A Software Engineering Approach to Quantitative Security Risk Modeling and Analysis using QFLan

Anonymous Author(s)

ABSTRACT

Domain-specific quantitative modeling and analysis approaches are essential to support software and system engineering in scenarios where *qualitative* approaches are inappropriate or unfeasible. In this paper, we generalize QFLan, a successful domain-specific approach to support quantitative modeling and analysis of highly configurable systems, by decoupling domain-specific components to ease their instantiation in new domains. We validate our proposal by instantiating the QFLan approach in one such new domain, namely security risk modeling and analysis. The result is a new tool, called RisQFLan, to support a software engineering approach to quantitative security risk modeling and analysis, which constitutes a significant contribution to the existing toolsets in that domain. We illustrate features of RisQFLan with three case studies from seminal approaches to risk analysis based on attack trees.

CCS CONCEPTS

• **Software and its engineering** → **Specification languages**; **Formal methods**; **Extra-functional properties**; • **Security and privacy** → **Formal methods and theory of security**; **Formal security models**.

KEYWORDS

Graphical security risk models, attack-defense trees, quantitative security, QFLan, statistical model checking, formal analysis tools

ACM Reference Format:

Anonymous Author(s). 2020. A Software Engineering Approach to Quantitative Security Risk Modeling and Analysis using QFLan. In *ASE 2020: 35th IEEE/ACM International Conference on Automated Software Engineering, September 21–25, 2020, Melbourne, Australia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Quantitative modeling and analysis approaches are essential to support software and system engineering in scenarios where *qualitative* approaches are inappropriate or unfeasible, for example due to complexity or uncertainty, or by quantitative nature of the properties of interest. Automated approaches to support quantitative modeling and analysis have been developed during the last decades, including generic as well as domain-specific approaches (cf., e.g., [4, 6, 7, 9, 10, 18, 20, 25, 34, 36]).

ASE 2020, September 21–25, 2020, Melbourne, Australia

ACM ISBN 978-1-4503-XXXX-X/20/09…$15.00
https://doi.org/10.1145/1122445.1122456

QFLan [7] is one such example of a successful domain-specific approach to support quantitative modeling and analysis of highly configurable systems, such as software product lines. QFLan combines a number of well-studied rigorous notions and techniques in an Eclipse-based domain-specific tool framework. The framework consists of a domain-specific language (DSL) tailored for product lines, and an analysis engine based on statistical model checking [1, 38]. The authors of [7] showed the robustness and scalability of QFLan by analyzing large instances of case studies that could not be analyzed before.

Research question. This paper addresses the following research question: *can the success story of QFLan be exported to other domains*?

Contributions. We answer positively by

(1) generalizing the QFLan approach by decoupling domain-specific components to ease its instantiation in new domains;
(2) instantiating the QFLan approach in a new domain, namely risk modeling and analysis.

The result of (1) and (2) is a new tool (RisQFLan) to support a *software engineering approach to quantitative security risk modeling and analysis*, and constitutes a significant contribution to the existing toolset in that area. In particular, RisQFLan can be used to

(a) build rich models by combining popular features from existing formalisms;
(b) complement the analysis of existing tools that support risk modeling.

Regarding (a), the DSL of RisQFLan has been designed to include the most popular features of existing formalisms based on attack trees so they can be combined in the same model. Subsets of the RisQFLan DSL, indeed, can be used to capture classes of existing models. In addition, RisQFLan allows one to specify specific dynamic threat profiles, a feature that is supported only recently by a few other approaches in a limited way [4, 17, 19, 34, 35, 39]. We illustrate (b) by showing 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 the analyses provided by existing tools.

Synopsis. Section 2 provides an introduction to the domain of security risk modeling with attack trees. Section 3 presents a first contribution of the paper, namely the generic QFLan approach to domain-specific quantitative modeling and analysis. Section 4 describes the main contribution of the paper, namely RisQFLan, a QFLan instance to support security risk modeling and analysis. Section 5 shows the flexibility of RisQFLan by illustrating how features from three influential classes of attack trees can be specified in RisQFLan and how the RisQFLan analysis capabilities can be used to complement and enrich the analyses provided by existing tools. Section 6 discusses related work. Section 7 draws conclusions and outlines avenues for future work.

2 RISK MODELING AND ANALYSIS WITH GRAPH-BASED SECURITY MODELS

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 is used later in Section 4 to illustrate how the QFLan approach has been instantiated in that domain.

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 [28, 41, 44] constitute a popular family of graph-based security models for which several approaches have been developed over the last years (cf., e.g., the surveys [22, 31, 49]), aiming at providing scalable and usable methods for specifying vulnerabilities and countermeasures, their interplay, and their key attributes such as cost and effectiveness. Attack 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.

Essentially, an attack-tree diagram is an and/or tree, whose nodes represent attack goals or defensive measures and whose sub-trees represent refinements of such goals and measures. Figure 1 shows an attack tree modeling our running example. The root of the tree 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 to open the vault (`OpenVault`) *or* to 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 that 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 combing `LearnCombo` and `GetToVault` with 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--3`).

Attack trees can also include defensive mechanisms to deal with or prevent attack threats. In the example scenario there are two defensive mechanisms. First, a `LockDown` *countermeasure* that is triggered by (successful or not) blow up attacks and, 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 measure in our example is a *defense* `Memo` that is 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-tree 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 like Secur*IT*ree [2], ADTool [26], SPTool [27], and ATTop [35] to mention a few (further examples can be found in the surveys [22, 31, 49]).

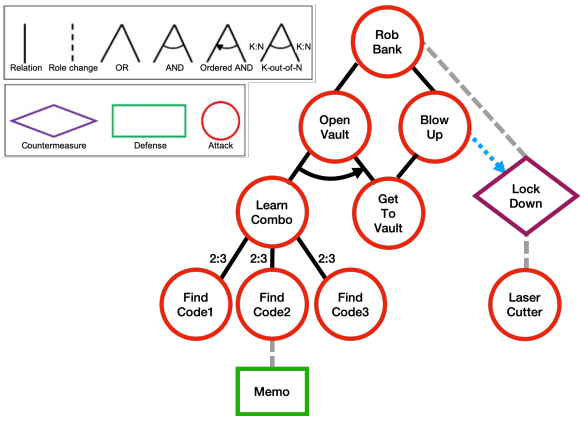

Figure 1: Attack-defense diagram

Some of the standard analyses conducted on attack-tree diagrams regard the feasibility of attacks (e.g. *can the attacker activate some actions that will result in the achievement of its main goal?*), the likelihood of attacks (e.g. *what is the probability that the main goal is achieved* or the cost of attacks (e.g. *what is the cheapest successful attack for the attacker?*). Analysis techniques are often based on constraint solving, optimization and statistical techniques. Section 4.2 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 more amenable for instantiations in domains beyond the one for which it was conceived (configurable systems like Software Product Lines).

