Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 337 (2018) 135–154

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Stochastic-based Semantics Of Attack-Defense Trees For Security Assessment

Karim Lounis[1,2]

Interdisciplinary Centre for Security, Reliability and Trust (SnT)
University of Luxembourg
Luxembourg, Luxembourg

Abstract

Losses caused by cyber-attacks are considerably increasing each year. The need for an optimal security assessment methodology that combines mathematical foundations with practical and user-friendly representations has become imperatively crucial. In this paper, we propose a stochastic security assessment methodology that adopts attack-defense trees model to represent security scenarios, and stochastic models to perform both qualitative and quantitative assessment. We illustrate our approach with a simple but realistic example study.

Keywords: Graphical Security Models, Attack-Defense Trees, Attack-trees, Security Assessments, CTMCs, Stochastic Petri-nets.

1 Introduction

Cyber-attacks are becoming more and more complex, distributed, sophisticated, and organized. Most businesses and industries are well aware of the figurative need to have a financially affordable, reliable, and powerful security assessment methodology that provides visualization of security scenarios, and conveys security information to non-experts in a very intuitive way. Over the last decade, a great research effort has been devoted to design the best security assessment methodology, where several graphical security models have been proposed in the literature [3,4,10]. The optimal methodology should combine mathematical foundations with practical and intuitive representation features. We found the attack-defense trees methodology [3] to be a favorable candidate which responds to such criteria.

[1] The research leading to the results presented in this work received funding from the European Commission's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 318003 (TREsPASS) and Fonds National de la Recherche Luxembourg under the grant $C13/IS/5809105$ (ADT2P).

[2] Email: karim.lounis@uni.lu

https://doi.org/10.1016/j.entcs.2018.03.038

ADTrees (Attack-Defense Trees) [3] are defined as a formal, and graphical methodology used for representing security scenarios by systematically enumerating the different actions that an attacker might undertake to perform a security breach, as well as the different actions that a defender might apply to stop, mitigate or delay attacker's actions from being successfully realized. The existing semantics for ADTrees (e.g Propositional, De Morgan Lattice, Equational, and Multisets) are purely denotational and do not express any notion of execution, or actions ordering. Therefore our main and first contribution will rely on proposing an operational semantics based on stochastic Petri-nets for ADTrees in order to be able to handle aspects like action ordering and dependencies.

Moreover, there is a considerable paucity of qualitative underpinnings supporting ADTrees analysis. Thus, our second contribution in this work will consist in interpreting the behavioral qualitative properties of Petri-nets in the context of security in general, and ADTrees in particular. Thanks to the isomorphic relation between finite continuous time Markov chains and bounded stochastic Petri-nets [16], we can perform both quantitative and qualitative security analysis by shifting from one model to another. In fact, to perform qualitative security analysis, we apply the well-known structural and behavioral analysis of Petri-nets. For quantitative analysis, we generate the corresponding finite CTMC (Continuous Time Markov Chain) of a given bounded Petri-net model and apply the classical analytical approach for probabilistic and time-based security analysis. Finally, to our knowledge, ADTrees do not specify any refinement operator describing actions to be performed in parallel. Therefore, we propose a new refinement operator expressing parallel actions, which we call disjunctive parallel refinement. In addition to that, countermeasures in ADTrees are quite abstract and general from the execution aspect, inspired by [18], who proposed one class of countermeasure called delayed-type countermeasure, we propose an other class of countermeasures that we call blocking-type countermeasures.

The main contributions of this paper are the four folds:

(i) We propose a stochastic operational semantics in terms of stochastic Petri-nets for attack-defense trees as well as attack-trees.

(ii) We extend the existing attack-defense trees' refinement operators with a new type of refinement operator expressing parallelism.

(iii) We extend the existing attack-defense trees' countermeasure types with a new type of countermeasure called blocking-type countermeasure.

(iv) We interprete the behavioral qualitative properties of Petri-net models in the context of security to perform qualitative analysis of attack-defense trees as well as attack-trees.

The remainder of the paper is organized as follows: in sections 2 we give an overview on attack-defense trees, and extend the formalsim with respect to refinements operator, and countermeasures. In section 3, we briefly present stochastic Petri-nets. In section 4, we introduce the operational semantics for attack-defense

trees, and provide formal framework for it. In section 5, we present then report the results of the qualitative and quantitative analysis performed on a simple but realistic example study. We discuss related works in section 6, and conclude the paper in section 7.

2 Basic concepts

Given that our goal is to improve the analysis of attack–defense scenarios, modeled using ADTree, in this section we first introduce the ADTree methodology, then extend the ADTree refinement specification. Finally, we present the concept of countermeasures in ADTrees, then propose a classification for it.

2.1 Attack–Defense Trees

We consider an attack–defense scenario to be a game between two players, the proponent (denoted by p) and the opponent (denoted by o), and represent the scenario using Attack–Defense Trees. An ADTree is a node-labeled rooted tree where the root node represents the main goal of the proponent. When the root of an ADTree is an attack node, the proponent is an attacker and the opponent is a defender. Conversely, when the root is a defense node, the proponent is a defender and the opponent is an attacker. The children of a given node represents its refinement into sub-goals. The refinement of a node is typically either disjunctive, conjunctive. or sequential-conjunctive. The goal of a disjunctively refined node is achieved when at least one of its children's goals is achieved and the goal of a conjunctively refined node is achieved when all its children's goals are achieved. Finally, a sequential-conjunctive refinement allows us to model that a certain goal is reached if and only if all its subgoals are reached in a precise order. Each node can have one child of the opposite type, representing the node's counteraction, which can be refined and countered again. Thus, an attack node may have several children which refine the attack and one child which defends against the attack. The defending child in turn may have several children which refines the defense and one child that is an attack node and counters the defense. The leaves of an ADTree represent the basic actions which need not be refined any further. Graphically, the attacker actions are depicted with red circles, and defender actions with green squares. We depict a conjunctive refinement of a node by connecting the edges going from this node to its children of the same type with an arc. We depict attack nodes by circles and defense nodes by rectangles, as shown in Figure 1. Refinement relations are indicated by solid edges between nodes and countermeasures are indicated by dotted edges.

Example 2.1 Figure 1 illustrates an ADTree which models a security scenario in which one attacker tries to perform attacks on a LTE-network subscribers. The attacker runs two compound attacks in parallel, a denial of service over the infrastructure, and tries to remotely control the users equipments. We model this using a parallel disjunction refinement ($\widetilde{\vee}$). In order to launch the denial of service, the attacker conjunctively exhaust RAN connexions and ($\wedge$) overload the DNS infras-

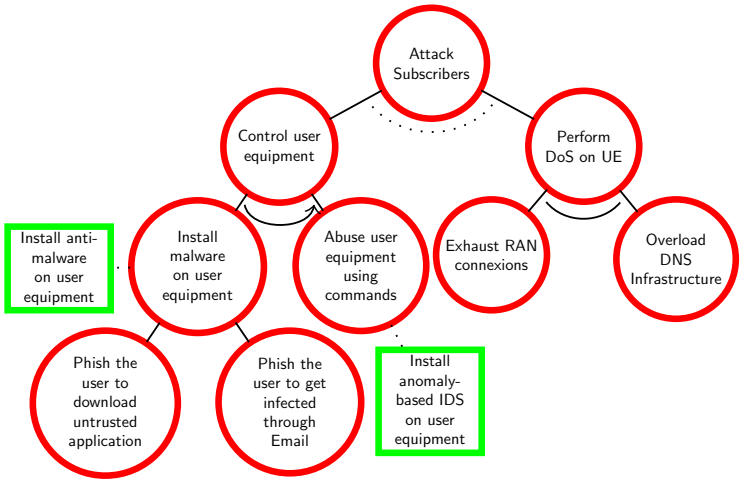

Fig. 1. Attack-Defense tree for attacking LTE-subscribers, where red circles are attacker actions, and green squares are defender actions

tructure. Meanwhile, to control the users equipments he sequentially tries to install a malware on the users equipments then ($\overrightarrow{\wedge}$) abuse the equipments by running some dangerous commands. To installing malware on users equipments, the attacker runs either phishing attack through application stores, or ($\vee$) phishing through emails. From the other side, the defender set up an anomaly-based IDS, which by detecting dangerous commands, turns the user equipments off preventing further damages (C_B). He also have a malware detector, which by detecting a malware signature deletes it (C_D).

