Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

TC 11 Briefing Papers

Quantitative Security Risk Modeling and Analysis with RisQFLan

Maurice H. ter Beek[a,*], *Axel Legay*[b], *Alberto Lluch Lafuente*[c], *Andrea Vandin*[c,d]

[a] *ISTI–CNR, Pisa, Italy*
[b] *UCLouvain, Belgium*
[c] *DTU, Lyngby, Denmark*
[d] *Sant'Anna School of Advanced Studies, Pisa, Italy*

ARTICLE INFO

Article history:
Received 1 January 2021
Revised 23 May 2021
Accepted 19 June 2021
Available online 25 June 2021

Keywords:
Graph-based security risk models
Attack-defense trees
Probabilistic model checking
Statistical model checking
Formal analysis tools

ABSTRACT

Domain-specific *quantitative* modeling and analysis approaches are fundamental in scenarios in which *qualitative* approaches are inappropriate or unfeasible. In this paper, we present a tool-supported approach to quantitative graph-based security risk modeling and analysis based on attack-defense trees. Our approach is based on QFLan, a successful domain-specific approach to support quantitative modeling and analysis of highly configurable systems, whose domain-specific components have been decoupled to facilitate the instantiation of the QFLan approach in the domain of graph-based security risk modeling and analysis. Our approach incorporates distinctive features from three popular kinds of attack trees, namely enhanced attack trees, capabilities-based attack trees and attack countermeasure trees, into the domain-specific modeling language. The result is a new framework, called RisQFLan, to support quantitative security risk modeling and analysis based on attack-defense diagrams. By offering either exact or statistical verification of probabilistic attack scenarios, RisQFLan constitutes a significant novel contribution to the existing toolsets in that domain. We validate our approach by highlighting the additional features offered by RisQFLan in three illustrative case studies from seminal approaches to graph-based security risk modeling analysis based on attack trees.

1. Introduction

Quantitative modeling and analysis approaches are essential to support software and system engineering in scenarios where *qualitative* approaches are inappropriate or unfeasible, e.g. due to complexity or uncertainty, or by the quantitative nature of the properties of interest. Automated approaches to support quantitative modeling and analysis have been developed extensively during the last decades, including generic as well as domain-specific approaches (cf., e.g., Aslanyan et al. (2016); ter Beek and Legay, 2019; ter Beek et al., 2020; Bernardo et al. (2016); Bozga et al. (2012); Hahn et al. (2019); Hartmanns and Hermanns (2015); Katoen and Larsen (2012); Kumar et al. (2015); Kumar and Stoelinga (2017)).

* Corresponding author.
E-mail address: maurice.terbeek@isti.cnr.it (M.H. ter Beek).
https://doi.org/10.1016/j.cose.2021.102381

QFLan ter Beek et al., 2020 is one example of a successful domain-specific approach to support quantitative modeling and analysis of highly configurable systems, such as software product lines. QFLan combines several well-studied rigorous notions and techniques in an Eclipse-based domain-specific tool framework. It consists of a domain-specific language (DSL) tailored for configurable systems, and an analysis engine based on statistical model checking (SMC) Agha and Palmskog (2018); Legay et al. (2019). In ter Beek et al., 2020, we showed the robustness and scalability of QFLan by analyzing large instances of case studies that could not be analyzed before.

In this paper, we generalize the QFLan approach by decoupling domain-specific components and instantiating the QFLan approach in a new domain: risk modeling and analysis.

The result, called RisQFLan, is a new framework to support graph-based *quantitative security risk modeling and analysis*. It constitutes a significant novel contribution to existing toolsets in that domain. In particular, RisQFLan can be used to:

1. build rich models by combining distinctive features from existing formalisms for risk modeling and analysis;
2. enhance the analysis of existing tools for risk modeling.

Regarding 1), the DSL of RisQFLan has been designed to include the most significant features of existing formalisms based on attack trees, such that they can be combined in the same model. Subsets of the RisQFLan DSL, indeed, can thus be used to capture classes of well-established modeling formalisms. In addition, RisQFLan allows one to focus on specific dynamic threat profiles, a feature that is supported only recently by very few approaches (Aslanyan et al. (2016); Gadyatskaya et al. (2016); Hansen et al. (2017); Kumar et al. (2015, 2018b); Lenin et al. (2014)) and in a limited way (cf. the detailed discussion in Section 8).

We validate feature 2) by showing in Section 7 how three influential classes of risk models based on attack trees can be specified in RisQFLan, and how the RisQFLan analysis capabilities can be used to complement and enrich those provided by existing toolsets. This is an advantage offered with respect to the existing tools. In particular, RisQFLan includes an additional analysis engine based on exact probabilistic model checking that is not inherited from QFLan, which comes with a statistical model checking engine Clarke et al. (2018).

Synopsis

Section 2 introduces the domain of graph-based security risk modeling with attack-defense trees. Section 3 presents a first contribution of the paper: a generalization of the QFLan approach to domain-specific quantitative modeling and analysis. Sections 4–6 describe the main contributions of the paper to support security risk modeling and analysis: the RisQFLan DSL in Section 4, its formal semantics in Section 5 and the analysis capabilities of the RisQFLan tool in Section 6. Section 7 validates the flexibility of RisQFLan by illustrating in detail how features from three influential classes of attack trees can be specified in RisQFLan and how the RisQFLan analysis capabilities can be successfully used to complement and enrich the analyses provided by existing tools. Section 8 discusses related work. Section 9 draws conclusions and outlines future work.

2. Graph-based risk modeling and analysis

This section provides a brief introduction to the specific domain of risk modeling and analysis with graph-based security models. For this purpose, we use as running example the risk assessment of a *"bank robbery"* scenario, which will also be used in Section 4 to illustrate RisQFLan.

Graph-based security models offer an intuitive and effective means to represent security scenarios in complex systems, by combining intuitive visual features with formal semantics, which can then be used for formal analysis. *Attack trees* and their variants Kordy et al. (2011); Mauw and Oostdijk (2005); Schneier (1999) constitute a popular family of graph-based security models for which several approaches have been developed over the last decades (cf., e.g., the surveys Hong et al. (2017); Kordy et al., 2014; Widel et al. (2019)). Attack trees are related to the well-established fault tree formalism and they have been around for decades. They aim at providing scalable and usable methods for specifying vulnerabilities and countermeasures, their interplay and their key attributes such as cost and effectiveness. Attack trees (and attack-defense trees) thus serve as a basis for quantitative risk assessment, which helps to determine, for instance, where defensive resources are best spent to protect a system. Attack trees are effectively used in practice by both the public and private sector (e.g. by European and US aerospace, defense and intelligence organizations, as well as by health care providers, critical infrastructure companies, and financial organizations).

In their simplest form, attack-defense diagrams are and/or-trees whose nodes represent either attack goals or defensive measures, and with sub-trees representing refinements of such goals and measures. Fig. 1 shows an attack-defense diagram modeling our running example. The tree's root represents the main threat under analysis, i.e. robbing a bank (`RobBank`).

Attack nodes can be refined in several ways by identifying necessary sub-goals and combining them in different ways, e.g. with disjunction, (ordered) conjunction, etc. In our example, the attacker has two options to achieve its main goal: either open the vault (`OpenVault`) *or* blow it up (`BlowUp`). This is specified in the tree with corresponding nodes as children of node `RobBank`, combined in a disjunctive way. Another kind of refinement illustrated in our example is the following: in order to open the vault (`OpenVault`), the attacker needs to *first* learn its combo (`LearnCombo`) *and then* get to the vault (`GetToVault`). This is specified by combining `LearnCombo` and `GetToVault` through an ordered conjunction. A last example of refinement is used to model that for security reasons *two out of three* of the vault's opening codes are required (`FindCode1`–`FindCode3`). Instead, blowing it up only requires to get to the vault.

Attack-defense diagrams can also include defensive mechanisms to deal with or to prevent attack threats. In the example scenario, there are two defensive mechanisms. First, a `LockDown` *countermeasure*, triggered by (successful or not) blow up attacks that, once active, mitigates bank robbery attacks. The rationale is that the vault is sealed to prevent robbery when an explosion is detected. The second defensive

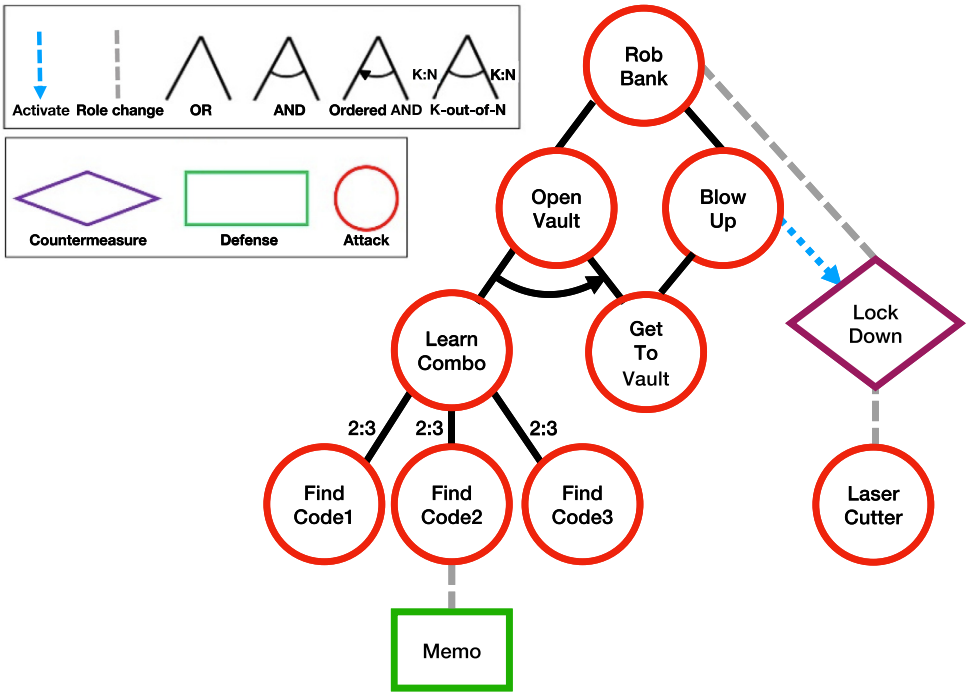


Fig. 1 – An attack-defense diagram.

measure in our running example is a *defense* `Memo`, permanently active against attacks trying to find opening code 2 (`FindCode2`). The interplay between such a defensive countermeasure and the corresponding attack nodes is also typically depicted visually, as in our example. Defensive mechanisms, in turn, can also be affected (e.g. disabled or mitigated) by attacks. For instance, in our example an attack with a `LaserCutter` can break the `LockDown`.

Attack-defense diagrams, besides being a useful tool for modeling and informally reasoning on security risk scenarios, often also have a formal meaning that lies at the basis of formal reasoning, typically supported by effective software tools. Next to a number of academic tools, like ADTool Kordy et al. (2013), SPTool Kordy et al. (2016a), and ATTop Kumar et al. (2018b) (cf. surveys Hong et al. (2017); Kordy et al., 2014; Widel et al. (2019) for further examples), there are several commercial tools for attack trees with decades of history. Isograph offers AttackTree (https://www.isograph.com/software/attacktree/) as part of a suite of decision tree tools. 2T Security (assured service provider of the UK's National Cyber Security Centre) offers Risk*Tree* (https://risktree.2t-security.co.uk/): a process for understanding, recording, and managing risks, using workshops to build so-called RiskTrees that define the risks. Amenaza Technologies offers Secur*ITree* Amenaza Technologies Limited (2006) (https://www.amenaza.com/), which is capable of performing both attack tree and fault tree analysis.

Standard analyses conducted on attack-defense diagrams typically regard the feasibility of attacks (e.g. *can the attacker activate some actions that will result in the achievement of her/his main goal?*), their likelihood (e.g. *what is the probability that the main goal is achieved?*), or their cost (e.g. *what is the cheapest successful attack for the attacker?*). Analysis techniques are often based on constraint solving, optimization, and statistical techniques. Section 6 will provide some of these analyses applied to our running example.

3. Generalizing the QFLan approach

This section describes how the QFLan architecture was made amenable for instantiation in domains beyond the one for which it was conceived (configurable systems like software product lines), and how its analysis capabilities were enriched.

The Original QFLan Architecture

We first summarize the original QFLan architecture as presented in ter Beek et al., 2020, organized in two layers: the Graphical User Interface (`GUI`), devoted to modeling, and the `CORE` layer, devoted to analysis. Both layers are wrapped in an Eclipse-based tool embedding the third-party statistical analyzer MultiVeStA Gilmore et al. (2017); Sebastio and Vandin (2013); Vandin et al. (2021).[1] MultiVeStA is a *model-agnostic* tool, which can be integrated with existing probabilistic simulators to enrich them with distributed statistical analysis capabilities. Section 6 discusses and exemplifies MultiVeStA's analysis capabilities used in our tools. QFLan is an open-source tool. The components of the `GUI` layer are:

- a `QFLAN Editor` with editing support that is typical of a modern Integrated Development Environment (IDE), developed in the `XTEXT` framework, and a `MultiQuaTEx Editor` for property specification in the MultiQuaTEx language Sebastio and Vandin (2013); Vandin et al. (2021);

[1] https://github.com/andrea-vandin/MultiVeStA/wiki

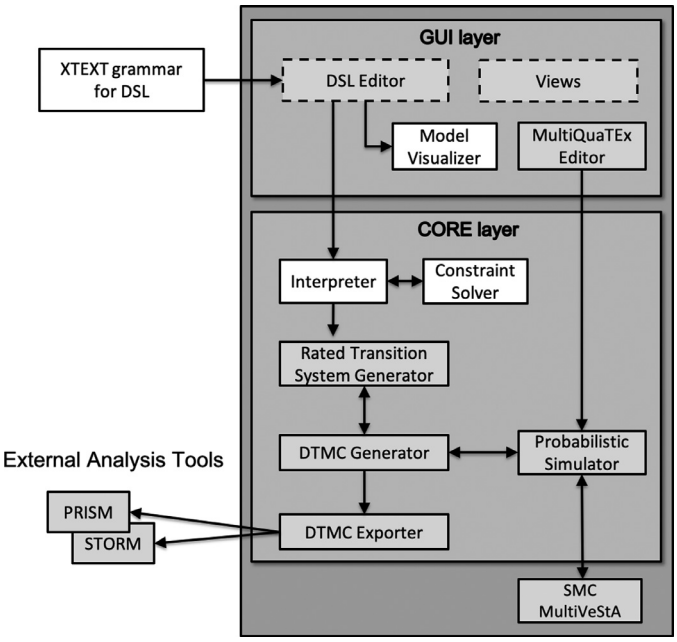


Fig. 2 – The refactored QFLan architecture.