The original QFLan architecture. We start by summarizing the original QFLan architecture as presented in [7]. Figure 2 provides a high-level overview of the architecture, which is 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 that embeds the third-party MultiVeStA [45] statistical analyzer. QFLan is an open-source tool.

The components of the `GUI` layer are

- a `QFLAN Editor` with text editing support typical of modern integrated development environments developed within the `XTEXT` framework;
- a `MultiQuaTEx Editor` for property specification in the MultiQuaTEx language;
- a set of `Views`, including a project explorer, a diagnosis console, and a plot viewer for displaying analysis results.

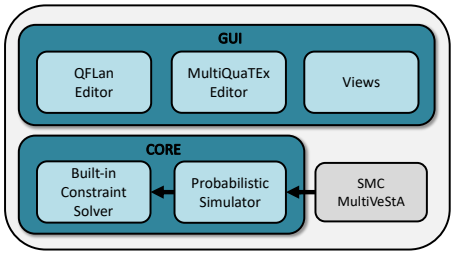

Figure 2: The QFLan architecture reproduced from [7]

The components of the CORE layer are

- a `Probabilistic Simulator`, which is an interpreter of the formal semantics as probabilistic processes;
- a `Built-in Constraint Solver` used by the simulator to check constraints during simulation.

The refactored QFLan architecture. The architecture illustrated in Fig. 3 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 `XTEXT grammar for DSL`, the `Interpreter`, and the `Constraint Solver` (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).

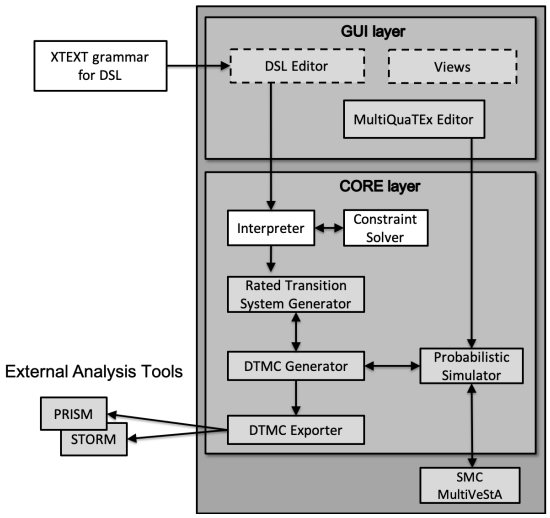

Figure 3: The refactored QFLan architecture

The `GUI` layer basically remains unchanged, with the only exception that we have made it explicit in Fig. 3 that the `DSL Editor` is generated automatically from an `XTEXT grammar for DSL`.

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

- an `Interpreter` and associated `Constraint Solver`, which implement the formal semantics of the DSL based on rated transition systems (transition systems whose transitions are decorated with rates);
- a `Rated Transition System Generator` relying on the `Interpreter` to generate rated transition systems on-the-fly;
- a `DTMC Generator`, which relies on the `Rated Transition System Generator` to normalize rated transition system into on-the-fly generated Discrete-Time Markov Chains (DTMC);
- a `Probabilistic Simulator` of models, which is now separated from the above components and is able to simulate an obtained DTMC without generating it entirely by exploiting the on-the-fly `DTMC Generator`;
- a `DTMC Exporter` which can generate an entire explicit DTMC by exploiting the `DTMC Generator` and export it in the input format of known DTMC analyzers like the probabilistic model checkers PRISM [37] and STORM [16].

4 RisQFLan: AN INSTANTIATION OF QFLan

This section describes RisQFLan, a domain-specific instantiation of QFLan in the security risk domain described in Section 2. We first describe the DSL of RisQFLan in Section 4.1 and then present the analysis capabilities of the RisQFLan tool in Section 4.2.

4.1 RisQFLan DSL

In this section, we illustrate the DSL of RisQFLan through the running example, whose attack-defense diagram is depicted in Fig. 1. In the RisQFLan specification, nodes are declared in specific blocks as shown in Code 1. Note that countermeasure nodes require to indicate the attack node(s) that can trigger them.

```
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

Our attack-defense diagrams relate nodes according to two types of relations: *refinements* structure an offensive (defensive, resp.) node into a set of offensive (defensive, resp.) sub-nodes, while *role-changes* specify how to oppose an offensive (defensive, resp.) node by a defensive (offensive, resp.) node. Each node has at most one refinement and at most one role-change. In Fig. 1, edges are implicitly directed downwards. Typical for our approach is that nodes may have multiple parents, which is convenient to specify one attack (defense) node that affects multiple defenses (attacks) or an attack (defense) node that refines many attacks (countermeasures).

```
1 begin attack diagram
2  RobBank -> { OpenVault, BlowUp }
3  OpenVault -OAND-> [ LearnCombo, GetToVault ]
4  BlowUp -> { GetToVault }
5  LearnCombo -K2-> { FindCode1, FindCode2, FindCode3 }
6  RobBank -> { LockDown }
7  LockDown -> { LaserCutter }
8  FindCode2 -> { Memo }
9 end attack diagram
```

Code 2: Attack-defense diagram

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 become effective only upon (attack detection and) activation of the countermeasure. **AND** and **OR** refinements originate from the seminal works on attack trees [44]. **OAND** refinements stem from *enhanced* and *improved attack trees* [11, 40] 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* [43]. 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* [28],

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 many other approaches [31], attack nodes may be decorated with attributes like cost or detection rates, thus facilitating quantitative analyses [4, 19, 30]. 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 [44]. 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, we consider both successful and failed attack attempts to compute the value of an attribute of an attack node. Moreover, we allow attributes also for defensive nodes.

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

Code 3: Attributes

In [2, 23], an attribute called *noticeability* is a behavioral metric used to indicate the likeliness for an attack attempt to be noticed. Following *attack countermeasure trees* [43], we turn this notion into a first-class citizen of RisQFLan, called *attack detection rate*, which influences the activation of countermeasures. More precisely, such a rate determines the probability for an attack attempt to be detected, and it triggers the activation of the affected countermeasures; 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 [26, 28], 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 [43] 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 $1 - 0.3$.

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

Code 5: Defense effectiveness (ALL denotes any attacker)

In our models, 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 [7, 47]. Explicit attack behavior enables the analyses of specific attacker types, like script kiddies, insiders, and hackers, with the advantage of being able to evaluate system vulnerabilities for those attacker types making more sense for the security scenario under study. Furthermore, 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 [26]).

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 6). Figure 4 (and its corresponding Code 6) 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 also try get-to-vault attacks, required by both. OpenVault requires to try to learn the combo, which in turn requires to try to find at least two codes.

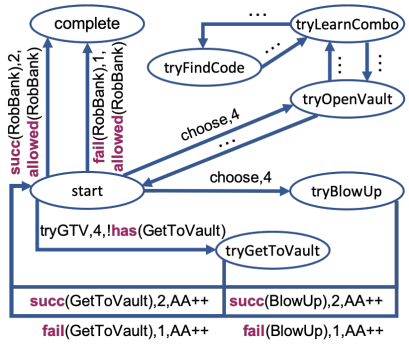

Figure 4: Attack behavior

Attacker actions can be *user-defined* for scenario-specific behavior not directly related to activation of nodes, like those declared in Code 7 (try is part of the tryOpenVault, tryLearnCombo, and tryFindCode attacks not detailed in Fig. 4/Code 6).

RisQFLan also provides 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 5, we will see how attackers can backtrack via the predefined action remove.

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

Code 7: Actions

The behavior of an attack scenario 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. Among those, their relative rates are used to probabilistically choose one (e.g., from start to complete with probability $\frac{1}{1+2}$).

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

Code 8: Action constraints

Transitions can thus also contain *guards*, like allowed, used to attempt RobBank in start only if all required sub-attacks succeeded. Another example is ! **has**, used to forbid the transition to tryGetToVault if one already succeeded to GetToVault. RisQFLan also supports *action constraints*, acting as guards on any transition executing a given action (while transition guards constrain single transitions). As defined in Code 8, any transition with action choose is disabled as soon as one succeeds to open or blow up the vault.