We now introduce some notation in order to formally represent ADTrees. Let $\mathbb{S} = \{p, o\}$ be a set of types where $\overline{p} = o$ and $\overline{o} = p$, and $\vee_k^s$, $\wedge_k^s$ and $\overrightarrow{\wedge}_k^s$ be unranked functions representing the disjunctive, conjunctive, and sequential conjunction refinement operators respectively. The binary function c^s ($s \in \mathbb{S}$) connects an action of type s with action of the opposite type $\overline{s}$. Finally, let $\mathbb{B} = \mathbb{B}^p \cup \mathbb{B}^o$ where the elements of $\mathbb{B}^p$ and $\mathbb{B}^o$ be typed constants, which we refer to as the basic actions of the proponent (p) and the opponent (o) respectively. The signature of an ADTree can then be denoted as $\Sigma = (\mathbb{S}, \mathbb{F})$ where $\mathbb{F} = \{\vee_k^s, \wedge_k^s, \overrightarrow{\wedge}_k^s, c^s\} \cup \mathbb{B}$.

Definition 2.2 An Attack–Defense Tree is a typed-term defined over Σ, and generated by the following BNF-grammar, where $s \in \mathbb{S}$, and $k \in \mathbb{N}$:

$$t ::= b^s \mid \vee^s(t, \ldots, t) \mid \wedge^s(t, \ldots, t) \mid \overrightarrow{\wedge}^s(t, \ldots, t) \mid c^s(t, t)$$

2.2 Extending Attack-Defense Trees refinements

It has been widely recognized that parallel actions are crucial in security, as they constitute the main key of cyber-attacks performed on large scaled systems. However, abide the different existing types of refinements operator used in ADTrees, there is no refinement operator expressing parallel actions execution. Moreover, the use of only disjunctive refinement is not enough to capture such security scenarios.

Recent works [5, 9, 12, 13] considered disjunctive refinement as a refinement operator where an attacker has to execute at least one refinement action among many, which means that the attacker may have all the refinements executed. This sounds to be a kind of parallel refinement operator, however, we define the disjunction as a choice operator, where among all refinements, only one will be executed by the attacker. In fact, the disjunction operator assumes the refinement actions to be mutually exclusive.

Therefore, we introduce a new type of refinement operator to represent parallel execution of actions. We name this operator parallel disjunction refinement. This new type of refinement will allow an attacker/defender to perform more than one action at the same time, expecting that the successful termination of at least one action is enough to proceed to the next action. We use dots to graphically represent this type of refinement.

Definition 2.3 Let $k \in \mathbb{N}$, let $t_1, \ldots, t_k \in \mathbb{T}_\Sigma$, and let $\wedge$, and $\vee$ be the *conjunction* and *disjunction refinement operators*. Then a *parallel disjunction operator* is a refinement operator defined by an unranked function $\widetilde{\vee}^s_k$, such that:

$$\widetilde{\vee}^s(t_1, \ldots, t_k) = \widetilde{\vee}^s(t_1, \ldots, t_{k-1}) \vee (t_k \wedge \widetilde{\vee}^s(t_1, \ldots, t_{k-1})) \vee t_k$$

Example 2.4 Let us consider three basic actions $b_0, b_1, b_2 \in \mathbb{B}$ that a player plan to run in parallel. The goal of the player will be achieved if at least one of the following successful combination happens $\{b_0, b_1, b_2, b_0b_1, b_0b_2, b_1b_2, b_0b_1b_2\}$.

2.3 Countermeasures in Attack-Defense Trees

An ADTree consists of two types of nodes: one representing attacker's actions and the other corresponding to the defender. The original definition of ADTree [3] does not distinguish between various types of defense nodes. Relatively recently, to improve modeling capabilities and security analysis, studies have considered classification of defense nodes. For example, [5, 9, 12, 13] consider defenses that instantaneously negate the effect of an attack whereas [18] study defenses that delay the success of an attack. The former have been considered largely implicitly where the corresponding attacker's action are removed from the model. In this paper, we distinguish between the following two types of defense nodes.

(i) *Delaying-type* defense node: Similarly to [18], we consider these types of countermeasures to model situations where the defender, for example, changes the IP address of a host to act against a scanning attack. The attacker in this case is forced to re-perform the reconnaissance step of her attack.

(ii) *Blocking-type* defense nodes: These types of countermeasures are a generalization of the ones proposed in [5, 9, 12, 13]. An attacker is prevented (blocked) from executing an action if the corresponding defense is either in place or applied successfully before the attacker's action is completed. For example, an attacker cannot transfer files via ftp if the defender has stopped running the ftp service on the target host. In contrast to the delayed-impact countermeasures, this type of defenses do not allow the attacker to re-perform the same attack

is forced to choose an alternative.

Let c_D^s and c_B^s refer to the delayed-impact and blocking-impact countermeasures respectively. The signature of an ADTree (see Definition 2.2) can then be defined as $\Sigma = (\mathbb{S}, \mathbb{F})$ where $\mathbb{F} = \{\vee_k^s, \wedge_k^s, \overrightarrow{\wedge}_k^s, \widetilde{\vee}_k^s, c_D^s, c_B^s\} \cup \mathbb{B}$ and the ADTree can be defined as follows.

Definition 2.5 An Attack–Defense Tree is a typed-term defined over Σ, and generated by the following BNF-grammar, where $s \in \mathbb{S}$, and $k \in \mathbb{N}$:

$$t ::= b^s \mid \vee^s(t, \ldots, t) \mid \wedge^s(t, \ldots, t) \mid \overrightarrow{\wedge}^s(t, \ldots, t) \mid \widetilde{\vee}^s(t, \ldots, t) \mid c_B^s(t, t) \mid c_D^s(t, t)$$

Example 2.6 By labeling the basic actions of the ADTree in Figure 1 as; 'Download untrusted application' b_0^p, 'Get infected by email' b_1^p 'Abuse UE' b_2^p, 'Exhaust RAN connexion' b_3^p, 'Overload DNS infrastructure' b_4^p, 'IDS anomaly detection' b_1^o, and 'Anti-Malware' b_0^o. The resulting ADTerm of the ADTree in Figure 1 is:

$$t = \widetilde{\vee}^p\left(\overrightarrow{\wedge}\left(c_D^p\big(\vee^p(b_0^p, b_1^p), b_0^o\big), c_B^p\big(b_2^p, b_1^o\big)\right), \wedge^p\left(b_3^p, b_4^p\right)\right)$$

3 Stochastic Petri Nets

Petri-nets are mathematical yet graphical models that are widely used to represent systems that exhibit concurrency, parallelism, synchronization and nondeterminism [20]. They have been used to study qualitative security properties such as the take-grant model [1] and lattice-based structures describing access control policies [2]. Several complex systems have been modeled [6,8] using various extensions of using Petri-nets [11,14–16]. To ensure that the semantics of ADTrees presented in this paper are consistent with the semantics in [18], we focus here on *bounded stochastic petri-nets* (SPNs) since they are isomorphic to finite continuous time Markov chains (CTMCs) [16].

Graphically, SPNs are composed of two main components: *places* that represent resources (depicted as circles ◯) and *transitions* that represent actions on resources (depicted by bars []). Places contain tokens that denote the number of resources available. Places and transitions are connected by directed arcs (→), where the places which feed a transition with tokens are defined as its *preset* and the places which are fed by a transition are defined as its *postset*. In a stochastic Petri-net, transitions are controlled by an exponentially distributed firing time i.e., each transition is attributed an exponential rate that determines the mean time before the transition is fired.

Definition 3.1 A stochastic Petri-net is a 6-tuple (P, T, I, O, M_0, Λ), where P is a *finite set of places*, T a finite set of transitions, $I = \{(p,t) \mid p \in P,\ t \in T\}$ a finite set of input arcs (preset), $O = \{(t,p) \mid t \in T,\ p \in P\}$ a finite set of output arcs (postset), M_0 the initial marking indicating the amount of tokens initially available in each place, and $\Lambda\colon T \to \mathbb{R}^+$ a function which associates an exponential rate with each transition.