- a set of `Views`, including a project explorer, a diagnosis console and a plot viewer for displaying analysis results.

The components of the CORE layer are:

- a `Probabilistic Simulator`, which is an interpreter of the formal semantics as probabilistic processes. This interacts with the external statistical analyzer MultiVeStA to obtain SMC capabilities;
- a `Built-in Constraint Solver` used by the simulator to check constraints during simulation.

The Refactored QFLan Architecture

The architecture illustrated in Fig. 2 decouples domain-specific components of the QFLan architecture from domain-generic ones. The domain-specific components that need to be provided to instantiate the architecture in a new domain are the following: the `XTEXT grammar for DSL`, the `Interpreter`, the `Constraint Solver` and the `Model Visualizer` (differentiated from other components by their blank background). The remaining components are either existing domain-generic components (solid border) or domain-specific components, automatically generated by XTEXT (dashed border).

The GUI layer basically remains unchanged, except that Fig. 2 makes explicit that the `DSL Editor` is generated automatically from an `XTEXT grammar for DSL`. Moreover, it was extended with a `Model Visualizer` component to offer an automatic visual representation of the model at hand. This is obtained by providing an encoding of the models' features of interest in the DOT language.[2]

The main changes in the refactored QFLan architecture however concern the CORE layer, whose new components are:

- an `Interpreter` and a `Constraint Solver`, implementing the formal semantics of the DSL based on rated transition systems (transition systems with rate-decorated transitions). Together these components compute the possible *target* states (model configurations) reachable in one step from a given *source* state. The `Interpreter` produces each target state by trying to apply each of the rules of the semantics of our DSL, as described in detail in Section 5. To do so, it consults the `Constraint Solver` to solve the conditions needed to apply the rules. The `Constraint Solver` is heavily based on the one developed in ter Beek et al., 2020, which showed to be more efficient than other available solvers for the class of constraints under consideration.
- a `Rated Transition System Generator` relying on the `Interpreter` to generate rated transition systems on-the-fly. More precisely, the `Rated Transition System Generator` iteratively invokes the `Interpreter` to generate on demand the state space of the system under consideration. This means that for each state, it can generate all possible target states together with their (probabilistic) rates.
- a `DTMC Generator` that uses the `Rated Transition System Generator` to normalize rated transition systems into on-the-fly generated Discrete-Time Markov Chains (DTMC). This is achieved by transforming sets of rated transitions from a source state to its target states, into a probabilisitic distribution of target states, where the probability assigned to each target state is proportional to the rate of its transition. This normalization procedure is described in detail in Section 5.
- a `DTMC Exporter`, which generates an entire DTMC by using the `DTMC Generator` and exports it in the input format of the well-known probabilistic model checkers PRISM Kwiatkowska et al. (2011)

[2] https://www.graphviz.org/doc/info/lang.html

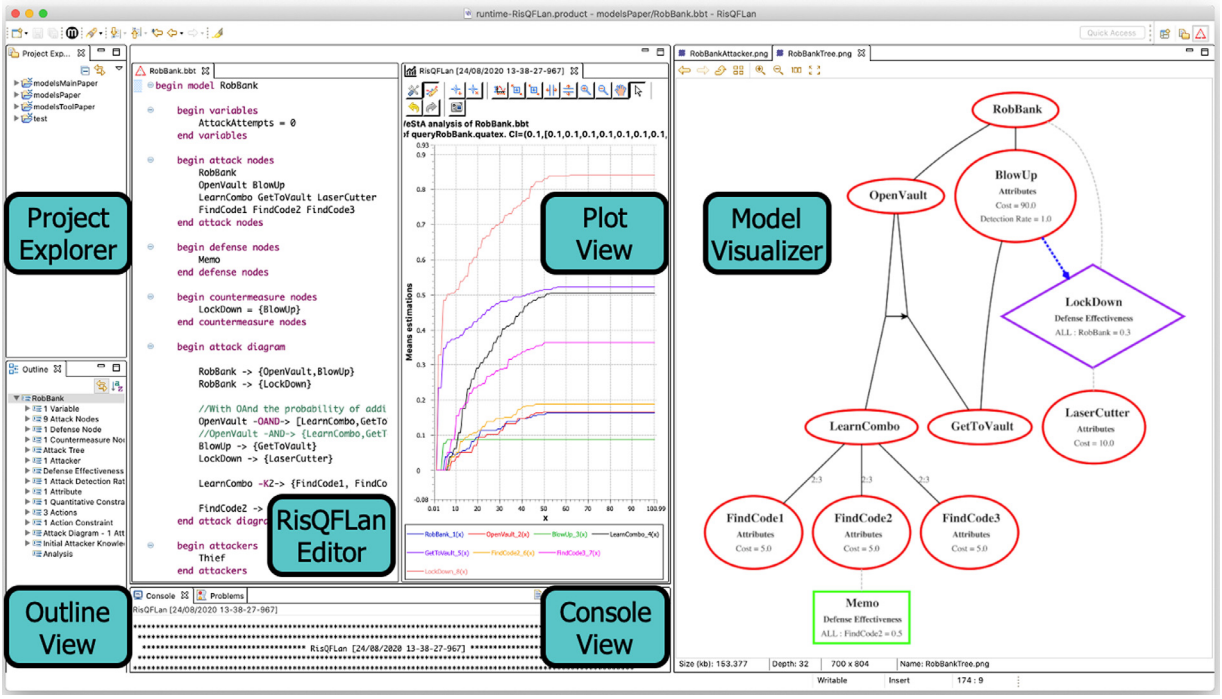


Fig. 3 – A screenshot of RisQFLan.

(https://www.prismmodelchecker.org) and STORM Dehnert et al. (2017) (https://www.stormchecker.org). This component uses the `DTMC Generator` in combination with suitable data structures to store the entire state space so to ensure that no state is missed or explored twice. The generated space of states with probabilistic transitions (i.e. a DTMC) is then exported in the expected text format of the above mentioned tools, together with additional information regarding the properties of interest.

- a `Probabilistic Simulator`, which is now separated from the above components and which is able to simulate a DTMC without fully generating it using the on-the-fly `DTMC Generator`. In particular, the `Probabilistic Simulator` interacts with the `Interpreter`, the `Constraint Solver`, and the `DTMC Generator` to perform probabilistic simulations. These are obtained by iteratively selecting only one of the one-step next states, performing probabilistic choices based on the probabilities computed by the `DTMC Generator`.

4. RisQFLan **DSL**

This section describes RisQFLan, a domain-specific instantiation of QFLan in the security risk domain described in Section 2. A screenshot of RisQFLan is provided in Fig. 3, depicting the implemented components from the GUI layer in Fig. 2. We describe here the DSL of RisQFLan, while its formal semantics is given in Section 5 and its analysis capabilities are presented in Section 6. We illustrate the DSL of RisQFLan through the running example, whose attack-defense diagram is depicted in Fig. 1 (and in Fig. 3).

```
begin attack nodes
 RobBank OpenVault BlowUp
 LearnCombo GetToVault
 FindCode1 FindCode2
 FindCode3 LaserCutter
end attack nodes

begin defense nodes
 Memo
end defense nodes

begin countermeasure nodes
 LockDown = {BlowUp}
end countermeasure nodes
```

Code 1 – Nodes.

In the DSL, nodes are declared in specific blocks, cf. Code 1. Note that countermeasure nodes require to indicate the attack node(s) that may trigger them.

Our attack-defense diagrams relate nodes by two types of relations: (i) *refinements* shape offensive (defensive, resp.) nodes into a set of offensive (defensive, resp.) sub-nodes; (ii) *role-changes* state how to oppose offensive (defensive, resp.) nodes by defensive (offensive, resp.) nodes. Each node has at most one refinement and at most one role-change. Typical for our approach is that nodes may have multiple parents, which is convenient to specify an attack (defense) node that affects multiple defenses (attacks) or an attack (defense) node that refines many attacks (countermeasures).

We offer `OR`, `AND`, `OAND` (ordered `AND`), and `k-out-of-n` refinements for attack and countermeasure nodes. Defense nodes model static, atomic defenses that cannot be refined. Countermeasures are also atomic, but they can be refined with defense nodes to permit *reactive* defense nodes that be-

```
begin attack diagram
 RobBank -> {OpenVault, BlowUp}
 OpenVault -OAND-> [LearnCombo, GetToVault]
 BlowUp -> {GetToVault}
 LearnCombo -K2-> {FindCode1, FindCode2, FindCode3}
 RobBank -> {LockDown}
 LockDown -> {LaserCutter}
 FindCode2 -> {Memo}
end attack diagram
```

Code 2 – Attack-defense diagram.

```
begin attributes
 Cost = {LaserCutter = 10, BlowUp = 90,
         FindCode1 = 5, FindCode2 = 5, FindCode3 = 5}
end attributes
```

Code 3 – Attributes.

come effective only upon (attack detection and) activation of the refined countermeasure. AND and OR refinements originate from the seminal works on attack trees Schneier (1999). OAND refinements stem from *enhanced* and *improved attack trees* Çamtepe and Yener (2007); Lv and Li (2011) and are used to model ordered attacks: sub-nodes can be activated in any order but only the correct order activates the parent node. The k-out-of-n refinements are inspired by *attack countermeasure trees* Roy et al., 2012.

Lines 2–5 of Code 2 show how to declare attack diagrams in RisQFLan. The square brackets of OAND indicate that order matters: OpenVault requires LearnCombo and GetToVault *in that order*. K2 expresses that *at least two* of the three sub-attacks of LearnCombo are required. Inspired by other formalisms supporting both attack and defense mechanisms, like *attack-defense trees* Kordy et al. (2011), a *role-changing* relation describes the attack a countermeasure or defense works against (e.g. LockDown defends against RobBank) or vice versa (e.g. LaserCutter neutralizes LockDown). Lines 6-8 of Code 2 show that attack, defense and countermeasure nodes can additionally have a *role-changing* relation with a child of the opposite role, an opponent node affecting its activation.

As in other approaches Kordy et al., 2014, attack nodes may be decorated with attributes, like cost or detection rates, for quantitative analyses Aslanyan et al. (2016); Hansen et al. (2017); Kordy et al. (2012). The cost of (attempting) an attack, like the attribute Cost in Code 3, may be used to impose constraints. The default value is 0, e.g. Cost(GetToVault) = 0. The cumulative value for the entire scenario, often the cost associated to a (sub-system rooted in a) node, is the sum of the costs of its active descendants Schneier (1999). However, the total cost of an attack should not reflect only the cost of *successful* sub-attacks, as this would be a best-case scenario. Therefore, in RisQFLan we consider both successful and failed attack attempts to compute the value of an attribute of an attack node. Furthermore, we allow attributes also for defensive nodes.

In Amenaza Technologies Limited (2006); Ingoldsby (2013), a *noticeability* attribute is a behavioral metric used to indicate the likeliness of an attack attempt to be noticed. Following *attack countermeasure trees* Roy et al., 2012, we make this notion a first-class citizen of RisQFLan, called *attack detection rate*, which influences activation of countermeasures. More precisely, such a rate determines the probability for an attack attempt, whether successful or not, to be detected, and it triggers the activation of the affected countermeasures, in the sense that higher detection rates lead to more likely activation of countermeasures. The default value is 0, i.e. an attack is undetectable. Code 4 shows that an attempt to blow up a vault is always noticed.

```
begin attack detection rates
 BlowUp = 1.0
end attack detection rates
```

Code 4 – Attack detection rates.

In Kordy et al. (2013, 2011), an attack node is *disabled* if it is affected by a defense. However, a common conception in security is that nothing is 100% secure. Therefore, we include the notion of *defense effectiveness* from Roy et al., 2012 to specify the probability for a defense node to be effective against a combination of attack nodes and attack behavior. The rationale is that different attackers might be affected differently, even when attempting the same attack (e.g. a security guard is efficient against a thief, but not against a military attack). The default value is 0, i.e. the defense has no effect. Code 5 states that Memo scales the probability of succeeding in FindCode2 attacks by $1 - 0.5$, whereas LockDown scales that of RobBank by $1 - 0.3$.

4.1. *Attack behavior*

An important feature of our models is that defensive behavior is *reactive*, while attackers are *proactive*. RisQFLan allows to fine tune security scenarios by defining explicit *attack behavior*, implicitly constrained by an attack-defense diagram. The combination of attack-defense diagrams and explicit (probabilistic) attack behavior was motivated by work on configurable systems ter Beek et al., 2020; Vandin et al. (2018), where QFLan was validated on several case studies. In particular, it was shown that QFLan could handle significantly larger instances of configurable elevator systems than existing approaches. Explicit

```
begin defense effectiveness
 Memo(ALL, FindCode2) = 0.5, LockDown(ALL, RobBank) = 0.3
end defense effectiveness
```

Code 5 – Defense effectiveness (`ALL` denotes any attacker).

```
begin actions
 choose tryGTV try
end actions
```

Code 6 – Actions.

attack behavior enables the analyses of specific attacker types, like script kiddies, insiders, and hackers, which has the advantage of being able to evaluate system vulnerabilities for those attacker types that make more sense for the security scenario at hand. Moreover, it enables novel types of analysis to complement the classical best- and worst-case evaluations of attack graphs (like the bottom-up evaluation in ADTool Kordy et al. (2013)).

Attack behavior is modeled as rated transition systems, whose transitions are labeled with the action being executed and a rate (used to compute the probability of executing the action), and possibly with effects (updates of variables) and guards (conditions on the action's executability), in this order (e.g. Lines 12-13 in Code 7). Fig. 4 (and its corresponding Code 7) sketches an attacker, named `Thief`, that `starts` by choosing to attempt an open vault (`tryOpenVault`) or blow up (`tryBlowUp`) attack. Independently of this choice, s/he can try get-to-vault attacks (`tryGetToVault`), required by both strategies. `OpenVault` requires to try to learn the combo, which in turn requires to try to find at least two codes.

Attacker actions can be *user-defined* for scenario-specific behavior not directly related to node activation, such as `choose`, `tryGTV`, and `try` in Code 6 (where `try` is part of the attacks `tryOpenVault`, `tryLearnCombo`, and `tryFindCode`, not further detailed in Fig. 4 and Code 7).

RisQFLan also provides a set of predefined attacker actions, like `succ` and `fail` for a successful or failed attack *attempt*, resp., modeled by a probabilistic choice between `succ` and `fail` actions, whose associated rates determine the success likelihood together with (the effectiveness of) the involved defenses. In Section 7, we will see how attackers can apply backtracking strategies via the predefined action `remove`.