```
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 used (open/blow), 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 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 6: Attack behavior

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

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

Code 9: Variables

In addition to the constraints imposed by the attack diagrams, transition guards, and action constraints, attack behavior may be constrained by quantitative constraints in the form of Boolean expressions involving (inequalities or arithmetic expressions over) reals, attributes, and variables.

In our example we constrain the cost of attacks, which is particularly interesting given that attack behavior may model failed attacks, by the quantitative constraint in Code 10: the accumulated cost of an attack may not exceed 100.

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

Code 10: Quantitative constraints

Attack behavior is completed with an initial setup, which specifies the attacker and initially accomplished attacks (if any). The latter enrich expressiveness, because one can assign an initial advantage to an attacker. Indeed, an attack-defense diagram models all possible attacks, but some attackers (e.g., insiders) might already have access to critical components. This is convenient as a diagram's sub-trees may be ignored without their explicitly removal. The attacker in Code 11 already has the first code.

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

Code 11: Initial setup

4.2 RisQFLan Supported Analysis

RisQFLan supports quantitative analysis of probabilistic attack scenarios by means of statistical model checking (SMC) [1, 38], thus providing a complementary analysis capability to what other risk analysis tools typically offer. The analysis of RisQFLan is obtained thanks to the integration with MultiVeStA [45], a framework for enriching simulators with SMC capabilities. We opted for SMC 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. We showcase two analysis capabilities of RisQFLan on our running example, as well as probabilistic simulation and DTMC exporting facilities.

Analysis while varying simulation steps. We start to study the probabilities of activating attacks and countermeasures while varying the simulation step. 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, considering `FindCode1` is always active, plus countermeasure `LockDown`). Each property can be an arithmetic expression of nodes (evaluating to 1 or 0 if the node is active or not, resp.), variables, or attributes. The properties are considered in all 100 steps, totalling 800 properties.

Each such actual property p_i denotes a random variable X_i which gets a real value assigned in each simulation. MultiVeStA estimates the expected value $E[X_i]$ of each of the 800 properties (reusing the same simulations) as the mean $\overline{x}_i$ of n independent simulations, with n large enough to guarantee an (α, δ) *confidence interval* (CI): $E[x_i]$ belongs to $[\overline{x}_i - \delta/2, \overline{x}_i + \delta/2]$ with statistical confidence $(1-\alpha)\cdot 100\%$. The CI is given by **`alpha`** and **`default delta`** (but a property-specific δ 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.

```
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

Figure 5 shows the analysis results. Recall from Fig. 1 that `RobBank` requires `OpenVault` or `BlowUp`. The probability to activate `RobBank` reaches about 0.6 after 100 steps, while those of `OpenVault` and `BlowUp` reach 0.45 and 0.55, resp. We know from Code 8 that the latter cannot both be activated, so one should be able to activate `RobBank` with probability almost 1 after 100 steps.

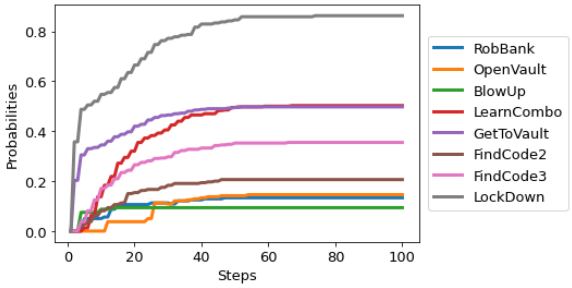

Figure 5: Analysis result of the properties in Code 12

Instead, the actual probability is scaled down by the probabilistic choice from `start` 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 that the probability to activate `BlowUp` should be higher than that of `LockDown`. However, this is not true: it reaches about 0.53 after 80 steps. This is explained by the fact that both succeeded and failed `BlowUp` attempts are detected (cf. success (`succ(BU)`) and failure (`fail(BU)`) in Fig. 4). Interestingly, if we add `LaserCutter` to the initial configuration, then the probability of activating `LockDown` remains 0, as it is inhibited by `LaserCutter`.

Analysis at the verification of a condition. The second analysis capability of RisQFLan presented here regards computing properties at the verification of a given condition. We exemplify this by computing the probability for each attack to be the first attempted and succeeded one. At the same time, we also study the average number of steps necessary to perform the first attack attempt. Code 13 expresses these 9 properties (one probability per attack node plus the average number of steps). The only difference with respect to Code 12 is that the **`from-to-by`** pattern is replaced by the keyword **`when`**, specifying that the properties should be evaluated in the first state satisfying `AttackAttempts == 1`. Moreover, the list of properties of interest now also includes **`steps`**, for which we give a specific delta, evaluated as the average number of steps computed to reach the first state satisfying the required condition.

```
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

Overall the analysis required 400 simulations, performed in a few seconds on a standard laptop machine. 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 previously succeed. Consistently with Fig. 5, `GetToVault` has higher probability than `FindCode2` and `FindCode3`. Intuitively, this depends on the way the attacker's behavior is defined. As sketched in Fig. 4, starting from state `start` we only have to perform one step to try

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	0.23	0	0.04	0.32	2.61

`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 having activated the countermeasure `LockDown`. This means failed BlowUp attempts were detected. Finally, Table 1 also shows that, on average, 2.61 steps are needed to perform one attack attempt. Indeed, in state `start` no attack attempt is allowed, so two steps are necessary to attempt `GetToVault` or `BlowUp` attacks, while three are necessary for `FindCode` attempts.

Simulating and exporting. RisQFLan models can be debugged performing probabilistic simulations: Code 14 prints every chosen state and other information of the simulation for debugging.

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

Code 14: Log generation

RisQFLan's `DTMC Exporter` generates entire explicit DTMCs and exports them in the input format of the probabilistic model checkers PRISM and STORM. Code 15 shows how to export the DTMC of our running example for external analysis labeling with `"hasRB"` all states in which the rob bank attack succeeded.

