Harùspex: a Suite to Assess and Manage ICT Risk by Simulating Threat Agents

Fabrizio Baiardi & Federico Tonelli
Dipartimento di Informatica
Università di Pisa, Pisa, Italy.

Daniela Pestonesi & Valentino Angeletti
Enel Ingegneria e Ricerca
Pisa, Italy.

ABSTRACT: Harùspex is a suite that supports a scenario-based assessment. In each scenario, intelligent agents compose elementary attacks against an ICT system to reach some predefined goals. Some Harùspex tools build the models of the target system and of the agents of interest. Using these model, further tools apply a Monte Carlo method with multiple, independent simulations of the agent attacks and return a sample to compute statistics of interest. Other Harùspex tools use the statistics to produce some security stress curves that evaluate the system robustness and select countermeasures to improve it. After describing our approach and the tools, we detail the assessment of a turbo gas power generation plant.

1 INTRODUCTION

We consider the quantitative, probabilistic risk assessment and management of an information and communication technology, ICT, system, in general an ICT network, under attack by intelligent, goal-oriented agents. These agents escalate their privileges by composing the attacks enabled by system vulnerabilities into sequences of attacks or complex attacks. Each sequence may involve distinct network nodes and exploits the privileges returned by previous attacks to execute the following ones till the agent acquires all the privileges of interest. This paper describe a model-based assessment that considers a scenario with the target system and the agents that attack it. After building the system and the agent models, we run several, independent simulations of the attacks. In each simulation, we collect information on how each agent selects and executes attacks, and how the system reacts to produce a sample to compute statistics to assess and manage the risk. Since the proposed framework is model based, we can assess a system even before its deployment.

To partially automate the assessment, we have developed the Harùspex suite, an integrated set of tools that build the simulation models, apply the Monte Carlo method, and analyze the samples this method returns. As a further support, we have defined the security stress to simplify the evaluation of ICT robustness.

This paper is structured as follows. Sect. 2 briefly reviews works on security metrics, vulnerabilities, and attack simulation. Sect. 3 describes the Harùspex tools to build the models and simulate the agent attacks. Sect. 4 defines the tools to analyze the output of the simulation. It also introduce the security stress, the measure we have defined to simplify the evaluation of the robustness of an infrastructure. Sect. 5 describes the adoption of the suite and of the security stress to assess and manage the ICT risk due to an industrial control system, ICS, that supervises and manages a turbo gas power production plant. Lastly, we draw some conclusions.

With respect to our previous works (Baiardi & Sgandurra 2013; Baiardi, Corò, Tonelli, & Guidi 2013b; Baiardi, Corò, Tonelli, & Guidi 2013a), the original contribution of this paper is the adoption of the suite to assess and manage the risk posed by a real power production plant. This assessment exploits synthetic metrics such as the security stress.

2 RELATED WORKS

This section briefly reviews related works on attacks, their description, simulations and metrics we have considered to evaluate ICT robustness.

(Tankard 2011) discusses about the privilege esca-
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 rights to execute $at$ |
| $res(at)$   | the resources to execute $at$ |
| $post(at)$  | the rights granted if $at$ succeeds |
| $succ(at)$  | the success probability of $at$ |
| $time(at)$  | the execution time of $at$ |
| $\lambda(ag)$ | the look-ahead of $ag$ |


All the metrics in (Vaughn Jr., Henning, & Siraj 2003; Schudel & Wood 2000; Langweg 2006) evaluate the robustness of an ICT infrastructure under attack by intelligent agents but none integrates these metrics with the simulation of the attacks. The metric in (Wang, Jajodia, Singhal, Cheng, & Noel 2014) is focused on the discovery of zero-day vulnerabilities. The one in (Pamula, Jajodia, Ammann, & Swarup 2006) is similar to security stress as it considers the amount of work to attack a system. (Howard 1998) computes the probability that an agent reaches a goal but it neglects alternative attacks for the same goal.