Attack behavior is executed by considering, at each step, the outgoing transitions from the current state admitted by the attack diagram and by further constraints discussed below. Normalizing the sum of the rates of these transitions to 1 leads to a DTMC, while probabilistic simulations are obtained by selecting one transition probabilistically using the transition rates (e.g. from `start` to `complete` with probability $\frac{1}{1+2}$).

Transitions can contain *guards*, like `allowed`, used to attempt `RobBank` in `start` only if all required sub-attacks succeeded (cf. Lines 6-7 in Code 7), or `!has`, used to forbid the transition to `tryGetToVault` if one already succeeded to `GetToVault` (cf. Line 8 in Code 7).

RisQFLan also supports *action constraints*, acting as guards on any transition executing a given action (while transition guards constrain single transitions). They are given as $\texttt{do}(act) \rightarrow b$, where *act* is an action and *b* is a Boolean expression over attributes. As defined in Code 8, any transition with action `choose` is disabled as soon as one succeeds to open or blow up the vault.

Transitions can also be labeled with *side-effects*: real-valued variables updated upon a transition's execution. Variables model context information, thus allowing for rich descriptions of system states, of attackers and of defenses, greatly facilitating the expression of constraints and the analysis phase. Code 9 defines variable `AttackAttempts` (`AA` in Fig. 4), which stores the number of attack attempts, updated each time a `succ` or `fail` action occurs as attempt to rob the bank.

In addition to constraints imposed by attack diagrams, transition guards and action constraints, attack behavior may be constrained by quantitative constraints in the form of Boolean expressions involving (arithmetic expressions or inequalities over) reals, attributes and variables. In Code 10, we constrain to 100 the maximum accumulated cost of an attack, of particular interest since attack behavior may model failed attacks.

Attack behavior is completed with an initial setup specifying the attacker and any initially accomplished attack(s). The latter enrich expressiveness, since one can assign an initial advantage to attackers: an attack-defense diagram models all possible attacks, but some attackers (e.g. insiders) may already have access to critical components.

This is convenient as a diagram's sub-trees may be ignored without their explicit removal. Due to Code 11, the attacker `Thief` already has one code.

Note that RisQFLan provides a programming-like environment that may be attractive to software developers, but it integrates at the same time a graphical component shown in Fig. 3, which may make it more attractive for security experts. The DSL moreover has a formal semantics, defined next.

5. RisQFLan operational semantics

5.1. RisQFLan *models and configurations*

In this section, we provide a formal definition of the ingredients composing RisQFLan models. In order to improve readability, we provide references to the corresponding code blocks from Section 4 when relevant, which show ho the components of the model are actually specified in our DSL.

A RisQFLan model $\mathcal{S}$ is defined as a septuple $\mathcal{S} = \langle \mathcal{N}, \mathcal{D}, \mathcal{V}, \mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{P} \rangle$, where

- $\mathcal{N} = \mathcal{N}_a \uplus \mathcal{N}_d \uplus \mathcal{N}_c$ is a set of nodes divided into attack nodes $\mathcal{N}_a$, defense nodes $\mathcal{N}_d$ and countermeasure nodes $\mathcal{N}_c$ (Code 1);

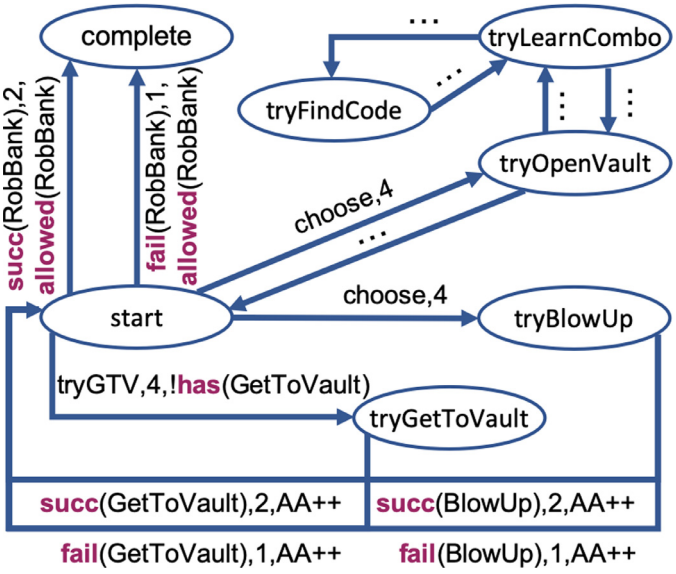


Fig. 4 – Attack behavior.

```
begin attacker behavior
 begin attack
  attacker = Thief
  states = start, tryOpenVault, tryLearnCombo, tryFindCombo, tryGetToVault, tryBlowUp, complete
  transitions =
   start -(succ(RobBank), 2, allowed(RobBank)) -> complete, //If I open or blow up the vault, then I can rob the bank
   start -(fail(RobBank), 1, allowed(RobBank)) -> complete,
   start -(tryGTV, 4, !has(GetToVault)) -> tryGetToVault, //Whatever strategy was used, I must get to the vault
    tryGetToVault -(succ(GetToVault), 2, {AttackAttempts=AttackAttempts+1}) -> start,
    tryGetToVault -(fail(GetToVault), 1, {AttackAttempts=AttackAttempts+1}) -> start,
    start -(choose, 4) -> tryOpenVault, //This is the strategy where I open the vault
    tryOpenVault -(succ(OpenVault), 2, {AttackAttempts=AttackAttempts+1}, has(LearnCombo) and has(GetToVault)) -> start,
    tryOpenVault -(fail(OpenVault), 2, {AttackAttempts=AttackAttempts+1}, has(LearnCombo) and has(GetToVault)) -> start,
    tryOpenVault -(try, 2, has(LearnCombo) and !has(GetToVault)) -> start, //I know the combo but did not get to the vault
    tryOpenVault -(try, 5, !has(LearnCombo)) -> tryLearnCombo,
     ... //Similar for tryLearnCombo and then tryFindCode
   start -(choose, 4) -> tryBlowUp, //This is the strategy where I blow up the vault
   tryBlowUp -(succ(BlowUp), 2, {AttackAttempts=AttackAttempts+1}) -> start,
   tryBlowUp -(fail(BlowUp), 1, {AttackAttempts=AttackAttempts+1}) -> start
 end attack
end attacker behavior
```

Code 7 – Attack behavior.

```
begin action constraints
 do(choose) -> !(has(OpenVault) or has(BlowUp))
end action constraints
```

Code 8 – Action constraints.

```
begin variables
 AttackAttempts = 0
end variables
```

Code 9 – Variables.

```
begin quantitative constraints
 {value(Cost) <= 100}
end quantitative constraints
```

Code 10 – Quantitative constraints.

- $\mathcal{D}$ is a set of attacker actions. The set $\mathcal{D}$ contains all actions succ(n_a), fail(n_a), and remove(n_a), where $n_a \in \mathcal{N}_a$, and additionally user-defined actions (Code 6);
- $\mathcal{V}$ is a set of variables (Code 9);
- $\mathcal{A}$ is a set of attackers names (Code 7);
- $\mathcal{B}$ is a set of attacker behaviors (Code 7);
- $\mathcal{C}$ is a set of constraints on the (presence/absence of) nodes, their attributes, and on (user-defined) variables. Such constraints are formed by the hierarchical constraints (built with -OR->, -AND->, -OAND->, and -Kn->, Code 2), action constraints (of the form do(act) $\rightarrow b$, where $act \in \mathcal{D}$ and b is a Boolean expression over attributes, Code 8) and quantitative constraints (Boolean expressions enriched with

```
begin init
 Thief = {FindCode1}
end init
```

Code 11 – Initial setup.

special attributes like allowed(n_a) and has(n_a), with $n_a \in \mathcal{N}_a$, Code 10);
- $\mathcal{P} : \mathcal{N} \to \mathbb{R}$ is a set of node properties, distinguishing attributes decorating nodes (Code 3), attack detection rates decorating attack nodes (functions $\mathcal{N}_a \to [0, 1]$, Code 4) and defense effectiveness decorating defense nodes (functions $(\mathcal{N}_d \cup \mathcal{N}_c) \times \mathcal{N}_a \times \mathcal{A} \to [0, 1]$, Code 5).

We introduce the notion of configuration for a RisQFLan model and equip it with an operational semantics based on rated transition systems. A *configuration* of a RisQFLan model $\mathcal{S}$ is a tuple $\langle C, s \rangle$, where s is a state of attack behavior of $\mathcal{S}$ and C is a set of constraints consisting of:

- all constraints of the model $\mathcal{S}$;
- a predicate $has(n)$ for each currently active node $n \in \mathcal{N}$;
- constraints of form $t(n_a) < t(n'_a)$, for $n_a, n'_a \in \mathcal{N}_a$, denoting that n_a was activated before n'_a, necessary to support OAND refinements;
- an assignment of form $att(n) = x$ for each attribute att and node $n \in \mathcal{N}$ to denote the value of the attribute for the node n, with $x \in \mathbb{R}$;
- assignments of form $value_a(att) = x$ and $value_{def}(att) = x$ for each attribute att to denote its cumulative attacker and defender value, with $x \in \mathbb{R}$;
- an assignment of form $v = x$ for each variable $v \in \mathcal{V}$, with $x \in \mathbb{R}$;
- an assignment of form $dr(n_a) = x$ for each attack node n_a to denote detection rate of n_a, with $x \in \mathbb{R}$;
- a set $detect(n_c) \subseteq \mathcal{N}_a$ for each countermeasure node n_c to denote the attack nodes that can be detected by n_c.

Let $\mathcal{M}$ denote the set of all configurations for a RisQFLan model $\mathcal{S}$. We restrict to configurations $\langle C, s \rangle$ such that C is consistent, i.e. all constraints are satisfied, denoted by $con(C)$. As we will see in Proposition 1, this property is preserved by the operational semantics: no inconsistent configuration can be reached from a consistent one. We will use $\oplus$ to denote union of constraint sets, $\ominus$ for subtraction and $\vdash$ for entailment.

5.2. RisQFLan *dynamics*

The dynamics of RisQFLan configurations is given as rated transition systems that specify how a configuration $\langle C, s \rangle$ can evolve into a configuration $\langle C', s' \rangle$ with a certain rate r. Such evolution occurs as the consequence of the attacker trying to perform an action and the defender eventually reacting to mitigate it. We denote such an evolution with a transition of the form $\langle C, s \rangle \xrightarrow{r} \langle C', s' \rangle$. In general, the dynamics is defined by a multi-relation $\to \subseteq \mathbb{N}^{\mathcal{M} \times \mathbb{R}^+ \times \mathcal{M}}$ induced by the rules of Fig. 5. We use a multi-relation since we have to account for multiple copies of the same transition with the same rate, as the probabilistic interpretation requires to 'sum' such rates. Indeed, as we shall see, the dynamics of a configuration is ultimately defined as a discrete-time Markov chain, upon which the analysis of RisQFLan is based.

The rules share some premises and effects. First, all rules need an attack behavior transition of the form $s \xrightarrow{\alpha,r,u,g} s'$, with current state of the attacker s, action α, rate r and memory update u, such that the executability conditions of the transition guard g hold. This is imposed by $exe(C, \alpha, g)$, defined as:

$$exe(C, \alpha, g) = \begin{cases} false & \text{if } C \nvdash g \\ false & \text{if } C = C' \oplus (do(\alpha) \to C'') \text{ and } C' \nvdash C'' \\ false & \text{if } \alpha = add(n_a) \text{ and } has(n_a) \in C, \text{ with } n_a \in \mathcal{N}_a \\ false & \text{if } \alpha = fail(n_a) \text{ and } has(n_a) \in C, \text{ with } n_a \in \mathcal{N}_a \\ false & \text{if } \alpha = remove(n_a) \text{ and } has(n_a) \notin C, \text{with } n_a \in \mathcal{N}_a \\ true & \text{otherwise} \end{cases}$$

Second, all rules require the resulting store to be consistent. Further conditions vary from rule to rule, as we will explain. By applying a rule on a configuration $\langle C, s \rangle$ due to a local transition $s \xrightarrow{\alpha,r,u,g} s'$, we obtain a configuration $\langle C', s' \rangle$, where C' is obtained by applying the effects u on the variables in C (denoted $u(C, \alpha)$) and by possibly (de)activating nodes. In addition, $u(C, \alpha)$ updates cumulative attack and defense attribute values, as explained in Section 4. The semantics of $u(C, \alpha)$ is as expected, and not presented for conciseness.

We now describe each rule in detail.

Rule ACT executes user-defined actions: node activations are not altered by this rule so its effects are limited to variables.

Rule ADD is triggered by actions $add(n_a)$: with probability $dr(n_a)$, it may activate the set $c(n_a, C)$ of countermeasure nodes able to detect n_a that are not already active or inhibited by an active attack node n'_a. The set $c(n_a, C)$ is defined as follows, where -RC-> denotes a role-changing relation:

$$\{\, n_c \mid n_a \in detect(n_c) \wedge has(n_c) \notin C \wedge \neg \exists n'_a . \\ (has(n'_a) \in C \wedge (n_c \text{ -RC-> } n'_a) \in C) \,\}$$

Upon the execution of the rule, the constraint store is updated with the new attack node n_a, which is recorded to be the last active attack node of the store ($t(n) < t(n_a)$). Furthermore, the constraint store is also updated with each countermeasure node in $c(n_a, C)$. Another effect is that all defenses that have n_a as opponent are deactivated. The rate of the obtained transition is not necessarily the original rate r of the attack behavior transition. In fact, r might be scaled by the defense effectiveness of the active defenses against n_a in the newly obtained store, denoted by $de(C', n_a, s) \in [0, 1]$. We distinguish three cases: (i) if n_a has no role-changing relation, it is 1; (ii) if n_a has a defense opponent n_d, it is the effectiveness of n_d for n_a and the current attacker; (iii) if n_a has a countermeasure opponent n_c, it is the product of the effectiveness of n_c and that of any defense node that refines it, for n_a and the attacker $\mathcal{A}$. Finally, we have to multiply the rate by the probability of activating the countermeasures, $dr(n_a)$. Rule ADDNOC is similar, but it covers the case in which the countermeasures $c(n_a, C)$ do not get activated.