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

Code 15: DTMC export

5 VALIDATION

A variety of extensions of attack-tree models exist and no single approach has emerged as the ultimate solution [22, 23, 31, 49]. 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 its analysis capabilities can be used to complement and enrich the analyses provided by existing tools. The tool and the full specifications of the models are available at https://github.com/risqflan/RisQFLan/wiki.

5.1 Case Study 1: Ordered Attacks

This section shows that the RisQFLan DSL can be used to model features from *enhanced attack trees*, an extension of basic attack trees proposed in [11], and that RisQFLan hence complements the analysis capabilities of [11] with statistical model checking 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.

5.1.1 Ordered Attacks to "Bypassing 802.1x". As illustrative example we use one of the case studies modeled and evaluated in [11], namely an enhanced attack tree modeling complex (ordered) attacks on wireless lans using protocol IEEE 802.11. Figure 6, reproduced from [11], 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 and, ultimately, attacker actions.

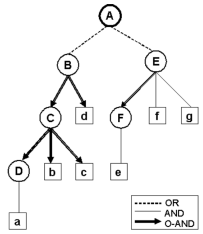

Figure 6: Enhanced attack tree for "Bypassing 802.1x" [11]

```
begin attack diagram
 A -OR-> { B, E }
 B -OAND-> [ C, d ]
 C -OAND-> [ D, b, c ]
 D -AND-> { a }
 E -OAND-> [ F, fg ]
 fg -AND-> { f, g }
 F -> { e }
end attack diagram
```

Code 16: Attack tree of Fig. 6 in RisQFLan

5.1.2 Specifying Ordered Attacks in RisQFLan. Code 16 shows how the enhanced attack tree of Fig. 6 is modeled in RisQFLan. It is worth observing how the ordering relation is modeled. The original model in Fig. 6 prescribes that: (i) to achieve attack B, attack goal C must be achieved before d (cf. Line 3 in Code 16); (ii) to achieve attack C, attack goal D must be achieved before b, which itself must be achieved before c (cf. Line 4 in Code 16); and (iii) to achieve attack E, attack goal F must be achieved before f and g (cf. Lines 6-7 in Code 16). Note that in the RisQFLan specification, the auxiliary node fg is used to group the conjunction of actions f and g.

5.1.3 Complementing the Analysis of [11] with RisQFLan. The main analysis feature of the approach in [11] consists of inspecting activity logs to recognise potential attacks as per the specified enhanced attack trees. With RisQFLan one can complement such analyses with statistical verification on the average behaviour of specific attacker profiles. To illustrate this we modeled four attacker profiles:

- `Best`: an attacker that knows one of the optimal orders of actions to perform to achieve the main attack goal;
- `AverageA`: an attacker that randomly tries attack goals and actions until the main attack goal is achieved or a wrong order has led to failure;
- `AverageB`: like `AverageA` but can undo goals/actions to backtrack;
- `Worst`: like `AverageA` but chooses actions with a probability inversely proportional to the order used by `Best`.

Figure 7 shows the results of the analysis and how the attacker profiles converge to different attack success probabilities. 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.

5.2 Case Study 2: Noticeability

In this section, we show that the RisQFLan DSL can be used to model features from *capabilities-based attack trees* [23], an extension of basic attack trees offered in the commercial attack tree-based risk assessment tool Secur*IT*ree [2]. This shows that RisQFLan complements the models of Secur*IT*ree with explicit dynamic attack behavior[1] and its analysis capabilities with statistical model checking of attacker profiles. We do so by illustrating how the notion

[1] Amenaza has similar plans for Secur*IT*ree v5.1 (T. Ingoldsby, personal communication, April 1, 2020).

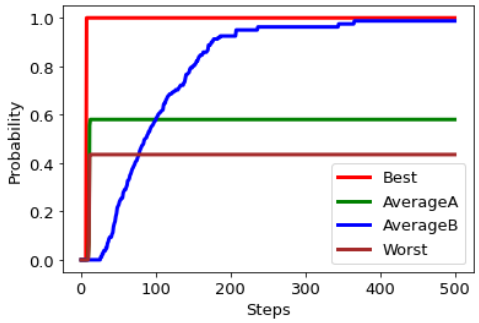

Figure 7: Statistical analysis on "Bypassing 802.1x"

of *noticeability*, one of the capability features of *capabilities-based attack trees*, can be specified in RisQFLan.

5.2.1 Noticeability Capabilities of BurgleHouse. As illustrative example we use two attack scenarios modeled and evaluated in [2], namely the Cat Burglar and Juvenile Delinquent scenarios from the BurgleHouse case study. Figure 8, reproduced from [2], depicts two capabilities-based attack trees. The idea is that a house can be burglarized by entering the house by carrying out two sub-goals: `WalkUpToHouse` and `PenetrateHouse`. The former is an attacker action, while the latter goal is further refined into sub-goals and, ultimately, attacker actions. 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 [2], namely *noticeability*. The values were kindly provided by Terry Ingoldsby of Amenaza Technologies Ltd. with a license for Secur*IT*ree v5.0.

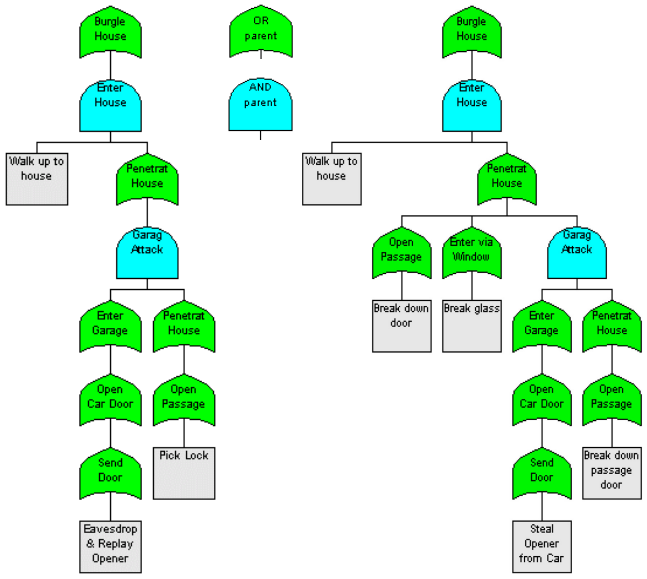

Figure 8: Capabilities-based attack trees for the scenarios Cat Burglar (left) and Juvenile Delinquent (right) [2]

5.2.2 Modeling Noticeability in RisQFLan. Code 17 and Code 18 show how the noticeability values of the Cat Burglar and Juvenile Delinquent scenarios, resp., are modeled as a `Noticeability` attribute in RisQFLan. Not surprisingly, walking up to the house is almost unnoticeable (cf. Line 2 in Code 17 and Code 18), whereas breaking a door or glass is likely to be noticed (cf. Line 3 in Code 18).

```
1 begin attributes
2  Noticeability = { WalkUpToHouse = 0.01 ,
3   EavesdropAndReplayOpenerCode = 0.05 , PickLock = 0.02 }
4 end attributes
```