3 THE HARUSPEX SUITE: RUNNING EXPERIMENTS

This section describes the kernel of the suite, the tools that build the models of the target system and of the agents and run the simulations of attacks in a scenario. The accuracy of these tools is critical for all the assessment. The following section describes the tools to analyze the samples the kernel tools produce. In the following we denote by user, the team that applies the suite to run an assessment. For the sake of brevity, here and in the following, assessment is a synonymous for probabilistic risk assessment, right and privilege of access right. Table 1 defines the abbreviations we use.

3.1 Modeling an Infrastructure

The builder is the tool that builds a model of the target system $S$ that describes the attacks agent can implement. The model of $S$ decomposes it into components, i.e. hardware/software modules. The vulnerabilities in these components enable some attacks (NIST; MITRE; Scarfone & Mell 2009). Haruspex supports both effective and potential vulnerabilities, i.e. those already known and those that the user suspects. There are several strategies to discover effective vulnerabilities, possible examples include a preliminary analysis or a vulnerability scanning of the various components. An assessment can also consider potential vulnerabilities and pair each of them with the probability it is discovered at a given time. To cover social engineering attacks, our methodology supports the modeling of users of $S$ as further components with the proper vulnerabilities (Jagatic, Johnson, Jakobsen, & Menczer 2007). If any vulnerability in $v(at)$ is effective, at is enabled and succeeds with a probability $succ(at)$, otherwise it fails. For each vulnerability $v$ of a node $n$ of $S$, the builder computes the $pre(at)$ and $post(at)$ of each attack $at$ that $v$ enables. These attributes of $at$ determine the sequence where $at$ appears. To compute these conditions, the builder exploits some de facto standard databases, e.g. the Common Vulnerability Enumeration (NIST; Baiardi, Corò, Tonelli, & Sgandurra 2014), and it classifies $v$ by matching some predefined patterns against its description. $pre(at)$ and the $post(at)$ depends upon the class of $v$. Furthermore, the builder discovers also if $at$ can be remotely exploited against $n$. In this way, the tool computes the nodes that can attack $n$. This information is used to compute attack sequences. The builder considers any attack sequence of $ag$ as a privilege escalations that may involve distinct nodes of $S$. Furthermore, for any $ag$ this sequence starts from a node $n_i$, where $ag$ owns some rights on the component that run on the node, and it ends in a node $n_f$, where $ag$ acquires at least one right of $g$. To discover privilege escalations that involve distinct nodes, the user describes to the builder the logical topology of $S$, because some constraints may prevent a node $n_i$ to directly interact with some other nodes $n_f$. The user also describe the components of $S$ that controls this topology such as firewalls.

3.2 Modeling Agents

The descriptor is the tools that builds agent models. For each agent, the user specifies four properties:
1) the initial rights;
2) the goal(s);
3) the selection strategy;
4) the value of $\lambda(ag)$.
Each goals is a set of rights that $ag$ reaches after acquiring all its rights and it results in an impact, i.e. a loss the owner of $S$. Even if the security policy forbids
ag to acquire any right in g, there is an impact only if, and when, ag reaches g. To implement at, ag needs the resources in res(at) and it should own, or acquire through previous attacks, the rights in pre(at). We model the intelligence of ag through a selection strategy that minimizes the execution time of the privilege escalation or maximizes the probability of reaching a goal. This strategy considers the goals, the current set of rights and the preferences of ag. 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 it cannot execute an attack.

\( \lambda (ag) \), the look-ahead of ag, is an attribute that describes the amount of information on S the strategy of ag by specifying to the length of the sequences it ranks. If some sequences with at most \( \lambda (ag) \) attacks lead to a goal, then one of them is selected. Otherwise, the strategy ranks these sequences according to the attributes of their attacks. Here, ag may select a sequence with useless attacks, i.e. they only grant rights useless to reach a goal. Some of the strategies ag can adopt are:

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

If \( \lambda (ag) = 0 \), then ag adopts a strategy, SmartSubnetFirst, that returns any attack with a precondition included in the current rights of ag and that increases these rights. This strategy privileges attacks that enable ag to enter another subnetwork, and it is Smart because it only considers attacks ag can execute. However, it may even return an attack at that is not enabled.