4 Stochastic Petri-Nets semantics for ADTrees

In this section we define SPNs-based semantics for ADTrees. First, we formulate each element of an ADTree – basic actions $\mathbb{B}$, refinements $\vee_k^s$, $\wedge_k^s$, $\overrightarrow{\wedge}_k^s$, $\widetilde{\vee}_k^s$, and countermeasures c_D^s, c_B^s – in terms of a SPN. Then, we present an approach that provides the SPN representing the entire ADTree. This approach first models each leaf node representing a basic action in the ADTree using a 1-transition SPN. Following the standard bottom-up procedure [4], the SPN representing all other nodes are derived by composing the SPNs corresponding to their children. This recursive process terminates at the root node where an SPN that represents the entire attack–defense scenario modeled by the ADTree is obtained. The SPNs semantics of ADTrees can therefore be defined as follows.

Definition 4.1 Let $\mathbb{T}_\Sigma$ be the set of all ADTrees defined over the signature Σ, and let $\mathbb{T}$ be the set of all SPNs. The SPNs semantics $[\![\cdot]\!]_{SPN}\colon \mathbb{T}_\Sigma \to \mathbb{Y}$ is a function which provides a SPNs representation for each ADTree.

To enable fine-grained modeling of the elements of ADTrees, similarly to [18], we explicitly distinguish between three types of places within SPNs. We consider that each SPN has a set of initial places denoted by P_0, a set of transitive places denoted by P_t, and a set of final places denoted by P_*. The original definition of a stochastic Petri-net (see Definition 3.1) is then adapted in order to explicitly enumerate the set of places as $P = P_0 \cup P_t \cup P_*$ and the set of input and output arcs as $K = I \cup O$. In our context, the markings in a SPN is driven by the following principle: the number of tokens in a place denotes the number of actions that an agent (attacker/defender) can perform in that place.

In what follows, we denote by $y_1^s, ..., y_k^s \in \mathbb{Y}$ stochastic Petri-nets, and by $b^s \in \mathbb{B}$ basic actions for a given player $s \in \mathbb{S}$, and let Γ^P denotes a property that we call linkability. We write $(p,t) \models \Gamma^P$ or $(t,p) \models \Gamma^P$ to say that a pair $(p,t) \in I$, or $(t,p) \in O$ is indeed an input, or output arc in a given stochastic Petri-net with a given set of places P.

4.1 Semantics for basic actions

We start by defining the semantics of the basic actions of ADTrees. When $t = b^s$, where $t \in \mathbb{T}_\Sigma$, $b^s \in \mathbb{B}$ and $s \in \mathbb{S}$, the function $[\![\cdot]\!]_{SPN}\colon \mathbb{T}_\Sigma \to \mathbb{Y}$ interprets the attack tree t (irrespective of attacker's or defender's action) as a SPN having a single stochastic transition $t_b^{y^s}$ with rate $\lambda_b^{y^s}$. This SPN has two places – an initial place P_0 and a final place P_*. The initial marking is $(1,0)$ and denotes that agent s can execute one action b^s with rate $\lambda_b^{y^s}$ from place P_0. Successful execution of this action provides the final marking $(0,1)$ in the SPN. Formally, $[\![b^s]\!]_{SPN} = \Psi_0(b^s)$ such that:

$$\Psi_0(b^s) = (P_0 \cup P_*, \{t_b^{y^s}\}, \{(p_0, t_b^{y^s}), (t_b^{y^s}, p_*)\}, \{\lambda_b^{y^s}\}, (M_0(P_0), M_0(P_*))) \quad (1)$$

$$= (\{p_0, p_*\}, \{t_b^{y^s}\}, \{(p_0, t_b^{y^s}), (t_b^{y^s}, p_*)\}, \{\lambda_b^{y^s}\}, (1,0)) \quad (2)$$

Example 4.2 Let us consider an ADTree $t = b_1^s$, where $b_1^s \in \mathbb{B}^s$ is a basic actions.

Then applying the function $\Psi_0(b_1^s)$ will produce a SPN $y_1^s \in \mathbb{Y}$ depicted in Figure 2-a. The SPN is composed of 2 places, one initial p_0, and one final p_*, linked together using one transition of rate $\lambda_1 \in \mathbb{R}^+$.

4.2 Semantics for conjunction refinement

We define the semantics of conjunction refinement of ADTrees. Basically, when $t = \wedge^s(t_1, \ldots, t_k)$, where $t \in \mathbb{T}_\Sigma$, $t_{1 \leq i \leq k} \in \mathbb{T}_\Sigma$, $k \in \mathbb{N}$ and $s \in \mathbb{S}$. The function $[\![\cdot]\!]_{SPN}\colon \mathbb{T}_\Sigma \to \mathbb{Y}$ interprets the attack tree t (irrespective of the type of player) as a SPN composed using the Cartesian product of all SPNs $y_1, \ldots, y_k \in \mathbb{Y}$ representing the refinements $t_{1 \leq i \leq k} \in \mathbb{T}_\Sigma$. This composed SPN has $|P^{y_1}| + \ldots + |P^{y_k}|$ places, where $|P^{y_i}|$ is the cardinality of the set of place P of a given SPN $y_{1 \leq i \leq k} \in \mathbb{Y}$. The involved SPNs are joined together in such a way that all of them will be executed in an irrelevant order. The initial marking is $(1, \ldots, 0)$ and denotes that agent s can execute one of the refinements $t_{1 \leq i \leq k} \in \mathbb{T}_\Sigma$ from place P_0. Successful execution of all refinements (involved SPNs) provides the final marking $(0, \ldots 1)$ in the SPN. Formally, $[\![\wedge^s(t_1, \ldots, t_k)]\!]_{SPN} = \Psi_1(y_1^s, \ldots, y_k^s) = (P, T, K, \Lambda, M_0)$ such that:

$$
\begin{cases}
P = \prod_{i=1}^{k} P^{y_i^s} = \bigcup \left\{ \begin{array}{l} P_0 = \{(p_0^{y_1^s}, \ldots, p_0^{y_k^s})\} \\ P_t = P - \{(p_0^{y_1^s}, \ldots, p_0^{y_k^s}), (p_*^{y_1^s}, \ldots, p_*^{y_k^s})\} \\ P_* = \{(p_*^{y_1^s}, \ldots, p_*^{y_k^s})\} \end{array} \right\} \\
T = \{t_0^{y_1^s}, \ldots, t_k^{y_1^s}, \ldots, t_0^{y_{k'}^s}, \ldots, t_k^{y_{k'}^s}\} \\
K = \{(p,t) \models \Gamma^P\} \bigcup \{(t,p) \models \Gamma^P\}, p \in P, \textit{ and } t \in T \\
\Lambda = \{\lambda_0^{y_1^s}, \ldots, \lambda_k^{y_1^s}, \ldots \lambda_0^{y_{k'}^s}, \ldots, \lambda_k^{y_{k'}^s}\} \\
M_0 = (M_0(p_0), \ldots, M_0(p_*)) = (1, \ldots, 0)
\end{cases}
$$

Example 4.3 Let us consider an ADTree $t = \wedge^s(b_1, b_2)$, where $b_1^s, b_2^s \in \mathbb{B}^s$ are two basic actions, and let $y_1^s, y_2^s \in \mathbb{Y}$ be their corresponding SPNs with rates $\lambda_1, \lambda_2 \in \mathbb{R}^+$ respectively. Then applying the function $\Psi_1(y_1^s, y_2^s)$ will produce the SPN depicted in Figure 2-b. The SPN is composed of 4 places, one initial p_0, one final p_*, and two transient places p_1 and p_2.

