Assessing and Managing Risk by Simulating Attack Chains

F.Baiardi, F.Tonelli
Department of Computer Science,
University of Pisa
Pisa, Italy
[baiardi,tonelli]@di.unipi.it

A.D. Ruggiero Di Biase
VI Division – C4I Systems and Trasformation
Defence General Staff
Rome, Italy
sesto.cdo.sede@smd.difesa.it

Abstract—Haruspex is a suite of tools to assess and manage the risk posed by an information and communication technology system. The suite is built around the application of a Monte Carlo method to a scenario where intelligent agents implement chains of attacks to reach their goals. Some tools build a description of the agents, the target system, its vulnerabilities and the resulting attacks. Another tool applies a Monte Carlo method to this description, simulates the building of attack chains by the agents and it returns a database with samples it collects in the simulations. Further tools analyze this database to select countermeasures. To validate the suite and verify it truthfully models attackers, it has been adopted in Locked Shield 2014, a network defense exercise with participants from 17 nations. The results of this exercise validate the designs of the tools.

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

I. INTRODUCTION

We consider the probabilistic risk assessment and management of an information and communication technology, ICT, system where intelligent agents compose elementary attacks into attack chains. To evaluate the resulting risk, we consider several scenarios, each describing the target system and some agents and adopts a Monte Carlo method. For each scenario, the method implements multiple simulations of how the agents select and implement attacks in a time interval. In each simulation samples of events of interest are collected to build a sample to compute statistics of interest. The tools in the Haruspex suite support an assessment by:

- describing the target system and the agents,
- applying the Monte Carlo method,
- selecting countermeasures.

After presenting the suite, we discuss its validation in Locked Shield 2014, a real-time network defence exercise organized by the NATO Cooperative Cyber Defence Centre of Excellence and where 12 defending teams were put against one attacking team. This exercise is a unique opportunity to validate the overall suite. In fact, we have assessed the system, deployed the countermeasures and evaluated how they improve the system robustness with respect to the attacking team. It is seldom the case that information on real attackers is available to evaluate the accuracy of simulations.

Sect. II of the paper reviews related works on vulnerabilities, attacks, agents and their simulation. Sect. III presents the suite tools and introduces those that select countermeasures. Sect. IV describes the suite validation.

We briefly review previous works on attack, vulnerabilities, their description, risk assessment and management.

II. BACKGROUND


Most works that discuss attack chains neglect their automated discovery [2]. [23] analyzes attack chains involving distinct nodes but it cannot discover all the chains. [22] computes the success probability of a complex attack without considering its relation with the agent selection strategy. [24] and [25] discuss attack chains and countermeasures.

III. THE HARUSPEX SUITE

Here and in the following, assessment is a synonymous for probabilistic assessment and right is a shorthand for access right. The adopted definition quantifies risk [26] in terms of the loss for the system owner and of the probability of this loss.
A. An Haruspex Scenario

A scenario includes $S$, the target of the assessment, and some or one agent that attack $S$. We outline the description of a scenario before defining the tools to build it.

System Components. $S$ is decomposed into interconnected components. Each component is a module described in terms of its operations, its vulnerabilities and the elementary attacks these vulnerabilities enable. To describe social engineering attacks users of $S$ are modeled as further components [27]. The security policy of $S$ defines the rights of users and components. Each right enables the owner to invoke one operation.

Vulnerabilities. Each vulnerability enables some elementary attacks, or attacks, that grant some rights to an agent. The suite considers both public and suspected vulnerabilities. The latter is paired with the probability of being discovered at a given time. Haruspex deduces public vulnerabilities from the vulnerability database provided by NIST.

Elementary Attacks. Haruspex does not define the actions of an elementary attack $at$ and pairs it with some attributes [1]:
- $v(at)$, the vulnerabilities that enable $at$,
- $succ(at)$, the success probability of $at$,
- $pre(at)$, the rights to execute the actions of $at$,
- $res(at)$, the resources to execute the actions of $at$,
- $post(at)$, the rights acquired if $at$ is successful,
- $time(at)$, the time to execute the actions of $at$.

An agent can implement $at$ if it owns the resources in $res(at)$ and the rights in $pre(at)$ that it may acquire through previous attacks. If vulnerabilities in $v(at)$ are public, $at$ is enabled and succeeds with a probability $succ(at)$ otherwise it always fails.

Threat Agents. Any agent $ag$ is a user of $S$. The attributes of $ag$ define the resources it can access, its initial rights, i.e. the operations it is entitled to invoke, and its goals. Any goal $g$ is a set of rights that $ag$ reaches after acquiring the corresponding rights. $g$ is paired with an impact, a loss that the owner of $S$ pays per each unit of time $ag$ owns the rights in $g$.

