in the nodes of S. This database is produced by merging the outputs of the vulnerability scanning of the nodes of S. Distinct scanners can be applied to distinct nodes and the user can insert, remove or edit, any vulnerability in the database. The *builder* discovers global vulnerability, e.g. sets of correlated vulnerabilities where two vulnerabilities are correlated if the attacks they enabled can be sequentialized into a plan because the pre condition of an attack is included in the post condition of the other. By discovering the global vulnerabilities of S, the *builder* computes most of the information to describe S.

The discover of global vulnerabilities is built around a classification of each vulnerability v of a component of S into one of seven classes. The classification of v is driven by the Common Vulnerabilities and Exposures (CVE) description of the attacks v enables. In this way, the pre and post conditions of the attacks enabled by v are deduced from the class of v . By considering the CVE description of v, the *builder* determines also other attributes of the attacks that v enables

such as their success probability. We refer to [4] for a detailed discussion of the current implementation and the accuracy of the classification.

The *builder* can work in several modes. In the one of interest for this paper, at first the *builder* computes the attack surfaces of each node in S by merging information on attack post conditions with the one on the logical interconnection topology to discover which nodes can be attacked from a given one. This information is fundamental to discover alternative plans of an agent. After computing the attack surfaces, the *builder* returns any vulnerability in each component of S, the attacks it enables and their pre and post conditions.

3.3 Attack Sequences vs Plans

We recall some properties of attack plans and then introduce some parameters that define how they are selected and implemented by the agents.

A plan of ag is one of the sequence that ag can implement to reach a goal. As an example, a plan pl of ag to control a node n' of an ICT network may include three attacks. The first attack gains access to an account on a network node n. Then, ag uses this account and n as a step stone to attack n' and control a further account. Lastly, ag becomes the administrator of n' through a privilege escalation attack. A sequence that grants to ag the rights in $g$but that includes further attacks beside those in pl is not a plan because there is a shorter sequence of attacks to g.

Being intelligent, when ag selects the attack to implement, it should only consider the plans that lead to a goal and rank them according to some predefined strategy that considers, among others, the number of attacks, the time to execute the attacks, or their success probability. After selecting a plan, ag implements its first attack and then it repeats the selection to take into account newly discovered vulnerabilities. However, when selecting a plan, ag needs to acquire and elaborate some information on S. Suppose, as an example, that ag can implement one of two attacks, at_1 and at_2. To select the shortest sequence to a goal, ag has to acquire information on the attacks that can be executed after at_1 and after at_2 and so on. Hence, to select plans to reach a goal, ag has to analyze sequences with a length that depends upon both S and the current rights of ag. If, as it always happens in a real attack, the length of the sequences is bounded a priori, then ag may be forced to select a sequence without knowing if leads to a goal. This implies that ag may implement sequences that are not plans, eg some of their attacks are useless to reach a goal, due to lack of information on S. From now on, to take this case into account, we say that ag selects a sequence rather than a plan.

3.3.1 Information Gathering The selection strategy of ag defines how ag balances costs and benefits of alternative sequences [41]. As previously discussed, alternative strategies may be defined and they share a parameter $\lambda(ag)$, the *look-ahead* of ag. $\lambda(ag)$ is the number of attacks that the selection strategy of ag considers to rank a sequence. If $\lambda(ag) = 0$, then ag neglects any information

on S and randomly selects an attack according to the current rights of ag, even if it is not enabled.

To rank a sequence with an attack enabled by a vulnerability of c, ag gathers information through a vulnerability scanning of c. This scanning requires a time depending on c. Since this time is paid only the first time ag ranks a subplan with one attack enabled by a vulnerability of c, the time a ranking requires increases with the number of components to be scanned that, in turn, increases with $\lambda(ag)$. To model insiders, each agent may be paired with the components it knows and does not have to scan.

3.3.2 Attack Graphs An attack graph $AttGr(S, ag)$ is a direct, acyclic, labeled graph that represents distinct sequences of ag. Each node n of $AttGr(S, ag)$ represents the set $r(n)$ of rights that ag owns. The current status of ag is the last node of $AttGr(S, ag)$ reached by ag. Since node transitions are related to elementary attacks, there is an arc from n_1 to n_2 labeled by at if $r(n_1)$ includes $pre(at)$ and $r(n_2)$ is equal to $r(n_1) \cup post(at)$. A node n is *initial* if the security policy of S entitles ag to the rights in $r(n)$. A node n is $final$ if it is the first node on a path from an initial node where $r(n)$ includes the rights of a goal g. Each path from an initial node to a final one defines a sequence to reach the corresponding goal. An agent ag reaches a node n after acquiring any right in $r(n)$ or, equivalently, by successfully implementing all the attacks labeling the arcs on the path from an initial node to n. As an example, a path with four nodes and three arcs represents a three attacks sequence. If some vulnerabilities enabling at are suspected, e.g not known yet, then ag can cross the corresponding arcs only when they are known.