4.3 Semantics for disjunction refinement

We define the semantics of disjunction refinement of ADTrees. Basically, when $t = \vee^s(t_1, \ldots, t_k)$, where $t \in \mathbb{T}_\Sigma$, $t_{1 \leq i \leq k} \in \mathbb{T}_\Sigma$, $k \in \mathbb{N}$ and $s \in \mathbb{S}$. The function $[\![\cdot]\!]_{SPN}\colon \mathbb{T}_\Sigma \to \mathbb{Y}$ interprets the attack tree t (irrespective of the type of player) as a SPN composed of k SPNs $y_{1 \leq i \leq k} \in \mathbb{Y}$ in such a way that only one SPN can independently succeed. This is realized by merging together the initial place of all involved SPN into one common initial place. The initial marking is $(1, \ldots, 0)$ and denotes that agent s can execute one of the refinements $t_{1 \leq i \leq k} \in \mathbb{T}_\Sigma$ from place P_0. Successful execution of the i^{th} SPN provides one final marking

$(0, \ldots, M_0(p_i), \ldots, 0)$, where $M_0(p_i) = 1$, and p_i is the final place of the i^{th} SPN. Formally, $[\![\vee^s(t_1, \ldots, t_k)]\!]_{SPN} = \Psi_2(y_1^s, \ldots, y_k^s) = (P, T, K, \Lambda, M_0)$ such that:

$$\begin{cases} P = \bigcup \left\{ \begin{array}{l} P_0 = \{(p_0^{y_1^s}, \ldots, p_0^{y_k^s})\} \\ P_t = \bigcup_{i=1}^{k} P_t^{y_i} \times \prod_{j \neq i} P_0^{y_j} \\ P_* = \bigcup_{i=1}^{k} P_*^{y_i} \times \prod_{j \neq i} P_0^{y_j} \end{array} \right\} \\ T = \{t_0^{y_1^s}, \ldots, t_k^{y_1^s}, \ldots, t_0^{y_{k'}^s}, \ldots, t_k^{y_{k'}^s}\} \\ K = \{(p,t) \models \Gamma^P\} \bigcup \{(t,p) \models \Gamma^P\}, p \in P, \text{ and } t \in T \\ \Lambda = \{\lambda_0^{y_1^s}, \ldots, \lambda_k^{y_1^s}, \ldots \lambda_0^{y_{k'}^s}, \ldots, \lambda_k^{y_{k'}^s}\} \\ M_0 = (M_0(p_0), \ldots, M_0(p_*)) = (1, \ldots, 0) \end{cases}$$

Example 4.4 Let us consider an ADTree $t = \vee^s(b_1, b_2)$, where $b_1^s, b_2^s \in \mathbb{B}^s$ are two basic actions, and let $y_1^s, y_2^s \in \mathbb{Y}$ be their corresponding SPNs with rates $\lambda_1, \lambda_2 \in \mathbb{R}^+$ respectively. Then applying the function $\Psi_1(y_1^s, y_2^s)$ will produce the SPN depicted in Figure 2-b. The SPN is composed of 3 places, one initial p_0, and two final p_1, and p_2.

4.4 Semantics for sequential conjunction refinement

We define the semantics of sequential conjunction refinement of ADTrees. Basically, when $t = \overrightarrow{\wedge}^s(t_1, \ldots, t_k)$, where $t \in \mathbb{T}_\Sigma$, $t_{1 \leq i \leq k} \in \mathbb{T}_\Sigma$, $k \in \mathbb{N}$ and $s \in \mathbb{S}$. The function $[\![\cdot]\!]_{SPN}$: $\mathbb{T}_\Sigma \rightarrow \mathbb{Y}$ interprets the attack tree t (irrespective of the type of player) as a SPN composed of k involved SPNs $y_{1 \leq i \leq k} \in \mathbb{Y}$ sequentially linked together one after the other. This is realized by merging together the final place of the $k-1^{th}$ SPN with the initial place of the k^{th} SPN. The initial marking is $(1, \ldots, 0)$ and denotes that agent s can execute one of the refinements $t_{1 \leq i \leq k} \in \mathbb{T}_\Sigma$ from place P_0. Successful execution of the all refinements (involved SPNs), in a sequential right-to-left order, provides the final marking $(0, \ldots 1)$ in the SPN. Formally, $[\![\overrightarrow{\wedge}^s(t_1, \ldots, t_k)]\!]_{SPN} = \Psi_3(y_1^s, \ldots, y_k^s) = (P, T, K, \Lambda, M_0)$ such that:

$$\begin{cases} P = \bigcup \left\{ \begin{array}{l} P_0 = \{(p_0^{y_1^s}, \ldots, p_0^{y_k^s})\} \\ P_t = \bigcup_{i=1}^{k-1} P_*^{y_i} \times P_0^{y_{i+1}} \bigcup P_t^{y_i} \times P_0^{y_{i+1}} \bigcup P_*^{y_i} \times P_t^{y_{i+1}} \\ P_* = \{(p_*^{y_1^s}, \ldots, p_*^{y_k^s})\} \end{array} \right\} \\ T = \{t_0^{y_1^s}, \ldots, t_k^{y_1^s}, \ldots, t_0^{y_{k'}^s}, \ldots, t_k^{y_{k'}^s}\} \\ K = \{(p,t) \models \Gamma^P\} \bigcup \{(t,p) \models \Gamma^P\}, p \in P, \text{ and } t \in T \\ \Lambda = \{\lambda_0^{y_1^s}, \ldots, \lambda_k^{y_1^s}, \ldots \lambda_0^{y_{k'}^s}, \ldots, \lambda_k^{y_{k'}^s}\} \\ M_0 = (M_0(p_0), \ldots, M_0(p_*)) = (1, \ldots, 0) \end{cases}$$

Example 4.5 Let us consider an ADTree $t = \overrightarrow{\wedge}^s(b_1, b_2)$, where $b_1^s, b_2^s \in \mathbb{B}^s$ are two basic actions, and let $y_1^s, y_2^s \in \mathbb{Y}$ be their corresponding SPNs with rates $\lambda_1, \lambda_2 \in \mathbb{R}^+$ respectively. Then applying the function $\Psi_3(y_1^s, y_2^s)$ will produce the SPN depicted in Figure 2-d. The SPN is composed of 3 places, one initial p_0, one final p_*, and one transient places p_1.

4.5 Semantics for parallel disjunction refinement

We define the semantics of parallel disjunction refinement of ADTrees. Basically, when $t = \widetilde{\vee}^s(t_1, \ldots, t_k)$, where $t \in \mathbb{T}_\Sigma$, $t_{1 \leq i \leq k} \in \mathbb{T}_\Sigma$, $k \in \mathbb{N}$ and $s \in \mathbb{S}$. The function $[\![\cdot]\!]_{SPN}$: $\mathbb{T}_\Sigma \to \mathbb{Y}$ interprets the attack tree t (irrespective of the type of player) as a SPN composed by the Cartesian product of all SPNs $y_1, \ldots, y_k \in \mathbb{Y}$ representing each attack tree $t_{1 \leq i \leq k} \in \mathbb{T}_\Sigma$ of the refinement. The involved SPNs are joined together in such a way that all of them will be executed in an irrelevant order. The initial marking is $(1, \ldots, 0)$ and denotes that agent s can execute one of the refinements $t_{1 \leq i \leq k} \in \mathbb{T}_\Sigma$ from place P_0. Successful execution of at least one refinements (involved SPNs) provides one final marking $(0, \ldots, M_0(p_i), \ldots, 0)$, where $M_0(p_i) = 1$, and p_i is a final place encompassing at least one final place of the k involved SPN. Formally, $[\![\widetilde{\vee}^s(t_1, \ldots, t_k)]\!]_{SPN} = \Psi_4(y_1^s, \ldots, y_k^s) = (P, T, K, \Lambda, M_0)$ such that:

$$\begin{cases} P = \bigcup \left\{ \begin{array}{l} P_0 = \{(p_0^{y_1^s}, \ldots, p_0^{y_k^s})\} \\ P_t = P - P_0 \bigcup P_* \\ P_* = \bigcup_{i=1}^{k} \bigcup_{j \neq i}^{k} P_*^{y_i^s} \times P^{y_j^s} \end{array} \right\} \\ T = \{t_0^{y_1^s}, \ldots, t_k^{y_1^s}, \ldots, t_0^{y_{k'}^s}, \ldots, t_k^{y_{k'}^s}\} \\ K = \{(p,t) \models \Gamma^P\} \bigcup \{(t,p) \models \Gamma^P\}, p \in P, \text{ and } t \in T \\ \Lambda = \{\lambda_0^{y_1^s}, \ldots, \lambda_k^{y_1^s}, \ldots \lambda_0^{y_{k'}^s}, \ldots, \lambda_k^{y_{k'}^s}\} \\ M_0 = (M_0(p_0), \ldots, M_0(p_*)) = (1, \ldots, 0) \end{cases}$$

Example 4.6 Let us consider an ADTree $t = \widetilde{\vee}^s(b_1, b_2)$, where $b_1^s, b_2^s \in \mathbb{B}^s$ are two basic actions, and let $y_1^s, y_2^s \in \mathbb{Y}$ be their corresponding SPNs with rates $\lambda_1, \lambda_2 \in \mathbb{R}^+$ respectively. Then applying the function $\Psi_3(y_1^s, y_2^s)$ will produce the SPN depicted in Figure 2-e. The SPN is composed of 3 places, one initial p_0, one final p_*, and one transient places p_1.