Rules FAIL and FAILNOC are similar to ADD and ADDNOC, but the attack node is not activated because they regard the fail action which model failed attack attempts. Finally, rule REM models the deactivation of an attack node.

$$[\textsc{Act}]\ \frac{s \xrightarrow{act,r,u,g} s' \qquad exe(C, act, g) \qquad C' = u(C, act) \qquad con(C')}{\langle C, s\rangle \xrightarrow{r} \langle C', s'\rangle}$$

$$[\textsc{Add}]\ \frac{\begin{array}{c} s \xrightarrow{add(n_a),r,u,g} s' \qquad exe(C, add(n_a), g) \\ C' = u(C, add(n_a)) \oplus has(n_a) \oplus \bigoplus_{n_c \in c(n_a,C)} has(n_c) \oplus \bigoplus_{\{\, n \in \mathcal{N}_a \mid has(n) \in C \,\}} t(n) < t(n_a) \ominus \left(\bigoplus_{\{\, n \mid n \ \text{-RC->} \ n_a \,\}} has(n) \right) \qquad con(C') \end{array}}{\langle C, s\rangle \xrightarrow{r \cdot de(C', n_a, s) \cdot dr(n_a)} \langle C', s'\rangle}$$

$$[\textsc{AddNoC}]\ \frac{\begin{array}{c} s \xrightarrow{add(n_a),r,u,g} s' \qquad exe(C, add(n_a), g) \\ C' = u(C, add(n_a)) \oplus has(n_a) \oplus \bigoplus_{\{\, n \in \mathcal{N}_a \mid has(n) \in C \,\}} t(n) < t(n_a) \ominus \left(\bigoplus_{\{\, n \mid n \ \text{-RC->} \ n_a \,\}} has(n) \right) \qquad con(C') \end{array}}{\langle C, s\rangle \xrightarrow{r \cdot de(C', n_a, s) \cdot (1 - dr(n_a))} \langle C', s'\rangle}$$

$$[\textsc{Fail}]\ \frac{s \xrightarrow{fail(n_a),r,u,g} s' \qquad exe(C, fail(n_a), g) \qquad C' = u(C, fail(n_a)) \oplus \bigoplus_{n_c \in c(n_a,C)} has(n_c) \qquad con(C')}{\langle C, s\rangle \xrightarrow{r \cdot de(C', n_a, s) \cdot dr(n_a)} \langle C', s'\rangle}$$

$$[\textsc{FailNoC}]\ \frac{s \xrightarrow{fail(n_a),r,u,g} s' \qquad exe(C, fail(n_a), g) \qquad C' = u(C, fail(n_a)) \qquad con(C')}{\langle C, s\rangle \xrightarrow{r \cdot de(C', n_a, s) \cdot (1 - dr(n_a))} \langle C', s'\rangle}$$

$$[\textsc{Rem}]\ \frac{\begin{array}{c} s \xrightarrow{remove(n_a),r,u,g} s' \qquad exe(C, remove(n_a), g) \\ C' = u(C, remove(n_a)) \ominus \left(has(n_a) \oplus \bigoplus_{\{\, n \mid has(n) \in C \,\}} t(n) < t(n_a) \right) \qquad con(C') \end{array}}{\langle C, s\rangle \xrightarrow{r} \langle C', s'\rangle}$$

Fig. 5 – The operational semantics.

It is easy to see that the semantic rules ensure consistency is preserved along sequences of configurations, since consistency is a premise in every rule, and hence in every transition.

Proposition 1. *Let $\mathcal{S}$ be a RisQFLan model and $\langle C, s\rangle$ be a configuration such that C is consistent. Then for any configuration $\langle C', s'\rangle$ such that $\langle C, s\rangle \rightarrow^* \langle C', s'\rangle$ it holds that C′ is consistent.*

The probabilistic interpretation of rated transition systems yields DMTCs. A DTMC is a tuple $\langle \Gamma, \Pi\rangle$ where Γ is a set of states and $\Pi : \Gamma \rightarrow [0, 1]$ is a probability transition function, i.e. such that for all $s \in \Gamma$, $\sum_{s' \in \Gamma} \Pi(s, s') = 1$. The DTMC semantics of a rated transition system is obtained by normalising the rates into [0.1] such that in each state/configuration, the sum of the rates of its outgoing transitions equals one. So, for a rated transition system $\rightarrow$ on a set of configurations $\mathcal{M}$ we obtain the DTMC $\langle \mathcal{M}, \Pi\rangle$ where, for each pair of states $s, s' \in \mathcal{M}$, the probability transition function Π is defined by

$$\Pi(s, s') = \begin{cases} \frac{\sum_{(s,r,s') \in \rightarrow} r}{\mathbf{out}(s)} & \text{if } \mathbf{out}(s) > 0 \\ 1 & \text{if } \mathbf{out}(s) = 0 \text{ and } s = s' \\ 0 & \text{otherwise} \end{cases}$$

where **out** denotes the outdegree of a configuration. Note that self-loops with probability 1 are added to configurations without outgoing transitions. The DTMC semantics of RisQFLan models is used in our analyses, described in the next section.

6. RisQFLan supported quantitative analyses

RisQFLan supports the quantitative analysis of probabilistic attack scenarios by means of statistical model checking (SMC) Agha and Palmskog (2018); Legay et al. (2019) as well as probabilistic model checking (PMC) Baier and Katoen (2008), thus providing additional analysis capabilities to what other risk analysis tools typically offer.

SMC is concerned with running a sufficient number of (probabilistic) simulations of a system model to obtain statistical evidence (with a predefined level of statistical confidence) for the quantitative properties to be checked. Compared to obtaining exact results (with 100% confidence) with exact analysis techniques like (probabilistic) model checking, SMC offers unique advantages over exhaustive (probabilistic) model checking. Most importantly, SMC scales better. First, there is no need to generate entire state spaces, thus avoiding the combinatorial state-space explosion problem typical of model checking Clarke et al. (2018). Second, the set of simulations to be carried out can be trivially distributed and run in parallel, thus scaling better with hardware resources. Mul-

tiVeStA, indeed, can be run on multi-core machines, clusters or distributed computers with a nearly linear speedup. Another advantage concerns its uptake in industry. Compared to model checking, SMC is simple to implement, understand and use, and it requires no specific modeling effort other than a system model that can be simulated and checked against quantitative properties. In fact, SMC is more and more being applied in industry Arnold et al. (2017); Bao et al. (2019); Basile et al. (2018, 2019a, 2020, 2017, 2019b); ter Beek et al., 2016; Cappart et al. (2017); Ferrari et al., 2020; Filipovikj et al. (2016); Garavel et al. (2020); Gilmore et al. (2014); Puch et al. (2018).

In RisQFLan, the SMC analysis is obtained thanks to the integration of the internal DTMC simulator with MultiVeStA Gilmore et al. (2017); Sebastio and Vandin (2013); Vandin et al. (2021), a framework for enriching simulators with SMC capabilities, while the PMC analysis is obtained thanks to RisQFLan's DTMC exporting capabilities in a format supported by PRISM Kwiatkowska et al. (2011) and STORM Dehnert et al. (2017). SMC is necessary because the RisQFLan DSL has high expressivity, allowing for potentially unbounded variables and high variability in the models, thus often giving rise to large or infinite state spaces. PMC can instead be used for models with finite state spaces for exact analyses.

Next we showcase two SMC analysis capabilities of RisQFLan on our running example. PMC cannot be used in this case as the model has an infinite state space. We will showcase PMC analyses using PRISM in Section 7.

6.1. Analysis while varying simulation steps

We start by studying the probabilities of activating attacks and countermeasures while varying the simulation step.

Expressing Properties This is expressed in Code 12. The pattern `from-to-by` specifies that we are interested in the first 100 steps. We list 8 properties of interest (one per attack node, considering `FindCode1` active, plus the countermeasure `LockDown`). Properties can be an arithmetic expression of nodes (evaluating to 1 or 0 if the node is active or not, resp.), variables, or attributes. The 8 properties are considered in each of the first 100 steps, totaling 800 actual properties.

Statistical Reliability and Confidence Intervals Each of these 800 actual properties p_i denotes a random variable X_i which gets a real value assigned in each simulation (e.g. property '`FindCode2` *at step 10*' evaluates to 1 in all simulations where the attacker is able to successfully perform such attack in the first 10 steps, and 0 otherwise). As discussed in Sebastio and Vandin (2013); Vandin et al. (2021), MultiVeStA estimates the expected value $E[X_i]$ of each of the 800 properties (reusing the same simulations) as the mean $\bar{x}_i$ of n independent simulations, with n large enough to guarantee an (α, δ) *confidence interval* (CI) (cf. (Law, 2015, Chapter 9)). In other words, $E[x_i]$ belongs to $[\bar{x}_i - \delta/2, \bar{x}_i + \delta/2]$ with statistical confidence $(1 - \alpha) \cdot 100\%$.[3] The required CI is given by `alpha` and `default delta` (but property-specific ones could be used instead). Finally, `parallelism` states how many local processes should be launched to distribute the simulations. Overall the analysis required 400 simulations, performed in 16 seconds on a standard laptop machine.

Interpretation of the Results Fig. 6 shows the results. In all plots in this paper, we only provide the estimated expected values without showing the computed CI, because they are guaranteed to have width at most δ and the required statistical confidence. Recall (Fig. 1, Code 2) that `RobBank` requires `OpenVault` or `BlowUp`. The probability to activate `RobBank` starts growing after step 4, stabilizing at 0.17, while those of `OpenVault` and `BlowUp` reach 0.15 and 0.11, resp. We know from Code 8 that they cannot both be activated, so one should be able to activate `RobBank` with probability almost 0.26. Instead, the actual probability is scaled down by $\frac{2}{3}$ due to the probabilistic choice from `start` to `complete` in Fig. 4: `RobBank` can either succeed or fail.

Note that `LockDown` has a high probability to be activated, reaching about 0.85 after 60 steps. This is coherent with Code 5, stating that any `BlowUp` attempt is detected. One might expect the probability to activate `BlowUp` to be higher than that of `LockDown`, as the former triggers the latter. However, this is not true. This is explained by the fact that both succeeded and failed `BlowUp` attempts are detected (cf. success `succ(BlowUp)` and failure `fail(BlowUp)` actions in Fig. 4). Interestingly, if we added `LaserCutter` to the initial configuration, then the probability of activating `LockDown` would remain 0, as it is inhibited by `LaserCutter`.

6.2. Analysis at the verification of a condition

We can also compute properties evaluated as soon as a given condition verifies rather than while varying the simulation step. Here we compute the probability for each attack node to be the first attempted and succeeded, as well as the average number of steps needed to perform the first attempt.

Expressing Properties Code 13 expresses these 9 properties (one probability per attack node plus the average number of steps). Note that the `from-to-by` pattern is replaced by `when` to specify that the properties should be evaluated in the first state satisfying `AttackAttempts == 1`. Moreover, the list of properties of interest now includes `steps`, evaluated as the average number of steps computed to reach the first state satisfying the required condition.

Statistical Reliability and Confidence Intervals As for Code 12, we set $\delta = 0.1$ and $\alpha = 0.1$. However, for `steps` we specify a specific delta (`[delta = 0.5]`), because we deem $\delta = 0.1$ to be unnecessarily small for this property. Overall, the analysis required 400 simulations, performed in a few seconds on a standard laptop machine.

Interpretation of the Results The analysis results are provided in Table 1. The first four attack nodes have probability 0 of being the first attempted and succeeded attack. This is coherent with the diagram in Fig. 1, as such attacks are not leaves of the diagram and thus require other attacks to succeed first. Consistently with Fig. 6, `GetToVault` has higher probability than `FindCode2` and `FindCode3`. Intuitively, this depends

[3] We chose $\alpha = 0.1$ because it is common in statistical estimations (cf., e.g., Law (2015)). Indeed, modelers are often satisfied with 90% confidence intervals (alternatively, 95% or 99% are also used commonly). Instead, we chose $\delta = 0.1$ because all properties are estimated as a value in the interval [0,1], therefore we deem a *precision* of approximately $\frac{\delta}{2} = 0.05$ to be reasonable.

```
begin analysis
 query = eval from 1 to 100 by 1 :
  {RobBank, OpenVault, BlowUp, LearnCombo, GetToVault,
   FindCode2, FindCode3, LockDown}
 default delta = 0.1 alpha = 0.1 parallelism = 1
end analysis
```

Code 12 – Analysis of the scenario.

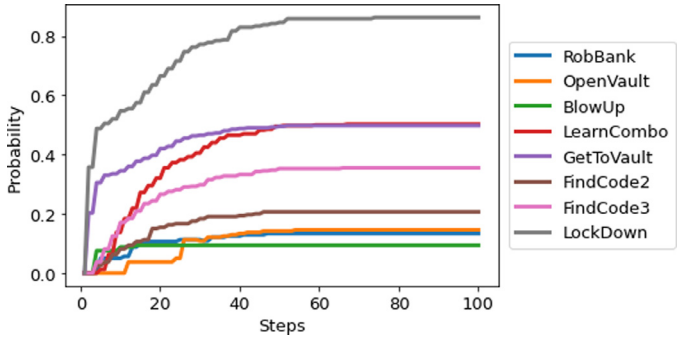


Fig. 6 – Analysis result of the properties in Code 12.

```
begin analysis
 query = eval when {AttackAttempts == 1} :
  {RobBank, OpenVault, BlowUp, LearnCombo, GetToVault,
   FindCode2, FindCode3, LockDown, steps[delta = 0.5]}
 default delta = 0.1 alpha = 0.1 parallelism = 1
end analysis
```

Code 13 – Analysis of the scenario.

Table 1 – Analysis result of the properties in Code 13.

Rob Bank	Open Vault	Blow Up	Learn Combo	GetTo Vault	Find Code2	Find Code3	Lock Down	steps
0	0	0	0	0.27	0	0.01	0.32	2.51

on the way the attacker's behavior is defined. As sketched in Fig. 4 and specified in Code 7, starting from state `start` we only have to perform one step to try `GetToVault` attacks, while to try finding a code requires traversing two more states, in each of which other competing actions are enabled. In turn, `FindCode2` has lower probability (belonging to the interval [0,0.05] due to the imposed CI) than `FindCode3` due to the defense `Memo`. Interestingly, we note a probability of 0.32 of activating the countermeasure `LockDown`. This means that failed `BlowUp` attempts were detected. Table 1 also shows that, on average, 2.51 steps are needed to perform one attack attempt. Indeed, in state `start` no attack attempt is allowed, so two steps are needed to attempt `GetToVault` or `BlowUp` attacks, while three are needed for `FindCode` attempts.

6.3. *Simulating and exporting*

RisQFLan models can be debugged by performing probabilistic simulations. Code 14 prints (in file `sim.log`) all chosen states and other useful information of the simulation suitable for debugging. RisQFLan's DTMC `Exporter` can generate entire DTMCs and export them in the input format accepted by the probabilistic model checkers PRISM or STORM.

