Quantitative Verification and Synthesis of Attack-Defence Scenarios

Aslanyan, Zaruhi; Nielson , Flemming ; Parker, David

DOI:
10.1109/CSF.2016.15

Quantitative Verification and Synthesis of Attack-Defence Scenarios

Zaruhi Aslanyan, Flemming Nielson
DTU Compute
Technical University of Denmark
Denmark
Email: {zaas,fnie}@dtu.dk

David Parker
School of Computer Science
University of Birmingham
Birmingham, United Kingdom
Email: d.a.parker@cs.bham.ac.uk

***Abstract*—Attack-defence trees are a powerful technique for formally evaluating attack-defence scenarios. They represent in an intuitive, graphical way the interaction between an attacker and a defender who compete in order to achieve conflicting objectives. We propose a novel framework for the formal analysis of quantitative properties of complex attack-defence scenarios, using an extension of attack-defence trees which models temporal ordering of actions and allows explicit dependencies in the strategies adopted by attackers and defenders. We adopt a game-theoretic approach, translating attack-defence trees to two-player stochastic games, and then employ probabilistic model checking techniques to formally analyse these models. This provides a means to both verify formally specified security properties of the attack-defence scenarios and, dually, to synthesise strategies for attackers or defenders which guarantee or optimise some quantitative property, such as the probability of a successful attack, the expected cost incurred, or some multi-objective trade-off between the two. We implement our approach, building upon the PRISM-games model checker, and apply it to a case study of an RFID goods management system.**

Keywords: attack-defence trees, stochastic games, formal verification, probabilistic model checking

I. Introduction

Keeping systems secure against the threat of attacks is a crucial problem, which becomes increasingly difficult as attacks become more sophisticated and systems more complex. This necessitates thorough investigations of the possible attack scenarios for a system, along with the appropriate response mechanisms needed. Formal graphical models can help identify and understand the security threats to the system and to study the possible countermeasures.

One such approach is *attack trees*, a widely used graphical modelling formalism introduced by Schneier [1] for representing and evaluating the security of attack scenarios in a structured, hierarchical way. The main intuition behind attack trees is to split a complex goal into sub-goals and basic actions [1]. Various approaches have also been proposed for the automatic construction of attack trees from descriptions of attack scenarios [2], [3].

However, attack trees describe only the attacker's behaviour and do not consider possible defences undertaken to counter the attacks. To overcome this limitation, further extensions of attack trees for capturing the defender's behaviour have been studied. *Attack-defence trees*, introduced by Kordy et al. [4],

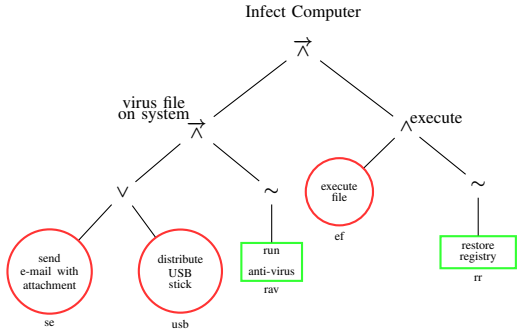


Fig. 1. An example of an attack-defence tree for infecting a computer.

are an extension of the formalism with countermeasures. They enhance attack trees with the options available to a defender. Attack-defence trees have been used to evaluate real-life scenarios such as an RFID goods management system [5] and various case studies within the European project TREsPASS [6]. Other applications of attack-defence trees are investigated in the research project ADT2P [7]. A simple example of an attack-defence tree, which we will return to as running example throughout the paper, is shown in Figure 1. It represents a scenario in which an attacker attempts to infect a computer with a virus: circle nodes denote the attacker's actions and square nodes denote the defender's actions.

In addition to capturing the relationship between attacker and defender actions, attack-defence trees can be augmented with *attributes*, representing a variety of *quantitative* measures of interest, such as the success probability or cost associated with basic actions [8]. A bottom-up traversal of the tree can then be performed to identify, for example, attacks that succeed with high probability or incur minimal costs. Typically, such an analysis focuses on one specific aspect of the system, however it is also possible to consider optimisation of multiple criteria, for example using Pareto efficiency [9].

In this paper, we propose a novel framework for the formal analysis of quantitative properties of attack-defence scenarios using an extension of attack-defence trees. We allow the trees to incorporate information about the temporal ordering of some actions or subgoals (similar ideas have been put forward in, for example, [10]). We then propose a novel class of at-

tack/defence strategies that incorporate *dependencies*, allowing decisions about which actions are taken by the attacker or defender to be based on which actions have been taken earlier and the outcomes of those actions.

Formally, these strategies are defined as *decision trees*. For an attack-defence scenario, modelled as an attack-defence tree, and specific strategies for both the attacker and defender, we give an operational semantics defining the resulting behaviour. We allow attack-defence trees to be annotated with the probability of success or failure of individual basic actions, so the semantics takes the form of a Markov model, more precisely a discrete-time Markov chain.

In order to formally analyse attack-defence scenarios modelled in this way, we employ *quantitative verification* techniques, in particular *probabilistic model checking*. This uses variants of temporal logic to formally specify quantitative measures of interest of the system, which can then be automatically checked against a state-based probabilistic model.

These techniques, and associated verification tools have been developed for a wide range of probabilistic models. For attack-defence trees, it is natural to model the interactions between attacker and defender as a two-player game [11]. So, in our setting, where quantitative and probabilistic aspects are essential, we use *stochastic two-player games* [12], building upon the probabilistic model checking techniques for stochastic games proposed in [13], [14], and implemented in the PRISM-games model checking tool [15].

This approach uses a temporal logic called rPATL (probabilistic alternating-time temporal logic with rewards), a generalisation of the well known logic ATL (alternating temporal logic). rPATL allows us to explicitly reason about the strategies available to the competing players, and provides a variety of operators to specify quantitative properties of these strategies relating to the probability of certain events occurring, or a cost or reward measure associated with them.

Probabilistic model checking of rPATL on stochastic games allows us to take two distinct approaches to the analysis of an attack-defence scenario: we can *verify* security properties of them (e.g., "whatever the attacker does, the defender can always guarantee that the probability of a successful attack is at most 0.001"); or we can *synthesise* strategies for a player with respect to some goal (e.g., "what strategy for the attacker maximises the probability of a successful attack, regardless of the actions employed by the defender?"). These logics provide expressiveness which goes beyond the standard queries typically used on attack-defence scenarios [8]. Furthermore, we use an extension of the basic rPATL-based approach for *multi-objective probabilistic model checking* of stochastic games [14]. This allows us to analyse multiple, conflicting objectives (e.g., "what strategy minimises the cost incurred by the attacker, whilst guaranteeing a successful attack with probability at least 0.5?"), and also to compute the Pareto curve associated with the objectives.

In order to use these game-theoretic verification techniques within our proposed framework, we define a translation from attack-defence trees to stochastic two-player games. This model captures the set of all strategies available to the attacker and defender players, and the resulting (Markov chain) semantics that results for each pair of such strategies. Using probabilistic model checking, we can either verify a security property on this game model, or synthesise a strategy achieving or optimising a desired property. In the latter case, a player strategy generated from the stochastic game as a result is then converted into a decision tree for either the attacker or defender. We implement our approach and illustrate it on an example of a Radio-Frequency Identification (RFID) warehouse goods management system [5].

Organisation of the paper. In Sect. II we provide background material on stochastic systems and the logic rPATL. Our formalism for attack-defence trees and strategies, as well as their meaning is presented in Sect. III. In Sect. IV we describe our proposed translation from attack-defence trees to two-player stochastic games. The results of the evaluation are discussed on a case study for a RFID system in Sect V. We describe related work in Sect. VI and conclude in Sect. VII.

II. Preliminaries

We begin with some background on the probabilistic models used in the paper, Markov chains and stochastic two-player games, and their analysis using probabilistic model checking.

A. Discrete-time Markov Chains

Definition 1 (DTMC) *A discrete-time Markov chain (DTMC) is a tuple $\mathcal{D} = (S, s_0, P, AP, L)$, where:*

- *S is a set of states and $s_0 \in S$ is an initial state;*
- *$P : S \times S \to [0, 1]$ is a transition probability function such that $\sum_{s' \in S} P(s, s') = 1$ for all states $s \in S$;*
- *AP is a set of atomic propositions; and*
- *$L : S \to 2^{AP}$ is a labelling function.*

The transition probability function indicates the probability $P(s, s')$ of moving from state s to state s' in a single transition. A *path* through a DTMC is a sequence of states $\pi = s_0 s_1 \cdots$ where $s_i \in S$ and $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$. We write $Path_s$ for the set of all infinite-length paths starting in a state s. To reason about quantitative properties of a DTMC, we use a probability measure Pr_s over the set $Path_s$ of infinite paths, which can be defined in standard fashion [16].

We also define *reward structures* of the form $r : S \times S \to \mathbb{R}_{\geq 0}$, which we use as a mechanism to model both costs and rewards associated with a DTMC model.

B. Stochastic Two-Player Games

Definition 2 (STG) *A (turn-based) stochastic two-player game is a tuple $\mathcal{M} = (\Pi, S, s_0, \alpha, (S_A, S_D, S_P, S_\circledcirc), P, T, AP, L)$, where:*

- *$\Pi = \{A, D\}$ is a set of players,*
- *$S = S_A \uplus S_D \uplus S_P \uplus S_\circledcirc$ is a finite set of states, partitioned into attacker states (S_A), defender states (S_D), probabilistic states (S_P) and final states ($S_\circledcirc$)*
- *$s_0 \in S$ is an initial state,*
- *α is a finite, non-empty set of actions,*

- $P : S_P \times S \to [0,1]$ *is a probabilistic transition function such that for all probabilistic states* $s \in S_P$, $\sum_{s' \in S} P(s,s') = 1$,
- $T : (S_A \cup S_D) \times \alpha \to S$ *is a transition function,*
- AP *is a set of atomic propositions, and*
- $L : S \to 2^{AP}$ *is a labelling function.*

An STG has two players, which, for this paper, we fix as A (attacker) and D (defender). Each player controls a subset of the states: *A* chooses between available actions $a \in \alpha$ in states $s \in S_A$, while *D* does so for states $s \in S_D$. The choice between outgoing transitions in probabilistic states $s \in S_P$ is made probabilistically, by the probabilistic transition function P. States in $S_\circledcirc$ are terminating and have no transitions. As for DTMCs, paths are (finite or infinite) sequences of connected states. The sets of all infinite and finite paths starting in state s are $Path_s$ and $Path_s^{fin}$, respectively.

