Iterative Selection of Cost-Effective Countermeasures for Intelligent Threat Agents

F.Baiardi, F.Tonelli, A. Bertolini, R. Bertolotti
Dipartimento di Informatica, Università di Pisa, Pisa, Italy
Email: {haruspex}@di.unipi.it

Abstract—We describe the tools of the Haruspex suite to manage the risk due to intelligent agents. The suite applies a Monte Carlo method and it return a statistical samples on the attacks these agents implement. Some tools of the suite analyzes these samples to select the countermeasures to deploy. The tools work in an iterative way that alternates the selection of countermeasures and the application of the Monte Carlo method. This takes into account that an intelligent agent may select distinct attacks to replace those affected by the countermeasures. Lastly, we apply the tools to three industrial control systems.

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

I. INTRODUCTION

Intelligent threat agents, or agents, chain the attacks enabled by the system vulnerabilities into sequences to reach some pre-defined set of privileges. This results in a privilege escalation [1] that uses the privileges granted by the previous attacks to implement the following ones till collecting all the privileges in a goal. The escalation may involve distinct network nodes.

Haruspex is an integrated set of tools to support a probabilistic risk assessment of an ICT system. The suite adopts a scenario based approach where each scenario includes the target system and some agents. Some Haruspex tools build the models to describe a scenario. Another tool, the engine, uses these models to apply a Monte Carlo method that runs multiple simulations of the agent attacks and to collect statistical samples. Two suite tools, the manager and the planner, use these samples to select cost effective countermeasures.

The paper is structured as follows. Sect. 2 briefly reviews related works on vulnerabilities, attacks, agents and their simulation. Sect. 3 presents the tools of the suite that produce the information to assess a system. Sect. 4 describes the manager and the planner and it outlines the algorithm we have devised to select cost effective countermeasures. Sect. 5 exemplifies the application of all the suite to manage the risk posed by three versions of an industrial control system to supervise a power generation plant. Lastly, sect. 6 draws some conclusions and outlines some future works.

II. RELATED WORKS

We review previous works on attacks, plans, their description, and countermeasures. Then, we recall the Haruspex tools related to those described in this paper.

[2]–[7] analyze the simulation of attacks against ICT infrastructures. Intelligent, goal oriented agents have been analyzed with reference to terrorism [8], [9], [10] presents a formal model of plans similar to the one of the Haruspex suite. [11] considers goal oriented attackers. [12] describes the prerequisites and the effects of an attack and pairs it with the proper countermeasure. [8], [13], [14] describe how the deployment of countermeasures affects the behavior of threat agents. [15]–[17] models countermeasures and hardening through attack graphs. [18], [19] discuss the selection of countermeasures for control systems.

Countermeasure selection is related to the sequence of attacks of an agent. [20] analyzes sequences of attacks involving distinct computing nodes of an infrastructure and it is focused on the compromised level of a node. The resulting approach does not enumerate all sequences. M2D2 is a formal data model for IDS alert correlation [21]. [22] supports the discovery of the sequences of attack of each agent but it analyzes each sequence in isolation and it neglects the probability the agent selects it.

The tools of the Haruspex suite we describe in this paper are applied after assessing the risk through three other tools: the builder, descriptor and the engine [23], [24]. The builder returns a model of the target system. Starting from the vulnerabilities that affects the system components, the model describes the attacks that enable the agents to illegally acquire some privileges, e.g. access rights or rights to invoke some operations. Each vulnerability may be known or suspected. Haruspex pairs each suspected vulnerability with the probability it is discovered at time $t$. The pre and the post conditions of an attack at describe, respectively, the rights to implement at and those that at grants if it is successful. The descriptor is the tool that returns the model of a threat agent. A threat agent, or agent (ag), is an attacker 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 we pair it with a loss for the owner of $S$ that occurs when, and if, $ag$ owns these rights. For each agent, the user supplies to the descriptor information on its goals, the resources it can access, the operations $ag$ is entitled to invoke, and the strategy it applies to select the attacks. This strategy models the preferences and the priorities of $ag$ [25].