To model the time to select an attack, we assume that ag scans a node \( n \) the first time it considers a sequence with an attack enabled by a vulnerability of c that runs on \( n \). The scanning occurs only once because it returns all vulnerabilities in the components on \( n \) and it takes a time depending upon \( n \) and the number of its vulnerabilities. Larger values of \( \lambda (ag) \) result in a more accurate selection that may avoid useless attacks, on the other hand the time ag spends is directly proportional with the value of \( \lambda (ag) \) due to the larger number of scanning. An insider can avoid this time because there are some nodes it already knows.

3.3 Simulation Engine

The engine is the tool that receives the models of S and of the agents in the scenario of interest and applies the Monte Carlo method to the scenario by executing an experiment with several independent runs. Each run simulates, for the same time interval, the agents attacks and the discovery of potential vulnerabilities. To guarantee run independence, the engine re-initializes the state of S and of any agent before starting a new run. Each experiment returns a database with the samples collected in the various runs.

In each time step, the engine considers whether some potential vulnerabilities are discovered or not and any idle agent that still has to reach a goal. For each agent ag, the engine considers its current rights and computes the set of the sequences with, at most, \( \lambda (ag) \) attacks that ag can select. If this set is empty, ag is busy for the selection time only. Otherwise, the engine applies the selection strategy of ag and it simulates at, the first attack of the sequence the strategy returns. Then, it sets ag busy for \( time(at) \) plus the selection time. If at succeeds and ag has reached a goal, the engine updates the corresponding impact. ag retries a failed attack for \( nr(ag) \) times before invoking again its selection strategy. Each run produces a sample that records the sequence each agent has implemented, the goals it has reached and the time spent. The sample collected in an experiment is used to compute statistics of interest. The confidence level of these statistics depends upon the number of runs because the engine collects one sample in each run. An experiment ends either after executing the specified number of runs or when a predefined statistic reaches the required confidence level.

4 ANALYZING THE OUTPUT OF EXPERIMENT

This section discusses the analysis of the samples the engine returns to minimize the risk in a scenario. At first, the section describes the two tools to select cost-effective countermeasures. Then, it introduces a measure to evaluate ICT robustness. For the sake of simplicity, in the following we assume a scenario includes only one agent, \( ag \) with one goal g.

4.1 Selecting Countermeasures

The two tools that cooperate to select a cost-effective set of countermeasures are the planner and the manager.

4.1.1 Discovering the Plans of an Agent

At first, the planner analyzes the database that the engine returns and removes useless attacks from the sequences ag executes to reach g. We recall that an attack is "useless" the agent does not exploit the rights it grants to reach g. In the following, a plan is any attack sequence that leads an agent to a goal and does not include useless attacks. By considering plans rather than sequences, we increase cost effectiveness because we never select countermeasures for useless attacks. The planner maps each sequence s of ag to reach g into the corresponding plan \( p(s, g) \)
through a backward scan of $s$. This scan removes attacks that grants rights that belong neither to $g$ nor to the precondition of attacks previously inserted into $p(s,g)$. This algorithm is correct if $ag$ only executes attacks that increase its rights and $s$ does not interleave distinct plans. To solve this problem, we also map any permutation of $s$ that is a sequence, e.g. the rights granted by its first $i-1$ attacks enable the $i-th$ one. After computing all the plans, the planner computes the success probability of each plan as the percentage of the runs where $ag$ reaches $g$ through a sequence mapped into the plan.

4.1.2 Iterative Selection Countermeasures

The manager receives a parameter, $lowrisk$ and the output of an experiment and selects a cost effective set of countermeasure that reduce the risk to, at most, $lowrisk$. We assume that for each attack there is a countermeasure that decreases its success probability. As an example, a patch for a vulnerability $vuln(at)$ results in the failure of $at$, while longer password or longer encryption keys decrease the probability that $ag$ guesses them. The extensions to this assumption are trivial. Under these assumption, $lowrisk$ defines the highest success probability of $ag$ the user accepts.