As described in the following, to simulate the strategy of ag, the *engine* builds a subset of $AttGr(S, ag)$ starting from the node reached by ag.

3.4 Monte Carlo Simulation of Agent Attacks

Starting from a scenario description, the *engine* returns a database with samples collected in an experiment with several independent runs that simulate, for the same time interval, the agent attacks and the discovery of suspected vulnerabilities. At each time step of a run, the *engine* determines which suspected vulnerabilities are discovered. Then it considers each agent ag that still has to reach at least one goal and it is idle or it has just completed the execution of an attack. After building an attack graph $AttGr(S, ag)$ that includes all the sequences with at most $\lambda(ag)$ attacks that ag may select given its current access rights. Then, the *engine* applies the selection strategy of ag to determine the attack at that ag implements. If at is enabled, the *engine* simulates it and ag will be busy for $time(at)$ plus the time of the selection. Anytime at successful, the *engine* checks if ag has reached a goal and updates the impact due to ag. If at fails, it is repeated for a user-defined number of times before selecting a distinct attack. At the end of a run, the *engine*collects samples it inserts into a database. A sample includes information on the attacks each agent has implemented, the

goals it has reached, the time to reach a goal, the number of agents that have executed each attack, the number of successful execution of an attack. Then, to guarantee that runs are independent, the *engine* reinitializes the state of S and of any agent and starts a new run.

The confidence level of statistics computed through the sample database depends upon the number of runs in an experiment because each run contributes to the database with exactly one sample. For each experiment, the user can specify either the number of runs or the confidence level for some predefined statistics. In the latter case, Haruspex starts a new run until reaching this level.

3.5 Selection Strategies

Currently, the user can pair ag with one of four strategies:

1. random: returns each attack ag can implement with the same probability,
2. maxprob: returns the sequence with the best success probability,
3. maxincr: returns the sequence granting the largest set of rights,
4. maxeff: returns the sequence with the best ratio between success probability and execution time.

If $\lambda(ag) > 0$, then ag considers the set Cas with the sequences with at most $\lambda(ag)$ attacks that ag can implement, where the first attack is enabled and it increases the current rights of ag. If Cas is empty, ag is idle. Otherwise, if a sequence in Cas leads to a goal of ag, the strategy only ranks the sequences in Cas leading to a goal. Otherwise, it ranks all the sequences in Cas. Upon receiving the first sequence in the ranking, ag implements its first attack.

4 Selecting Countermeasures

This section describes the *planner* and the *manager*, the tools to select the countermeasures to minimize the risk in a scenario.

Here and in the following, for the sake of simplicity, we assume that a scenario includes one agent ag with one goal g. Extensions to several agents and/or several goals are straightforward.

4.1 Discovering Plans

The *planner* analyzes the sample database to remove useless attacks from the sequences of ag to reach g and discover the plans ag implements and their success probabilities. By focusing the selection of countermeasures on the plans rather than on the sequences we increase cost effectiveness by deploying countermeasures for attacks that are useful to reach g.

Given a sequence s of attacks that ag has implemented to reach g, the *planner* executes a backwards scan of s to map it into the plan $p(s, g)$ with the attacks of s useful to achieve g. In the following,

- n is the length of s,
- $s(i)$ is the $i-th$ attack of s, where i is bounded by n,
- $tp(s,g)$ is a sequence that is initialized with $s(n)$, the last attack of s,
- $useful(i)$ the set with the rights ag needs before executing $s(i)$ to achieve g.

Initially, $useful(n) = pre(s(n)) \cup (g - post(s(n)))$, e.g. before executing $s(n)$ the set of useful rights includes those in the precondition of $s(n)$ and the rights in g that do not belong to the post condition of $s(n)$. Informally, before executing at, the last attack in s, the useful rights are those to execute at and those in the goal that cannot be acquired through at itself.