A *strategy* for player $i \in \Pi$ is a function $\sigma_i : S^* S_i \to Dist(\alpha)$, where $Dist(\alpha)$ is the set of discrete probability distributions over α. Informally, the strategy resolves the choice of actions for the player during their turns, based on the previous history of the STG. A strategy σ_i is *memoryless* if for any $\pi, \pi' \in S^*$ and $s \in S_i$, $\sigma_i(\pi s) = \sigma_i(\pi' s) = \sigma_i(s)$. A strategy is *deterministic* if it always selects actions with probability 1, and *randomised* otherwise. The set of all strategies for player i is denoted Σ_i.

Given a pair σ_A, σ_D of strategies for both players, the resulting behaviour of the STG is represented by an induced DTMC, from which we can obtain a probability measure $Pr_s^{\sigma_A,\sigma_D}$ over $Path_s$. If the strategies are both memoryless, the induced DTMC has the same state space S as the STG.

Like for DTMCS, we also consider reward structures. For an STG, these take the form $r_A : S_A \times \alpha \to \mathbb{R}_{\geq 0}$ and $r_D : S_D \times \alpha \to \mathbb{R}_{\geq 0}$, annotating the transitions controlled by the attacker and defender, respectively.

C. Probabilistic Model Checking

In this paper, we use probabilistic model checking, which provides automated analysis of quantitative properties, formally specified in temporal logic, against various probabilistic models, including DTMCs and STGs. For DTMCs, we can use the logic PCTL [17] and its extensions with cost and reward operators [18]. Mostly, in this paper, we apply probabilistic model checking to STGs, the basis for which is the temporal logic rPATL [13], which combines PCTL (plus rewards) with alternating temporal logic (ATL).

Definition 3 (rPATL Syntax) *The syntax of rPATL is:*

$$\phi ::= \mathit{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle P_{\bowtie q}(\psi) \mid \langle\langle C \rangle\rangle R^r_{\bowtie x}(F^* \phi)$$
$$\psi ::= X\phi \mid \phi\, U^{\leq k} \phi \mid \phi\, U \phi$$

where $a \in AP$, $C \subseteq \Pi$, $\bowtie \in \{<, \leq, >, \geq\}$, $q \in \mathbb{Q} \cap [0,1]$, $x \in \mathbb{Q}_{\geq 0}$, *r is a reward structure,* $* \in \{0, \infty, c\}$ *and* $k \in \mathbb{N}$.

The coalition operator $\langle\langle C \rangle\rangle \Phi$ means that the coalition of the players $C \subseteq \Pi$ has a strategy to satisfy Φ, regardless of the strategies of the players $\Pi \backslash C$. In this work, C is either $\{A\}$ or $\{D\}$. Using shorthand $F\phi \equiv \mathit{true}\, U\, \phi$, an example formula is $\langle\langle A \rangle\rangle P_{\geq 0.1}(F\, success)$, which is true if the attacker player has a strategy that guarantees to reach a state labelled with *success* (e.g., denoting a successful attack), regardless of the strategy of the defender. The R operator reasons about expected cumulated reward (or cost) until a ϕ-state is reached. The parameter $*$ specifies the result if ϕ is not reached, but is not needed in this paper. We refer the reader to [13] for full details of the semantics of rPATL.

In this paper, we use several other types of properties. First, we use numerical queries, such as $\langle\langle A \rangle\rangle P_{max=?}(F\, success)$, which directly returns the optimal value associated with the property (e.g., in this case, the maximum probability of a successful attack achievable by the attacker, given any possible strategy of the defender). Secondly, we use the multi-objective extensions [14] of rPATL provided by the PRISM-games model checker [15]. For example: $\langle\langle A \rangle\rangle (R^{r_A}_{\leq 500}[F end] \wedge P_{\geq 0.005}[F\, success])$ asks if there is an attacker strategy which reaches *success* with probability at least 0.005 *and* with an expected total cost of at most 500. For all the types of properties explained here, PRISM-games also synthesises strategies with the specified properties.

III. Attack-Defence Trees

In this section, we present the key ingredients of our formalism: attack-defence trees, to represent attack-defence scenarios, and strategies (in the form of decision graphs), to represent the behaviour of the attacker and defender. We start by defining the syntax and terminology for each of these, in Sect. III-A and III-B, respectively. Then, in Sect. III-C, we describe their formal semantics.

A. Attack-Defence Trees

An attack-defence tree is a graphical model representing the interaction between two players (denoted by τ): the attacker ($\tau = A$) and the defender ($\tau = D$). The root of the tree represents the main goal of an attack-defence scenario for a given player τ (e.g., for the attacker, it represents a successful attack of the system). The leaves of the tree represent the *basic actions* that players can perform in order to achieve their goals. If performed, each basic action can either succeed or fail, the likelihood of which is specified by a separate annotation with probabilities (see Sect. III-C). The internal nodes of the tree show how the basic actions can be combined and how they interact with each other.

The abstract syntax of an attack-defence tree is presented in the top part of Table I, and is split into rules for a tree t and a non-sequential tree nt (see below). A tree is either a leaf or the application of a tree operator to one or two subtrees. A leaf a is a basic action of either the attacker or the defender. We denote the attacker's and defender's sets of basic actions by Act_A and Act_D, respectively. We assume these are disjoint and write $Act = Act_A \uplus Act_D$ for the set of all basic actions. The special leaves `true` and `false` represent trivially successful and trivially failed actions, respectively.

TABLE I
THE SYNTAX OF ATTACK-DEFENCE TREES AND THE TYPE SYSTEM FOR DEFINING WELL-FORMED TREES.

$$t ::= nt \mid \overrightarrow{\wedge}(t_1, t_2) \mid \overrightarrow{\vee}(t_1, t_2)$$
$$nt ::= a \mid \wedge(nt_1, nt_2) \mid \vee(nt_1, nt_2) \mid \sim nt \mid \texttt{true} \mid \texttt{false}$$

$$\vdash a : A \text{ if } a \in Act_A \qquad \vdash a : D \text{ if } a \in Act_D$$

$$\frac{\vdash t_1 : \tau \;\; \vdash t_2 : \tau}{\vdash \overrightarrow{\wedge}(t_1, t_2) : \tau} \qquad \frac{\vdash t_1 : \tau \;\; \vdash t_2 : \tau}{\vdash \overrightarrow{\vee}(t_1, t_2) : \tau}$$

$$\frac{\vdash nt_1 : \tau \;\; \vdash nt_2 : \tau}{\vdash \wedge(nt_1, nt_2) : \tau} \qquad \frac{\vdash nt_1 : \tau \;\; \vdash nt_2 : \tau}{\vdash \vee(nt_1, nt_2) : \tau} \qquad \frac{\vdash nt : \tau}{\vdash \sim nt : \tau'} \; \tau' = \tau^{-1}$$

$$\vdash \texttt{true} : \tau \qquad \vdash \texttt{false} : \tau$$

Tree operators include the standard conjunction $\wedge$ and disjunction $\vee$, as well as three additional operators: sequential conjunction $\overrightarrow{\wedge}$, sequential disjunction $\overrightarrow{\vee}$ and the $\sim$ construct representing player alternation. The conjunction operator $nt = \wedge(nt_1, nt_2)$ requires that the goals of both nt_1 and nt_2 are achieved in order for the goal of nt to be achieved. The disjunction operator $nt = \vee(nt_1, nt_2)$ requires that at least one sub-tree nt_1 or nt_2 is achieved.

The sequential variants additionally impose an ordering on the sub-trees. Sequential conjunction $t = \overrightarrow{\wedge}(t_1, t_2)$ requires that the goals of both t_1 and t_2 are achieved and that the former is performed before the latter. Sequential disjunction $t = \overrightarrow{\vee}(t_1, t_2)$ similarly requires t_1 to be performed before t_2 (assuming both of them are). Intuitively, in a sequential disjunction it only makes sense to *attempt* t_2 after failing in *achieving* t_1. In order to simplify the technical developments, we require that the sequential operators only occur *above* the conjunction and disjunction operators in a tree. Thus, we disallow trees such as $\wedge(\overrightarrow{\wedge}(a, b), c)$ for basic actions a, b, c. This is imposed by the division of the syntax into trees t and *non-sequential* trees nt.

Each node of attack-defence tree is associated with one of the two players. The $\sim$ operator changes the goal of a tree by changing it to the opposite player (i.e., switching between attacker and defender). For instance, if $t = \sim t'$ and t' belongs to the attacker, then the tree t belongs to the defender.

Well-formedness. The syntax in Table I is overly liberal since it does not explicitly associate players to nodes. The bottom part of Table I shows a simple type system which enforces such an association by defining a *well-formedness* condition for trees. We assign type τ to all basic actions of the player τ. Both variants of the conjunction and disjunction operators have the same type as both of their sub-trees. Finally, the $\sim$ operator flips the type of its sub-tree from τ to τ^{-1} (where $A^{-1} = D$ and $D^{-1} = A$).

Phases. In an attack-defence tree t of the form described above, we associate each maximal non-sequential sub-tree with a *phase*. We say that the maximal non-sequential sub-trees $nt_1, \cdots, nt_n$ divide the tree t into *phases* $\mathfrak{p}_1, \cdots, \mathfrak{p}_n$, where any two non-sequential sub-trees are connected with a sequential operator. Thus, the number of phases in a tree is one more than the number of sequential operators. We denote by *Phases* the set of all phases in a tree. We indicate by $Act_{\mathfrak{p}_i,A}$ and $Act_{\mathfrak{p}_i,D}$, respectively, the set of attacker and defender basic actions in phase $\mathfrak{p}_i$ (non-sequential tree nt_i), and by $Act_{\mathfrak{p}_i}$ the set of all basic actions in phase $\mathfrak{p}_i$, i.e., $Act_{\mathfrak{p}_i} = Act_{\mathfrak{p}_i,A} \uplus Act_{\mathfrak{p}_i,D}$.

Example 1 *Let us introduce an example that we will develop throughout the paper. We consider a modified version of a simple scenario borrowed from [19], where an attacker wants to infect a computer with a virus. In order to do so, the attacker needs to put the virus file on the system and only after that execute it. The attacker can transmit the virus either by sending an e-mail with an attachment or by distributing a USB stick to a user of the system. The defender, on the other hand, can try to prevent the attack by running an anti-virus program. Once the virus file is on the computer, the attacker can execute it either directly or through a third person. The defender can counteract this by restoring the registry to a previous checkpoint. The corresponding attack-defence tree is shown in Figure 1, where we label leaves for ease of reference. The syntactic term corresponding to the full tree is:*

$$t = \overrightarrow{\wedge}(\overrightarrow{\wedge}(\vee(se, usb), \sim rav), \wedge(ef, \sim rr))$$

The tree t has three non-sequential sub-trees: $t_1 = \vee(se, usb)$, $t_2 = \sim rav$, *and* $t_3 = \wedge(ef, \sim rr)$ *and thus three phases.*

B. Strategies as Decision Trees

In the standard model of attack-defence trees [4], a possible attack (or defence) is a *set* of basic actions for one player which will be performed. Given an attack and a defence, the tree defines the success or failure of these according to the way that the basic actions are combined within the attack-defence tree (e.g., if the root node belongs to the attacker, the attack is successful if the tree evaluates to true).