The manager runs a first experiment and then enters a sequence of iterations. In each iteration, it applies the planner to the output of the last experiment to discover the plans $ag$ has executed. Then, it selects a subset of these plans and determines a set of countermeasures that affect these plans. Then, it updates the model of $S$ to mimic their deployment and runs it a new experiment to discover how $ag$ reacts to these countermeasures, i.e. it discovers any new plan $ag$ implements to replace those affected by the countermeasures. When the new experiment ends, the manager checks whether the risk is still higher than $lowrisk$ and, in this case, it starts a new iteration.

Since a countermeasure for $at$ affects all the plans that share $at$, the selection of countermeasures for plans in $Sp$ focuses on the attacks these plans share to reduce both the number of countermeasures and their overall cost. Hence, before selecting the countermeasures, the tool removes all redundant attacks. An attack is redundant if all the plans, that share it, share also another distinct attack with a cheaper countermeasure. Then, the manager maps any non-redundant attack $at$ into $S(at) = \{p_1, \ldots, p_k\}$, the plans of $Sp$ that share $at$. A set of countermeasures for the attacks in $Sat = \{at_1, \ldots, at_j\}$ affects all the plans in $Sp$ if the elements of $Satp = \bigcup (S(at)) \forall at \in Sat$ defines a coverage of $S$. The coverage problem is NP-Complete and the manager solves it by considering all the alternative sets $Sat$. We have experimentally verified that, given the number of plans and the attacks they share, the resulting execution time is still acceptable. According to our results, if the execution time is very large, then there is a large number of alternative plans that $ag$ can implement. This requires a completely redesign of $S$ rather than the selection of a few countermeasures. Anyway, to further reduce the execution time, we abort the building of a set as soon as its cost exceeds the current optimum.

A feature that strongly influences the selection is the set of plans the manager considers at each iteration. There are two approaches: global and incremental. The $i-th$ iteration of a global approach selects countermeasures that affect the set $S_i$ with all the plans $ag$ has implemented in all previous iterations. This includes the plans affected by the countermeasures previously selected. Hence, the manager may select a set of countermeasures that differs from the one selected in a previous iteration. This is possible if the new plans discovered in the following iterations share distinct attacks with those previously discovered. Instead, each iteration of the incremental approach considers the plans discovered in the last experiment, selects countermeasures for these plans and add them to those to be deployed. The global approach exploits at best shared attacks to minimize the number of countermeasures to deploy at the cost of a larger computational complexity. Currently, the manager adopts a global approach where $S_i$ includes all the plans previously considered and a subset, $Cpi$, of the plans that $ag$ executes in the $i-th$ iteration. Plans are inserted into $Cpi$ starting from those with the largest success probability. The selection ends as soon as the overall success probability of the remaining plans is lower than $lowrisk$. This reduces the computational overhead, but neglects that the success probability of a plan may strongly increases after deploying countermeasures for other ones. As a compromise, the user can bound the size of $Cpi$ as a fixed percentage of successful plans.

We handle attacks with no countermeasure by pairing them with a countermeasure with an infinite cost. Hence, if the manager returns a coverage with an infinite cost, at least one plan includes attacks without countermeasure. The success probability of this plan may be a lower bound on the success probability of $ag$ that risk management can achieve.

4.2 Security Measures

Due to the large number of details in the output of an experiment, we have defined a new measure to synthesize this output and simplify the analysis of the robustness of an ICT system.

4.2.1 Countermeasure-based Measures

Some robustness metrics consider the number of countermeasures to deploy. As an example, they consider the number of countermeasures to reduce ICT risk to a user define value. Some of these metrics also consider the cost of countermeasure and return information on the weakest components or the number of steps to reach a goal. This information enables
a user to understand which infrastructure components should be improved from a robustness perspective.