Given $useful(j)$, the algorithm does not insert $at = s(j)$ at the beginning of $tp(s,g)$ if and only if:

1. no right in $post(at)$ belongs to $useful(j)$,
2. before executing at, ag already owns the rights in the intersection between $post(at)$ and $useful(j)$, i.e. each of these rights is an initial right of ag or it belongs to the post conditions of an attack in $\{s(1), ..., s(n-1)\}$.

Before analyzing $s(j-1)$, we assign $useful(j-1)$. If $s(j)$ is useful, then $useful(j-1) = (useful(j) - post(s(j))) \cup pre(s(j))$, e.g. we remove from $useful(j)$ the rights in the post condition of $s(j)$ and add those in its precondition. Otherwise, $useful(j-1) = useful(j)$. At the end of the scanning, $p(s,g) = tp(s,g)$.

This algorithm assumes that ag only implements attacks that increase its rights and it may not return the correct plan if s interleaves several plans. This can be discovered by mapping into a plan not only s but also any permutation of s where the first $j-1$ attacks enable the execution of the $j-th$ one.

After discovering the plans of ag, the *planner* computes the success probability of a plan p as the percentage of runs that have successfully implemented p. This considers that distinct sequences may implement the same plan.

4.2 Selecting Countermeasures for a Set of Plans

The *manager* applies the *planner* to the sample database to discover the plans ag implements. Starting from these plans, it determines the countermeasures to deploy to reduce the corresponding risk. An input parameter, $lowrisk$, defines the highest success probability of ag that may be accepted.

In the following, we assume that for any attack at there is a countermeasure that decreases its success probability. As an example, the patching of a vulnerability in $vuln(at)$ results in the failure of at, while a longer password or a longer encryption key decrease the probability they are guessed. $c(at)$ denotes the cost of a countermeasure for at. At the end, we discuss how to handle attacks with no countermeasure.

The *manager* has an iterative behavior where, after selecting some countermeasures, it updates the description of S to model their deployment and runs another experiment with this version to discover how ag reacts to countermeasure, e.g. which plans ag implements to replace those affected by the deployed countermeasures. The new experiment discovers whether, when attacking S, ag

neglects some plans it can successfully implement against the new version of S. Any of these plans is denoted as a dependent one because ag selects it only when other ones are affected by countermeasures. Only a new experiment can discover dependent plans as they cannot be deduced from the output of an experiment where ag neglects them. If the new experiment shows that the success probabilities of dependent plans is larger than $lowrisk$, the $manager$ selects and deploys further countermeasures, runs another experiment and so on.

Two features that strongly influence number of selected countermeasures are the plans that the $manager$ considers at each iteration. In a first solution, at the $i-th$ iteration the tool deploys countermeasures affecting a set S_i of plans that depends upon all the plans that ag implements in any previous iteration. Hence, these countermeasures may completely differ from those the $manager$ has considered in any previous iteration. Instead, in the incremental approach, at each iteration the $manager$ only considers the dependent plans discovered at that iteration and it extends the previous set of countermeasures with those that affect these dependent plans.

In the current version, S_i includes all the plans considered in the previous iterations and a subset,Cp_i, of the plans ag executes in the $i-th$ iteration. We insert plans into Cp_i starting from those with the largest success probability and stop as soon as the sum of the success probabilities of the remaining plans is lower than $lowrisk$. This heuristic strategy reduces the computational overhead but neglects that the success probability of a plan may strongly increases after deploying some countermeasures for other plans. The user can handle cases where the agent executes a large number of plans by bounding the size of Cp_i as a fixed percentage of successful plans.

4.3 Countermeasures Selection

To minimize the number of countermeasures, we focus the selection of countermeasures for plans in Sp on the attacks these plans share because a countermeasure for at affects all the plans where at appears.

To describe the selection of countermeasures for the plans in Sp, we consider all the attacks they implement and pair each of these attack at with $Sp(at) = \{i_1, ..., i_k\}$, the set of the indexes of the plans in Sp that share at. Now, to compute a set of countermeasures for all the plans in Sp with the lowest cost we compute a coverage [42] with the minimal cost of $\{1, ..., n\}$ through the elements of $Sa = \bigcup(Sp(at))\forall at \in Sp$. When computing a coverage we neglect an attack at_1 if all the plans that share at_1 also share another attack at_2 with a cheaper countermeasure. After removing any attack that can be neglected, the $manager$ considers any possible coverage and select the cheapest one. Given the number of plans and the attacks they share, the resulting execution time is acceptable even if the coverage problem is NP-Complete. As a matter of fact, if both the previous number are very large, then ag can attack S in several, distinct ways, e.g. through a large number of distinct plans, and this requires a rather extensive redesign of S rather than some countermeasures. To further reduce the execution time, we abort the building of a set as soon as its cost exceeds the current best.