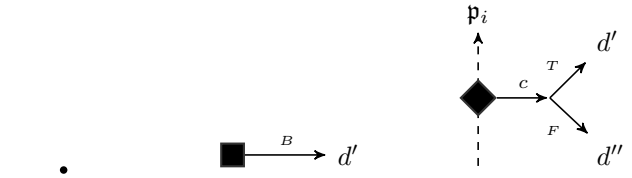
In this work, we take a different approach, proposing a more powerful class of *strategy* for attackers and defenders which can incorporate *dependencies* on events that happened previously when deciding which basic action to perform. In particular, a strategy's choice can depend on the attempt, success or failure of another basic action from an earlier phase. We represent these strategies, for either attacker or defender, by means of *decision trees*, which illustrate the relationship between phases in an intuitive way.

The abstract syntax of a decision tree is presented in Table II. We assume that this defines a strategy for an attack-defence tree with n phases $\mathfrak{p}_1, \cdots, \mathfrak{p}_n$ and, for convenience, we add a "dummy" phase $\mathfrak{p}_{n+1}$ denoting the end of the strategy. In the syntax, d_i^τ represents a sub-tree defining a strategy for player τ (where $\tau = A, D$) which starts in phase $\mathfrak{p}_i$. So the root node of a decision tree is of the form d_1^τ.

In phase $\mathfrak{p}_i$ (for $1 \leq i \leq n$), a sub-tree can be either: (i) an *action node* $B.d_{i+1}^\tau$, indicating that player τ performs

TABLE II
THE SYNTAX OF A DECISION TREE

d_i^τ	::=	$B.d_{i+1}^\tau$ where $B \subseteq Act_{\mathfrak{p}_i,\tau}$
d_i^τ	::=	$\mathit{if}(c_i^\tau, d_i'^\tau, d_i''^\tau)$
d_{n+1}^τ	::=	stop
c_i^τ	::=	$a?$ if $\tau = A$ and $a \in Act_{\mathfrak{p}_i,D}$
c_i^τ	::=	$\mathfrak{p}_j?$ if $1 \leq j < i$
c_i^τ	::=	$c_i'^\tau \wedge c_i''^\tau$
c_i^τ	::=	$c_i'^\tau \vee c_i''^\tau$
c_i^τ	::=	$\neg c_i'^\tau$



(a) stop node (b) action node (c) decision node

Fig. 2. Graphical representation of decision tree nodes.

the (possibly empty) set of basic actions $B \subseteq Act_{\mathfrak{p}_i,\tau}$ in that phase and then proceeds to sub-tree d_{i+1}^τ; or (ii) a *decision node* of the form $\mathit{if}(c_i^\tau, d_i'^\tau, d_i''^\tau)$, which branches based on the condition c_i^τ. Once n phases have passed, the decision tree moves to a stop node, representing the end.

Decision nodes represent conditional dependencies: deciding whether to perform some basic actions or to move to another phase can be conditional on events that have already occurred. For instance, in phase $\mathfrak{p}_i$, a player may wish to only perform an action in the case of a failed attempt at performing some action in phase $\mathfrak{p}_{i-1}$. A decision node consists of a condition c_i^τ and two possible sub-trees, where execution of the strategy moves to the first subtree $d_i'^\tau$ if the condition is true, or the second sub-tree $d_i''^\tau$ otherwise.

Conditions are expressed as Boolean expressions over atomic conditions of two types, $\mathfrak{p}_j?$ or $a?$. A condition of the form $\mathfrak{p}_j?$, in a phase $\mathfrak{p}_i$ decision tree node (where $j < i$), asks about the success of an earlier phase. The success of a phase is determined by an evaluation of the corresponding node of the subtree (more precisely, the root node of the corresponding maximal non-sequential tree), which we will define more precisely in the next section.

In the case of a decision tree for an attacker strategy (i.e., where $\tau = A$), we also allow conditions $a?$ that ask whether some defender action $a \in Act_{\mathfrak{p}_i,D}$ was performed within the current phase $\mathfrak{p}_i$ (note that we ask whether it was attempted, not whether it was successful, since the latter is not yet known). We do *not* allow the reverse: where defenders ask about attacker action's performed in the same phase. In our work, we assume that the attacker has slightly more power/information than the defender in this respect, favouring a conservative approach to verifying the absence of attacks.

Graphical representation. The graphical representation of each node in a strategy is shown in Figure 2. A stop node is represented by a black dot, and occurs only as a leaf at the end of the tree. An action node $B.d'$ is represented by a black square with an outgoing edge to the successor node d' labelled with the set of performed actions B. A conditional node $\mathit{if}(c, d', d'')$ is drawn as a black diamond. It has an outgoing transition labelled with the condition c and two transitions, labelled with "T" (true) and "F" (false), corresponding to the "then" and "else" branches of the "if" statement. In the rightmost node in Figure 2 (the decision node), we also illustrate the use of vertical dashed lines to mark the boundaries of phases.

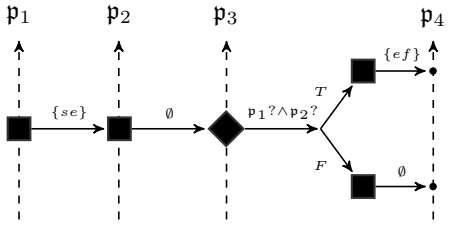


Fig. 3. An attacker strategy for the attack-defence tree of Fig 1.

Randomisation. Although, for clarity, we omit it from the presentation above, we also consider *randomisation* in decision trees. More precisely, instead of just allowing the attacker or defender to choose to execute a set of actions B in a given phase, we allow them to probabilistically select between several different action sets. In terms of the syntax, in addition to action nodes $d = B.d'$ for $B \subseteq Act_{\mathfrak{p}_i,\tau}$, we allow $d = \mu.d'$, where μ is a discrete probability distribution over $Act_{\mathfrak{p}_i,\tau}$, indicating that each action set B_i may be picked, with probability $\mu(B_i)$, before proceeding to the sub-tree d'.

Example 2 *Figure 3 illustrates a possible attacker strategy for the tree in Figure 1. The strategy is represented by the decision tree with the following syntactic term:*

$$d_1^A = \{se\}.\emptyset.\mathit{if}(\mathfrak{p}_1? \wedge \mathfrak{p}_2?, \{ef\}.\mathit{stop}, \emptyset.\mathit{stop})$$

According to the strategy d_1^A, in the first phase, the attacker sends an e-mail with an attachment. As there are no attacker actions in the second phase, the attacker does nothing and moves to the third phase. Before performing an action in the third phase, the attacker checks the success of the previous phases $\mathfrak{p}_1, \mathfrak{p}_2$. In case of success, i.e., if the attacker successfully sent an e-mail in phase $\mathfrak{p}_1$ and the defender did nothing or failed to run the anti-virus program in phase $\mathfrak{p}_2$, the attacker executes the file; otherwise, they do nothing.

Example 3 *Figure 4 illustrates a possible defender strategy for the same tree. The difference for a defender strategy with respect to an attacker strategy is that in each phase the defender knows about the success of the previous phases but not about actions attempted by attacker in that phase. The defender strategy is represented by the following term:*

$$d_1^D = \emptyset.\mathit{if}(\mathfrak{p}_1?, \{rav\}.\mathit{if}(\mathfrak{p}_1? \wedge \mathfrak{p}_2?, \{rr\}.\mathit{stop}, \emptyset.\mathit{stop}), \emptyset.\emptyset.\mathit{stop})$$

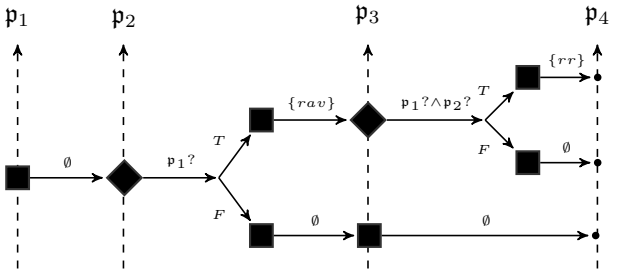


Fig. 4. A defender strategy for the attack-defence tree of Fig 1.

In the first phase, the defender does not have any actions to perform. In the second, it runs the anti-virus program if $\mathfrak{p}_1$ *was successful, that is, if the attacker succeeded in sending a virus over e-mail; otherwise, they do nothing. In the third phase, depending on the success of the previous phases, the defender either restores the registry or does nothing.*

C. Semantics of Attack-Defence Trees

So far we have presented the syntax of attack-defence trees and explained how strategies can be represented by means of decision trees. We now formalise, given a tree and strategies for both players, the semantics of the resulting attack-defence scenario. In classical attack-defence trees, this means determining the value for the root node of the attack tree. The computation depends on the type of semantics chosen for the basic actions, e.g., Boolean, probabilistic, etc. Various evaluation approaches can be used. For example, a bottom-up evaluation (without sequential operator) for the Boolean and probabilistic case can be found in [8] and for multi-objective (Pareto) analyses in [9].

Since our attack-defence trees incorporate temporal ordering (through the sequential operators) and our strategies involve dependencies on earlier events, we define our semantics as a state-based model capturing the possible sequences of events that can unfold. As discussed below, basic actions can fail stochastically, so the semantics is in fact represented as a (discrete-time) Markov chain. We will also annotate this model with costs or rewards (e.g., to represent the cost of implementing a strategy) and, later, will show how the semantic model can be analysed against a range of quantitative measures using probabilistic model checking. This will allow us to check for the existence (or absence) of particular strategies, for example: "what is the maximum probability of a successful attack?" or "can the expected cost of an attack be less than 500 whilst maintaining an attack success probability of at least 0.005?".

Probabilities and costs. We associate each basic action a with a probability $p(a)$ that it is achieved successfully if a is performed. Moreover, we assume that each basic action a has a cost $c(a)$ of performing it.[1] Note that, the probabilities of individual basic actions are all assumed to be independent of each other. Similarly, the costs incurred by each action and by each player are all independent. The relationship between actions (and the resulting impact this has in terms of, e.g., the probability of a successful attack) is captured by the structure of the attack tree.

Semantics. We define the DTMC semantics for attack-defence tree t, with n phases, represented by non-sequential trees $t = t_1, \ldots, t_n$, probability and cost functions p and c, and decision trees d_1^D, d_1^A. The semantics is given by the function build in Table III. It constructs the DTMC, which takes the form of a tree, recursively, each call returning a new root state.