The engine uses the models of the system and of the agents in a scenario to run an experiment that consist of a set of independent runs. These runs simulate, for the same time interval, the discovery of suspected vulnerabilities and the agent attacks. At the end of each run, the engine produces a sample. This sample records, among others, the sequence of attacks of each agent, the goals it has reached, the time to
reach a goal, and the number of executions and failures of an attack. Then, the engine reinitializes the state of both S and the agents and starts a new run. The confidence level of statistics computed through an experiment depends upon the number of runs because each run returns exactly one sample. In an experiment, the engine starts a new run until reaching the required level. To minimize the time of an experiment, a highly parallel version of the engine runs on an IBM system with 96 cores.

III. HARUSPEX SUITE: SELECTING COUNTERMEASURES

This section describes the planner and the manager, the tools to select countermeasures. For simplicity, we assume that a scenario includes one agent ag with one goal g. Extensions to a set of agents and/or of goals are straightforward. In the following, we assume that risk is a function of the impacts of the agents and of the probability they occur.

A. Discovering Plans

The planner analyzes the sample database to discover the plan corresponding to each sequence the agent has implemented. By focusing the selection of countermeasures on the plans we increase cost effectiveness by deploying countermeasures only for attacks useful to reach g.

The planner maps a sequence s of attacks to reach g into the plan p(s,g) through a backwards scan of s that copies into p(s,g) the attacks useful to achieve g. In the following, we denote by n the length of s, by s(i) the i-th attack of s and by tp(s,g) a sequence initialized with s(n)

Let useful(i) be the the rights ag needs before executing s(i) to achieve g. Initially, useful(n) = pre(s(n)) ∪ (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 the last attack in s, the useful rights are those to execute this attack and those in the goal that it does not grant.

Our algorithm considers useful(j) and it 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 post(at) ∩ useful(j), i.e. each right 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))) ∪ 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. It always return the correct plan provided that s does not interfere more than one plan.

To consider that distinct sequences may corresponds to the same plan, after discovering the plans of ag, the planner computes the success probability of each plan. This probability is the percentage of runs where ag has successfully implemented it.

B. Selecting Countermeasures for a Set of Plans

The manager determines the countermeasures to deploy starting from the plans of ag that the planner returns. An input of the tool, lowrisk, defines the highest success probability of ag accepted by the owner of S.

In the following, we assume that for any attack at there is a countermeasure to decrease its success probability. As an example, the patching of a vulnerability results in the failure of any attack it enables, while a password with non numerical characters decreases the success probability of a dictionary attack. c(at) denotes the cost of a countermeasure for at.

After running a first experiment, the manager has an iterative behavior where, after selecting some countermeasures, it runs another experiment to discover how ag reacts to countermeasure, e.g. which plans ag implements against the new system version 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. We denote these plans as dependent ones because ag selects them only when some countermeasures affect the other ones. Only a new experiment can discover dependent plans because ag neglects them in the previous ones. If, in this experiment, dependent plans have success probability larger than lowrisk, the manager starts a new iteration. Otherwise, the manager terminates and returns the countermeasure selected in the last iteration.

The number of countermeasures the manager returns depends upon the plans it considers at each iteration. In a global approach, at the i-th iteration the tool deploys countermeasures affecting a set Si of plans that depends upon those that ag implements in the previous iterations. Hence, these countermeasures may completely differ from those selected in a previous iteration. In the incremental approach, at each iteration the manager inserts in the countermeasure to deploy those selected for the dependent plans discovered in the previous iteration.

Currently, the manager adopts a global approach where Si includes all the plans considered in the previous iterations and a subset, Cp0, of the plans ag executes in the i − th iteration. We insert plans into Cp0 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 heuristics 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 agents that execute a large number of plans by bounding the size of Cp0 as a fixed percentage of successful plans.

C. Countermeasures Selection

Since a countermeasure for at affects all the plans sharing at, we minimize the number of countermeasures for some plans by focusing the selection on countermeasures for the attacks shared among these plans.