Code 17: Noticeability of "Cat Burglar" specified in RisQFLan

```
1 begin attributes
2  Noticeability = { WalkUpToHouse = 0.01 ,
3   BreakDownDoor = 0.3 , BreakGlass = 0.3 ,
4   StealOpenerFromCar = 0.2 , BreakDownPassageDoor = 0.1 }
5 end attributes
```

Code 18: Noticeability of "Juvenile Delinquent" in RisQFLan

5.2.3 Complementing the Analysis of [2] with RisQFLan. One of the analysis features of Secur*IT*ree 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, Secur*IT*ree 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 4.1). However, RisQFLan can also complement such analyses with statistical 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:

`Best`: an attacker that knows an optimal, most unnoticeable order of actions to perform to achieve the main attack goal;
`AverageA`: an attacker that randomly tries attack goals and actions until the main attack goal is achieved;
`AverageB`: like `AverageA` but can undo goals/actions to backtrack;
`Worst`: like `AverageA` but chooses actions with a probability inversely proportional to `Best`.

Figures 9 and 10 show how the attacker profiles converge to different average noticeability values of the attacks.[2] 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 the figures, in both scenarios all attackers indeed 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, EnterViaWindow, GarageAttack}`). In fact, the average noticeability value of the `Best` attacker is just over 3, that of the `AverageA` attacker is close to 6, that of the `Worst` attacker is close to 7, and also in this case the repeated attack attempts of the `AverageB` attacker guarantee (s)he will be noticed.

RisQFLan thus allows to analyze the risk of getting caught for different types of behavior of a concrete Cat Burglar or Juvenile Delinquent and to estimate who runs less risk. Secur*IT*ree does not consider such explicit dynamic attack behavior in its risk analyses. However, Secur*IT*ree offers advanced analysis functionalities to estimate the risk of scenarios by combining impact of attacks and so-called capabilistic attack propensity, expressed by considering ease (e.g. cost or resources) and benefits (e.g. rewards) of attacks.

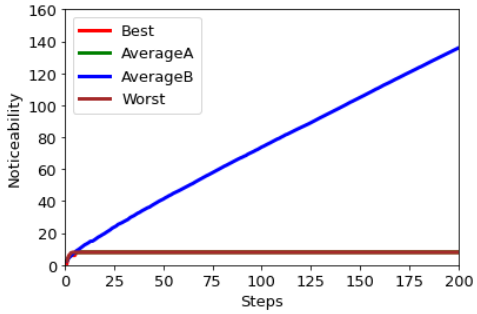

Figure 9: Statistical analysis on "Cat Burglar"

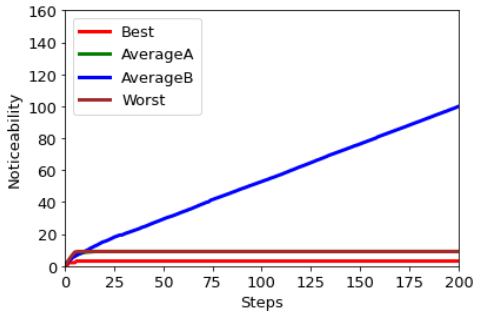

Figure 10: Statistical analysis on "Juvenile Delinquent"

5.3 Case Study 3: Countermeasures

Similar to the previous sections we focus on an influential approach to attack trees, namely *attack countermeasure trees* [43], which has inspired some of the modeling features of RisQFLan. We show how RisQFLan DSL primitives can be used to 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.

5.3.1 Countermeasures against "Resetting BGP". As illustrative example we use one of the case studies modeled and evaluated in [43], namely an attack countermeasure tree modeling defensive mechanisms against resetting attacks on the Border Gateway Protocol (BGP). Figure 11, reproduced from [11], illustrates the attack countermeasure tree modeling such a scenario. The main idea is to model

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

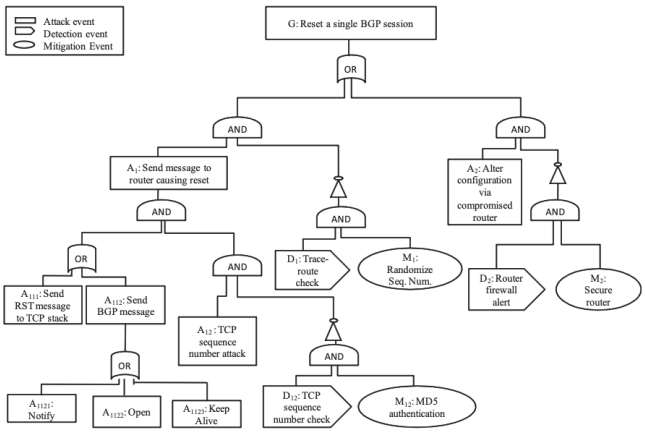

Figure 11: Countermeasure tree for "Resetting BGP" [43]

a known denial-of-service attack on the BGP: the attacker tries to reset a BGP session again and again to prevent communication.

Such an attack consists of several steps, some of which can be detected and mitigated using well-known techniques. Like TCP sequence number attacks (A12) can be detected with TCP sequence number checks (D12). A mitigation mechanism for such attacks is using MD5 authentication (M12).

5.3.2 Countermeasures in RisQFLan. Code 19 shows how the attack countermeasure tree of Fig. 11 is modeled in RisQFLan. In particular: (i) detection events D12, D1, and D2 are modeled as countermeasure nodes and the attacks A12, A1, and A2 they intend to detect, resp., are specified accordingly (cf. Line 10 of Code 19); (ii) measure events M12, M1, and DM2 are modeled as defense nodes (cf. Line 6 in Code 19) and the attacks A12, A1, and A2 they mitigate, resp., are specified in the attack effectiveness block (cf. Line 33 in Code 19); and (iii) the relation between a detection event D and a corresponding triggered mitigation event M is modeled in RisQFLan by specifying that the defense node D is a refinement of countermeasure node M (cf. Lines 22-24 in Code 19).

```
1  begin attack nodes
2   G A1 A111 A112 A1121 A1122
3   A1123 A12 A2 OR1
4  end attack nodes
5
6  begin defense nodes
7   M12 M1 M2
8  end defense nodes
9
10 begin countermeasure nodes
11  D12 = { A12 }
12  D1 = { A1 }
13  D2 = { A2 }
14 end countermeasure nodes
15
16 begin attack diagram
17  G -OR-> { A1, A2 }
18  A1 -AND-> { OR1, A12 }
19  OR1 -OR-> { A111, A112 }
20  A112 -OR->
21      { A1121, A1122, A1123 }
22  D12 -AND-> { M12 }
23  D1 -AND-> { M1 }
24  D2 -AND-> { M2 }
25 end attack diagram
26
27 begin attack detection rates
28  A1 = 0.5 ,
29  A12 = 0.5 ,
30  A2 = 0.5
31 end attack detection rates
32
33 begin defense effectiveness
34  M12(ALL, A12) = 0.5
35  M1(ALL, A1) = 0.5
36  M2(ALL, A2) = 0.5
37 end defense effectiveness
```

Code 19: Attack tree of Fig. 11 in RisQFLan

5.3.3 Complementing the Analysis of [43] with RisQFLan. The approach described in [43] includes a rich variety of analyses for attack countermeasure trees, including success probabilities, costs and impact for attack goals and defensive mechanisms. With RisQFLan one can complement such analyses with statistical verification of the average behavior of specific attacker profiles. To illustrate this we modeled three attacker profiles:

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

Figure 12 shows the results of the analyses and how the attacker profiles converge to different attack success probabilities. 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.