Within a phase of an attack tree, the order in which players perform actions is not determined. But in the DTMC, this needs to be made explicit. Since we assume, as discussed earlier, that the attacker knows the actions attempted by the defender in the current phase, in the DTMC semantics, the defender is the first to move in each phase.

The build function operates recursively over the two decision trees: each call is of the form $\mathsf{build}(d^\tau, d^{\tau^{-1}}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i)$, where the parameters have the following meaning. The first two parameters correspond to a player's decision tree d_i^τ and the opposite player's decision tree $d_i^{\tau^{-1}}$ that still have to be evaluated. The third parameter represents the next player to move, and is used to identify the end of each phase, i.e, it is the end of the phase if $\tau = A$. The remaining parameters record the phases that were successful ($\mathit{Succ}_\mathfrak{p}$), the set of actions attempted in the current phase (Done_i) and the ones that succeeded (Succ_i). At the top-level, the function is called as $\mathsf{build}(d_1^D, d_1^A, D, \emptyset, \emptyset, \emptyset)$.

If the decision tree is an *if*-clause, $d_i^\tau = \mathit{if}(c_i^\tau, d_i'^\tau, d_i''^\tau)$, we evaluate the condition c_i^τ over the success of the previous phases and, when $\tau = A$, also over the set of actions attempted in the current phase. The evaluation $[\![c_i^\tau]\!](\mathit{Succ}_\mathfrak{p}, \mathit{Done}_i)$ is a standard Boolean evaluation, where c_i^τ is a Boolean expression and $(\mathit{Succ}_\mathfrak{p}, \mathit{Done}_i)$ give the truth values of the variables. If $\tau = D$ we can omit the component Done_i from the evaluation. The DTMC is constructed recursively, from either $d_i'^\tau$ or $d_i''^\tau$, depending on whether c_i^τ evaluates to true.

If the root of the decision tree is an action node containing action a, i.e., $d_i^\tau = (B \cup \{a\}).d_i^{\tau^{-1}}$, we create a DTMC state labelled with a, with outgoing transitions corresponding to the success or failure of executing a (with probability $p(a)$ and $1-p(a)$). We also label the transitions with the cost $c(a)$. The successor states are constructed recursively, adding a to Done_i and, if appropriate, Succ_i.

If the set of actions in the action node is empty, $d_i^\tau = \emptyset.d_{i+1}^\tau$, and the current player is D, it means that the defender does not have any more moves in phase i and we need to start exploring the attacker decision tree.

On the contrary, if the action set is empty and the current player is A, then we are at the end of the phase. Hence, we evaluate the success of phase i based on the set Succ_i, $[\![t_i]\!](\mathit{Succ}_i)$, where $[\![t_i]\!]$ is the Boolean formula of which the non-sequential sub-tree t_i is a parse tree and the actions in the formula are 1 if the actions are in the set Succ_i and 0 otherwise. If phase i was successful, we add $\mathfrak{p}_i$ to the set

[1] Determining realistic estimates for the probabilities and costs for basic actions is a research topic in itself and is outside the scope of this work.

TABLE III
THE FUNCTION build DESCRIBING THE SEMANTICS OF AN ATTACK-DEFENCE TREE AS A DTMC.

$$\mathsf{build}(\mathit{if}(c_i^\tau, {d'_i}^\tau, {d''_i}^\tau), {d_i^\tau}^{-1}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) = \begin{cases} \mathsf{build}({d'_i}^\tau, {d_i^\tau}^{-1}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) & \text{if } [\![c_i^\tau]\!](\mathit{Succ}_\mathfrak{p}, \mathit{Done}_i) \\ \mathsf{build}({d''_i}^\tau, {d_i^\tau}^{-1}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) & \text{if } \neg[\![c_i^\tau]\!](\mathit{Succ}_\mathfrak{p}, \mathit{Done}_i) \end{cases}$$

$\mathsf{build}((B \cup \{a\}).d_{i+1}^\tau, {d_i^\tau}^{-1}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) =$ new state s with:
$L(s) = \{a\}, P(s, s') = p(a), P(s, s'') = 1 - p(a)$ where:
$s' = \mathsf{build}(B.d_{i+1}^\tau, {d_i^\tau}^{-1}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i \cup \{a\}, \mathit{Succ}_i \cup \{a\})$
$s'' = \mathsf{build}(B.d_{i+1}^\tau, {d_i^\tau}^{-1}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i \cup \{a\}, \mathit{Succ}_i)$
and: $r(s, s') = r(s, s'') = c(a)$

$$\mathsf{build}(\emptyset.d_{i+1}^\tau, {d_i^\tau}^{-1}, D, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) = \mathsf{build}({d_i^\tau}^{-1}, d_{i+1}^\tau, A, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i)$$

$$\mathsf{build}(\emptyset.d_{i+1}^\tau, {d_{i+1}^\tau}^{-1}, A, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) = \begin{cases} \mathsf{build}({d_{i+1}^\tau}^{-1}, d_{i+1}^\tau, D, \mathit{Succ}_\mathfrak{p} \cup \{\mathfrak{p}_i\}, \emptyset, \emptyset) & \text{if } [\![t_i]\!](\mathit{Succ}_i) \\ \mathsf{build}({d_{i+1}^\tau}^{-1}, d_{i+1}^\tau, D, \mathit{Succ}_\mathfrak{p}, \emptyset, \emptyset) & \text{if } \neg[\![t_i]\!](\mathit{Succ}_i) \end{cases}$$

$\mathsf{build}(\mathit{stop}, \mathit{stop}, A, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) =$ new state s with:
$L(s) = \{\mathit{success}\}$ if $[\![t]\!](\mathit{Succ}_\mathfrak{p}, \mathit{Succ}_i)$ and $\{\mathit{failure}\}$ otherwise

TABLE IV
PROBABILITIES AND COSTS FOR THE BASIC ACTIONS IN THE EXAMPLE.

Label	Name of the Node	Success probability	Cost
se	send e-mail with attachment	0.2	20
usb	distribute USB stick	0.6	80
rav	run anti-virus	0.7	70
ef	execute file	0.75	50
rr	restore registry	0.85	65

$\mathit{Succ}_\mathfrak{p}$. We start the new phase $\mathfrak{p}_{i+1}$ with player D to move next, and resetting the sets Done_i and Succ_i to empty.

Finally, if the decision trees for both players consist of the stop node, and A is to move next, then we are at the end of both strategies. We create a final node in the DTMC and label it with the result of the evaluation of the tree t over the success of all phases.

Randomisation. As mentioned in Sect. III-A, we also consider random selection of actions in decision trees. These can be added to the semantics in Table III in straightforward fashion: a node $d = \mu.d'$ results in a single DTMC state with one outgoing transition for each element of the support of μ, each of which is a normal action node of the form $d'' = B.d'$.

Properties of DTMCs. Once we have obtained a DTMC, we can verify the properties of interest by means of probabilistic model checking [20]. Below, we will see some examples of security properties verified on the DTMC corresponding to the tree given in Figure 1.

Example 4 *Consider the example attack-defence tree given in Figure 1, the strategies for attacker and defender given in Figures 3 and 4, and the probability and cost values for basic actions listed in Table IV. Figure 5 shows the resulting DTMC semantics. We verify the following security properties: "What is the success probability of an attack?" and "Is the expected cost of an attack smaller than or equal to 500 while the success probability is greater than or equal to 0.005?". The first one is expressed in PCTL as the formula $P_{=?}[F\,success]$ and the obtained result is 0.00675, while the second one is expressed in PCTL as the formula $R_{\leq 500}[F\,success] \wedge P_{\geq 0.005}[F\,success]$ which evaluates to* `true`.

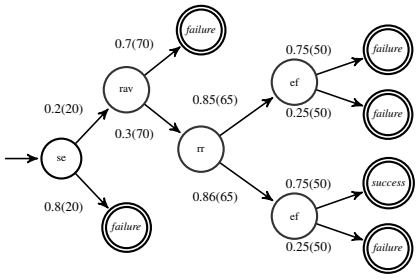


Fig. 5. The DTMC for attack-defence tree from Figure 1 and decision trees d_1^A, d_1^D from Figures 3 and 4 (see Example 4).

IV. GAME-BASED VERIFICATION AND STRATEGY SYNTHESIS

In this paper, we use probabilistic model checking techniques to evaluate attack-defence scenarios using the formalism proposed in the previous section. In particular, we aim to verify whether certain types of attack are impossible, or to synthesise attack or defence strategies satisfying some formally specified property. The basic idea is to transform an attack-defence tree into a stochastic two-player game, in which the players are the attacker and defender, and strategies (in the sense of the stochastic game) correspond to strategies represented by decision trees over an attack tree.

In this section, we explain the transformation of an attack tree to a game and describe how probabilistic model checking can be applied to answer the kinds of questions posed above. We also then explain how to extract decision tree strategies from the results of model checking.

A. Construction of the Stochastic Game

Given an attack-defence tree t with n non-sequential sub-trees (phases) we transform the tree t to a stochastic two-player game (STG) in two steps. First we transform each sub-tree to a game and then combine the games by means of the sequential composition, mimicking the behaviour of a sequential operator connecting corresponding non-sequential sub-trees.

Before explaining the algorithm, it is worthwhile discussing the behaviour of the players in the trees. Each sub-tree represents the static behaviour of the players, i.e., each player makes a choice of their actions independently and the outcome of each action affects only the overall result of the sub-tree and not the other basic actions. Thus, in a game corresponding to a sub-tree, we consider the set of attempted actions for each player instead of one action at a time. Moreover, similarly to a DTMC, we cannot generate games without fixing an order of the players. We assume that the attacker has more information

than the defender, thus in a game the defender will be the first to move. On the contrary, a tree consisting of two non-sequential sub-trees t_1, t_2 combined with a sequential operator illustrates the dynamic behaviour of the players. Here, the choices of the basic actions in t_2 might depend on the outcome of t_1. We take care of this in the sequential composition of two sub-trees, formalised below.

Algorithm 1 displays how to transform a non-sequential tree t_i to a game. The transformation first considers all nondeterministic transitions of the defender and of the attacker, and then the probabilistic transitions. We start with the initial state s_0 belonging to the defender. For all subsets $B \subseteq Act_{\mathfrak{p}_i,D}$ of the defender actions we have an outgoing edge from s_0 entering an attacker state labelled with the subset B. The outgoing edges are labelled with the sum of the costs of the actions in the subset B, $\sum_{a\in B} c(a)$. For each attacker state we do a similar construction, i.e., each attacker state has as many outgoing edges as the subsets $C \subseteq Act_{\mathfrak{p}_i,A}$ of the attacker actions. Similarly, the edges are labelled with the sum of the costs $\sum_{a\in C} c(a)$ and they enter the probabilistic states labelled with the corresponding subset C. Each probabilistic state has two outgoing edges. One of the edges enters the final state labelled with *success* and is labelled with the sum of the success probabilities of the actions that evaluates the tree t_i to true. The other edge enters the final state labelled with *failure* and is labelled with the sum of the failure probabilities. The final states labelled with *success* and *failure* are also instrumented with $\mathfrak{p}_i = T$ meaning that the phase $\mathfrak{p}_i$ was successful, and $\mathfrak{p}_i = F$ meaning that the phase $\mathfrak{p}_i$ failed, respectively.