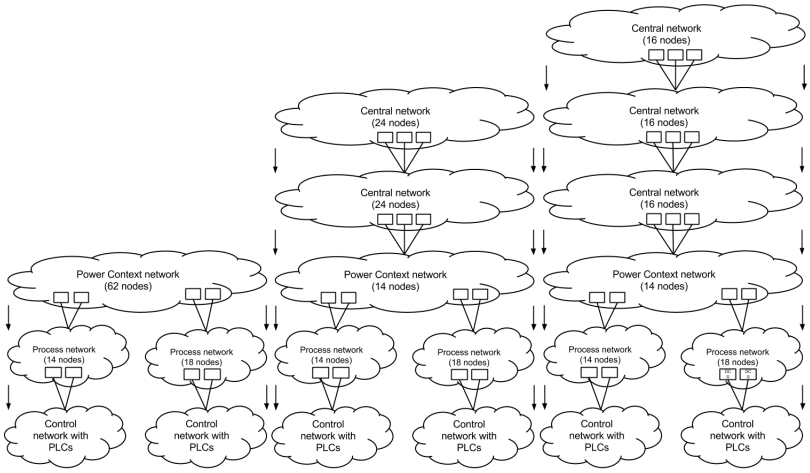

Figure 1. The three versions of the experimental ICT infrastructure

The adopted solution clearly shows the disadvantages of an incremental approach. In fact, by considering at each iteration all the plans to be stopped, we minimize the number of countermeasures by exploiting the best information on the attacks these plans share. Instead, when the incremental approach selects at the $i-th$ iteration the attacks to be affected by a countermeasure, it cannot anticipate the dependent plans the agent implements in the following iterations and of the attacks these plans share with those that are actually considered.

To handle attacks with no countermeasure, first of all when building a coverage of a set of plans we assume that the cost of some countermeasure may be infinite. If the *manager* return a coverage with an infinite cost, then there is at least one plan that only include attack with no countermeasure. The success probability of this plan is a lower bound on the agent success probability.

5 Case Study and Evaluation of Results

We have applied the Haruspex suite to three distinct versions of an ICT infrastructure with SCADA components to control power generation. The infrastructure includes 98 nodes segmented into subnets. There are four kind of subnets: Central, Power Context, Process, and Control. The nodes in a Central subnet are assigned to the company intranet users. The operators use the nodes in a Power Context subnet to manage the SCADA system. The SCADA servers and clients that act as the supervision and control system of the electric power production process belong to a Process subnet. Finally, the PLC systems that control some devices in the plant belong to a Control subnet. In the first version of the infrastructure the Power Context subnet and the Central one have been merged.

Table 2. Details of the Manager Iterations in the Three Assessments

Iter.	N. of plans	N. of patch	N. of plans	N. of patch	N. of plans	N. of patch
1	29	5	563	5	2192	5
2	64	10	271	5	1026	5
3	6	11	275	7	1042	5
4	6	12	673	7	1027	7
5	0	12	688	8	1961	7
6	-	-	213	10	1945	7
7	-	-	109	11	1960	8
8	-	-	109	12	833	10
9	-	-	0	12	455	11
10	-	-	-	-	440	12
11	-	-	-	-	0	12

Fig.1 shows the three versions of the infrastructure that include, respectively, five, seven and eight subnetworks.

In the first version, the Central subnet includes 48 nodes, the Power Context includes 14 nodes. Process subnet 1 and 2 include, respectively, 14 and 18 nodes. Each Process subnet is connected to a Control subnet with two PLC devices. Three nodes of the Central subnet are connected to the Power Context subnet. Two pairs of nodes in the Power Context network are connected to nodes in a Process subnet. Lastly, two nodes in each Process subnet are connected to the corresponding Control subnet. To increase the complexity of the test and to properly stress the suite tools, we have inserted further vulnerabilities in some nodes. Also the second infrastructure has 98 nodes, but the Central subnet is segmented into two subnets, each with 24 nodes. Lastly, the Central subnet of the third infrastructure is segmented into three subnets, each with 16 nodes.