Code 15 shows how to export the DTMC of our running example for external analysis, labeling with "hasRB" all states in which a RobBank attack succeeded.

```
begin simulate
 seed = 1 steps = 1
 file = "sim.log"
end simulate
```

Code 14 – Log generation.

```
begin exportDTMC
 file = "RobBank.pm"
 label with "hasRB" when has(RobBank)
end exportDTMC
```

Code 15 – DTMC export.

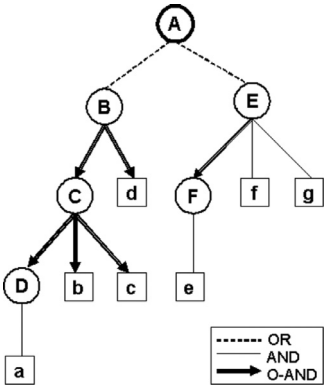


Fig. 7 – Enhanced attack tree for "Bypassing 802.1x" Çamtepe and Yener (2007).

```
1  begin attack diagram
2   A -OR-> {B, E}
3   B -OAND-> [C, d]
4   C -OAND-> [D, b, c]
5   D -AND-> {a}
6   E -OAND-> [F, fg]
7   fg -AND-> {f, g}
8   F -> {e}
9  end attack diagram
```

Code 16 – Enhanced attack tree of Fig. 7 specified in RisQFLan.

7. Validation of RisQFLan

A variety of extensions of attack-tree models exist and no single approach has so far emerged as the ultimate solution Hong et al. (2017); Ingoldsby (2013); Kordy et al., 2014; Wideł et al. (2019). This section shows the flexibility of RisQFLan by illustrating how features from three seminal and influential kinds of attack trees can be specified in RisQFLan, and how the latter's analysis capabilities can be used to complement and enrich the analyses provided by existing tools. To simplify the presentation, all MultiVeStA analyses in this section used the value 0.1 for both α and δ, even though larger δ could have been used in Section 7.2. The tool, its source code, and the models and analyses are available at https://github.com/risqflan/RisQFLan/wiki.

7.1. *Case study 1: Ordered attacks*

This section shows that the RisQFLanDSL can be used to model features from *enhanced attack trees*, an extension of basic attack trees proposed in Çamtepe and Yener (2007), and that RisQFLan hence complements the analysis capabilities of Çamtepe and Yener (2007) with (exact) PMC and SMC on specific attacker profiles. We do so by illustrating how *ordered attacks*, a key differentiating feature of such *enhanced attack trees*, can be specified in RisQFLan.

7.1.1. *Ordered attacks to "Bypassing 802.1x"*

As illustrative example, we use one case study from Çamtepe and Yener (2007), namely an enhanced attack tree modeling complex (ordered) attacks on wireless LANs using protocol IEEE 802.11. Fig. 7, reproduced from Çamtepe and Yener (2007), illustrates the enhanced attack tree. The main idea is that the authentication mechanism of the protocol can be compromised through hijacking authenticated sessions (B) or man-in-the-middle attacks (E). The sub-trees of B and E further refine both attacks into specific sub-goals.

7.1.2. *Specifying ordered attacks in RisQFLan*

Code 16 shows a model of the enhanced attack tree of Fig. 7 in RisQFLan. It is worth observing how the ordering relation is modeled. The original model in Fig. 7 prescribes that: (i) to achieve attack B, sub-goal C must be achieved before d (cf. Line 3 in Code 16); (ii) to achieve attack C, sub-goal D must be achieved before b, which itself must be achieved before c (cf. Line 14 in Code 16); and (iii) to achieve attack E, sub-goal F must be achieved before f and g (cf. Lines 6-7 in Code 16). Note that in the RisQFLan specification, auxiliary node fg is used to group the unordered conjunction of f and g.

7.1.3. *Complementing the analysis of Çamtepe and Yener (2007) with RisQFLan*

The main analysis feature of the approach in Çamtepe and Yener (2007) consists of inspecting activity logs to recognize potential attacks as per the specified enhanced attack trees. With RisQFLan this can be augmented with exact or statistical probabilistic verification on the average behavior of specific attacker profiles. To illustrate this we modeled four attacker profiles:

1. `Best`: an attacker that knows one of the optimal order of attacks to perform to achieve the main attack goal;
2. `AverageA`: an attacker randomly trying attacks until achieving the main attack goal or a wrong order led to failure;
3. `AverageB`: like `AverageA` but can undo attacks (backtrack);
4. `Worst`: like `AverageA` but chooses attacks with a probability inversely proportional to the order used by `Best`.

Fig. 8 presents the results of SMC analysis of each such attacker profile, showing that they converge to different attack success probabilities. We have also exported the corresponding DTMCs and analyzed them with PMC using PRISM. PRISM computed the same results for all attackers except for `AverageB`, whose DTMC is too large (due to backtracking in the attacker's strategy) for PRISM or STORM to be able to handle it. Attackers `Best` and `AverageB` obviously achieve the attack with probability 1, although the latter needs more time. The `AverageA` attacker is next, achieving a success probability slightly above 0.6, while the `Worst` attacker achieves an attack with probability about 0.4.

7.2. *Case study 2: Noticeability*

This section shows that the RisQFLan DSL can be used to model features from *capabilities-based attack trees* Ingoldsby (2013), an extension of basic attack trees offered in the commercial attack tree-based risk assessment tool SecurITree Amenaza Technologies Limited (2006). This

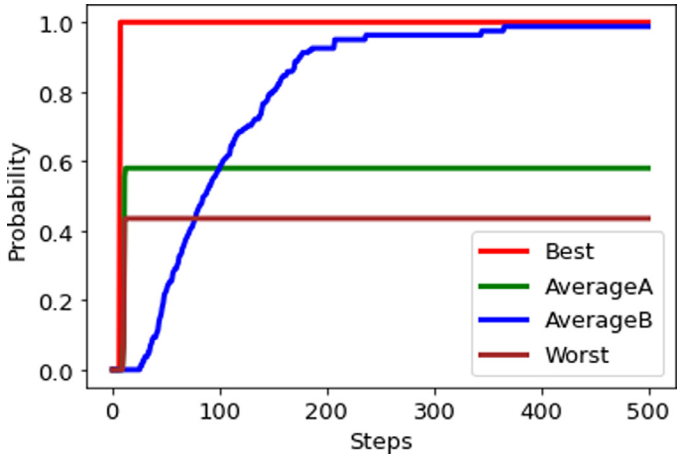


Fig. 8 – Statistical analysis on "Bypassing 802.1x".

```
begin attributes
 Noticeability = {WalkUpToHouse = 0.01,
  EavesdropAndReplayOpenerCode = 0.05, PickLock = 0.02}
end attributes
```

Code 17 – Noticeability of "Cat Burglar" specified in RisQFLan.

means RisQFLan complements the models of SecurITree with explicit dynamic attack behavior [4] and its analysis capabilities with analysis of attacker profiles. We illustrate how the notion of *noticeability*, one of the capability features of *capabilities-based attack trees*, can be specified in RisQFLan.

7.2.1. *Noticeability capabilities of BurgleHouse*

As illustrative example, we use two attack scenarios studied in Amenaza Technologies Limited (2006), namely the Cat Burglar and Juvenile Delinquent scenarios from the BurgleHouse case study. Fig. 9, reproduced from Amenaza Technologies Limited (2006), depicts two capabilities-based attack trees which can be easily encoded in the RisQFLan DSL using `OR` and `AND` refinements. The idea is that a house can be burglarized by entering the house by carrying out two sub-goals: `WalkUpToHouse` and `PenetrateHouse`. The latter is further refined into sub-goals. In the Cat Burglar scenario the house can only be penetrated via a `GarageAttack`, whereas in the Juvenile Delinquent scenario there are two further alternatives: opening the passage door by breaking it down or entering via the window by breaking the glass. We consider one of the three so-called behavioral indicators associated to attacker actions in Amenaza Technologies Limited (2006), namely *noticeability*. The values were kindly provided by Terry Ingoldsby of Amenaza Technologies Ltd. together with a license for SecurITree v5.0.

7.2.2. *Noticeability in RisQFLan*

Codes 17 and 18 show how the noticeability values of the Cat Burglar and Juvenile Delinquent scenarios, resp., are modeled as a `Noticeability` attribute in RisQFLan: walking up to the house is almost unnoticeable, while breaking a door or glass is more noticeable.

7.2.3. *Complementing the analysis of Amenaza Technologies Limited (2006) with RisQFLan*

One of the analysis features of SecurITree consists of the possibility to identify attack scenarios according to one or more behavioral indicators. For instance, by pruning the complete attack tree of the BurgleHouse case study with 29 nodes, SecurITree identified the above scenarios as corresponding to the specific capabilities of threat agents of the Cat Burglar and Juvenile Delinquent type (which avoid attacks that involve a risk of getting caught greater than 10% and 30%, resp., expressed through the noticeability criterion). Similarly, RisQFLan can limit its analysis to such type of scenarios by imposing quantitative constraints (cf. Code 10 in Section 4). However, RisQFLan can also augment such analyses with quantitative verification on the average behavior of specific attacker profiles as well as with estimation of the average noticeability of specific (successful) attacks. To illustrate this, we modeled four attacker profiles:

1. `Best`: an attacker that knows an optimal, most unnoticeable order of attacks to perform to achieve the main attack goal;
2. `AverageA`: an attacker that randomly tries attacks until the main attack goal is achieved;
3. `AverageB`: like `AverageA` but can undo attacks (backtrack);
4. `Worst`: like `AverageA` but chooses attacks with a probability inversely proportional to `Best`.

We analyzed these 4 attackers in the two scenarios using the SMC analysis capabilities of RisQFLan. Fig. 10 and Fig. 11

[4] Amenaza has similar plans for SecurITree v5.1 (T. Ingoldsby, personal communication, April 1, 2020).

```
begin attributes
 Noticeability = {WalkUpToHouse = 0.01,
  BreakDownDoor = 0.3, BreakGlass = 0.3,
   StealOpenerFromCar = 0.2, BreakDownPassageDoor = 0.1}
end attributes
```

Code 18 – Noticeability of "Juvenile Delinquent" specified in RisQFLan.

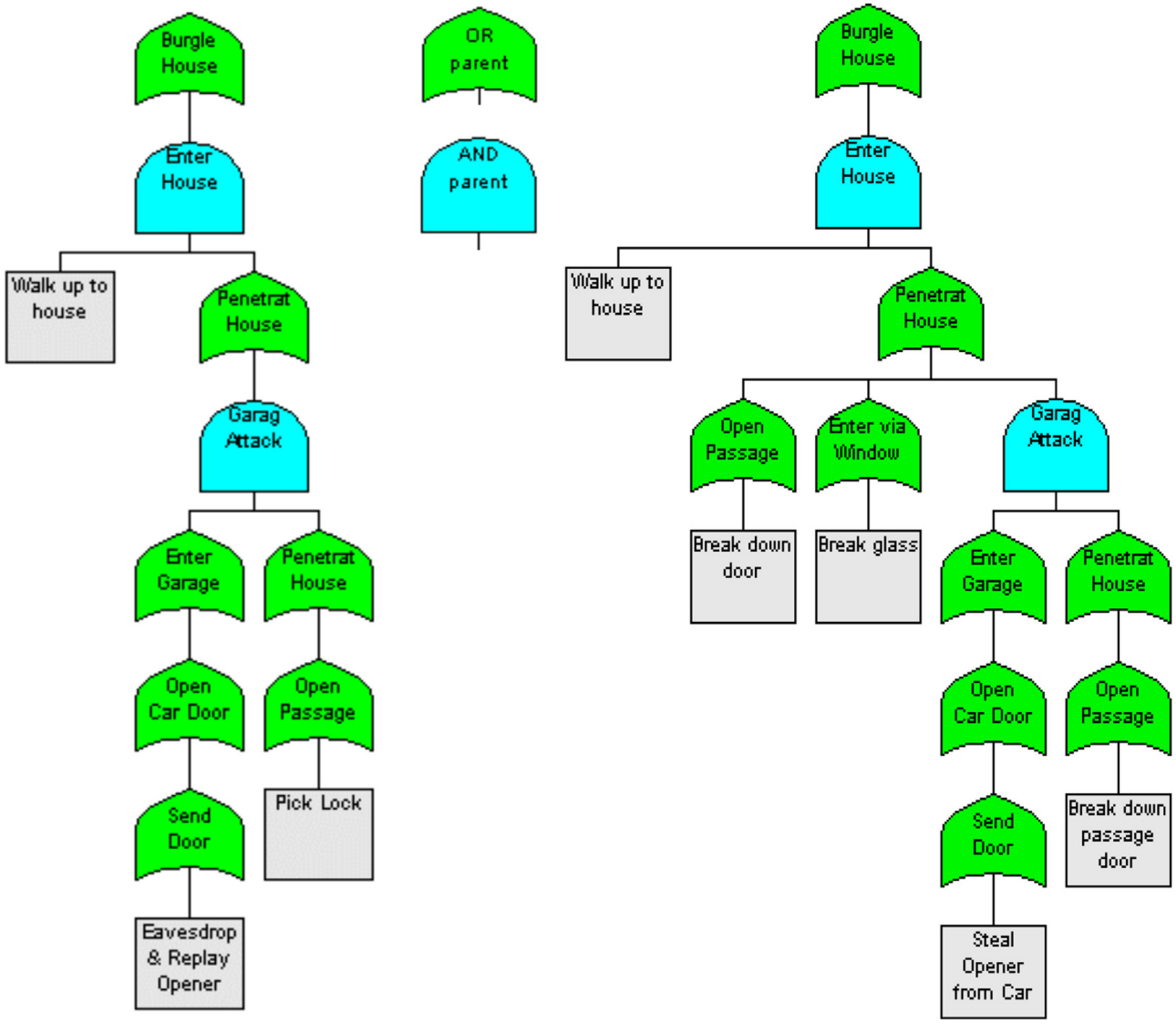


Fig. 9 – Capabilities-based attack trees: "Cat Burglar" (left) and "Juvenile Delinquent" (right) Amenaza Technologies Limited (2006).

show how the attacker profiles converge to different average noticeability values of the attacks [5] Note that, contrary to the case study presented in the previous section, none of the orders of attacks can result in failure. In fact, while not shown, in both scenarios all attackers succeed with probability 1, although in both cases attacker `AverageB` needs considerably more time. Moreover, in the Cat Burglar scenario, all successful attackers that cannot backtrack use the same set of actions. In fact, the average noticeability value of the `Best`, `AverageA`, and `Worst` attackers is 8, whereas the repeated attack attempts of the `AverageB` attacker guarantee that (s)he will be noticed.