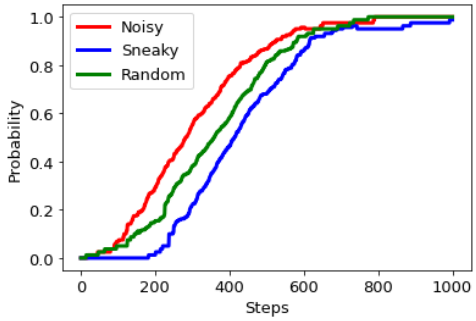

Figure 12: Statistical analysis on "Resetting a BGP session"

6 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 [21, 39], and the attack detection rates [2], ordered attacks [11], and countermeasures [43] treated in Section 5.1, Section 5.2, and Section 5.3, respectively. A recent study has 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 that illustrate each of them.

The first dimension, which was a major focus of the large-scale EU H2020 project TRESPASS [46], is that of generation approaches whose main objective is to generate attack trees from scenarios. The main difficulty is to find a compact and effective representation of the tree, knowing that structurally different attack trees can capture the same information. A representative contribution in this area concerns the ATSyRa toolset of Pinchinat et al. [42]. An original and crucial feature of this approach 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 [48], where attacks are generated from flow constraints using a SAT solver, and value-passing quality calculus is used to represent how an attacker can reach a given location. Our approach clearly is not concerned with the synthesis of attack trees. However, we observe that the high-level refactoring of ATSyRa perfectly matches the semantics of RisQFLan. Consequently, the two tools could be combined in a win-win situation in which RisQFLan would benefit from both the Eclipse plugin and the scenario generation of ATSyRa, while the latter would benefit from all analysis primitives of RisQFLan. We believe this conclusion to apply to many scenario generation tools.

The second dimension in [49] 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. [5, 49] and their references). This research trend is very similar to the one applied to feature diagrams [12], 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 [8]. It is worth noticing that the above mentioned approaches do not permit reasoning on the order of steps of the attack. Attack trees modeled in RisQFLan directly translate to Boolean functions, and thus the comparison with this line of research is outside our scope.

More recently, several researchers have suggested to extend attack tree representations with that of its environment, i.e. the attacker and the system under attack. As an example, in [17, 19], the authors not only consider the attack tree itself, but also a transition system representation of an attacker model. 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 [29]. In this context, contributions like [17] consider that defenses are fixed a priori, while the game-based approach of Aslanyan et al. [4] 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. [24]. RisQFLan follows the approach of Aslanyan et al., but uses Statistical Model Checking (SMC) [38], 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 [49] 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 or computing the Pareto optimal attack for two or more quantitative parameters [3]. However, as observed by Kordy and colleagues, minimal representations no longer exist [30, 32], 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 and colleagues [17, 19], already cited for the Boolean case. 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 [15] to compute the probability or cost of an attack. In addition, in case non-determinism is added to the attacker model, UPPAAL Stratego [14] can be used to synthesize quantitative strategies.

RisQFLan goes further than [17, 19] 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 them as well as between defenses that can moreover be added at runtime. However, RisQFLan does not offer non-determinism for the attacker. This may be needed to reason on the use of several strategies. A solution could be to combine RisQFLan's SMC engine with the Plasma Plugin for non-deterministic systems [13]. The approach by Aslanyan et al. [4] allows one to reason on causality between steps and non-deterministic attackers, but it is restricted to Boolean causalities called waves, and there is no DSL. Stoelinga and co-authors have also proposed several 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 [33]. Last but not least, compared to the three approaches mentioned above, RisQFLan is the only one that proposes a fully dedicated and maintained toolset, which is moreover open source.

7 CONCLUSION AND FUTURE WORK

We have instantiated QFLan in the domain of quantitative security risk modeling and analysis and applied the resulting tool RisQFLan to three case studies from the risk modeling and analysis domain. The generalization and subsequent instantiation of QFLan was feasible since it is open source. Also RisQFLan is open source, a distinguishing feature among the toolsets available in the domain. RisQFLan's DTMC exporting facilities moreover permit tool-chaining with probabilistic model checkers for models of sizes that do not require SMC. We believe that RisQFLan thus constitutes a significant contribution to the existing toolset for risk modelling analysis.

RisQFLan could be enriched in several directions. For example, we currently propagate the value of an attribute of a node of an attack-defense tree as the sum of the attribute's values of its descendants. In the future, We would like to offer the possibility to associate attribute-specific formulae to nodes, like in Secur*IT*ree.

It would also be interesting to synthesize the attackers with the best chance of success by omitting some of the rates from the transitions of the attack behavior model and rather derive them.

REFERENCES

[1] Gul Agha and Karl Palmskog. 2018. A Survey of Statistical Model Checking. *ACM Trans. Model. Comp. Simul.* 28, 1 (2018), 6:1–6:39. https://doi.org/10.1145/3158668

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

[3] Zaruhi Aslanyan and Flemming Nielson. 2015. Pareto Efficient Solutions of Attack-Defence Trees. In *Proceedings of the 4th International Conference on Principles of Security and Trust (POST'15) (Lecture Notes in Computer Science)*, Riccardo Focardi and Andrew C. Myers (Eds.), Vol. 9036. Springer, 95–114. https://doi.org/10.1007/978-3-662-46666-7_6

[4] Zaruhi Aslanyan, Flemming Nielson, and David Parker. 2016. Quantitative Verification and Synthesis of Attack-Defence Scenarios. In *Proceedings of the IEEE 29th Computer Security Foundations Symposium (CSF'16)*. IEEE, 105–119. https://doi.org/10.1109/CSF.2016.15

[5] Maxime Audinot, Sophie Pinchinat, and Barbara Kordy. 2017. Is My Attack Tree Correct?. In *Proceedings 22nd European Symposium on Research in Computer Security (ESORICS'17) (Lecture Notes in Computer Science)*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.), Vol. 10492. Springer, 83–102. https://doi.org/10.1007/978-3-319-66402-6_7

[6] Maurice H. ter Beek and Axel Legay. 2019. Quantitative variability modelling and analysis. *Int. J. Softw. Tools Technol. Transf.* 21, 6 (2019), 607–612. https://doi.org/10.1007/s10009-019-00535-1