5.1 Selecting Countermeasures to Deploy

All the agents we consider to assess the three versions initially own some rights on the Central Network and scan each node to discover its vulnerabilities. We have defined the other agent attributes by combining goals, strategies and lookahead values. The four possible goals are the control of the PLC devices in both Control Networks, the control of the PLC devices in any Control Network and the control of the PLC devices in a specific Control Network. For each goal, we consider one agent that adopts the random strategy and six agents that adopt, respectively, one of the other three strategies and one of two lookahead values, 1 and 2. The latter value minimizes the time to discover a sequence to reach a goal. In this way, the whole assessment considers 28 agents.

The confidence level of all the experiments is 95% on the components that an agent attacks to reach its goal. This requires from 150.000 to 500.000 runs in each experiment.

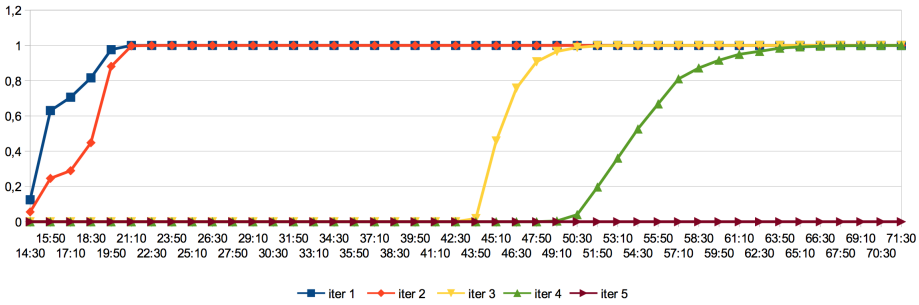

Figure 2. First version of the infrastructure: success probability as a function of time for the iterations of the manager

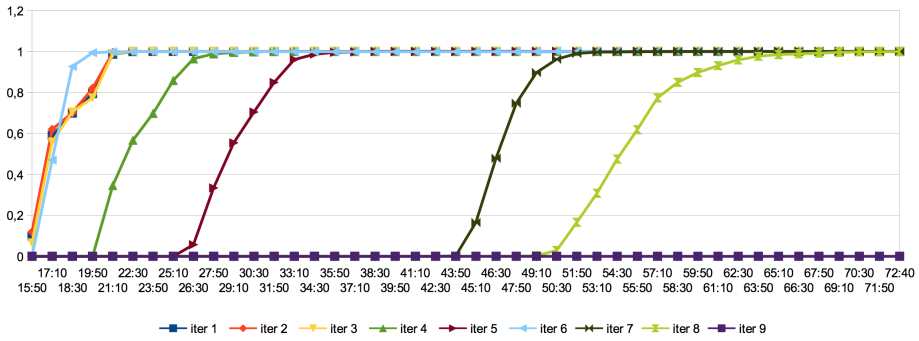

Figure 3. Second version of the infrastructure: success probability as a function of time for the iterations of the manager

In our experiments, the agent that reaches its goal in the shortest time is the one that aims to control the PLCs in any Control Network and adopts the $maxeff$ strategy with $\lambda = 2$.

We consider now for each infrastructure, the output of the *manager* for the considered agent ag under the assumption that we patch any vulnerability and this results in the failure of the corresponding attack.

For each *manager* iteration, Table 2 shows the number of plans the *planner* returns for ag and the one of countermeasures the *manager* selects. The leftmost columns refer to the first version, the central one to the second version and the rightmost columns to the third one. The *manager* finds the optimal set of countermeasures in, respectively, 5, 9 and 11 iterations. The number of plans to be affected increases anytime ag discovers new plans to the goal.

Fig 2, 3 and 4 shows the success probability as a function of the time ag has available to reach its goals. The value of the curve at time t is computed as the percentage of the runs in an experiment where ag reaches its goal within t. Each figure shows the change in the success probability of ag due to the