Selection Strategies. To reach any goal, $ag$ selects and implements an attack chain. A sequence of attacks is a chain if both any attack in the sequence increases the rights of $ag$ and the first $j$-1 attacks grant the rights in the precondition of the $j$-th one. $ag$ applies its selection strategy to discover the best attack to reach its goals according to the rights it currently owns and the vulnerabilities in the components of $S$. To minimize its effort to reach a goal, $ag$ aims to implement a plan, a chain where any grants some rights useful to reach the goal. However, this only happens if $ag$ can access complete information on $S$ and properly exploits it.

The look-ahead of $ag$, $\lambda(ag)$, is the parameter that determines how $ag$ selection strategy exploits available information on $S$. This strategy ranks a set of chains and returns the first attack of the best chain. The set includes all the chains with, at most, $\lambda(ag)$ attacks provided that $ag$ can execute the first attack of the chain. If no such chain exists, $ag$ is idle till some suspected vulnerability is discovered. If any chain grants the rights in a goal, then its first attack is returned. If no chain leads to a goal, then the ranking considers other attributes and it may return a useless attack. However, larger values of $\lambda(ag)$ may slow down the selection.

B. Describing the Target System

To build a description of $S$, Haruspex runs a vulnerability scan of each node. Then, the builder is the suite tool that computes the description of $S$ starting from the output of the scans and the logical topology of $S$. Initially, the builder classifies the vulnerabilities to determine the attributes of each attack $at$. In particular, the tool computes $pre(at)$ and $post(at)$ and if $at$ may be remotely exploited [2]. To simplify the implementation of the selection strategy, the builder considers how elementary attacks can be composed into chains. In particular, it assumes that each chain results in a global escalation. A global escalation is a sequence of privilege escalations, where each escalation involves a distinct node of $S$. The sequence begins in a node $n_i$ where $ag$ owns some rights on the components in $n_i$ and it ends in a node $n_f$ with some components $ag$ aims to control. The sequence includes further nodes if the logical topology prevents $n_i$ from attacking $n_f$.

The builder does not consider the vulnerabilities that cannot be discovered by the scanning. Furthermore, it constrains an agent privilege escalation by assuming that only an agent that fully controls, e.g. is an administrator of, a node can launch an attack from it. To model social engineering, the corresponding vulnerabilities and attacks have to be manually inserted into the description the builder returns.

C. Describing the Threat Agents

The descriptor is the tool to describe the agents in a scenario. For each agent $ag$, the user defines the initial rights, the goals, the resources, the selection strategy and $\lambda(ag)$. If $\lambda(ag)=0$, $ag$ can only adopt the random strategy that selects, with the same probability, each attack $ag$ can implement. Otherwise, $ag$ can adopt one of

- maxProb: ranks chains according to their success probability,
- maxIncr: ranks chains according to the rights they grant,
- maxEff: ranks chains according to the ratio between success probability and execution time.

D. The Simulation Engine

The engine executes an experiment with several independent runs. Each run simulates, for the same time interval, the agents attacks and the discovery of suspected vulnerabilities. An experiment returns a database with the samples collected in the various runs.
At each time step of a run, the engine determines which suspected vulnerabilities are discovered. Then, the engine considers each idle agent, applies a selection strategy and, it executes the attack at it returns. If at is not enabled it immediately fails and ag is idle. Otherwise, ag will be busy for the next time(at) steps plus the time to select at. At the end of this interval, if at has succeeded, the engine checks if ag has reached a goal and updates ag impact ag repeats a failed attack for, at most, nr(ag) times before invoking the selection strategy. nr(ag) is a further attribute of ag.

The engine assumes that ag scans a node n to discover its vulnerabilities. This requires a time depending upon n. n is scanned the first time ag ranks a chain with an attack enabled by a vulnerability of n. Hence, larger values of λ(ag) result both in a better visibility and in a slower selection due to a larger number of nodes to be scanned. To model insiders, ag can know some nodes before starting its attacks.

The engine returns a database with samples on the agent attacks, the goals they have reached and the confidence level for some predefined statistics. In the latter case, the engine starts a new run until reaching this level.

E. Countermeasure Selection

The planner and the manager are the tools that drive the selection of countermeasure. They assume that for some attacks there is a countermeasure to decrease its success probability. As an example, the patching of a vulnerability results in the failure of any elementary attack it enables, while longer passwords or longer encryption keys decrease the probability an agent guess them.