4.2.2 Security Stress
The synthetic measure of ICT robustness we have defined is the security stress of $S$ due to $ag$ that aims to achieve $g$. Formally, we define the security stress at time $t$ as the cumulative probability distribution $Str^{S}_{ag,g}(t)$ that $ag$ reaches $g$ within $t$. $Str^{S}_{ag,g}(t)$ is monotone non-decreasing in $t$ and $Str^{S}_{ag,g}(0) = 0$. Instead of time, we may consider the number of steps that is the number of successful and failed attacks that $ag$ implements to reach $g$. This number evaluates the effectiveness of mechanisms to reduce the success probability of attacks.

To better explain the stress definition, assume that $t_0$ is the lowest time where $Str^{S}_{ag,g}(t) \neq 0$. If $t_0$ does not exist then $ag$ never reaches $g$ when attacking $S$. Furthermore, $t_1$, if it exists, is the lowest time when $Str^{S}_{ag,g}(t) = 1$. 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. After $t_0$, the shape of $S$ begins to change under the attacks of $ag$ till it cracks at $t_1$, when $S$ has completely surrendered to the force of $ag$. $t_1$ is also the ultimate time, because $ag$ is always successful when $t \geq t_1$. $t_1 - t_0$ evaluates how long $S$, partially, resists to the force that $ag$ applies to achieve $g$. $Str^{S}_{ag,g}$ is the inverse of a survival function (La Corte & Scatà 2011) as it plots the success probability of $ag$ instead of the one that $S$ survives $ag$ attacks.

The security stress evaluates in a more accurate way the robustness of $S$ than metrics that consider just a single factor, such as the average time, or the average number of attacks, to reach $g$. This is due to the relation between $Str^{S}_{ag,g}$ and several attributes of both $ag$ and $S$ such as the number and the length of the sequences to reach $g$, $time(at)$ and the selection strategy of $ag$. As an example, distinct strategies change, among others, the number of the useless attacks and this influences the time to reach a goal and the corresponding stress. Attributes such as $succ(at)$ are related to the stress because they determine the average time to successfully implement each attack $ag$ selects.

To generalize the definition of $Str^{S}_{ag,g}$ to a finite set of goal $Sg$, we assume that $ag$ stops its attacks after reaching a goal. Under this assumption, $Str^{S}_{ag,Sg}(t)$ is the probability that $ag$ is idle after $t$. To consider a set of agents $Sag$, we define the most dangerous agent in $Sag$ as the one that results in the largest stress value at any time. Otherwise, $Str^{S}_{Sag,Sg}(t)$ is the weighted sum of the stresses due to the agents in $Sag$. In the same way, we generalize $Str^{S}_{Sag,Sg}(t)$, to a set $Sg$ of several goals. Given the output of an experiment, we approximate $Str^{S}_{ag,g}(t)$ as the percentage of collected samples where $ag$ reaches $g$ before $t$. The confidence level of the approximation depends upon the one of the experiment.

5 RISK ASSESSMENT AND MANAGEMENT OF AN ICS
This section describes an assessment that has adopted Haruspex and where the target is an ICS that supervises a turbo gas power generation plant. The goal of the assessment is to evaluate the probability that some agent acquires the control of power generation and the computation of countermeasures to minimize this probability. Since the system has already been deployed, we have run Nessus Vulnerability Scanner Pro Edition to discover all the standard vulnerabilities that affect its nodes (Beale, Deraison, Meer, Temminger, & Walt 2004). The adoption of Pro version was been forced because only this version includes SCADA plug-ins. Since the PLCs are not standard components, we have also developed some dedicated plug-in modules that integrate the standard one in the scanning of these components. After building the proper models, we have run several Haruspex experiments to produce the statistical samples. Each experiment consists of 50,000 runs and achieves at least a 95% confidence level on the components that the agent attacks to reach its goal. We assume that the owner of the ICS does not supply any information about the agents that can attack the ICS. This is the reason why each experiment considers several agents, one for each possible combination of the agent parameters. Obviously, the number of agents in a scenario may be strongly reduced anytime more accurate information is available, e.g. the agent selection strategy is known. Each experiment adopts ten days, or 240 hours, as the time-limit for each run because we assume attacks are detected after this limit.