However, in the Juvenile Delinquent scenario, even successful attackers may have made use of different sets of actions, due to the three different ways to penetrate the house (`PenetrateHouse -OR-> {OpenPassageDoor,`

[5] To make the differences visible, the noticeability values of the Cat Burglar and Juvenile Delinquent scenarios were multiplied by 10 and 100, resp.

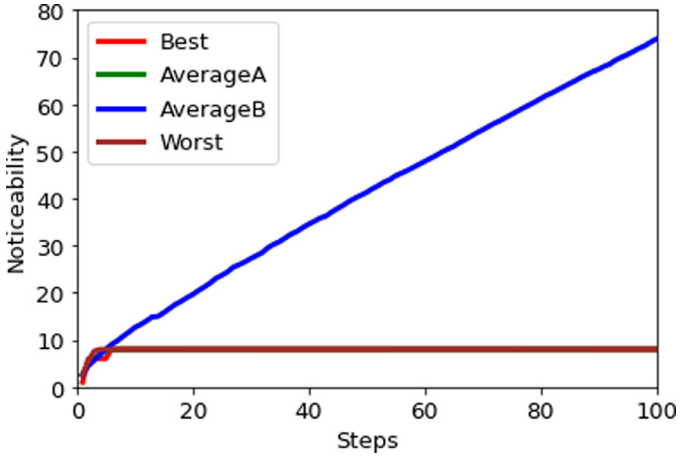


Fig. 10 – Statistical analysis on "Cat Burglar".

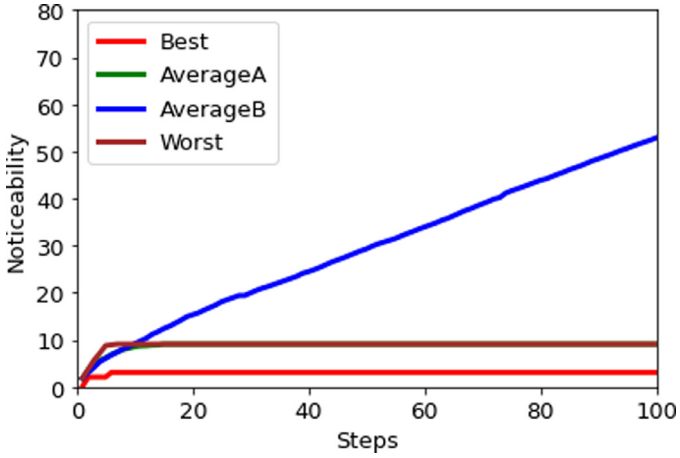


Fig. 11 – Statistical analysis on "Juvenile Delinquent".

`EnterViaWindow, GarageAttack`}). In fact, the average noticeability value of the `Best` attacker is just over 3, that of the `AverageA` and `Worst` attackers is just over 9, while also in this case the repeated attack attempts of the `AverageB` attacker guarantee that (s)he will be noticed.

RisQFLan thus allows to analyze the risk to get caught for different types of behavior of a concrete Cat Burglar or Juvenile Delinquent and to estimate who runs less risk. SecurITree considers such explicit dynamic attack behavior in a slightly different way. It offers advanced analysis functionalities to estimate the risk of scenarios by combining the impact of attacks and the so-called capabilistic attack propensity, which is expressed by considering feasibility (e.g. cost or resources) vs. benefits (rewards) and detriments incurred in attacks.

7.3. *Case study 3: Countermeasures*

As in the previous sections, we focus on an influential approach to attack trees, *attack countermeasure trees* Roy et al., 2012, which has inspired some of RisQFLan's modeling features. We show how RisQFLan DSL can specify the novel reactive defense mechanisms that were introduced in attack countermeasure trees, namely *detection events* that model defensive mechanisms to detect that an attack is being attempted and *measure events* that model defensive mechanisms to mitigate the effect of an attack.

7.3.1. *Countermeasures against "Resetting BGP"*

As illustrative example, we use a case study from Roy et al., 2012, namely an attack countermeasure tree modeling defensive mechanisms against resetting attacks on the so-called Border Gateway Protocol (BGP). Fig. 12, reproduced from Roy et al., 2012, depicts the attack countermeasure tree for this scenario. The idea is to model a known denial-of-service attack on the BGP: the attacker tries to reset a BGP session again and again to prevent communication. Such attacks consist of several steps, some of which can be detected and mitigated with well-known techniques (e.g. `TCP sequence number attacks` (`A12`) can be detected with `TCP sequence number checks` (`D12`), and a mitigation mechanism for such attacks is using `MD5 authentication` (`M12`)).

7.3.2. *Countermeasures in RisQFLan*

Code 19 shows how to model the attack countermeasure tree of Fig. 12 in RisQFLan.

In particular, we remark the following:

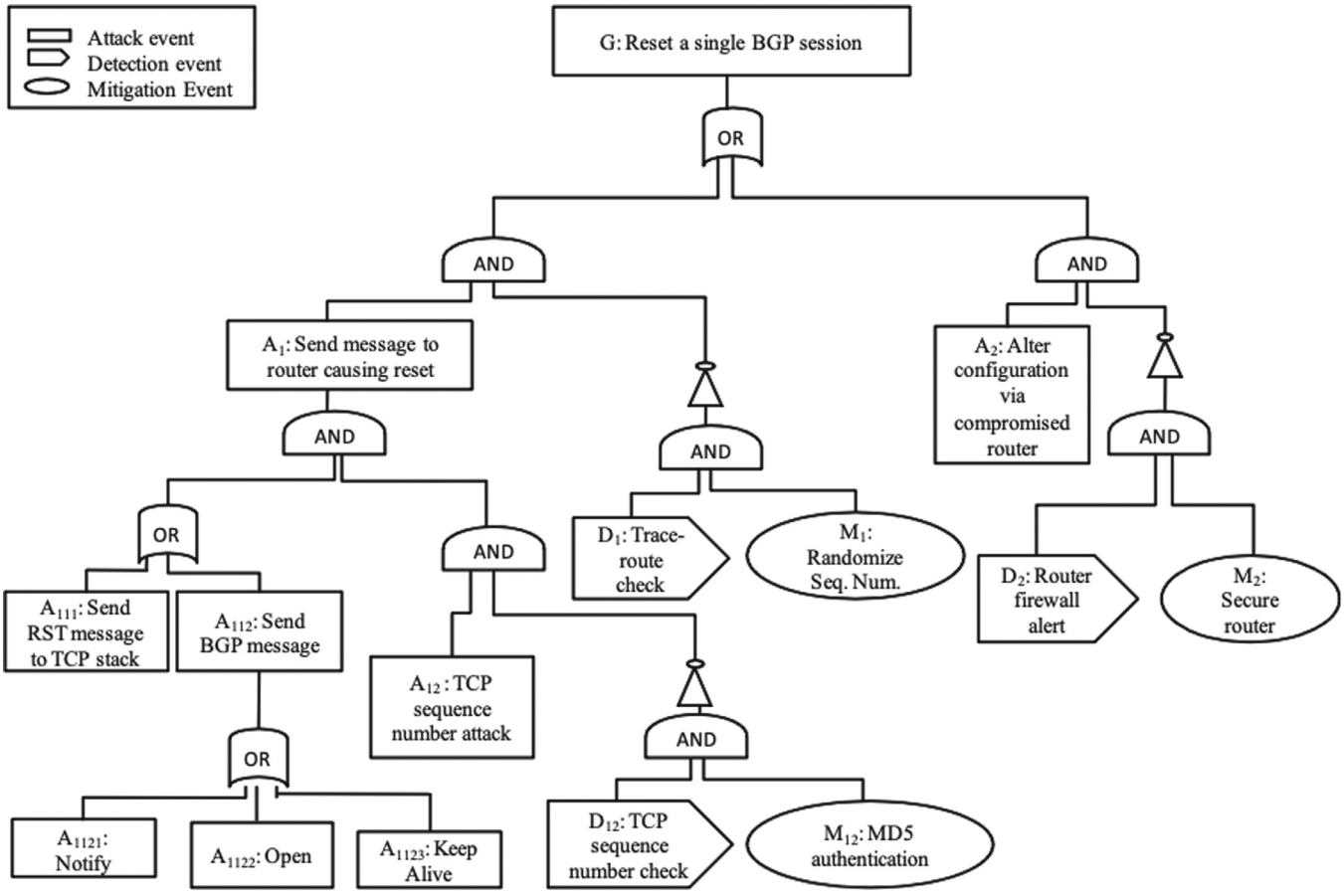


Fig. 12 – Countermeasure tree for "Resetting BGP" Roy et al., 2012.

```
begin attack nodes
 G A1 A111 A112 A1121 A1122 A1123 A12 A2 OR1
end attack nodes

begin defense nodes
 M12 M1 M2
end defense nodes

begin countermeasure nodes
 D12 = {A12}, D1 = {A1}, D2 = {A2}
end countermeasure nodes

begin attack diagram
 G -OR-> {A1, A2}
 A1 -AND-> {OR1, A12}
 OR1 -OR-> {A111, A112}
 A112 -OR->{A1121,A1122,A1123}
 D12 -AND-> {M12}
 D1 -AND-> {M1}
 D2 -AND-> {M2}
end attack diagram

begin attack detection rates
 A1 = 0.5, A12 = 0.5, A2 = 0.5
end attack detection rates

begin defense effectiveness
 M12(ALL, A12) = 0.5, M1(ALL, A1) = 0.5, M2(ALL, A2) = 0.5
end defense effectiveness
```

Code 19 – Countermeasure tree of Fig. 12 specified in RisQFLan.

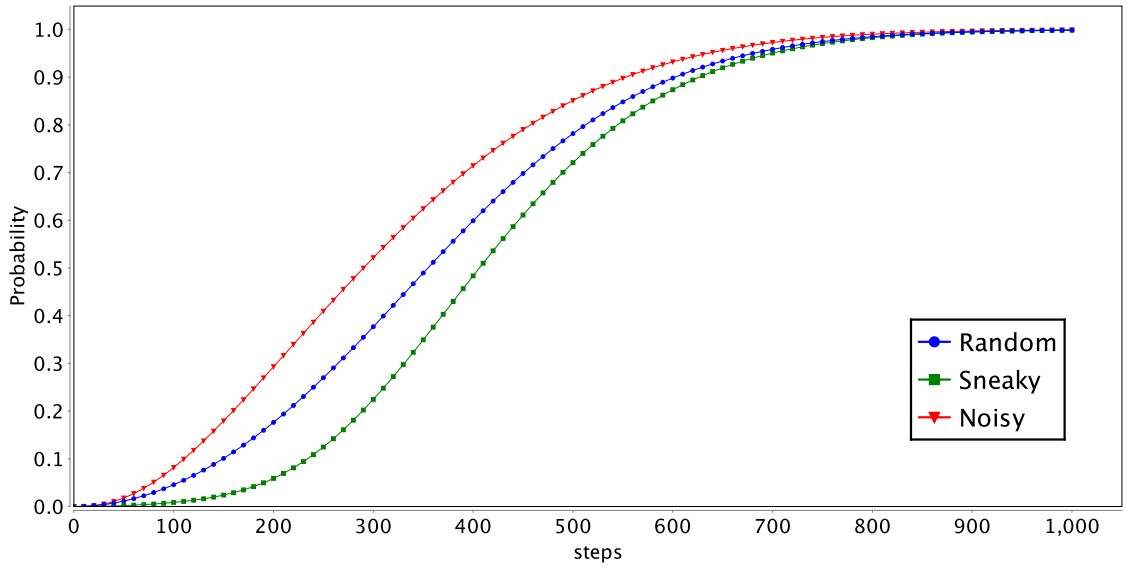


Fig. 13 – Exact PMC analysis on "Resetting BGP".

- detection events D12, D1 and D2 are modeled as countermeasure nodes; the attacks A12, A1 and A2 they intend to detect, resp., are specified accordingly (cf. Line 9 in Code 19);
- measure events M12, M1 and DM2 are modeled as defense nodes (cf. Line 5 in Code 19); the attacks A12, A1 and A2 they mitigate, resp., are specified as attack effectiveness block (cf. Line 27 in Code 19);
- the relation between a detection event D and its triggered mitigation event M is modeled in RisQFLan by specifying defense node D as a refinement of countermeasure node M (cf. Line 18-20 in Code 19).

7.3.3. *Complementing the analysis of Roy et al., 2012 with RisQFLan*

The approach in Roy et al., 2012 includes rich analyses for attack countermeasure trees, including success probabilities, costs and impact of attacks and defensive mechanisms. RisQFLan can augment such analyses with quantitative verification of specific attacker profiles. To illustrate this, we modeled three profiles:

`Random`: an attacker that randomly tries attacks until the main attack goal is achieved;
`Noisy`: like `Random` but tries attacks for which countermeasures exist with higher probability with respect to those for which no countermeasure exists;
`Sneaky`: like `Random` but tries attacks for which countermeasures exist with lower probability with respect to those for which no countermeasure exists.

We analyzed this scenario using the PMC functionalities of PRISM. Indeed, the DMTCs for the attackers could be generated by RisQFLan, and handled by PRISM. We `labeled with hasG` all states `when has(G)` was satisfied. The property we studied is the probability of success at each step, suitably formulated in the property specification language of PRISM. Fig. 13, generated by PRISM, shows the results of the analyses: since all attackers are given the chance to try again and again, they are all eventually successful, but they differ with respect to the amount of time needed to succeed. Paradoxically, the `Noisy` attacker converges faster, which means that the detection and measure events are not as effective as they should be.

8. Related work

There is a large body of related work. Throughout the paper, we indicated some sources of inspiration, like attack profiles specified as automata to describe possible attack steps and their costs Hermanns et al. (2016); Lenin et al. (2014) and the attack detection rates Amenaza Technologies Limited (2006), ordered attacks Çamtepe and Yener (2007) and countermeasures Roy et al., 2012 treated in Sections 7.1, 7.2, and 7.3, resp. A recent study by Widel et al. Widel et al. (2019) classified existing approaches integrating attack tree-based modeling and formal methods along three dimensions. We believe that RisQFLan can act as a unification of those dimensions. In this section, we detail the dimensions and relate RisQFLan to existing approaches.

The first dimension (a major focus of the large-scale EU project TREsPASS TREsPASS, 2016) is the generation of attack trees from scenarios. The main difficulty is to find a compact and effective representation, knowing that structurally different trees can capture the same information. A representative contribution in this area is the ATSyRa toolset Pinchinat et al. (2015). An original and crucial feature of ATSyRa is the support for high-level actions (which can be seen as a sub-goal of the attacker) to specify how sequences of actions can be abstracted and structured. Those high-level actions can later be used in a refactorization and hence better representation of the tree. The contribution is packed up in an elegant Eclipse plugin which makes it easily accessible to the uninitiated. Another contribution is the

process-algebraic generation approach from Vigo and the Nielsons Vigo et al. (2014), where attacks are generated from flow constraints using a SAT solver, and a value-passing quality calculus is used to represent how an attacker can reach a given location. Our approach is not concerned with the synthesis of attack trees, but those techniques and tools could be combined in RisQFLan, to complement them with analysis capabilities.