So far we have described how to transform each non-sequential sub-trees to a stochastic two-player game. We combine the games corresponding to sub-trees by means of the sequential composition. Consider two sub-trees t_1, t_2 connected with a sequential operator $op \in \{\overrightarrow{\wedge}, \overrightarrow{\vee}\}$, $t = op(t_1, t_2)$, and the corresponding games $\mathcal{M}_1, \mathcal{M}_2$. The sequential composition of two games is presented in Algorithm 2, and is as follows. Assume $\mathcal{M}_1$ has m final states. We create m disjoint copies of $\mathcal{M}_2$, denoted $\mathcal{M}_2^1, \cdots, \mathcal{M}_2^m$. For each final state j of $\mathcal{M}_1$ labelled with *success* we connect $\mathcal{M}_2^j$ with $\mathcal{M}_1$ by replacing the final state j of $\mathcal{M}_1$ with the starting state of $\mathcal{M}_2^j$ and adding the label $\mathfrak{p}_i = T$ to the starting state of $\mathcal{M}_2^j$. Similarly, for each final state j of $\mathcal{M}_1$ labelled with *failure* we connect $\mathcal{M}_2^j$ with $\mathcal{M}_1$ by replacing the final state j of $\mathcal{M}_1$ with the starting state of $\mathcal{M}_2^j$ and add the label $\mathfrak{p}_1 = F$ to the starting state of $\mathcal{M}_2^j$. We evaluate and re-label (if needed) each final state of $\mathcal{M}_2^j$ based on the set *Done* of performed actions on the path from the starting state of $\mathcal{M}_1$ till the final state, $[\![t]\!](\mathit{Done})$, where $[\![t]\!]$ is the Boolean formula of which the tree t is a parse tree.

In the special case where there are only sequential conjunctions, we can optimise the construction of the game by merging together all final states labelled with *success* and all final states labelled with *failure*. Observe that merging the final states together does not cause a lose of information in the history of a game.

Algorithm 1 Transformation of non-sequential tree to STG.

Input: a non-sequential tree t_i with probabilistic function p and cost function c for basic actions, and $Act_{\mathfrak{p}_i,A}, Act_{\mathfrak{p}_i,D}, Act = Act_{\mathfrak{p}_i,A} \uplus Act_{\mathfrak{p}_i,D}$ sets
Output: STG $(\Pi, S, s_0, \alpha, (S_A, S_D, S_P, S_\circledcirc), P, T, AP, L)$

$\Pi \leftarrow \{A, D\}$; $\alpha \leftarrow 2^{Act}$; $AP \leftarrow Act \uplus \{\mathit{success}, \mathit{failure}\}$;
$P \leftarrow \emptyset$; $T \leftarrow \emptyset$; $F_1 \leftarrow \emptyset$; $F_2 \leftarrow \emptyset$;
$S_A \leftarrow \emptyset$; $S_D \leftarrow \emptyset$; $S_P \leftarrow \emptyset$; $S_\circledcirc \leftarrow \emptyset$;
Create state s_D; $S_D \leftarrow S_D \cup \{s_D\}$; $s_0 \leftarrow s_D$;
for all $B \subseteq Act_{\mathfrak{p}_i,D}$ **do**
  Create state s_A; $S_A \leftarrow S_A \cup \{s_A\}$; $F_1 \leftarrow F_1 \cup \{s_A\}$;
  $T(s_D, B) \leftarrow s_A$; $L(s_A) \leftarrow B$;
  $r_D(s_D, B) \leftarrow \sum_{a\in B} c(a)$;
end for
for all $s_A \in F_1$ **do**
  for all $C \subseteq Act_{\mathfrak{p}_i,A}$ **do**
    Create state s_P; $S_P \leftarrow S_P \cup \{s_P\}$; $F_2 \leftarrow F_2 \cup \{s_P\}$;
    $T(s_A, C) \leftarrow s_P$;
    $L(s_P) \leftarrow C \cup B$ where $B \subseteq L(s_A)$;
    $r_A(s_A, C) \leftarrow \sum_{a\in C} c(a)$;
  end for
end for
Create states s_s, s_f; $S_\circledcirc \leftarrow S_\circledcirc \cup \{s_s, s_f\}$;
$L(s_s) \leftarrow \{\mathit{success}, \mathfrak{p}_i = T\}$; $L(s_f) \leftarrow \{\mathit{failure}, \mathfrak{p}_i = F\}$;
for all $s_P \in F_2$ **do**
  let $p = \sum_{E\subseteq BC s.t. eval(t,E)} \prod_{a\in E} p(a) \prod_{a\in BC\setminus E} 1 - p(a)$ where $BC \subseteq L(s_P)$;
  $P(s_P, s_s) \leftarrow p$; $P(s_P, s_f) \leftarrow 1 - p$;
end for
$S \leftarrow S_A \uplus S_D \uplus S_P \uplus S_\circledcirc$;

Algorithm 2 Sequential composition of two sub-trees.

Input: an attack-defence tree $t = op(t_1, t_2)$, $op \in \{\overrightarrow{\wedge}, \overrightarrow{\vee}\}$ and corresponding STGs $\mathcal{M}_1, \mathcal{M}_2$
Output: STG $(\Pi, S, s_0, \alpha, (S_A, S_D S_P, S_\circledcirc), P, T, AP, L)$

Let m be the numer of final states in $\mathcal{M}_1$;
Create m disjoint copies $\mathcal{M}_2^1, \cdots, \mathcal{M}_2^m$ of $\mathcal{M}_2$;
Merge $\mathcal{M}_1, \mathcal{M}_2^1, \cdots, \mathcal{M}_2^m$;
Replace each final state j labelled with *"success"* of $\mathcal{M}_1$ with the starting state of $\mathcal{M}_2^j$;
Add the label $\mathfrak{p}_1 = T$ to the starting state of $\mathcal{M}_2^j$;
Replace each final state j labelled with *"failure"* of $\mathcal{M}_1$ with the starting state of $\mathcal{M}_2^j$;
Add the label $\mathfrak{p}_1 = F$ to the starting state of $\mathcal{M}_2^j$;
Change the label of each final state of $\mathcal{M}_2^j$ base on the evaluation $[\![t]\!](\mathit{Done})$;

Example 5 *Let us construct a stochastic two-player game from the tree t, displayed in Figure 1, by following the steps described above. First we transform each basic sub-tree to a game through Algorithm 1. Figure 6 presents the constructed games for each basic sub-tree. As we can see, each game has first the nondeterministic transitions of the defender, then*

the nondeterministic transitions of the attacker and finally the probabilistic transitions. We combine the constructed games by means of the sequential composition, as explained in Algorithm 2. As the tree has only sequential conjunction, we merge the final states with same label during the sequential composition. The full game for the attack-defence tree t is illustrated in Figure 7.

B. Probabilistic Model Checking of Stochastic Games

In the previous section we proposed a transformation from attack-defence trees to stochastic two-player games. The main focus of this section is to show how to evaluate security properties over all possible strategies and how to synthesise optimal attack (or defence) strategies. We start with a discussion of the security properties of interest and then discuss their representation in the temporal logic rPATL. This allows us to perform our analysis of attack-defence trees using the existing model checking techniques implemented in PRISM-games.

Security properties. We can phrase a great many useful quantitative questions on attack-defence scenarios, concerning either one player or both players. It is worth observing that a question might refer to one or both players depending on the parameters they are formulated over. For example, cost-related questions refer to one player: for computing the cost of an attack we do not require the cost of the defender actions. On the other hand, probability-related questions refer to both players, i.e., if the attacker succeeds with probability p then the defender succeeds with probability $1 - p$.

In this work we characterise the basic actions of an attack-defence scenario with the success probability and the cost of an attack and a defence. We then study properties both with one objective, e.g., "is there an attack which is successful with probability greater than or equal to 0.03?" or "what is the maximum success probability of an attack?", and with multiple objectives, such as "can we achieve an attack with an expected cost of at most 500 and a success probability of at least 0.005?".

Verification of security properties. Formal verification is used to determine whether or not the system under study exhibits certain precisely specified properties. For verifying security properties of stochastic two-player games, we exploit probabilistic model checking of rPATL (probabilistic alternating-time temporal logic with rewards) [13]. This logic allows us to express a wide range of properties. For instance, the first single-objective property above is expressed in rPATL as the formula $\langle\langle A\rangle\rangle P_{\geq 0.03}[F\, success]$, while the property with multiple objectives is expressed as the formula $\langle\langle A\rangle\rangle(R^{r_A}_{\leq 500}[F\, success] \wedge P_{\geq 0.005}[F\, success])$.

Model checking systematically explores all states and transitions in the model to check whether it satisfies the given property. Moreover, probabilistic model checking of rPATL also allows us to synthesise strategies for a player with respect to a given property. For instance, we can determine which is the optimal strategy for the attacker in terms of maximising the success probability of the attack, for all possible strategies that the defender may choose. In fact, we can also determine, at the same time, what the best strategy for the defender to ensure that the probability of success does not exceed this.

For verification of multi-objective properties we use an extension of rPATL model checking [14]. The extension allows us both to verify security properties and to synthesise strategies for a player, e.g., "what strategy of the attacker ensures that the expected cost of an attack is at most 500, while the success probability is at least 0.005?". In addition, we can compute the Pareto curve of achievable objectives.