4.6 Semantics for blocking-type countermeasure

We define the semantics of the blocking-type countermeasure. Basically, when $t = c_B^s(t^s, t^{\overline{s}})$, where $t^s, t^{\overline{s}} \in \mathbb{T}_\Sigma$, and $s \in \mathbb{S}$. The function $[\![\cdot]\!]_{SPN}$: $\mathbb{T}_\Sigma \to \mathbb{Y}$ interprets the attack tree t (irrespective of the type of player) as a SPN composed by the Cartesian product of the two involved SPNs $y^s, y^{\overline{s}} \in \mathbb{Y}$ representing respectively attack tree t^s and $t^{\overline{s}}$. The involved SPNs are joined together in such a way that both actions (attack/defense) evolve in parallel. The countermeasure influences on the succession of the countered-action as follows: If the countered-action is executed, then a set of assets are gained e.g. password for Gmail-account, then if the countermeasure is executed before the previously gained assets are used (there is a chance to use the assets), those assets are blocked from being used (e.g. got disconnected from the Intranet after applying an IP-filter rule as countermeasure). However, if the countermeasure is executed before the countered-action, then the execution of the later is assumed to be successful with blocked assets i.e. there is no chance to use the assets as there was in the previous scenario (e.g. got the Gmail-password, but the attacker should be connected to the Intranet). Note that when both SPNs for the proponent and opponent reach their final places, they actually reach some special and common final places (dead places) that we denote by P_d, such that $P_d = P_*^{y^s} \times P_*^{y^{\overline{s}}}$ and $P_d \subset P_t$. These places represent sink places to the countered-player. Formally, $[\![c_B^s(t^s, t^{\overline{s}})]\!]_{SPN} = \Phi_0(y^s, y^{\overline{s}}) = (P, T, K, \Lambda, M_0)$ such that:

$$
\begin{cases}
P = \bigcup \left\{ \begin{array}{l} P_0 = P_0^{y^s} \times P_0^{y^{\overline{s}}} \\ P_t = P - P_0 \bigcup P_* \\ P_* = P_*^{y^s} \times P_0^{y^{\overline{s}}} \bigcup P_*^{y^s} \times P_t^{y^{\overline{s}}} \end{array} \right\} \\
T = \{t_0^{y^s}, \ldots, t_k^{y^s},\ t_0^{y^{\overline{s}}}, \ldots, t_k^{y^{\overline{s}}}\} \\
K = \{(p,t) \models \Gamma^P\} \bigcup \{(t,p) \models \Gamma^P\}, p \in P,\ \textit{and}\ t \in T \\
\Lambda = \{\lambda_0^{y^s}, \ldots, \lambda_k^{y^s},\ \lambda_0^{y^{\overline{s}}}, \ldots, \lambda_k^{y^{\overline{s}}}\} \\
M_0 = (M_0(p_0), \ldots, M_0(p_*)) = (1, \ldots, 0)
\end{cases}
$$

Example 4.7 Let us consider an ADTree $t = c_B^s(b^s, b^{\overline{s}})$, where $b^s \in \mathbb{B}$ and $b^{\overline{s}} \in \mathbb{B}$ are two basic actions for two opposite players, and let $y^s, y^{\overline{s}} \in \mathbb{Y}$ be their corresponding SPNs with rates $\lambda^s, \lambda^{\overline{s}} \in \mathbb{R}^+$ respectively. Then applying the function $\Phi_0(y^s, y^{\overline{s}})$ will produce the SPN depicted in Figure 2-f. The SPN is composed of 4 places, one initial p_0, one final p_*, and two transient places p_1 and p_2, where p_2 is a dead place (black state).

4.7 Semantics for delaying-type countermeasure

We define the semantics of the delaying-type countermeasure. Basically, when $t = c^s_D(t^s, t^{\overline{s}})$, where $t^s, t^{\overline{s}} \in \mathbb{T}_\Sigma$, and $s \in \mathbb{S}$. The function $[\![\cdot]\!]_{SPN}$: $\mathbb{T}_\Sigma \to \mathbb{Y}$ interprets the attack tree t (irrespective of the type of player) as a SPN composed of two opposite SPNs, $y^s \in \mathbb{Y}$ and $y^{\overline{s}} \in \mathbb{Y}$, in such a way that the countermeasure cancels the achievement of the countered-action if the late was executed i.e. if the attack is executed, the system reach a final place, then if the countermeasure is executed, the system is brought back to the initial place (canceling the achievement of the attacker). Contrary to the previous countermeasure (i.e. blocking-type), the delaying-type countermeasure allows the countered-player to re-start his action again, which is not possible in the blocking-type countermeasure. Formally, $[\![c^s_D(t^s, t^{\overline{s}})]\!]_{SPN} = \Phi_1(y^s, y^{\overline{s}}) = (P, T, K, \Lambda, M_0)$ such that:

$$\begin{cases} P = \bigcup \left\{ \begin{array}{l} P_0 = P_0^{y^s} \times P_*^{y^{\overline{s}}} \\ P_t = P_t^{y^s} \times P_0^{y^{\overline{s}}} \bigcup P_t^{y^{\overline{s}}} \\ P_* = P_*^{y^s} \times P_0^{y^{\overline{s}}} \end{array} \right\} \\ T = \{t_0^{y^s}, \ldots, t_k^{y^s},\ t_0^{y^{\overline{s}}}, \ldots, t_k^{y^{\overline{s}}}\} \\ K = \{(p,t) \models \Gamma^P\} \bigcup \{(t,p) \models \Gamma^P\}, p \in P,\ \textit{and}\ t \in T \\ \Lambda = \{\lambda_0^{y^s}, \ldots, \lambda_k^{y^s},\ \lambda_0^{y^{\overline{s}}}, \ldots, \lambda_k^{y^{\overline{s}}}\} \\ M_0 = (M_0(p_0), \ldots, M_0(p_*)) = (1, \ldots, 0) \end{cases}$$

Example 4.8 Let us consider an ADTree $t = c^s_D(b^s, b^{\overline{s}})$, where $b^s \in \mathbb{B}$ and $b^{\overline{s}} \in \mathbb{B}$ are two basic actions for two opposite players, and let $y^s, y^{\overline{s}} \in \mathbb{Y}$ be their corresponding SPNs with rates $\lambda^s, \lambda^{\overline{s}} \in \mathbb{R}^+$ respectively. Then applying the function $\Phi_0(y^s, y^{\overline{s}})$ will produce the SPN depicted in Figure 2-g. For a givne player $s \in \mathbb{S}$, the SPN is composed of 2 places, one initial p_0, and one final p_*.

5 Security Assessment of the ADTree Example

We report the analysis conducted to evaluate the qualitative as well as the quantitative security aspect of the scenario discussed in Section 2. We considered two misuse cases, the ADTree in Figure 1, and the ATree (Figure 1 modulo defense nodes). We have particularly generated the whole stochastic Petri-nets for the previous two trees using a java-code that we have developed for this purpose, and performed security assessment. In the following we start by performing the qualitative analysis, we present the general approach, then show how to apply it in the security context in general, and on ADTrees in particular. The second part will consist in quantitative analysis of both cases to show numerically the impact of countermeasures.