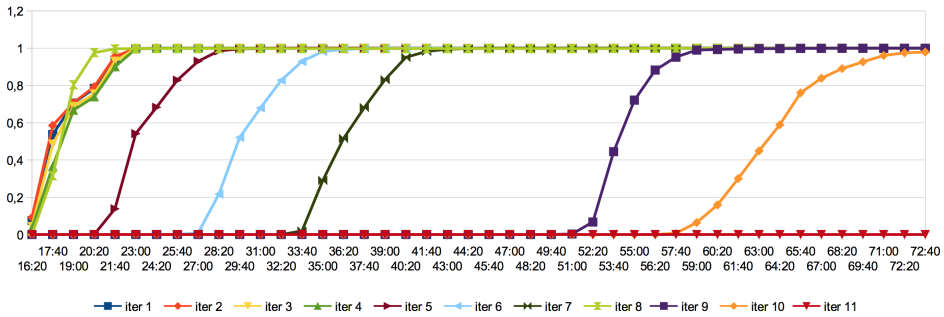

Figure 4. Third version of the infrastructure: success probability as a function of time for the iterations of the manager

countermeasures the *manager* selects in the $i-th$ iteration. The curve overlaps the x axis at the last iteration, when all the plans are stopped.

6 Conclusion

We have outline the Haruspex suite, a set of tools to assess and manage in an automatic way the risk due to an ICT system under attack by agents that escalate their privileges through attack sequences. Then, we have discussed the *manager*, the tool of the suite that computes a cost effective set of countermeasures. This tool runs a sequence of Haruspex experiments. Each experiment in this sequence determines how the agents react to the countermeasure selected using the results of the previous experiments. In this way, we take into account an intelligent agents may react to countermeasures and change the attacks it implements. Future developments of the Haruspex suite concerns the definition of a set of measures to simplify the evaluation of the robustness of a system.

References

1. Tankard, C.: Advanced persistent threats and how to monitor and deter them. Network Security **2011**(8), 16–19 (2011)
2. Baiardi, F., Sgandurra, D.: Assessing ict risk through a monte carlo method. Environment Systems and Decisions, 1–14 (2013)
3. Baiardi, F., Corò, F., Tonelli, F., Guidi, L.: Qsec: Supporting security decisions on an it infrastructure. In: Eighth CRITIS Conference on Critical Information Infrastructures Security, Amsterdam, The Netherlands (2013)
4. Baiardi, F., Corò, F., Tonelli, F., Guidi, L.: Gvscan: Scanning networks for global vulnerabilities. In: First International Workshop on Emerging Cyberthreats and Countermeasures, Regensburg, Germany (2013)
5. Baiardi, F., Corò, F., Tonelli, F., Sgandurra, D.: A scenario method to automatically assess ict risk. In: Proceedings of Euromicro PDP 2014, Turin, Italy (2014)

6. Gorodetski, V., Kotenko, I.: Attacks against Computer Network: Formal Grammar-Based Framework and Simulation Tool. In: Recent Advances in Intrusion Detection. Lecture Notes in Computer Science, vol. 2516, pp. 219–238. Springer, ??? (2002)
7. Kotenko, I.: Active vulnerability assessment of computer networks by simulation of complex remote attacks. Int. Conf. on Computer Networks and Mobile Computing, 40 (2003)
8. Helbing, D., Balietti, S.: How to do Agent Based Simulations in the Future (2011)
9. Conrad, S.H., LeClaire, R.J., O'Reilly, G.P., Uzunalioglu, H.: Critical national infrastructure reliability modeling and analysis. Bell Labs Technical Journal **11**(3), 57–71 (2006)
10. Brown, T., Beyeler, W., Barton, D.: Assessing infrastructure interdependencies: the challenge of risk analysis for complex adaptive systems. Int. Journal of Critical Infrastructures **1**(1), 108–117 (2004)
11. LeMay, E., Unkenholz, W., Parks, D., Muehrcke, C., Keefe, K., Sanders, W.: Adversary-driven state-based system security evaluation. In: Proc. of the 6th Int. Workshop on Security Measurements and Metrics. MetriSec '10, pp. 5–159. ACM, New York, NY, USA (2010)
12. Rios Insua, D., Rios, J., Banks, D.: Adversarial risk analysis. Journal of the American Statistical Association **104**(486), 841–854 (2009)
13. Buede, D.M., Mahoney, S., Ezell, B., Lathrop, J.: Using plural modeling for predicting decisions made by adaptive adversaries. Reliability Engineering and System Safety **108**(0), 77–89 (2012)
14. Cheung, S., Lindqvist, U., Fong, M.W.: Modeling multistep cyber attacks for scenario recognition. In: DARPA Information Survivability Conference and Exposition, 2003. Proceedings, vol. 1, pp. 284–2921 (2003)
15. Barnum, S.: Common attack pattern enumeration and classification (capec) schema description. Cigital Inc, http://capec. mitre. org/documents/documentation/CAPEC _Schema_Descr iption_v1 **3** (2008)
16. Diamant, J.: Resilient security architecture: A complementary approach to reducing vulnerabilities. Security Privacy, IEEE **9**(4), 80–84 (2011)
17. Bohme, R., Moore, T.: The iterated weakest link. Security Privacy, IEEE **8**(1), 53–55 (2010)
18. LeMay, E., Unkenholz, W., Parks, D., Muehrcke, C., Keefe, K., Sanders, W.: Model-based Security Metrics using ADversary VIew Security Evaluation (ADVISE). In: Proc. of the 8th Int. Conf. on Quantitative Evaluation of SysTems (QEST 2011) (2011)
19. Howard, J.D.: An analysis of security incidents on the internet 1989 - 1995. Ph.D Thesis (1998)
20. Engle, S., Whalen, S., Howard, D., Bishop, M.: Tree approach to vulnerability classification (2005)
21. Ammann, P., Pamula, J., Ritchey, R., Street, J.: A host-based approach to network attack chaining analysis. Technical Report SERC-TR-165-P (2005)
22. Cheung, S., Lindqvist, U., Fong, M.W.: Modeling multistep cyber attacks for scenario recognition. In: DARPA Inf. Survivability Conf. and Exposition, 2003, vol. 1, pp. 284–2921 (2003)
23. Zhang, Y., Paxson, V.: Detecting stepping stones (2000)
24. Morin, B., Mé, L., Debar, H., Ducassé, M.: M2d2: A formal data model for ids alert correlation