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 \( Sp(\text{at}) = \{i_1, ..., i_k\} \), the set of the indexes of the plans in \( Sp \) that share at. Now, we can determine a set of countermeasures for all the plans in \( Sp \) with the lowest cost by computing a coverage \([26]\) with the minimal cost of \( \{1, ..., n\} \) through the elements of \( Sa = \bigcup_{\text{at} \in Sp} \text{at} \). The computation of a coverage neglects an attack \( at_1 \) if all the plans that share \( at_1 \) also share \( at_2 \), a distinct attack with a cheaper countermeasure. The manager removes these attacks, considers any possible coverage and selects the cheapest one. Even if the coverage problem is NP-Complete, the resulting execution time is acceptable because the number of distinct plans is, in general, very low. In fact, if the number of distinct plans is very large and the plans share a low number of attacks, then \( ag \) can attack \( S \) through a large number of distinct plans. This requires an extensive redesign of \( S \) rather than deploying some countermeasures. To further reduce execution time, we abort the building of a set as soon as its cost exceeds the current best.

The adopted solution exploits at best shared attacks to minimize countermeasures. Instead, the incremental approach cannot anticipate the dependent plans the agent implements in the following iterations and the attacks they share with those implemented in the current one. As a counterpart, the incremental approach minimizes the number of plans that each iteration considers.

The computation of coverage handles the lack of countermeasures for some attacks by pairing each attack with a countermeasure with an infinite cost. If the manager returns a coverage with an infinite cost, then there is at least one plan with only attacks with no countermeasure. The success probability of this plan is a lower bound on the success probability of \( ag \).

IV. CASE STUDY AND EVALUATION OF RESULTS

We have applied the suite to three 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 interface the intranet users. The operators manage the SCADA system through nodes in a Power Context subnet. The SCADA servers and clients act as the supervision and control system of power production and they belong to a Process subnet. Finally, the PLC systems in a Control subnet control some devices in the plant. The first version of the infrastructure merges the Power Context subnet and the Central one.

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 a Process subnet. Lastly, two nodes in each Process subnet connect to the corresponding Control subnet. To increase the complexity of the study and to properly stress the suite tools, we have inserted further vulnerabilities in some nodes besides those returned by the scanning. In the second infrastructure, the Central subnet is segmented into two subnets, with 24 nodes. Lastly, in the third infrastructure, also the Central subnet is segmented into three subnets with 16 nodes.

A. Selecting Countermeasures to Deploy

In any version, all the agents initially own some rights on the Central Network and no information on other networks. We have defined the other agent attributes to cover all the cases of interest. The 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 have introduced one agent for each strategy that Haruspex supports and for each combination of the strategy parameters In this way, the whole assessment considers 28 agents. The confidence level of the experiments is 95% on the components that an agent attacks in its plan. This requires from 150.000 to 500.000 runs in each experiment.

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. This agent selects attacks according to both the time they take and their success probability.

<table>
<thead>
<tr>
<th>Iter</th>
<th>N. of plans</th>
<th>N. of patch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Table I. DETAIL OF MANAGER ITERATIONS: FIRST VERSION

<table>
<thead>
<tr>
<th>Iter</th>
<th>N. of plans</th>
<th>N. of cntrm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>563</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>271</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>275</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>673</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>688</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>213</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>109</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>109</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Table II. DETAIL OF MANAGER ITERATIONS: SECOND VERSION
Furthermore, it anticipates the effect of the two next attacks. For each infrastructure, we consider the countermeasures the manager selects this agent under the assumption that for each vulnerability there is a vulnerability that causes the failure of the enabled attacks.

For each iteration, the tables I-III show the number of plans of $ag$ and the one of countermeasures the manager selects. The manager finds an optimal set of countermeasures in, respectively, 5, 9 and 11 iterations.

Fig 2, 3 and 4 show the success probability as a function of the time $ag$ has available to reach its goals. The value at time $t$ is computed as the percentage of runs where $ag$ reaches its goal within $t$. Each figure shows how the countermeasures selected in the $i$-th iteration affect the success probability of $ag$. When no plan is successful, the curve overlaps the x axis.

V. Conclusion

We have outline the manager and the planner, the tools of the Haruspex suite that compute a cost effective set of countermeasures. The manager computes the countermeasure to deploy through a sequence of Haruspex experiments. Each experiment determines how the agents react to the countermeasure selected using the results of the previous ones. This takes into account that an agents reacts to countermeasures by changing its goal within $t$. Each figure shows how the countermeasures selected in the $i$-th iteration affect the success probability of $ag$.

REFERENCES


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

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

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