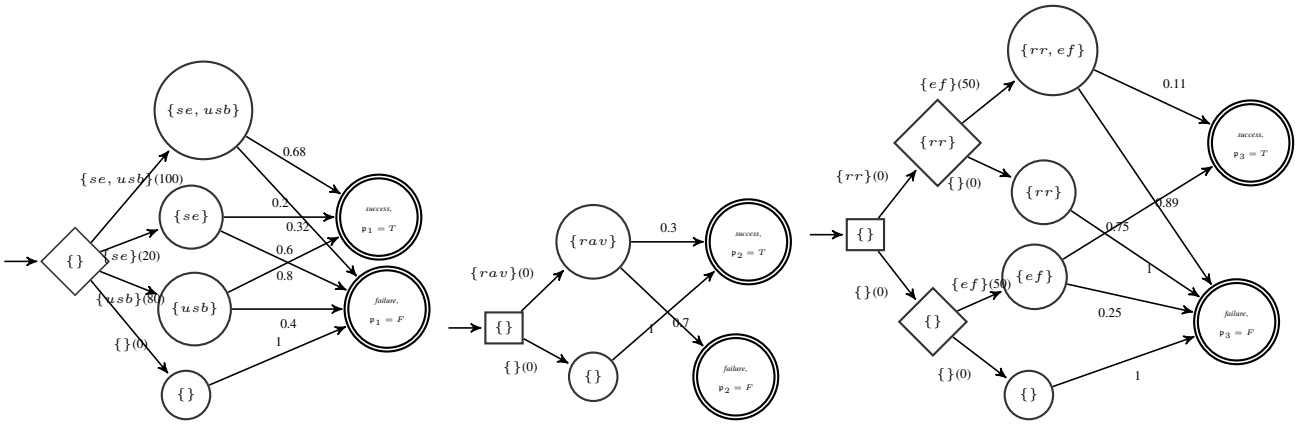
The model checking techniques described here are all implemented in PRISM-games [15], which we therefore employ for verification and strategy synthesis problems on attack-defence trees. PRISM-games also generates optimal strategies and, in the case of multi-objective queries, can compute and display graphically the Pareto curve associated with two objectives.

Correctness. We conclude this section by sketching the correctness of our approach, i.e., that the construction and analysis of the stochastic game described above yields the right answers to questions phrased in terms of attack-defence trees. This relies on the correspondence between an attack-defence tree t, as formalised in Sect. III, and the stochastic two-player game $\mathcal{M}$ whose construction is outlined in Sect.IV-A. More precisely, this depends on a correspondence between decision trees for t and their corresponding attacker or defender player strategies in the stochastic game $\mathcal{M}$.

In Sect. III-C, we gave a precise definition of the semantics of a pair of attacker/defender decision trees d^A, d^D in terms of a discrete-time Markov chain. Each decision tree d^τ has an equivalent strategy, say σ_τ, for player τ in $\mathcal{M}$. As mentioned in Sect. II-B, the behaviour of $\mathcal{M}$ under a pair of strategies σ_A, σ_D is also represented by a discrete-time Markov chain. It is the equivalence of these two Markov chains which underlies the correctness of the overall approach. An important issue here is the class of stochastic game strategies that we need to consider. For the properties used in this paper (those in the logic rPATL, and its multi-objective extension), it suffices to consider *memoryless* strategies, which makes the equivalence of the two Markov chains relatively straightforward.[2] The relationship between stochastic game strategies and decision is expanded upon in the following section.

Example 6 *Consider the game given in Figure 7. We use the tool PRISM-games to verify the security properties mentioned above. For example, the verification of the query $\langle\langle A\rangle\rangle P_{\geq 0.03}[F\, success]$ returns "false", meaning that there is no attack with success probability greater than or equal to 0.03. The verification of the quantitative query $\langle\langle A\rangle\rangle P_{max=?}[F\, success])$ computes the maximum success probability of an attack, which is 0.0229. Figure 7 also shows an optimal attacker strategy, marked in bold. We verify also multi-objective queries, such as $\langle\langle A\rangle\rangle(R^{r_A}_{\leq 500}[F\, success] \wedge P_{\geq 0.005}[F\, success])$. The property evaluates to "true" mean-*

[2]Multi-objective queries in stochastic games need infinite-memory in general [14], but our games are trees (or DAGs) so memoryless strategies suffice.



(a) An STG for $t_1 = \vee(se, usb)$ (b) An STG for $t_2 = \sim rav$ (c) An STG for $t_3 = \wedge(ef, \sim rr)$

Fig. 6. Transformation of basic sub-trees to games (attacker/defender/probabilistic states shown as diamonds/rectangles/circles).

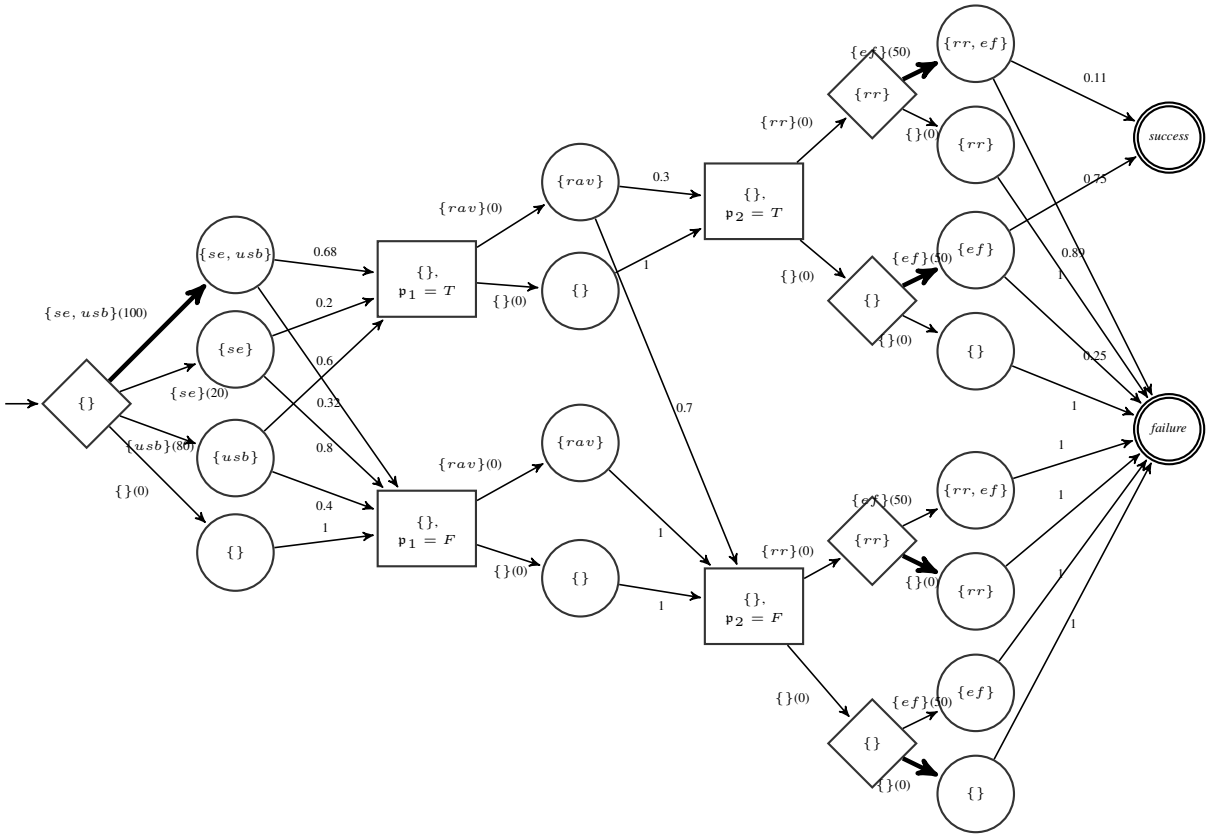


Fig. 7. The stochastic two-player game for attack-defence tree t, from Figure 1. An optimal strategy for the attacker player is marked in bold.

ing that there is an attack with cost at most 500 and success probability at least 0.005. Finally, Figure 8 illustrates the Pareto curve computed by PRISM-games when maximising probabilities and minimising cost of an attack.

C. Synthesising Strategies as Decision Trees

After synthesising an optimal strategy from the stochastic game, as described above, we can transform it to a corresponding decision tree. This provides a high-level, syntactic description of the strategy, in particular, capturing the dependencies within the strategy on the outcomes of earlier actions and choices by players. We now describe this process, first for an attacker, and then for a defender.

Attacker strategies. Synthesis of an attacker decision tree, from an STG $\mathcal{M}$ and attack strategy σ_A, is done using the recursive function $\mathsf{generateAD}(s, i)$, shown in Table V, which operates over the structure of $\mathcal{M}$. The first parameter s is a state of $\mathcal{M}$ and the second parameter i keeps track of the current phase (to

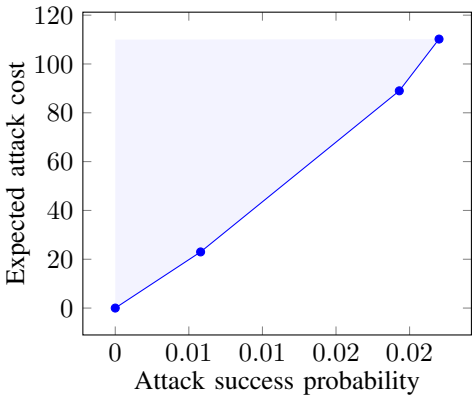


Fig. 8. Pareto curve illustrating the trade-off between attack success probability and expected attack cost over strategies in the running example.

be precise, the phase i associated with the decision tree node currently being created). At the top-level, we call the function as $\mathsf{generateAD}(s_0, 1)$, where s_0 is the initial state of $\mathcal{M}$.

By construction of the game $\mathcal{M}$ (see Sect.IV-A), its states are grouped by phase, and within each phase there are (possibly) defender and then (possibly) attacker states, followed by probabilistic states at the end of the phase. We treat the three classes of state separately.

If s is a defender state, $s = s_D \in S_D$, then the strategy σ_A will not have resolved the choice of actions in s_D and we need to consider each of the possible outgoing branches. These will be translated into *if* statements in the decision tree, which can ask whether a defender action was performed in the current phase (see Sect. III-B). This is done by calling $\mathsf{construct}(s_D, i, Act_{\mathfrak{p}_i,D}, \emptyset)$, explained below.

If the state s is an attacker state, $s = s_A \in S_A$, we create an action node with the set of attacker actions performed in state s_A, as specified by the strategy choice $\sigma_A(s_A)$, and the next node in the decision tree is generated recursively for the successor state $T(s_A, \sigma_A(s_A))$, which, by construction of the game, will be a probabilistic state.

For a probabilistic state, $s = s_P \in S_P$, we have reached at the end of current phase in the STG. We create a decision node whose condition depends on the success of the phase, and then recursively construct the decision tree for the successor states of s_P corresponding to the scenarios where the phase succeeded or failed. Notice that we create a decision node for phase i (i.e., a node d_i^A from Table II) which queries the state of the preceding phase $\mathfrak{p}_{i-1}$. Once we reach the end of the phases (indicated by $i = n{+}1$), we have reached the end of the STG and there are no further actions to be taken so we create a stop node in the decision tree.