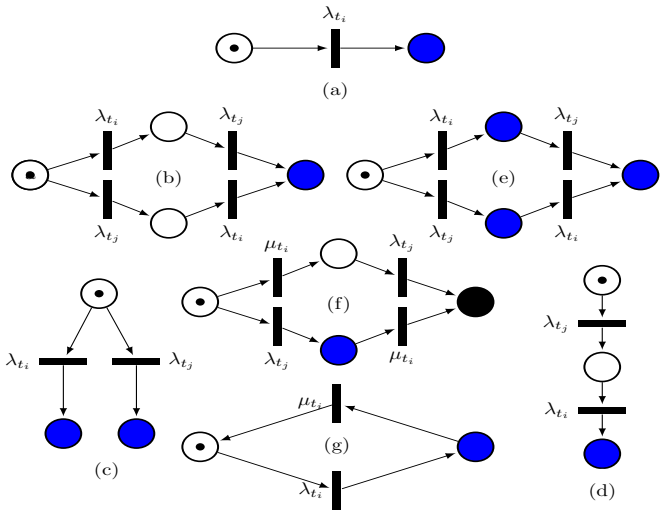

Fig. 2. Stochastic Petri-nets for a basic actions, countermeasures, and refinement, blue places are final places, black place is a dead place, and the places with a token are initial places.

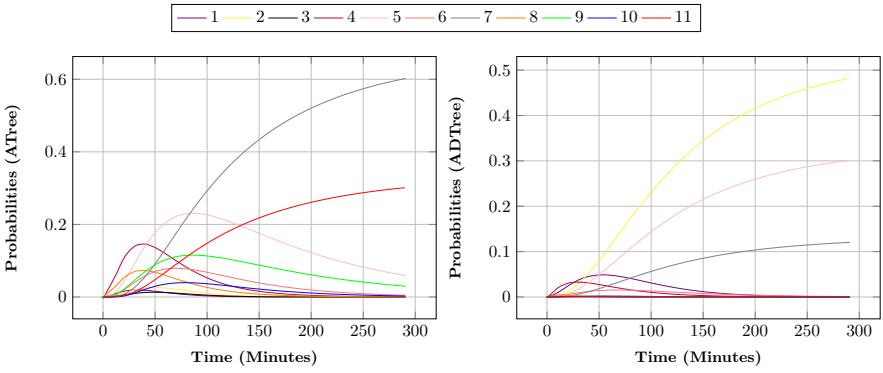

Fig. 3. Probability of breaching the network over time for each attack-path

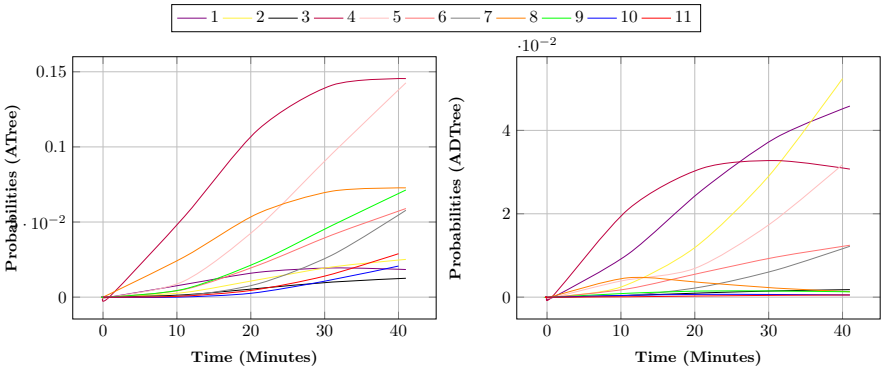

Fig. 4. Probability of breaching the network over half an hour for each attack-path

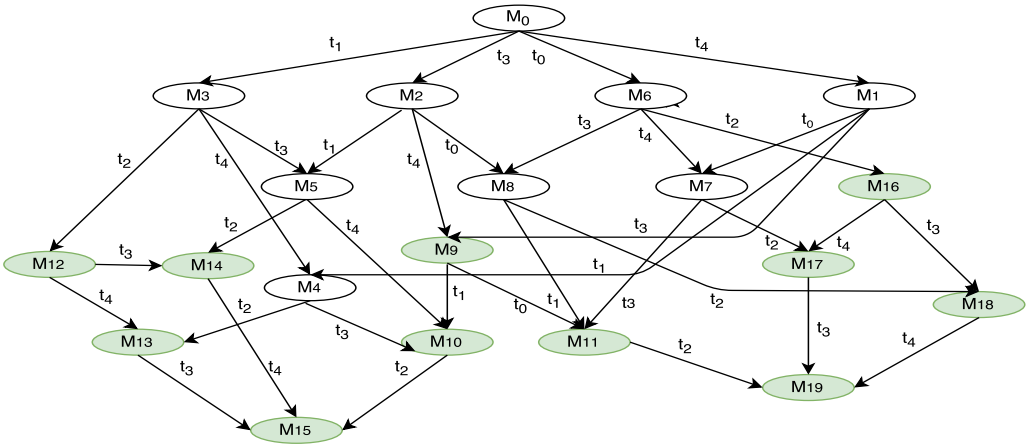

Fig. 5. RMG of the stochastic Petri-net model represneting the ADTree modulo defenses of Figure 1.

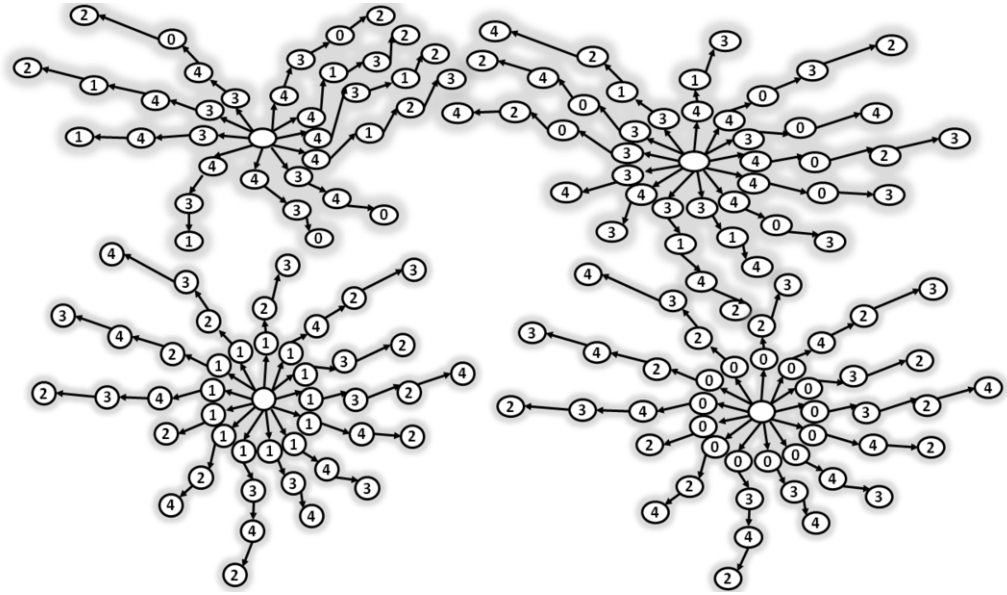

Fig. 6. Attack-paths elicited from structural analysis of ADTree modulo defenses of Figure 1.

5.1 Qualitative Security Assessment

Performing qualitative analysis on Petri-net models basically consists in two main phases, the first phase, called structural analysis, consists in generating either the covering-tree or the RMG (Reachable Markings Graph) of the used Petri-net model to extract the different transition sequences (e.g. cut sets), in our case attack-paths. The second part, called behavioral analysis, consists in checking a set of behavioral properties on the RMG.