25. Ingols, K., Chu, M., Lippmann, R., Webster, S., Boyer, S.: Modeling modern network attacks and countermeasures using attack graphs. In: Proc. of the Annual Computer Security Applications Conf., pp. 117–126. IEEE Computer Society, Washington, DC, USA (2009)
26. 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)
27. Bier, V.M., Oliveros, S., Samuelson, L.: Choosing what to protect: Strategic defensive allocation against an unknown attacker. Journal of Public Economic Theory **9**, 563–587 (2007)
28. Hausken, K., Bier, V.M.: Defending against multiple different attackers. European Journal of Operational Research **211**, 370–384 (2011)
29. Florencio, D., Herley, C.: Sex, Lies and Cyber-crime Survey. In: The Tenth Workshop on Economics of Information Security (2011)
30. Florencio, D., Herley, C.: Where Do All the Attacks Go? In: The Tenth Workshop on Economics of Information Security (2011)
31. Macal, C.M., North, M.J.: Tutorial on agent-based modelling and simulation. Journal of Simulation **4**(3), 151–162 (2010)
32. Rob, A.: A Survey of Agent Based Modelling and Simulation Tools. Technical Report DL-TR-2010-07, Science and Technology Facilities Council (2010)
33. Ghorbani, A., Bagheri, E., Onut, Zafarani, R., Baghi, H., Noye, G.: Agent-based Interdependencies Modeling and Simulation (AIMS). Technical report, Technical Rep. No. IAS-TR01-06, Intelligent and Adaptive Systems Research Group, Faculty of Computer Science, UNB (September 2006)
34. Casalicchio, E., Galli, E., Tucci, S.: 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, pp. 182–189. IEEE Computer Society, Washington, DC, USA (2007)
35. Arora, A., Hall, D., Piato, C.A., Ramsey, D., Telang, R.: Measuring the risk-based value of it security solutions. IT Professional **6**(6), 35–42 (2004)
36. Herrmann, A.: The quantitative estimation of it-related risk probabilities. Risk Analysis, (2012)
37. Alberts, C., Allen, J., Stoddard, R.: Risk-based measurement and analysis: Application to software security. Technical report, Software Engineering Inst., CMU (2012)
38. Konak, A., Coit, D.W., Smith, A.E.: Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering and System Safety **91**(9), 992–1007 (2006)
39. Deb, K.: Multi-objective optimization. In: Search Methodologies, pp. 273–316. Springer, ??? (2005)
40. Marler, R.T., Arora, J.S.: Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization **26**, 369–395 (2004)
41. Boddy, M., Gohde, J., Haigh, T., Harp, S.: Course of action generation for cyber security using classical planning. In: Proc. ICAPS 2005, pp. 12–21. AAAI Press, ??? (2005)
42. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J., Bohlinger, J. (eds.) Complexity of Computer Computations. The IBM Research Symposia Series, pp. 85–103. Springer, ??? (1972)