5.1 Structure of the ICS
Fig.1 shows the architecture of the target ICS that controls a plant that generates power through four turbo gas groups.

The backbone of this infrastructure consists of some switches and one main firewall that interconnects the switches and four subnets: the intranet network, the SMC network, the DMZ network, and the process network. A further subnet, the control network, is directly connected through some SCADA
servers belonging to the process network. The firewall that defines the perimeters of the subnets also acts as router and, together with the switches, it filters communications among subnets. Each subnet is flat and any two of its nodes can interact.

The business processes use the intranet subnet that contains more than one thousand of personal computer and some SCADA clients. The plant operators use the 11 nodes in the SMC network to control the power generation and exchange processes and the status of the plant. The node with the largest number of vulnerability, 10 critical ones, is the one that manages the graphic wall. Some clients in the SMC network interface the operator with some SCADA Servers. Each of these nodes has the largest number of medium-high vulnerabilities, 5. The DMZ network consists of 6 nodes and it provides services to other power plants and operators. In particular this subnet include an OSIsoft PI server, one of the most important nodes of the power plant because most of the power production depends upon the information it manages. The node that manages the diagnostic of this server has the highest number of critical vulnerabilities, 2. The Server PI does not have any critical vulnerability instead, but only 23 disclosure information vulnerabilities.

The process network includes several SCADA Servers that supervise and control electric power production. This subnet includes 138 nodes and it is logically divided into four parts, one foe each engine in the plan. Since most nodes in this subnet execute old software, this network is affected by several critical vulnerabilities. An an example, an Auxiliary Server that controls the first group has about 350 vulnerabilities and 69 are critical. Each group is directly connected through some switches to the control network. The control network includes 133 nodes. Among these nodes, 80 are programmable logical components, PLCs, and they are logically partitioned into five groups. Each PLC group is controlled by the corresponding node group. This group belongs to the process network and it manages the proper engine. The fifth group of nodes of the control network provides some common services to the power generation plant, e.g. the fire extinguisher service. Several nodes are redounded for availability and safety reasons. Some of those nodes are averagely affected by 20 critical vulnerabilities, except for one auxiliary node of the third group that is affected by 248 vulnerabilities.

According to the outputs of Nessus, the whole infrastructure is affected by more 3716 vulnerabilities that enable 13921 elementary attacks. We have verified that these attacks result in at least 390 distinct escalations.

5.2 Agents in the Scenario

To assess and manage the risk due to the ICS, we have considered an insider agent. This may be someone working in the plant that aims to violates the security policy of the ICS or an external agent that impersonates an operator after a successful phishing attack. We have modeled insiders as two sets of agents that initially control a node of the SMC network and aim to control, respectively, the different groups of SCADA Servers and the PLCs. We denote as AS\(\lambda\), the agents that aims to control the \(x\)-th group of SCADA serves and as AP\(\lambda\), those that aims to control the corresponding PLCs group. The AS\(\lambda\) agents aim to control of all the SCADA Servers belonging to the proper group. Instead, the AP\(\lambda\) agents aim to control or deny the service of all the PLCs in the proper group.

A further class, AP\(Gc\), model the agents that aims to control the fire extinguisher system in the common services group. The agents in each class cover three selection strategies, the maxIncr strategy, the maxEff strategy, and the SmartSubnetFirst one. Furthermore, we have considered \(\lambda = 1\) and \(\lambda = 2\) for the first strategy. Instead, for the second one we have considered only \(\lambda = 2\). For the third one only \(\lambda = 0\) is possible.

The assessment has run 36 experiments, 16 for the AS\(\lambda\) agents and 20 for the AP\(\lambda\) agents.