Structural analysis: This phase consists in generating the RMG of the Petri-net model [19]. Using dedicated algorithms, we can extract the different attack-paths

that the attacker might undertake. We have generated the RMG from the constructed stochastic Petri-nets (see Figure 5). The RMG for the ADTree contains 36 markings with a big number of transitions from the attacker and the defender. However, the RMG for the ATree is more clear and simple to illustrate. It contains 20 markings and only attackers' transitions. Note that, either we take the RMG of the ADTree or the RMG of the ATree, the attack-paths are the same. The structural analysis revealed more than 49 possible attack-paths (see Figure 6). Because of space restriction and clarity in presenting the results, we report only 20 attack-paths (see paths-column of the table in Figure 7) just to show the applicability of the analysis. In general, attack-paths differ mainly in the order in which the basic actions were performed. The SAP (Shortest Attack-Paths) are $\{\{b_4^p; b_3^p\}, \{b_3^p; b_4^p\}, \{b_0^p; b_2^p\}, \{b_1^p; b_2^p\}\}$. They represent the minimum number of basic actions to be performed by an attacker to reach his final goal.

Behavioral analysis: We define each behavioral property, provide an interpretation in the context of ADTrees, then check the satisfaction over the RMG.

- **Boundedness.** Let $\mathcal{A}$ denote the set of reachable markings, then a Petri-net is said to be bound when there exists an upper bound for the number of tokens available in every marking i.e. $\forall M \in \mathcal{A}, \forall p \in \mathbb{P}\colon \exists k \in \mathbb{N} \mid M(p) \leq k$. In security context, this property might be interpreted as follows: The presence of a token in a given place, represents the ability of the attacker and/or the defender to perform a particular action in a specific marking. It might also represent the number of attackers or defenders that are ready to attack and/or defend the system. Since there is only one token shared between the defender and the attacker as the ability to perform an action at a specific marking, our system is 1-bounded. We can also say that only one attacker with a specific profile (skills, knowledge) is enough to perform a security breach on our system i.e. a mono-hacking system.
- **Quasi-Liveness.** A system is said to be quasi-alive if for every given reachable marking, there exists at least one transition that can be fired from that marking. We formally express this property as: $\forall M \in \mathcal{A}\colon \exists t \in \mathbb{T} \mid M[> t$. Therefore, in the context of ADTrees, a player is quasi-alive if for a given reachable marking $M \in \mathcal{A}$, there exists at least one transition $t^s \in \mathbb{T}$ from the player type that can be fired from that marking. For example, the proponent is quasi-alive if and only if: $\forall M \in \mathcal{A}\colon \exists t^p \in \mathbb{T} \mid M[> t^p$. From the RMG mapping the ADTree execution, the attacker is considered to be quasi-alive since for every given marking, the attacker has always the ability and opportunity to perform an action. However, the defender is not quasi-alive since there exist some markings where the defender has nothing to perform i.e. $\exists M \in \mathcal{A}, \nexists t^o \in \mathbb{T} \mid M[> t^o$. Thus, the attack surface of the attacker is larger than the defender attack surface.
- **Pseudo-Liveness.** A system is said to be pseudo-alive if for every given transition, there exists at least one reachable marking where the transitions can be fired. Formally we write: $\forall t \in \mathbb{T}\colon \exists M \in \mathcal{A} \mid M[> t$. Therefore, in the context of ADTrees, a player is pseudo-alive if there is always a reachable marking where he can execute one of his actions. For example, if the budget of the attacker is

Paths	ENS_{ADT}	ENS_{AT}	Gr	Paths	ENS_{ADT}	ENS_{AT}	Gr
$b_4^p; b_1^p; b_2^p$	2.224399	1.448285	5	$b_4^p; b_0^p; b_2^p$	2.065829	1.335312	9
$b_1^p; b_4^p; b_2^p$	2.325961	1.670584		$b_0^p; b_4^p; b_2^p$	1.858243	1.335312	
$b_4^p; b_1^p; b_3^p$	2.224399	1.448258	2	$b_1^p; b_3^p; b_2^p$	2.138187	1.549213	6
$b_1^p; b_4^p; b_3^p$	2.325961	1.670584		$b_3^p; b_1^p; b_2^p$	1.757012	1.215310	
$b_3^p; b_1^p; b_4^p$	1.757012	1.215310		$b_3^p; b_0^p; b_2^p$	1.692329	1.163020	10
$b_1^p; b_3^p; b_4^p$	2.138187	1.549213		$b_0^p; b_3^p; b_2^p$	1.764357	1.274623	
$b_4^p; b_0^p; b_3^p$	2.065829	1.335312	3	$b_4^p; b_3^p$	1.907260	1.222326	1
$b_3^p; b_0^p; b_4^p$	1.692329	1.163020		$b_3^p; b_4^p$	1.627647	1.110723	
$b_0^p; b_4^p; b_3^p$	1.858243	1.335312		$b_1^p; b_2^p$	2.008822	1.444625	4
$b_0^p; b_3^p; b_4^p$	1.764357	1.274623		$b_0^p; b_2^p$	1.699674	1.222326	8

Fig. 7. Attack-paths, and expected number of steps for each group Gr.

less than 50$, then the attacker cannot perform an action which requires more than 100$. In the RMG, the attacker is considered to be pseudo-alive since for every possible attack action, the attacker/defender has the necessary resources to perform attacks/defenses specified by the ADTree.

- **Deadlock-freeness.** If a system contains a sink marking from where no transition can be fired, the system is not free of deadlocks: $\exists M \in \mathcal{A}\colon \forall t \in \mathbb{T} \mid M[t \not>$. In the context of security, if a marking is a dead-marking for the attacker and it is not referring to attacker goal, then the attacker can be stopped. Meanwhile, if it is a dead-marking for the defender, then the system will remain in a breached state after the attack. In our case, there is no dead-marking for the attacker, which means that the attacker will always have the opportunity to perform an attack. However, there are so many dead-marking for the defender since he is not quasi-alive. This means that if the system is found in a marking where the attacker is taking advantages, the system is considered to be paralizable.
- **Host-marking.** A reachable marking is qualified by a host-making if there exists a sequence of transition such that, once executed from any given marking, takes the system to that host-marking. Namely, if the initial marking is the host marking, the system is said to be recoverable. Formally, if $M_a \in \mathcal{A}$ is a host-marking then: $\forall M \in \mathcal{A}\colon \exists \sigma \in \mathbb{T}^* \mid M[> \sigma\ M_a$. In ADTrees, if the initial marking is a host-marking, then the system can always recover after a breach. Because of the defender who is not quasi-alive, there is no sequence of defender action that takes the system from a breached state to the initial non-breached state. Thus, the system is not recoverable.

5.2 Quantitative Security Assessment

Thanks to the isomorphic relation between bounded SPNs and finite CTMC, the quantitative analysis of stochastic Petri-nets consists in transforming the Petri-net model into its corresponding finite CTMC. This is done by merely transforming places into states, or by transforming each marking into a state of the CTMC. The markings where the attacker has achieved his goal will be represented by final states. Then, we apply the same quantitative approach described in [18]. We first estimate a mean time t for each basic action in the ADTree. This mean time will represent the inverse of the exponential rate. We have arbitrary chosen minutes as our time unit. Following the notation of basic action given in section 2, we arbitrary affect the following rates: $\lambda(b_0^p) = 1/60$, $\lambda(b_1^p) = 1/30$, $\lambda(b_2^p) = 1/20$, $\lambda(b_3^p) = 1/120$, $\lambda(b_4^p) = 1/60$, $\lambda(b_5^o) = 1/5$, and $\lambda(b_6^o) = 1/5$.