As mentioned above, defender states s_D are treated using an auxiliary recursive function $\mathsf{construct}(s_D, i, \mathit{LA}, \mathit{Done})$, which is also given in Table V. The first two parameters are as for $\mathsf{generateAD}$, the third, *LA*, is the set of the defender actions to be performed and the last, *Done*, is the set of actions already performed. The function iterates over the actions in *LA* (initially, the set $Act_{\mathfrak{p}_i,D}$ of all defender actions for phase i), each time removing an action a and creating a decision node

TABLE V
$\mathsf{generateAD}$: CONSTRUCTION OF ATTACKER DECISION TREE FROM STG AND ATTACKER PLAYER STRATEGY.

$\mathsf{generateAD}(s_D, i) = \mathsf{construct}(s_D, i, Act_{\mathfrak{p}_i,D}, \emptyset)$

$\mathsf{generateAD}(s_A, i) = \sigma_A(s_A).\mathsf{generateAD}(s_P, i+1)$
where $s_P = T(s_A, \sigma_A(s_A))$

$\mathsf{generateAD}(s_P, i) = \mathit{if}(\mathfrak{p}_{i-1}?, \mathsf{generateAD}(s', i), \mathsf{generateAD}(s'', i))$
where $P(s_P, s') > 0 \wedge$ "$\mathfrak{p}_{i-1} = T$" $\in L(s')$
and $P(s_P, s'') > 0 \wedge$ "$\mathfrak{p}_{i-1} = F$" $\in L(s'')$

$\mathsf{generateAD}(s_P, n+1) = \mathit{stop}$

$\mathsf{construct}(s_D, i, \mathit{LA} \cup \{a\}, \mathit{Done}) =$
$\mathit{if}(a?, \mathsf{construct}(s_D, i, \mathit{LA}, \mathit{Done} \cup \{a\}), \mathsf{construct}(s_D, i, \mathit{LA}, \mathit{Done}))$

$\mathsf{construct}(s_D, i, \emptyset, \mathit{Done}) =$
$\begin{cases} \mathsf{generateAD}(s', i) & \text{if } s' = T(s_D, \mathit{Done}) \in S_A \\ \emptyset.\mathsf{generateAD}(s', i+1) & \text{if } s' = T(s_D, \mathit{Done}) \in S_P \end{cases}$

TABLE VI
$\mathsf{generateDD}$: CONSTRUCTION OF DEFENDER DECISION TREE FROM STG AND DEFENDER PLAYER STRATEGY.

$\mathsf{generateDD}(s_D, i, DA) = \sigma_D(s_D).\mathsf{generateDD}(s', i+1, AD)$
where $s' = T(s_D, \sigma_D(s_D))$

$\mathsf{generateDD}(s_A, i, AD) = \mathsf{generateDD}(s_P, i+1, AD)$
where $s' = T(s_A, B)$ for some B

$\mathsf{generateDD}(s_A, i, DA) = \emptyset.\mathsf{generateDD}(s_P, i+1, AD)$
where $sP = T(s_A, B)$ for some B

$\mathsf{generateDD}(s_P, i, AD) =$
$\mathit{if}(\mathfrak{p}_{i-1}?, \mathsf{generateDD}(s', i, DA), \mathsf{generateDD}(s'', i, DA))$
where $P(s_P, s') > 0 \wedge$ "$\mathfrak{p}_{i-1} = T$" $\in L(s')$
and $P(s_P, s'') > 0 \wedge$ "$\mathfrak{p}_{i-1} = F$" $\in L(s'')$

$\mathsf{generateDD}(s_P, n+1, AD) = \mathit{stop}$

with condition $a?$ and recursively building the decision tree for the cases where the condition is true or false. This creates decision nodes that branch over the possible combinations. Once, the parameter *LA* is empty, we recursively construct the next part of the decision tree, using the outgoing transitions of the s_D state. These will either go to an attacker state s_A or directly to a probabilistic state s_P. In the latter case, we add an action node with an empty action set, indicating that the attacker performs no actions in this phase.

Defender strategies. The generation of a defender decision tree from a game and a defender tree is slightly different, since, here, the defender can ask only about the success of the previous phases, not any attacker actions from the current phase. Again, we use a recursive function operating over the states of the game $\mathcal{M}$. This function, $\mathsf{generateDD}(s, i, DA)$ is shown in Table VI and constructs a decision tree for a strategy σ_D of the defender player in $\mathcal{M}$. Parameters s and i are the current state and phase, as for $\mathsf{generateAD}$, above; parameter $\tau\tau^{-1}$ is used to keep track of the player. At the top-level, we call the function as $\mathsf{generateDD}(s_0, 1, DA)$.

If the state is a defender state, $s = s_D \in S_D$, we create an action node with the set of actions performed by the defender in s_D, obtained from the defender strategy σ_D, and proceed recursively using the successor of s_D chosen by σ_D.

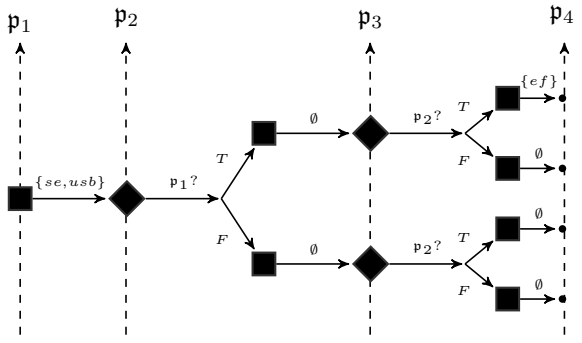


Fig. 9. An attack decision tree for the optimal attacker strategy highlighted in the stochastic game shown in Figure 7.

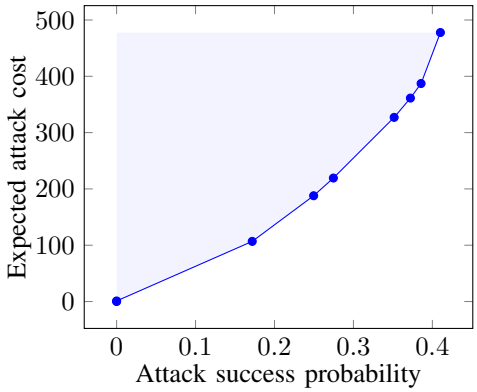


Fig. 10. Pareto curve illustrating the trade-off between attack success probability and expected attack cost over strategies for the RFID example.

If the state is an attacker state, $s = s_A \in S_A$, and the order of the players is AD, we just move to the next probabilistic state s_P. The state s_P is chosen nondeterministically. Note, that for any choice of s_P further construction on the decision tree is the same. If the order of the players is DA, this means that there is no defender action in the current phase. Thus, we create an empty set in the decision tree and move to the next probabilistic state s_P.

On probabilistic states, the function generateDD behaves the same as the function generateAD, as described above.

Finally, we note that the decision tree constructed as above can subsequently be optimised by merging identical subtrees and removing decision nodes with identical then/else branches.

Randomisation. As described in Sect. III, we also consider decision trees that incorporate randomisation. This is because optimal strategies for multi-objective properties may be randomised. Here, that means that strategy σ_A (or σ_D) may select a distribution over actions in a state s_A (or s_D), rather than a single action. The decision tree synthesis algorithms in Tables VI and V thus remain unchanged but the rules for s_A and s_D states, respectively generate random action nodes.

Example 7 *The stochastic game in Figure 7 also shows an optimal attacker strategy marked in bold. We show in Figure 9 the (optimised) attacker decision tree corresponding to the optimal attacker strategy.*

V. Implementation and Results

We have developed a prototype implementation of our techniques, comprising a converter from attack-defence trees, specified in XML, into stochastic games modelled in the input language of PRISM-games [15], available from [21]. The output of the tool can then be used to perform verification and strategy synthesis as described earlier.

We applied our approach to a real-life scenario studied in [5]: we consider part of a Radio-Frequency Identification (RFID) goods management system for a warehouse, modified by introducing temporal dependencies between actions.

The warehouse uses RFID tags to electronically identify all goods. In the attack-defence scenario that we consider, the attacker aims to physically remove some RFID tags after infiltrating the building.

In order to achieve this goal, the attacker has to first get into the premises and then into the warehouse. For getting into the premises the attacker can climb over the fence or enter through the main gate. The defender can protect against climbing by setting some barbed wire on the fence. To protect against the barbed wire the attacker can guard against barbs either by using a carpet over the barbs or by wearing protective cloths. Once the attacker succeeds in accessing the premises, they have to get into the warehouse. The attacker can achieve this subgoal either by entering through the door or by entering through the loading dock. The former action can be defended against by monitoring the door with biometric sensors.

The defender can prevent the attacker from attaining the main goal by monitoring the premises with security cameras. In order to overcome the camera issue the attacker can disable them either by shooting a strong laser at the cameras or by video looping the camera feed. The defender, in turn, can employ guards in order to patrol the premises and counter this kind of attack.

The corresponding attack-defence tree is given in Figure 11. The leaves (basic attack and defence actions) of the tree are decorated with success probability and cost values. The attack-defence tree has three phases: the first phase corresponds to the sub-tree with the root "get into premises", the second phase is the "get into warehouse" sub-tree, and the last phase is the sub-tree on the right of the main goal with the defender action on the root. The syntactic term corresponding to each phase and to the full tree is:

$$\begin{aligned} t_1 &= \vee(\wedge(ef, \sim \wedge(bw, \sim \vee(uc, pc))), tg) \\ t_2 &= \vee(\wedge(ed, \sim bs), ld) \\ t_3 &= \sim \wedge(sc, \sim \vee(lc, \wedge(vc, \sim eg))) \\ t &= \overrightarrow{\wedge}(\overrightarrow{\wedge}(t_1, t_2), t_3) \end{aligned}$$

The resulting stochastic game generated from the attack-defence by our approach has 1072 states and 2052 transitions. We verified a variety of properties, including the numerical property $\langle\langle A\rangle\rangle P_{max=?}[F\, success])$ that computes the maximum success probability of an attack (equal to 0.41), and the

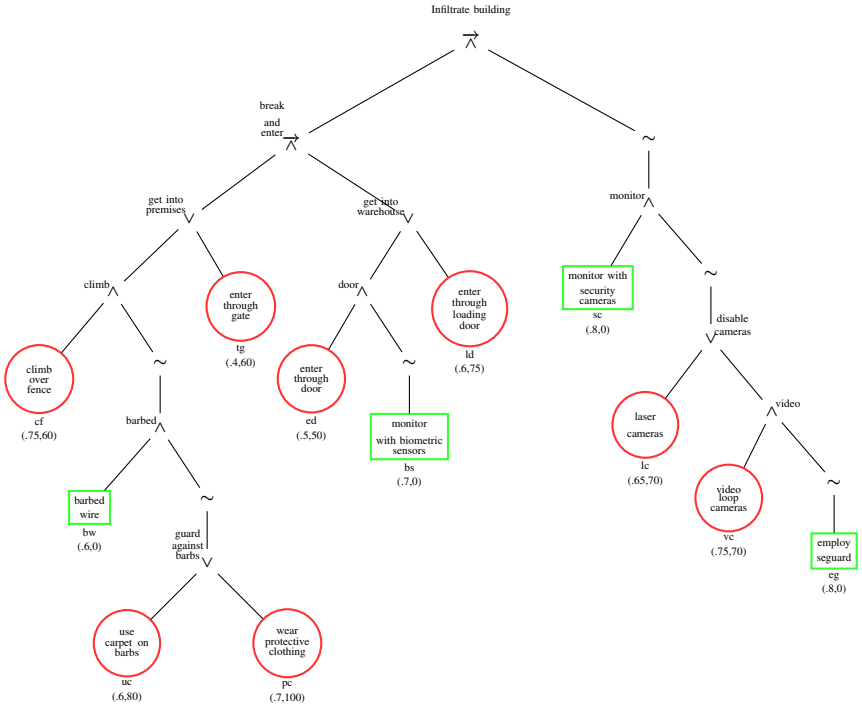


Fig. 11. Attack-defence tree for breaking and entering a building.

multi-objective qualitative property $\langle\langle A\rangle\rangle(R^{r_A}_{\leq 150}[F\,success] \wedge P_{\geq 0.1}[F\,success])$ (which evaluates to true, meaning that there is an attack with cost at most 150 and success probability at least 0.1). We also examine the trade-off between the probability of a successful attack at the expected cost of doing so. The Pareto curve generated for this pair of properties is shown in Figure 10.