For the sake of simplicity, we assume that a scenario includes one agent ag with one goal g.

The planner analyzes the engine output to discover the plans ag to achieve g and their success probabilities. To this purpose, it removes from each chain useless attacks. This increases the cost effectiveness of the selection by avoiding countermeasure for useless attacks. The planner computes the success probability of each plan p as the ratio between the number of runs where ag has implemented p and the number of runs in the experiment.

Starting from the output of the planner, the manager determines the countermeasures to reduce the risk due to S. A user input, hisu, defines the largest success probability of ag that can be accepted.

The manager works in an iterative way. After selecting, as described in the following, some countermeasures, it updates the description of S to model their deployment. Then, it runs an experiment with the new description to discover any plan that ag neglects when attacking S but that can be successfully implemented against the new version of S because it has not been affected by the countermeasures. Any of these plans is denoted as a dependent one because ag selects it only when those it previously selected are affected by countermeasures. If the new experiment shows that dependent plans have a success probability larger than hisu, the manager selects and deploys further countermeasures, runs another experiment and so on.

Since a countermeasure for at affects all the plans that implement it, the selection of countermeasure for a set of plans is focused on the attacks these plans share to reduce both the number of countermeasures and their overall cost.

To this purpose, given a set of plans Sp a set of countermeasures affects all the plans in Sp if it is a coverage [28], i.e. if it includes a countermeasure for at least one attack in each plan

IV. VALIDATING THE SUITE

The utility and the accuracy of the suite outputs fully depend upon the ability of the tools to mimic an accurate way how the agents select and implement their chains. We have validated the suite in the Locked Shields 2014 exercise as described in the following.

A. Locked Shield 2014

Locked Shields is a real-time network defense exercise, organized by the NATO Cooperative Cyber Defence Centre of Excellence, which involved participants from 17 nations. In the exercise, 12 defending teams were put against one attacking team, the red team.

The exercise scenario placed the teams in a fictional country of Berylia which industry fell under increasing cyber attacks. Day 1 started with low-level hacktivist campaigns and led to espionage and sabotage attacks against the networks of the defenders by the end of day 2. In addition to technical defense, the exercise includes a number of additional tasks such as legal assignments and forensics challenge, to make the exercise as lifelike as possible. The exercise is built up as a competitive game where the defending teams are scored based on their performance. Although the defending teams are competing with each other, the exercise encourages information sharing and cooperation.

The teams were participating from their home countries; exercise control was located in Tallinn, Estonia.

B. Using the Suite in the Exercise

One of the team involved in the exercise has used the suite to analyze the system to be defended and select vulnerabilities to be patched. Patching before the red team begins its attacks is the only countermeasure in the exercise. Since the time to patch is rather low, a further constrain the Haruspex manager considers to select vulnerabilities to patch is the time to apply the various patches. Here, the cost function to be minimized in the exercise considers the time to patch.

As previously said, to use the suite first of all we have to scan the nodes to be defended. This conflicts with the rule of the exercise about unknown nodes, e.g. a defender team cannot scan some of the nodes to defend. In other words, the team has to defend some systems it does not known and that could store some malware that attack some other nodes. This rule contradicts some of the basic Haruspex axioms and we
have handled it under the worst-case assumption that the attacker team fully controls these nodes.

To apply the Haruspex suite, we have considered one agent that has as its goals the system more valuable to the defender. To fully exploit the information this agent acquires about the system we have assumed its lookhead is equal to 3. We have run the experiment under the previous assumptions and then supplied its output to the manager. The manager returns a list of vulnerabilities to patch before the red team begins its attack. The team member manually applies the patches.

The system to be defended is affected by more than one thousand vulnerabilities and the list returned by the manager includes about 2% of these vulnerabilities.

C. Results of the Exercise

The team that has applied the Haruspex suite has scored the best results as far as concerns network defense in the exercise. Obviously this is the result of several team abilities, such as the one to deploy the patches and to monitor the network status. Hence, it should not be attributed to Haruspex only: however we believe that the good result in network defense confirms that the modeling of agent is realistic and accurate. As a consequence, the results of an Haruspex experiment can drive the selection of effective countermeasures.

V. CONCLUSION

This paper has discussed the design and the implementation of a suite of tools to support the risk assessment and management of an ICT system. The current version of this suite has been validated by adopting it in a defence exercise. The results of this adoption confirm that the suite can model the behavior of intelligent attackers.

Future development concerns the design and implementation of tools that automatically patches vulnerabilities according to a scheduling returned by the tools of the suite and the development of tools to assess a system that is currently designed.

REFERENCES