5.3 Result of Assessment

For the sake of brevity, in the following we only discuss the most dangerous AS and AP agents for each group. In all the experiments, the most dangerous agent is the one that adopts the MaxIncr strategy with \(\lambda = 2\). As far as concern the AS\(\lambda\) agents, Fig.2 and Fig.3 show the stress curves with respect to, respectively, the number of steps and the time that the agent takes to reach its goal(s). According to the stress curves, the AS\(G4\) agent is the first agent that reaches the goal. It executes 9 attacks and it takes about less
than 30 hours to control the fourth group of SCADA Servers. The next agent to reach its goal is the $AS_{G2}$ agent. Even this agent executes 9 attacks, but, with respect to $AS_{G4}$, after 4 attacks its stress is equal to zero. The execution of these 9 attacks requires more than 30 hours. The third agent is $AS_{G3}$. $AS_{G3}$ executes 14 attacks in about 33 hours to control the third group of SCADA Servers. The last agent is the $AS_{G1}$ one. It executes 145 attacks in just less than 240 hours to control the first group of SCADA Servers. The most significant difference between the first three agents and the last one is due to the fact that the first groups of SCADA Server are all up-to-date and run the last versions of software tools.

Fig 4 and Fig 5 show the stress curves of the $AP_{Gx}$ agents in terms of, respectively, number of steps and time. According to our experiments the fastest agent to reach a goal is the $AP_{G2}$ that takes about 87 hours and 71 attacks to control all PLCs that manage the engine of second group. The nearest stress curve is the one of $AP_{Gc}$ that executes 71 attacks that take about 87 hours. However, the stress curve of this agent is always lower than the first one. The third curve is the one of $AP_{G4}$ that takes about 93 hours and 89 attacks to reach its goal. The other two curves are pretty similar. In particular, the curve of $AP_{G3}$ is always lower than the one of $AP_{G1}$. $AP_{G1}$ implements 91 attacks and takes about 100 hours, instead, $AP_{G3}$ executes 217 attacks and takes just less than 240 hours.

### 5.4 Countermeasures

In its iterations to select countermeasures, the manager discovers about 193,000 plans for the $AS$ agents. The $AP$ agents, instead, implement more than half a million plans. After few iterations, the manager returns two sets of countermeasures that can stop respectively all $AS$ plans and all $AP$ plans. The first set includes 7 countermeasures that are mostly related to vulnerabilities of the SMB protocol. These countermeasures affect all the plans and patch any vulnerability in the process network nodes. The second set includes 34 countermeasures that are deployed in the control network and in the process one. The former concern some weaknesses of the SNMP protocol while the latter patch three vulnerabilities of the SMB protocol and the Windows Server Service Crafted RPC Requests.

Thus, to stop all the agents, less than 1.1% of the ICS vulnerabilities have to be patched. The manager computes the first set of countermeasure to deploy in 4 iterations and the second one in 10 iterations.

### 6 CONCLUSION

This paper has outlined Haruspex, an integrated set of tools to assess and manage ICT risk. These tools use scenarios and Monte Carlo simulations to model the behavior of intelligent and goal oriented agents, that escalate their privileges by composing attacks against one or several nodes of the target infrastructure. The paper has also described the adoption of Haruspex to assess and manage the ICT risk of an ICS that controls a real turbo gas power production plant. Among the outputs of the assessment, we recall the discovery of the most dangerous agents and of the time each agent takes to successfully attacks the ICS. The haruspex tools to manage the risk have computed a cost effective set of countermeasures. The effectiveness of these countermeasures is confirmed by the low number of vulnerabilities, 41, to be patched to guarantee that no agent can control power production. To run all experiments, the tools have used 180 hours of machine time. Most of this time is due to the engine and the manager. Each experiment has taken from 4 to 6 hours on an highly parallel IBM system with 96 cores in 16 clusters. The tools can exploit at best the available cores. For example, the engine maps distinct runs of an experiments onto distinct cores and the manager computes alternative coverages of a set of plans through threads mapped onto distinct cores. Future developments of this work concerns the assessment of most complex infrastructures, such as ICSs to control and manage smartgrids. Another development concerns the modeling of malware.

### REFERENCES


MITRE. CWE - common weakness enumeration. Technical report.