The second dimension in Widel et al. (2019) is that of giving a rigorous mathematical meaning to (extended) attack trees. The objective is to address a wide range of static problems, like comparing trees or enumerating the attacks. Well-known representations include Boolean function-based semantics, multisets, and linear logics (cf. Audinot et al. (2017); Widel et al. (2019), and their references). This research trend is very similar to the one applied to feature diagrams Czarnecki et al. (2008), and it is likely that many results from the software engineering community concerning product lines or configurable systems can be transferred to the security domain ter Beek et al., 2020. It is worth noticing that the above mentioned approaches do not permit reasoning on the order of steps of the attack, a distinguishing feature that our approach has adopted, together with the ability to undo attack action.

More recently, several researchers have suggested to extend attack tree representations with their environment, i.e. the attacker and the system under attack. For example, the authors of Gadyatskaya et al. (2016); Hansen et al. (2017) not only consider the attack tree itself, but also a transition system representation of an attacker model. The separation of the attack tree from the attacker model as we do in RisQFLan is fundamental to avoid confusion as explained by Mantel et al. Mantel and Probst (2019). This addition allows one to reason not only on static problems, but also on dynamic ones. For instance, one can make hypotheses on attack step sequences or extract correlations between step orders. In addition, the use of transition system-based representations allows one to encompass a model of the system under attack, and by consequences of (the order of) its defenses Kordy et al. (2014a). In this context, contributions like Gadyatskaya et al. (2016) consider that defenses are fixed a priori, while the game-based approach of Aslanyan et al. Aslanyan et al. (2016) allows one to propose them dynamically to react to specific orders of attack steps. Observe that the latter proposal generalizes the sequential conjunction approach of Jhawar et al. Jhawar et al. (2015). RisQFLan follows the approach of Aslanyan et al., but uses SMC Legay et al. (2019) (in addition to exact PMC), a simulation-based approach that is less precise but more effective than the exhaustive state-space exploration of the game-based approach. Moreover, RisQFLan offers a richer language to express constraints between attack steps and the behavior of the system under attack.

The third dimension proposed in Widel et al. (2019) is that of adding quantitative algorithms to reason on (extensions of) attack trees. This is achieved by enriching attack trees with quantitative information, like the cost or probability of an attack step. In this context, static techniques can still be used to answer extended membership queries such as computing the cost of an attack, the Pareto optimal attack for two or more quantitative parameters, or the optimal countermeasures Aslanyan and Nielson (2015); Fila and Widel (2019, 2020). However, as observed by Kordy et al., minimal representations no longer exist Kordy et al. (2012, 2016b), which drastically complicates both the comparison and the synthesis of trees. Quantitative analysis extends to the dynamic case, meaning one can benefit from all the recent work on quantitative formal verification, where the attacker model can remain non-deterministic or even become stochastic. One can then synthesize strategies of the attacker that belongs to the tree and for which the cost is at most a certain value. Over the last five years, a wide range of such techniques has been proposed. Some of those techniques were developed by Legay et al. Gadyatskaya et al. (2016); Hansen et al. (2017). These approaches rely on a quantitative representation of the attacker together with a timed automata-based model for the system. Defenses are provided a priori. The approaches were implemented in the UPPAAL framework, which allows one to use extensions like UPPAAL SMC David et al. (2015b) to compute the probability or cost of an attack. In case non-determinism is added to the attacker model, UPPAAL Stratego David et al., 2015 can be used to synthesize strategies.

RisQFLan goes further than Gadyatskaya et al. (2016); Hansen et al. (2017) by (i) proposing a DSL and (ii) allowing to not only quantify the number of attack steps, but also offering a rich process-algebraic language to impose conditions between steps as well as between defenses that can moreover be added at runtime. However, RisQFLan does not offer non-determinism for attackers. This may be needed to reason on the use of several strategies. A solution could be to add non-deterministic aspects to the DSL, and extend our DTMC exporter to an exporter for Markov decision processes, or in the input language of UPPAAL if also time aspects were to be considered, or to combine RisQFLan's SMC engine with the Plasma Plugin for non-deterministic systems D'Argenio et al., 2015. The approach by Aslanyan et al. Aslanyan et al. (2016) allows reasoning on causality between steps and non-deterministic attackers, but restricted to Boolean causalities called waves, and without DSL. Stoelinga et al. also proposed dynamic approaches to analyze attack trees via SMC. Those approaches are covered and extended by RisQFLan, especially concerning (i) the causality part and (ii) the DSL, which is restricted to the query part with LOCKS Kumar et al. (2018a). Finally, compared to the three approaches mentioned above, only RisQFLan proposes a fully dedicated and maintained open-source toolset.

9. Conclusion and future work

We instantiated QFLanin the quantitative security risk modeling and analysis domain, and applied the outcome, RisQFLan, to three case studies from well-known tools from the graph-based risk modeling and analysis domain. By enhancing the analysis features of these tools with either exact or statistical verification of probabilistic attack scenarios, RisQFLan constitutes a significant contribution to the domain's toolsets.

The generalization and subsequent instantiation of QFLan was feasible since it is open source, a distinguishing feature of RisQFLan among the toolsets available in the domain.

RisQFLan's DTMC exporting facilities moreover permit toolchaining with probabilistic model checkers for models of sizes that do not require SMC.

RisQFLan could be further enriched in several directions. First, we propagate the value of an attribute of a node as the sum of the attribute's values of its descendants. This could be generalized to attribute-specific formulae as in SecurITree, in which, e.g., the noticeability value of a node with n descendants $d_1, d_2, \ldots, d_n$ is computed as $1-((1-d_1)(1-d_2)\cdots(1-d_n))$. Most properties analyzed so far with RisQFLan concern logical requirements. Recently, SMC has also been used to compare system behavior via simulation Larsen et al. (2017). We could compare the behavior of two attackers via simulation or their effect on two different attack-defense diagrams. Finally, MultiVeStA has recently been redesigned and extended with novel statistical estimation techniques Vandin et al. (2021). One example is the ability to estimate properties *on-the-long-run*, i.e. at steady-state. Another example is the ability to compare the analysis results for different model configurations, to check whether there are statistically meaningful differences. For the future, we are interested in extending also RisQFLan with such analysis capabilities.

We also plan to consider non-deterministic and game aspects along the lines of Aslanyan et al. (2016); Gadyatskaya et al. (2016); Hansen et al. (2017), as discussed in detail in Section 8, as well as synthesis of attack profiles and countermeasures (cf., e.g., Fila and Wideł (2020)) for underspecified attack profiles.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Maurice H. ter Beek: Methodology, Validation, Formal analysis, Writing - original draft, Writing - review & editing, Visualization, Project administration. **Axel Legay:** Writing - original draft, Writing - review & editing. **Alberto Lluch Lafuente:** Conceptualization, Methodology, Validation, Formal analysis, Writing - original draft, Writing - review & editing, Project administration, Funding acquisition. **Andrea Vandin:** Conceptualization, Methodology, Software, Validation, Formal analysis, Data curation, Writing - original draft, Writing - review & editing, Project administration.

Acknowledgment

Supported by EU H2020 SU-ICT-03-2018 project 830929 CyberSec4Europe. We are grateful to Christian Bach and Christian Toftemann Bæk for their contribution to the development of RisQFLan as part of their M.Sc. projects. We thank the anonymous reviewers for their useful comments and suggestions that helped us to improve the presentation.

REFERENCES

Agha G, Palmskog K. A survey of statistical model checking. ACM Trans. Model. Comp. Simul. 2018;28(1) 6:1–6:39. doi:10.1145/3158668.

Amenaza Technologies Limited, 2006. The SecuITree®BurgleHouse Tutorial (a.k.a., Who wants to be a Cat Burglar?). 2.5 edition. https://www.amenaza.com/downloads/docs/Tutorial.pdf.

Arnold A, Baleani M, Ferrari A, Marazza M, Senni V, Legay A, Quilbeuf J, Etzien C. An application of SMC to continuous validation of heterogeneous systems. EAI Endorsed Trans. Indust. Netw. & Intellig. Syst. 2017;4(10). doi:10.4108/eai.1-2-2017.152154.

Aslanyan Z, Nielson F. Pareto Efficient Solutions of Attack-Defence Trees. In: Focardi R, Myers AC, editors. In: Proceedings of the 4th International Conference on Principles of Security and Trust (POST'15). Springer; 2015. p. 95–114. doi:10.1007/978-3-662-46666-7_6.

Aslanyan Z, Nielson F, Parker D. Quantitative Verification and Synthesis of Attack-Defence Scenarios. In: Proceedings of the 29th Computer Security Foundations Symposium (CSF'16). IEEE; 2016. p. 105–19. doi:10.1109/CSF.2016.15.

Audinot M, Pinchinat S, Kordy B. Is My Attack Tree Correct?. In: Foley SN, Gollmann D, Snekkenes E, editors. In: Proceedings of the 22nd European Symposium on Research in Computer Security (ESORICS'17). Springer; 2017. p. 83–102. doi:10.1007/978-3-319-66402-6_7.

Baier C, Katoen J-P. Principles of model checking. The MIT Press; 2008.

Bao R, Attiogbé JC, Delahaye B, Fournier P, Lime D. Parametric Statistical Model Checking of UAV Flight Plan. In: Pérez JA, Yoshida N, editors. In: Proceedings of the 39th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE'19). Springer; 2019. p. 57–74. doi:10.1007/978-3-030-21759-4_4.

Basile D, ter Beek MH, Ciancia V. Statistical Model Checking of a Moving Block Railway Signalling Scenario with Uppaal SMC: Experience and Outlook. In: Margaria T, Steffen B, editors. In: Proceedings of the 8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Verification (ISoLA'18). Springer; 2018. p. 372–91. doi:10.1007/978-3-030-03421-4_24.

Basile D, ter Beek MH, Ferrari A, Legay A. Modelling and Analysing ERTMS L3 Moving Block Railway Signalling with Simulink and Uppaal SMC. In: Larsen KG, Willemse T, editors. In: Proceedings of the 24th International Conference on Formal Methods for Industrial Critical Systems (FMICS'19). Springer; 2019. p. 1–21. doi:10.1007/978-3-030-27008-7_1.

Basile D, ter Beek MH, Legay A. Strategy Synthesis for Autonomous Driving in a Moving Block Railway System with Uppaal Stratego. In: Gotsman A, Sokolova A, editors. Proceedings of the 40th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE'20). Springer; 2020. p. 3–21.

Basile D, Di Giandomenico F, Gnesi S. Statistical Model Checking of an Energy-Saving Cyber-Physical System in the Railway Domain. In: Proceedings of the 32nd Symposium on Applied Computing (SAC'17). ACM; 2017. p. 1356–63. doi:10.1145/3019612.3019824.

Basile D, Fantechi A, Rucher L, Mandò G. Statistical model checking of hazards in an autonomous tramway positioning system. In: Collart-Dutilleul S, Lecomte T, Romanovsky AB, editors. In: Proceedings of the 3rd International Conference on Reliability, Safety, and Security of Railway Systems: Modelling, Analysis, Verification, and Certification (RSSRail'19). Springer; 2019. p. 41–58. doi:10.1007/978-3-030-18744-6_3.

Bernardo, M., De Nicola, R., Hillston, J. (Eds.), 2016. Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems. Vol. 9700 of Lecture Notes in Computer Science, Springer. 10.1007/978-3-319-34096-8

Bozga M, David A, Hartmanns A, Hermanns H, Larsen KG, Legay A, Tretmans J. State-of-the-Art Tools and Techniques for Quantitative Modeling and Analysis of Embedded Systems. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE'12). EDAA; 2012. p. 370–5. doi:10.1109/DATE.2012.6176499.

Çamtepe SA, Yener B. Modeling and detection of complex attacks. In: Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks (SecureComm'07). IEEE; 2007. p. 234–43. doi:10.1109/SECCOM.2007.4550338.

Cappart Q, Limbrée C, Schaus P, Quilbeuf J, Traonouez L-M, Legay A. Verification of Interlocking Systems Using Statistical Model Checking. In: Proceedings of the 18th International Symposium on High Assurance Systems Engineering (HASE'17). IEEE; 2017. p. 61–8. doi:10.1109/HASE.2017.10.

. Handbook of model checking. In: Clarke EM, Henzinger TA, Veith H, Bloem R, editors. Springer; 2018. doi:10.1007/978-3-319-10575-8.

Czarnecki K, She S, Wasowski A. Sample Spaces and Feature Models: There and Back Again. In: Proceedings of the 12th International Software Product Lines Conference (SPLC'08). IEEE; 2008. p. 22–31. doi:10.1109/SPLC.2008.49.

D'Argenio P, Legay A, Sedwards S, Traonouez L-M. Smart sampling for lightweight verification of Markov decision processes. Int. J. Softw. Tools Technol. Transf. 2015;17(4):469–84. doi:10.1007/s10009-015-0383-0.

David A, Jensen PG, Larsen KG, Mikučionis M, Taankvist JH. Uppaal Stratego. In: Baier C, Tinelli C, editors. In: Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15). Springer; 2015. p. 206–11.

David A, Larsen KG, Legay A, Mikucionis M, Poulsen DB. Uppaal SMC tutorial. Int. J. Softw. Tools Technol. Transf. 2015;17(4):397–415. doi:10.1007/s10009-014-0361-y.

Dehnert C, Junges S, Katoen J-P, Volk M. A Storm is Coming: A Modern Probabilistic Model Checker. In: Majumdar R, Kunčak V, editors. In: Proceedings of the 29th International Conference on Computer Aided Verification (CAV'17). Springer; 2017. p. 592–600. doi:10.1007/978-3-319-63390-9_31.

Ferrari A, Mazzanti F, Basile D, ter Beek MH, Fantechi A. Comparing Formal Tools for System Design: a Judgment Study. In: Proceedings of the 42nd International Conference on Software Engineering (ICSE'20). ACM; 2020. p. 62–74.

Fila B, Wideł W. Efficient Attack-Defense Tree Analysis using Pareto Attribute Domains. In: Proceedings of the 32nd Computer Security Foundations Symposium (CSF'19). IEEE; 2019. p. 200–15. doi:10.1109/CSF.2019.00021.

Fila B, Wideł W. Exploiting attack-defense trees to find an optimal set of countermeasures. In: Proceedings of the 33rd Computer Security Foundations Symposium (CSF'20). IEEE; 2020. p. 395–410. doi:10.1109/CSF49147.2020.00035.