[7] Maurice H. ter Beek, Axel Legay, Alberto Lluch Lafuente, and Andrea Vandin. 2020. A framework for quantitative modeling and analysis of highly (re)configurable systems. *IEEE Trans. Softw. Eng.* 46, 3 (2020), 321–345. https://doi.org/10.1109/TSE.2018.2853726

[8] Maurice H. ter Beek, Axel Legay, Alberto Lluch Lafuente, and Andrea Vandin. 2020. Variability meets Security. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-intensive Systems (VaMoS'20)*. ACM, 11:1–11:9. https://doi.org/10.1145/3377024.3377041

[9] Marco Bernardo, Rocco De Nicola, and Jane Hillston (Eds.). 2016. *Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems*. Lecture Notes in Computer Science, Vol. 9700. Springer. https://doi.org/10.1007/978-3-319-34096-8

[10] Marius Bozga, Alexandre David, Arnd Hartmanns, Holger Hermanns, Kim G. Larsen, Axel Legay, and Jan Tretmans. 2012. 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, 370–375. https://doi.org/10.1109/DATE.2012.6176499

[11] Seyit Ahmet Çamtepe and Bülent Yener. 2007. Modeling and detection of complex attacks. In *Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks (SecureComm'07)*. IEEE, 234–243. https://doi.org/10.1109/SECCOM.2007.4550338

[12] Krzysztof Czarnecki, Steven She, and Andrzej Wąsowski. 2008. Sample Spaces and Feature Models: There and Back Again. In *Proceedings of the 12th International Software Product Lines Conference (SPLC'08)*. IEEE, 22–31. https://doi.org/10.1109/SPLC.2008.49

[13] Pedro D'Argenio, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. 2015. Smart sampling for lightweight verification of Markov decision processes. *Int. J. Softw. Tools Technol. Transf.* 17, 4 (2015), 469–484. https://doi.org/10.1007/s10009-015-0383-0

[14] Alexandre David, Peter G. Jensen, Kim G. Larsen, Marius Mikučionis, and Jakob H. Taankvist. 2015. Uppaal Stratego. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15) (Lecture Notes in Computer Science)*, Christel Baier and Cesare Tinelli (Eds.), Vol. 9035. Springer, 206–211. https://doi.org/10.1007/978-3-662-46681-0_16

[15] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny B. Poulsen. 2015. Uppaal SMC tutorial. *Int. J. Softw. Tools Technol. Transf.* 17, 4 (2015), 397–415. https://doi.org/10.1007/s10009-014-0361-y

[16] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. 2017. A Storm is Coming: A Modern Probabilistic Model Checker. In *CAV (Lecture Notes in Computer Science)*, Rupak Majumdar and Viktor Kunčak (Eds.), Vol. 10427. Springer, 592–600. https://doi.org/10.1007/978-3-319-63390-9_31

[17] Olga Gadyatskaya, René R. Hansen, Kim G. Larsen, Axel Legay, Mads C. Olesen, and Danny B. Poulsen. 2016. Modelling Attack-defense Trees Using Timed Automata. In *Proceedings of the 14th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'16) (Lecture Notes in Computer Science)*, Martin Fränzle and Nicolas Markey (Eds.), Vol. 9884. Springer, 35–50. https://doi.org/10.1007/978-3-319-44878-7_3

[18] Ernst M. Hahn, Arnd Hartmanns, Christian Hensel, Michaela Klauck, Joachim Klein, Jan Kretínský, David Parker, Tim Quatmann, Enno Ruijters, and Marcel Steinmetz. 2019. The 2019 Comparison of Tools for the Analysis of Quantitative Formal Models. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: TOOLympics (TACAS'19) (Lecture Notes in Computer Science)*, Dirk Beyer, Marieke Huisman, Fabrice Kordon, and Bernhard Steffen (Eds.), Vol. 11429. Springer, 69–92. https://doi.org/10.1007/978-3-030-17502-3_5

[19] René R. Hansen, Peter G. Jensen, Kim G. Larsen, Axel Legay, and Danny B. Poulsen. 2017. Quantitative Evaluation of Attack Defense Trees Using Stochastic Timed Automata. In *Proceedings of the 4th International Workshop on Graphical Models for Security (GraMSec'17) (Lecture Notes in Computer Science)*, Peng Liu, Sjouke Mauw, and Ketil Stølen (Eds.), Vol. 10744. Springer, 75–90. https://doi.org/10.1007/978-3-319-74860-3_5

[20] Arnd Hartmanns and Holger Hermanns. 2015. In the quantitative automata zoo. *Sci. Comput. Program.* 112 (2015), 3–23. https://doi.org/10.1016/j.scico.2015.08.009

[21] Holger Hermanns, Julia Krämer, Jan Krcál, and Mariëlle Stoelinga. 2016. The Value of Attack-Defence Diagrams. In *Proceedings of the 5th International Conference on Principles of Security and Trust (POST'16) (Lecture Notes in Computer Science)*, Frank Piessens and Luca Viganò (Eds.), Vol. 9635. Springer, 163–185. https://doi.org/10.1007/978-3-662-49635-0_9

[22] Jin B. Hong, Dong Seong Kim, Chun-Jen Chung, and Dijiang Huang. 2017. A survey on the usability and practical applications of Graphical Security Models. *Comput. Sci. Rev.* 26 (2017), 1–16. https://doi.org/10.1016/j.cosrev.2017.09.001

[23] Terrance R. Ingoldsby. 2013. *Attack Tree-based Threat Risk Analysis*. Technical Report. Amenaza Technologies Limited. https://www.amenaza.com/downloads/docs/AttackTreeThreatRiskAnalysis.pdf

[24] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Rolando Trujillo-Rasua. 2015. Attack Trees with Sequential Conjunction. In *Proceedings of the 30th IFIP TC 11 International Conference on ICT Systems Security and Privacy Protection (SEC'15) (IFIP Advances in Information and Communication Technology)*, Hannes Federrath and Dieter Gollmann (Eds.), Vol. 455. Springer, 339–353. https://doi.org/10.1007/978-3-319-18467-8_23

[25] Joost-Pieter Katoen and Kim G. Larsen. 2012. Quantitative Modelling and Analysis. In *Proceedings of the 5th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Applications and Case Studies (ISoLA'12) (Lecture Notes in Computer Science)*, Tiziana Margaria and Bernhard Steffen (Eds.), Vol. 7610. Springer, 290–292. https://doi.org/10.1007/978-3-642-34032-1_27

[26] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. 2013. ADTool: Security Analysis with Attack–Defense Trees. In *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST'13) (Lecture Notes in Computer Science)*, Kaustubh Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro R. D'Argenio (Eds.), Vol. 8054. Springer, 173–176. https://doi.org/10.1007/978-3-642-40196-1_15