VI. Related Work

We now expand on the comparison with related work given in Sect. I. As mentioned earlier, Schneier developed attack trees as an approach to analyse the security of complex systems [1]. Various extensions of this model have been developed, including those that model dependencies between actions: Lv and Li [22] extended attack trees with sequential conjunction, considering the order on the execution of the basic actions in the tree; and Jhawar et al [10] gave a formal semantics of attack trees with sequential conjunction.

While most extensions study static attack trees, a few consider dynamic aspects. Arnold et al. [23] analysed the timing of attack scenarios using continuous-time Markov chains, but do not reason about strategies; [24] used priced time automata and the Uppaal model checker to analyse attack trees, but without probabilities. More recently, [25] explored how stochastic timed automata can be used to study attack-defence scenarios where timing plays a central role. None of these approaches use game-based models.

While attack trees focus on evaluating attack scenarios, other tree-structure representations incorporate countermeasures. Kordy et al. [4] formalised attack-defence trees for this purpose and they can be interpreted with various semantics to answer questions such as the vulnerability of the system to an attack or the minimum cost of an attack [4]. A formalisation of attack-defence trees similar to the one we used has been presented by Aslanyan and Nielson [9], where they proposed evaluation techniques for analysing trees with multiple conflicting parameters in terms of Pareto efficiency. Further developments on attack-defence trees have been carried out, such as combining the tree methodology with Bayesian networks for analysing probabilistic measures of attack-defence trees with dependent actions [19] and studying the relationship between such trees and binary zero-sum two-player games [11]. Moreover, Bistarelli et al. [26] used strategic games for analysing attack-defence scenarios presented with defence trees, an extension of attack trees with countermeasures only on the leaves.

Elsewhere, various studies have explored a game-theoretic approach to modelling security aspects of a system. In particular, stochastic games [12] have proven useful to model uncertainty and randomisation of security scenarios, and have been explored in several application domains. Lye and Wing [27] modelled the security of computer network as a stochastic game and computed Nash equilibrium strategies for the players. Ma et al. [28] presented a game-theoretic approach for studying rational attackers and defenders in the security of cyber-physical systems. Along similar lines, Vigo et al. [29] proposed a framework for modelling and analysing the security of cyber-physical systems by means of stochastic games.

VII. Conclusion

Attack-defence trees are a useful tool to study attack-defence scenarios and present the interaction between an attacker and a defender in an intuitive way. Security attributes, associated with the basic actions of attack-defence trees, provide the basis for various types of quantitative analysis.

In this paper, we explored the relationship between attack-defence trees and stochastic two-player games. We proposed a framework for evaluating security properties of attack-defence scenarios, by developing an extension of attack-defence trees in which temporal dependencies among subgoals can be expressed. In order to formally represent strategies for the players in presence of such dependencies, we have defined the novel concept of decision trees, whose semantics we have given in terms of discrete-time Markov chains. Moreover, we have shown how to encode an attack-defence tree into a stochastic two-player game, where it becomes natural to study the interaction between players and to account for quantitative and probabilistic aspects of a scenario. This allows us to exploit the power of probabilistic model checking techniques and tools, to verify security properties automatically and synthesise strategies for attacks and defences. These strategies can be converted to decision trees, linking the outcome of the verification on the game model to the original attack-defence tree, facilitating communication of the results to end-users.

We implemented our approach in a prototype tool and applied it to the example of an RFID goods management system, where the analysis gives insights on the points of the system open to attack and the corresponding effort to the attacker and likelihood of success.

Our current approach requires that sequential operators only occur above non-sequential operators in an attack-defence tree. In future work, we plan to generalise the approach and allow sequential operators to occur anywhere in a tree. Moreover, we plan to move from fully-observable games to partially-observable ones.

Acknowledgements

Part of the research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318003 (TREsPASS). Special thanks also go to Roberto Vigo for valuable comments.

References

[1] B. Schneier, "Attack Trees: Modeling Security Threats," *Dr. Dobb's Journal of Software Tools*, vol. 24, no. 12, pp. 21–29, 1999.

[2] R. Vigo, F. Nielson, and H. R. Nielson, "Automated generation of attack trees," in *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, 2014, pp. 337–350.

[3] M. G. Ivanova, C. W. Probst, R. R. Hansen, and F. Kammüller, "Attack tree generation by policy invalidation," in *Information Security Theory and Practice - 9th IFIP WG 11.2 International Conference, WISTP 2015 Heraklion, Crete, Greece, August 24-25, 2015 Proceedings*, 2015, pp. 249–259.

[4] B. Kordy, S. Mauw, S. Radomirovic, and P. Schweitzer, "Foundations of attack-defense trees," in *Formal Aspects of Security and Trust - 7th International Workshop, FAST 2010*, 2010, pp. 80–95.

[5] A. Bagnato, B. Kordy, P. H. Meland, and P. Schweitzer, "Attribute decoration of attack-defense trees," *IJSSE*, vol. 3, no. 2, pp. 1–35, 2012.

[6] The TREsPASS Consortium, "Project web page," Available at http://www.trespass-project.eu.

[7] The ADT2P Consortium, "Project web page," Available at http://wwwen.uni.lu/snt/research/research_projects2/adt2p.

[8] B. Kordy, S. Mauw, and P. Schweitzer, "Quantitative questions on attack-defense trees," in *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*, 2012, pp. 49–64.

[9] Z. Aslanyan and F. Nielson, "Pareto efficient solutions of attack-defence trees," in *Principles of Security and Trust - 4th International Conference, POST 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings*, 2015, pp. 95–114.

[10] R. Jhawar, B. Kordy, S. Mauw, S. Radomirovic, and R. Trujillo-Rasua, "Attack trees with sequential conjunction," in *ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, 2015, Proceedings*, 2015, pp. 339–353.

[11] B. Kordy, S. Mauw, M. Melissen, and P. Schweitzer, "Attack-defense trees and two-player binary zero-sum extensive form games are equivalent," in *Decision and Game Theory for Security - First International Conference, GameSec 2010, Berlin, Germany, November 22-23, 2010. Proceedings*, 2010, pp. 245–256.

[12] L. S. Shapley, "Stochastic games," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 39, no. 10, p. 1095, 1953.

[13] T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, and A. Simaitis, "Automatic verification of competitive stochastic systems," *Formal Methods in System Design*, vol. 43, no. 1, pp. 61–92, 2013.

[14] T. Chen, V. Forejt, M. Z. Kwiatkowska, A. Simaitis, and C. Wiltsche, "On stochastic games with multiple objectives," in *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, 2013, pp. 266–277.

[15] M. Kwiatkowska, D. Parker, and C. Wiltsche, "Prism-games 2.0: A tool for multi-objective strategy synthesis for stochastic games," in *Proc. 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'16)*, ser. LNCS. Springer, 2016.

[16] J. Kemeny, J. Snell, and A. Knapp, *Denumerable Markov Chains*, 2nd ed. Springer-Verlag, 1976.

[17] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[18] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, ser. LNCS (Tutorial Volume), M. Bernardo and J. Hillston, Eds., vol. 4486. Springer, 2007, pp. 220–270.

[19] B. Kordy, M. Pouly, and P. Schweitzer, "A probabilistic framework for security scenarios with dependent actions," in *Integrated Formal Methods - 11th International Conference, IFM 2014, Bertinoro, Italy, September 9-11, 2014, Proceedings*, 2014, pp. 256–271.

[20] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Formal Methods for Performance Evaluation, 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*, 2007, pp. 220–270.

[21] Http://www2.compute.dtu.dk/~zaas/ADT2PRISM.zip.

[22] W.-p. Lv and W.-m. Li, "Space based information system security risk evaluation based on improved attack trees," in *Multimedia Information Networking and Security (MINES), 2011 Third International Conference on*. IEEE, 2011, pp. 480–483.

[23] F. Arnold, H. Hermanns, R. Pulungan, and M. Stoelinga, "Time-dependent analysis of attacks," in *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, 2014, pp. 285–305.

[24] R. Kumar, E. Ruijters, and M. Stoelinga, "Quantitative attack tree analysis via priced timed automata," in *Formal Modeling and Analysis of Timed Systems - 13th International Conference, FORMATS 2015, Madrid, Spain, September 2-4, 2015, Proceedings*, 2015, pp. 156–171.

[25] H. Hermanns, J. Krämer, J. Krcál, and M. Stoelinga, "The value of attack-defence diagrams," in *Principles of Security and Trust - 5th International Conference, POST 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, 2016, pp. 163–185.

[26] S. Bistarelli, M. Dall'Aglio, and P. Peretti, "Strategic games on defense trees," in *Formal Aspects in Security and Trust, Fourth International Workshop, FAST 2006, Hamilton, Ontario, Canada, August 26-27, 2006, Revised Selected Papers*, 2006, pp. 1–15.

[27] K.-w. Lye and M. J. Wing, "Game strategies in network security," *International Journal of Information Security*, vol. 4, no. 1, pp. 71–86, 2005.

[28] C. Y. Ma, N. S. Rao, and D. K. Yau, "A game theoretic study of attack and defense in cyber-physical systems," in *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*. IEEE, 2011, pp. 708–713.

[29] R. Vigo, A. Bruni, and E. Yüksel, "Security games for cyber-physical systems," in *Secure IT Systems - 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, October 18-21, 2013, Proceedings*, 2013, pp. 17–32.