Filipovikj P, Mahmud N, Marinescu R, Seceleanu C, Ljungkrantz O, Lönn H. Simulink to UPPAAL Statistical Model Checker: Analyzing Automotive Industrial Systems. In: Fitzgerald J, Heitmeyer C, Gnesi S, Philippou A, editors. In: Proceedings of the 21st International Symposium on Formal Methods (FM'16). Springer; 2016. p. 748–56. doi:10.1007/978-3-319-48989-6_46.

Gadyatskaya O, Hansen RR, Larsen KG, Legay A, Olesen MC, Poulsen DB. Modelling Attack-defense Trees Using Timed Automata. In: Fränzle M, Markey N, editors. In: Proceedings of the 14th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'16). Springer; 2016. p. 35–50. doi:10.1007/978-3-319-44878-7_3.

Garavel H, ter Beek MH, van de Pol J. The 2020 Expert Survey on Formal Methods. In: ter Beek MH, Ničković D, editors. In: Proceedings of the 25th International Conference on Formal Methods for Industrial Critical Systems (FMICS'20). Springer; 2020. p. 3–69. doi:10.1007/978-3-030-58298-2_1.

Gilmore S, Reijsbergen D, Vandin A. Transient and Steady-State Statistical Analysis for Discrete Event Simulators. In: Polikarpova N, Schneider S, editors. In: Proceedings of the 13th International Conference on Integrated Formal Methods (IFM'17). Springer; 2017. p. 145–60. doi:10.1007/978-3-319-66845-1_10.

Gilmore S, Tribastone M, Vandin A. An Analysis Pathway for the Quantitative Evaluation of Public Transport Systems. In: Albert E, Sekerinski E, editors. In: Proceedings of the 11th International Conference on Integrated Formal Methods (IFM'14). Springer; 2014. p. 71–86. doi:10.1007/978-3-319-10181-1_5.

Hahn EM, Hartmanns A, Hensel C, Klauck M, Klein J, Kretínský J, Parker D, Quatmann T, Ruijters E, Steinmetz M. The 2019 comparison of tools for the analysis of quantitative formal models. In: Beyer D, Huisman M, Kordon F, Steffen B, editors. In: Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: TOOLympics (TACAS'19). Springer; 2019. p. 69–92. doi:10.1007/978-3-030-17502-3_5.

Hansen RR, Jensen PG, Larsen KG, Legay A, Poulsen DB. Quantitative Evaluation of Attack Defense Trees Using Stochastic Timed Automata. In: Liu P, Mauw S, Stølen K, editors. In: Proceedings of the 4th International Workshop on Graphical Models for Security (GraMSec'17). Springer; 2017. p. 75–90. doi:10.1007/978-3-319-74860-3_5.

Hartmanns A, Hermanns H. In the quantitative automata zoo. Sci. Comput. Program. 2015;112:3–23. doi:10.1016/j.scico.2015.08.009.

Hermanns H, Krämer J, Krcál J, Stoelinga M. The Value of Attack-Defence Diagrams. In: Piessens F, Viganò L, editors. In: Proceedings of the 5th International Conference on Principles of Security and Trust (POST'16). Springer; 2016. p. 163–85. doi:10.1007/978-3-662-49635-0_9.

Hong JB, Kim DS, Chung C-J, Huang D. A survey on the usability and practical applications of graphical security models. Comput. Sci. Rev. 2017;26:1–16. doi:10.1016/j.cosrev.2017.09.001.

Ingoldsby TR. In: Technical Report. Attack Tree-based Threat Risk Analysis. Amenaza Technologies Limited; 2013.

Jhawar R, Kordy B, Mauw S, Radomirovic S, Trujillo-Rasua R. Attack Trees with Sequential Conjunction. In: Federrath H, Gollmann D, editors. In: Proceedings of the 30th IFIP TC 11 International Conference on ICT Systems Security and Privacy Protection (SEC'15). Springer; 2015. p. 339–53. doi:10.1007/978-3-319-18467-8_23.

Katoen J-P, Larsen KG. Quantitative Modelling and Analysis. In: Margaria T, Steffen B, editors. In: Proceedings of the 5th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Applications and Case Studies (ISoLA'12). Springer; 2012. p. 290–2. doi:10.1007/978-3-642-34032-1_27.

Kordy B, Kordy P, van den Boom Y. SPTool – Equivalence Checker for SAND Attack Trees. In: Cuppens F, Cuppens N, Lanet J-L, Legay A, editors. In: Proceedings of the 11th International Conference on Risks and Security of Internet and Systems (CRiSIS'16). Springer; 2016. p. 105–13. doi:10.1007/978-3-319-54876-0_8.

Kordy B, Kordy P, Mauw S, Schweitzer P. ADTool: Security Analysis with Attack-Defense Trees. In: Joshi K, Siegle M, Stoelinga M, D'Argenio PR, editors. In: Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST'13). Springer; 2013. p. 173–6. doi:10.1007/978-3-642-40196-1_15.

Kordy B, Mauw S, Radomirović S, Schweitzer P. Foundations of Attack-Defense Trees. In: Degano P, Etalle S, Guttman J, editors. In: Proceedings of the 7th International Workshop on Formal Aspects in Security and Trust (FAST'10). Springer; 2011. p. 80–95. doi:10.1007/978-3-642-19751-2_6.

Kordy B, Mauw S, Radomirovic S, Schweitzer P. Attack-defense trees. J. Log. Comput. 2014;24(1):55–87. doi:10.1093/logcom/exs029.

Kordy B, Mauw S, Schweitzer P. Quantitative Questions on Attack-Defense Trees. In: Kwon T, Lee M-K, Kwon D, editors. In: Proceedings of the 15th International Conference on Information Security and Cryptology (ICISC'12). Springer; 2012. p. 49–64. doi:10.1007/978-3-642-37682-5_5.

Kordy B, Piètre-Cambacédès L, Schweitzer P. DAG-based attack and defense modeling: don't miss the forest for the attack trees. Comput. Sci. Rev. 2014;13–14:1–38. doi:10.1016/j.cosrev.2014.07.001.

Kordy B, Pouly M, Schweitzer P. Probabilistic reasoning with graphical security models. Inf. Sci. 2016;342:111–31. doi:10.1016/j.ins.2016.01.010.

Kumar R, Rensink A, Stoelinga M. LOCKS: a property specification language for security goals. In: Proceedings of the 33rd Symposium on Applied Computing (SAC'18). ACM; 2018. p. 1907–15. doi:10.1145/3167132.3167336.

Kumar R, Ruijters E, Stoelinga M. Quantitative Attack Tree Analysis via Priced Timed Automata. In: Sankaranarayanan S, Vicario E, editors. In: Proceedings of the 13th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'15). Springer; 2015. p. 156–71. doi:10.1007/978-3-319-22975-1_11.

Kumar R, Schivo S, Ruijters E, Yildiz BM, Huistra D, Brandt J, Rensink A, Stoelinga M. Effective Analysis of Attack Trees: A Model-Driven Approach. In: Russo A, Schürr A, editors. In: Proceedings of the 21st International Conference on Fundamental Approaches to Software Engineering (FASE'18). Springer; 2018. p. 56–73. doi:10.1007/978-3-319-89363-1_4.

Kumar R, Stoelinga M. Quantitative Security and Safety Analysis with Attack-Fault Trees. In: Proceedings of the 18th IEEE International Symposium on High Assurance Systems Engineering (HASE'17). IEEE; 2017. p. 25–32. doi:10.1109/HASE.2017.12.

Kwiatkowska MZ, Norman G, Parker D. PRISM 40: Verification of Probabilistic Real-Time Systems. In: Gopalakrishnan G, Qadeer S, editors. In: Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11). Springer; 2011. p. 585–91. doi:10.1007/978-3-642-22110-1_47.

Larsen KG, Legay A, Mikučionis M, Nielsen B, Nyman U. Compositional Testing of Real-Time Systems. In: Lecture Notes in Computer Science, Vol 10500. Springer; 2017. p. 107–24. doi:101007/978-3-319-68270-9_6.

Law AM. Simulation modeling and analysis. 5th. McGraw-Hill; 2015.

Legay A, Lukina A, Traonouez L-M, Yang J, Smolka SA, Grosu R. Statistical Model Checking. In: Lecture Notes in Computer Science, Vol 10000. Springer; 2019. p. 478–504. doi:101007/978-3-319-91908-9_23.

Lenin A, Willemson J, Sari DP. Attacker Profiling in Quantitative Security Assessment Based on Attack Trees. In: Bernsmed K, Fischer-Hübner S, editors. In: Proceedings of the 19th Nordic Conference on Secure IT Systems (NordSec'14). Springer; 2014. p. 199–212. doi:10.1007/978-3-319-11599-3_12.

Lv W-p, Li W-m. Space Based Information System Security Risk Evaluation Based on Improved Attack Trees. In: Proceedings of the 3rd International Conference on Multimedia Information Networking and Security (MINES'11). IEEE; 2011. p. 480–3. doi:10.1109/MINES.2011.94.

Mantel H, Probst CW. On the Meaning and Purpose of Attack Trees. In: Proceedings of the 32nd Computer Security Foundations Symposium (CSF'19). IEEE; 2019. p. 184–99. doi:10.1109/CSF.2019.00020.

Mauw S, Oostdijk M. Foundations of Attack Trees. In: Won D, Kim S, editors. In: Proceedings of the 8th International Conference on Information Security and Cryptology (ICISC'05). Springer; 2005. p. 186–98. doi:10.1007/11734727_17.

Pinchinat S, Acher M, Vojtisek D. ATSyRa: An Integrated Environment for Synthesizing Attack Trees (Tool Paper). In: Mauw S, Kordy B, Jajodia S, editors. In: Proceedings of the 2nd International Workshop on Graphical Models for Security (GraMSec'15). Springer; 2015. p. 97–101. doi:10.1007/978-3-319-29968-6_7.

Puch S, Fränzle M, Gerwinn S. Quantitative Risk Assessment of Safety-Critical Systems via Guided Simulation for Rare Events. In: Margaria T, Steffen B, editors. In: Proceedings of the 8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Verification (ISoLA'18). Springer; 2018. p. 305–21. doi:10.1007/978-3-030-03421-4_20.

Roy A, Kim DS, Trivedi KS. Attack Countermeasure Trees (ACT): towards unifying the constructs of attack and defense trees. Secur. Commun. Netw. 2012;5(8):929–43. doi:10.1002/sec.299.

Schneier B. Attack trees. Dr. Dobb's Journal 1999.

Sebastio S, Vandin A. MultiVeStA: Statistical Model Checking for Discrete Event Simulators. In: Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools (ValueTools'13). ACM; 2013. p. 310–15. doi:10.4108/icst.valuetools.2013.254377.

ter Beek MH, Legay A. Quantitative variability modelling and analysis. Int. J. Softw. Tools Technol. Transf. 2019;21(6):607–12. doi:10.1007/s10009-019-00535-1.

ter Beek MH, Legay A, Lluch Lafuente A, Vandin A. Statistical Model Checking for Product Lines. In: Margaria T, Steffen B, editors. In: Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques (ISoLA'16). Springer; 2016. p. 114–33.

ter Beek MH, Legay A, Lluch Lafuente A, Vandin A. Variability meets Security: Quantitative Security Modeling and Analysis of Highly Customizable Attack Scenarios. Proceedings of the 14th International Working Conference on Variability Modelling of Software-intensive Systems (VaMoS'20). ACM, 2020.

ter Beek MH, Legay A, Lluch Lafuente A, Vandin A. A framework for quantitative modeling and analysis of highly (re)configurable systems. IEEE Trans. Softw. Eng. 2020;46(3):321–45. doi:10.1109/TSE.2018.2853726.

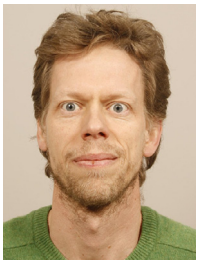
TREsPASS http://www.trespass-project.eu/.

Vandin A, ter Beek MH, Legay A, Lluch Lafuente A. QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems. In: Havelund K, Peleska J, Roscoe B, de Vink E, editors. In: Proceedings of the 22nd International Symposium on Formal Methods (FM'18). Springer; 2018. p. 329–37. doi:10.1007/978-3-319-95582-7_19.

Vandin, A., Giachini, D., Lamperti, F., Chiaromonte, F., 2021. Automated and Distributed Statistical Analysis of Economic Agent-Based Models. arXiv:2102.05405 **[econ.GN]**. https://arxiv.org/abs/2102.05405.

Vigo R, Nielson F, Nielson HR. Automated generation of attack trees. In: Proceedings of the 27th Computer Security Foundations Symposium (CSF'14). IEEE; 2014. p. 337–50. doi:10.1109/CSF.2014.31.

Wideł W, Audinot M, Fila B, Pinchinat S. Beyond 2014: formal methods for attack tree–based security modeling. ACM Comput. Surv. 2019;52(4). doi:10.1145/3331524. 75:1–75:36



Maurice ter Beek is senior researcher in the Istituto di Scienza e Tecnologie dell'Informazione (ISTI) of the Consiglio Nazionale delle Ricerche (CNR) in Pisa, Italy, and head of the Formal Methods and Tools (FMT) lab. He obtained his Ph.D. in Computer Science from Leiden University, The Netherlands, with a thesis introducing Team Automata. His research interests are in formal methods and model-checking tools, applied in particular to service computing, software product line engineering and railway systems.

Axel Legay is a professor at the Université Catholique de Louvain (UCLouvain, Belgium). He received his Ph.D. in Computer Science from the University of Liége, Belgium. His main research interests are in formal verification. He is a founder and major contributor of statistical model checking (a statistical variant of model checking effectively used in industry). He is referee for top journals and conferences in formal verification.

Alberto Lluch Lafuente is an associate professor at the Department of Applied Mathematics and Computer Science of the Technical University of Denmark (DTU). He received his Ph.D. in Computer Science from the Albert Ludwigs University of Freiburg im Breisgau, Germany. His research interests are in formal methods for safe and secure distributed systems, languages and models for concurrency and coordination, software engineering and artificial intelligence.

Andrea Vandin is tenure-track assistant professor in Computer Science at Sant'Anna School for Advanced Studies, Pisa, Italy. Previously he was associate professor at DTU, until 2019, assistant professor at IMT School for Advanced Studies Lucca, Italy, until 2017, and senior research assistant at University of Southampton, UK, until 2015. His research interests are the development of scalable techniques for formal quantitative system analysis. He is interested in applying his research in practice, thus providing tool support for most of his contributions.