Probabilistic analysis: The CTMC mapping the ATree of Figure 1 modulo defense nodes can be seen as the previously generated RMG of the ATree, where the markings are changed to states (e.g. M_0 to s_0). We have identified 11 final states, where the attacker has achieved his goal. Each final state can be reached through a group of attack-paths (see the table in Figure 7, column 1, and 5). Remember that we have considered only 20 attack-paths among 49. We have plotted the variation of the success probability of each group of attack-paths over time for ATtree in Figure 3-left. In the plot of Figure 3-left, except group 7 and 11, all other groups of attack-paths have a decreasing probability which converges to null. That is because the final states represented by both 7, and 11, are absorbing final states, which means that after a certain time spent (e.g. 10h), the system will reach a steady state, and the whole amount of probability will be shared between these two groups. However, we can zoom in inside the graph and try to analyse what was happening within the first 30 minutes (Figure 4-left). We can see that group 4 is the fastest growing probability function, this will correspond to the most probable group of attack-paths, in this case $\{b_1^p; b_2^p\}$ with a likelihood of 0.145. At this stage, we can qualify the attack-path $\{b_1^p; b_2^p\}$ as the MPAP (Most Probable Attack-Path). In the second case, the CTMC was bigger, it consisted of 36 states, among them 15 final states. This is because the execution of a countermeasure after the attacker has finished achieving his attack (already realized his goal), will take the system to another final state (i.e. the defense has no impact). The additional final states (i.e. 4 new final states) are merged to the final state that proceed them since countermeasures actions are not part of the attack paths. We can see from the plot of Figure 3-right, that groups of attack-paths 2, 5, and 7 are increasing in likelihood over time. This is because these groups corresponds to absorbing states (except for 11, which is this time decreasing). In order to understand the impact of the countermeasures that we have set, we zoom in the plot, and scrutinize what happened in the first 30 minutes (Figure 4-right). We remark that in the first 25 minutes, the attacker was most likely interested in undertaking the group of attack-paths 4, but at $t = 26$, it was equi-probable for the attacker to choose between group 1, and 4. Then, because the countermeasures are having a great

impact upon attack-paths of group 4, the attacker is most likely interested in undertaking attack-paths of group 1, which corresponds to the right-side of the ADTree. In long term, and as the attacker might exert his actions in parallel, he might be able to achieve (i.e. if no countermeasure executed) some parts of the left-side of the tree as can be confirmed by the graph of group 2, and 5. We can conclude that the countermeasures represented in the ADTree have a considerable impact on the security of the system, as it decreased the likelihood of breaching the system. Finally, and after identifying the most probable group of attack-paths, we can perform a ranking of attack-paths in a given group to identify the MPAP. This is done by computing the expected number of steps for each attack-path [18]. The results are illustrated in the table of Figure 7. If we consider the ADTree, and the group 2 as the MPG (Most Probable Group), the MPAP=$\{b_1^p; b_4^p; b_3^p\}$.

Time-based analysis: In this analysis, we mainly compute the mean time to breach a system. The approach is well explained in [18]. The computed MTTSF (Mean Time To Security Failure) revealed that that attacking the first configuration (ATree) takes in average half an hour (36'). However, the second configuration (ADTree) requires almost two hours (119') to be breached.

6 Related work

In 2010, the well-known attack-trees model [4] was extended to attack-defense trees [3], allowing security engineers to represent defense actions besides attacks actions in one tree-based layout. The later model allows a list of possible denotational semantics to be used [3]. However, non of these semantics allow the expression of actions ordering, which is practically the most used aspect in cyber-attacks. In [18], the authors proposed a CTMC-based semantics for attack-defense trees, which allows the expression of action ordering, and perform quantitative security analysis. However, besides security quantitative analysis, and to our knowledge, there is a considerable paucity of qualitative underpinnings supporting attack-defense trees analysis. From an overall picture, there have been a great number of security graphical models proposed in the literature years before attack-defense tree, such as attack-graphs [10]. Proposing security assessment techniques was the second face of this research immersion period, approaches based on Markov chains [5, 9], or Petri-nets [12, 13] were applied. However, none of the previously mentioned works [5, 9, 12, 13] have provided a classification for countermeasures or at least considered countermeasures in a more realistic way. This limitation was deeply discussed in [18], and the authors have given a first attempt to overcome this limitation by proposing two classes of countermeasures, immediate-impact, and delayed-impact. However, there still a large variety of countermeasures which have a different model, and belong to other classifications. Finally, and to our knowledge there is no type of refinement in the attack-defense tree specification expressing parallel actions [3, 4, 9, 18].

7 Conclusion

In this work, we have presented a stochastic operational semantics for ADTrees based on stochastic Petri-nets. This allowed us to perform both qualitative and quantitative security analysis. We have shown through a simple example that the proposed solution is useful in the sens it can help system designers to start security by design, or strengthening already existing infrastructure. This is a first attempt, that of course needs further improvements. In particular we plan to extend the framework to consider new types of countermeasures, qualitative properties to output a complete security assessment tool [17].

References

[1] Marc, Dacier., *A Petri Net Representation of the Take-Grant Model*, "Computer Security Foundations Workshop VI", 1993.

[2] Yixin, Jiang,. Chuang L., Hao, Y., and Zhangxi, T., *Security analysis of mandatory access control model*, "International Conference on Systems, Man & Cybernetics", 2004.

[3] Kordy, Barbara., Mauw, S., Radomirović, S., and Schweitzer, P., *Foundations of attack–defense trees*, "International Workshop on Formal Aspects in Security and Trust", 2010.

[4] Mauw, Sjouke., and Oostdijk, M., *Foundations of attack trees*, "International Conference on Information Security and Cryptology" , 2005.

[5] Madan, Bharat., Gogeva-Popstojanova, K., Vaidyanathan, K., and Trivedi, K., *Modeling and quantification of security attributes of software systems*, "International Conference onDependable Systems and Networks", 2002.

[6] Jonathan Billington., Geoffrey R. W., and Michael C. W., *PROTEAN: A High-Level Petri Net Tool for the Specification and Verification of Communication Protocols*, "IEEE Transactions on Software Engineering", 1988.

[7] Abraham, Subil., and Nair, Suku., *Predictive Cyber-security Analytics Framework: A non-homogenous Markov model for Security Quantification*, Computing Research Repository, **abs/1501.01901** (2015).

[8] Wil van der Aalst., *The Application of Petri Nets to Workflow Management*, Journal of Circuits, Systems, and Computers, **8** (1998).

[9] Arnold, Florian., Guck, D., Kumar, R., and Stoelinga, Mariële., *Sequential and parallel attack tree modelling*, "International Conference on Computer Safety, Reliability, and Security", 2015.

[10] Hughes, Todd., and Sheyner, O., *Attack scenario graphs for computer network threat analysis and prediction*, Wiley Online Library: Journal of Complexity, **9** (2003).

[11] Balbo, Gianfranco., *Introduction to Generalized Stochastic Petri Nets*, "Formal Methods for Performance Evaluation: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems", 2007.

[12] Pudar, S., and Manimaran, G., and Liu, C., *PENET: A practical method and tool for integrated modeling of security attacks and countermeasures*, In Computers & Security Journal, **28** (2009).

[13] Dalton, GC., Mills, Robert, F., Colombi, John, M., and Raines, Richard, A., and others, *Analyzing attack trees using generalized stochastic Petri nets*, "Information Assurance Workshop", 2006.

[14] Jensen, Kurt., *Coloured Petri nets: basic concepts, analysis methods and practical use*, "Springer Science & Business Media", 1995.

[15] Ramchandani, C., *Analysis of asynchronous concurrent systems by timed Petri nets*, "Massachusetts Institute of Technology", 1974.

[16] Molloy, Michael, K., *Performance analysis using stochastic Petri nets*, "IEEE Transactions on computers", 1982.

[17] Gadyatskaya, Olga., Jhawar, R., Kordy, P., Lounis, K., and Mauw, S., and Trujillo-Rasua, R., *Attack trees for practical security assessment: ranking of attack scenarios with ADTool 2.0*, "Proceedings of the 13th International Conference on Quantitative Evaluation of Systems", 2016.

[18] Lounis Karim., Jhawar, R., and Mauw, S., *A Stochastic Framework for Quantitative Analysis of Attack-Defense Trees*, "Proceedings of the 12th International Workshop on Security and Trust Management", 2016.

[19] Chiola, Giovanni., Carvajal-Schiaffino, R., *A reachability graph construction algorithm based on canonical transition firing count vectors*, "Proceedings of the 9th international Workshop on Petri Nets and Performance Models", 2001.

[20] Peterson, James, L., *Petri net theory and the modeling of systems*, "Prentice Hall PTR", 1981.