[27] Barbara Kordy, Piotr Kordy, and Yoann van den Boom. 2016. SPTool – Equivalence Checker for SAND Attack Trees. In *Proceedings of the 11th International Conference on Risks and Security of Internet and Systems (CRiSIS'16) (Lecture Notes in Computer Science)*, Frédéric Cuppens, Nora Cuppens, Jean-Louis Lanet, and Axel Legay (Eds.), Vol. 10158. Springer, 105–113. https://doi.org/10.1007/978-3-319-54876-0_8

[28] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. 2011. Foundations of Attack-Defense Trees. In *Proceedings of the 7th International Workshop on Formal Aspects in Security and Trust (FAST'10) (Lecture Notes in Computer Science)*, Pierpaolo Degano, Sandro Etalle, and Joshua Guttman (Eds.), Vol. 6561. Springer, 80–95. https://doi.org/10.1007/978-3-642-19751-2_6

[29] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. 2014. Attack-defense trees. *J. Log. Comput.* 24, 1 (2014), 55–87. https://doi.org/10.1093/logcom/exs029

[30] Barbara Kordy, Sjouke Mauw, and Patrick Schweitzer. 2012. Quantitative Questions on Attack-Defense Trees. In *Proceedings of the 15th International Conference on Information Security and Cryptology (ICISC'12) (Lecture Notes in Computer Science)*, Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon (Eds.), Vol. 7839. Springer, 49–64. https://doi.org/10.1007/978-3-642-37682-5_5

[31] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. 2014. DAG-based attack and defense modeling: Don't miss the forest for the attack trees. *Comput. Sci. Rev.* 13–14 (2014), 1–38. https://doi.org/10.1016/j.cosrev.2014.07.001

[32] Barbara Kordy, Marc Pouly, and Patrick Schweitzer. 2016. Probabilistic reasoning with graphical security models. *Inf. Sci.* 342 (2016), 111–131. https://doi.org/10.1016/j.ins.2016.01.010

[33] Rajesh Kumar, Arend Rensink, and Mariëlle Stoelinga. 2018. LOCKS: a property specification language for security goals. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC'18)*. ACM, 1907–1915. https://doi.org/10.1145/3167132.3167336

[34] Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. 2015. Quantitative Attack Tree Analysis via Priced Timed Automata. In *Proceedings of the 13th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'15) (Lecture Notes in Computer Science)*, Sriram Sankaranarayanan and Enrico Vicario (Eds.), Vol. 9268. Springer, 156–171. https://doi.org/10.1007/978-3-319-22975-1_11

[35] Rajesh Kumar, Stefano Schivo, Enno Ruijters, Buğra M. Yildiz, David Huistra, Jacco Brandt, Arend Rensink, and Mariëlle Stoelinga. 2018. Effective Analysis of Attack Trees: A Model-Driven Approach. In *Proceedings of the 21st International Conference on Fundamental Approaches to Software Engineering (FASE'18) (Lecture Notes in Computer Science)*, Alessandra Russo and Andy Schürr (Eds.), Vol. 10802. Springer, 56–73. https://doi.org/10.1007/978-3-319-89363-1_4

[36] Rajesh Kumar and Mariëlle Stoelinga. 2017. 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, 25–32. https://doi.org/10.1109/HASE.2017.12

[37] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV (Lecture Notes in Computer Science)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.), Vol. 6806. Springer, 585–591. https://doi.org/10.1007/978-3-642-22110-1_47

[38] Axel Legay, Anna Lukina, Louis-Marie Traonouez, Junxing Yang, Scott A. Smolka, and Radu Grosu. 2019. Statistical Model Checking. In *Computing and Software Science: State of the Art and Perspectives*, Bernhard Steffen and Gerhard J. Woeginger (Eds.). Lecture Notes in Computer Science, Vol. 10000. Springer, 478–504. https://doi.org/10.1007/978-3-319-91908-9_23

[39] Aleksandr Lenin, Jan Willemson, and Dyan P. Sari. 2014. Attacker Profiling in Quantitative Security Assessment Based on Attack Trees. In *Proceedings of the 19th Nordic Conference on Secure IT Systems (NordSec'14) (Lecture Notes in Computer Science)*, Karin Bernsmed and Simone Fischer-Hübner (Eds.), Vol. 8788. Springer, 199–212. https://doi.org/10.1007/978-3-319-11599-3_12

[40] Wen-ping Lv and Wei-min Li. 2011. 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, 480–483. https://doi.org/10.1109/MINES.2011.94

[41] Sjouke Mauw and Martijn Oostdijk. 2005. Foundations of Attack Trees. In *Proceedings of the 8th International Conference on Information Security and Cryptology (ICISC'05) (Lecture Notes in Computer Science)*, Dongho Won and Seungjoo Kim (Eds.), Vol. 3935. Springer, 186–198. https://doi.org/10.1007/11734727_17

[42] Sophie Pinchinat, Mathieu Acher, and Didier Vojtisek. 2015. ATSyRa: An Integrated Environment for Synthesizing Attack Trees - (Tool Paper). In *Proceedings of the 2nd International Workshop on Graphical Models for Security (GraMSec'15) (Lecture Notes in Computer Science)*, Sjouke Mauw, Barbara Kordy, and Sushil Jajodia (Eds.), Vol. 9390. Springer, 97–101. https://doi.org/10.1007/978-3-319-29968-6_7

[43] Arpan Roy, Dong S. Kim, and Kishor S. Trivedi. 2012. Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. *Secur. Commun. Netw.* 5, 8 (2012), 929–943. https://doi.org/10.1002/sec.299

[44] Bruce Schneier. 1999. Attack Trees. *Dr. Dobb's Journal* (1999). https://www.schneier.com/academic/archives/1999/12/attack_trees.html

[45] Stefano Sebastio and Andrea Vandin. 2013. MultiVeStA: Statistical Model Checking for Discrete Event Simulators. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools (ValueTools'13)*. ACM, 310–315. https://doi.org/10.4108/icst.valuetools.2013.254377

[46] TRESSPASS [n.d.]. H2020 project on robusT Risk basEd Screening and alert System for PASSengers and luggage. https://www.tresspass.eu/The-project

[47] Andrea Vandin, Maurice H. ter Beek, Axel Legay, and Alberto Lluch Lafuente. 2018. QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems. In *Proceedings of the 22nd International Symposium on Formal Methods (FM'18) (Lecture Notes in Computer Science)*, Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik de Vink (Eds.), Vol. 10951. Springer, 329–337. https://doi.org/10.1007/978-3-319-95582-7_19

[48] Roberto Vigo, Flemming Nielson, and Hanne R. Nielson. 2014. Automated Generation of Attack Trees. In *Proceedings of the 27th IEEE Computer Security Foundations Symposium (CSF'14)*. IEEE, 337–350. https://doi.org/10.1109/CSF.2014.31

[49] Wojciech Wideł, Maxime Audinot, Barbara Fila, and Sophie Pinchinat. 2019. Beyond 2014: Formal Methods for Attack Tree–Based Security Modeling. *ACM Comput. Surv.* 52, 4 (2019), 75:1–75:36. https://doi.org/10.1145/3331524