LOCKS: a property specification language for security goals

Preprint · March 2018
DOI: 10.1145/3167132.3167336

3 authors:

Rajesh Kumar
University of Twente
11 PUBLICATIONS **45** CITATIONS
SEE PROFILE

Arend Rensink
University of Twente
228 PUBLICATIONS **2,720** CITATIONS
SEE PROFILE

Mariëlle Stoelinga
University of Twente
92 PUBLICATIONS **2,120** CITATIONS
SEE PROFILE

LOCKS: a property specification language for security goals

Rajesh Kumar
Formal Methods and Tools
University of Twente
The Netherlands
r.kumar@utwente.nl

Arend Rensink
Formal Methods and Tools
University of Twente
The Netherlands
arend.rensink@utwente.nl

Mariëlle Stoelinga
Formal Methods and Tools
University of Twente
The Netherlands
m.i.a.stoelinga@utwente.nl

ABSTRACT

We introduce a formal specification language LOCKS, that allow security practitioners to express as well as compose security goals in a convenient manner. LOCKS supports the specification of the most common security properties over generic attributes, both for qualitative and quantitative goals. To make our language independent of a specific security framework, we evaluate LOCKS over a generic attack model, namely the structural attack model (SAM), which over-arches the most prominent graphical threat models. Furthermore, we equip our language with a concise grammar, type rules and denotational semantics, thus laying the foundations of an automated tool. We take a number of informal security goals from the literature and show how they can be formally expressed in our language.

CCS CONCEPTS

• **Security and privacy** → **Security requirements**; **Formal security models**;

KEYWORDS

Enterprise security, Quantitative security goals, Property specification language, Multi-objective query language, Threat models, Denotational semantics

ACM Reference format:
Rajesh Kumar, Arend Rensink, and Mariëlle Stoelinga. 2018. LOCKS: a property specification language for security goals. In *Proceedings of SAC 2018: Symposium on Applied Computing , Pau, France, April 9–13, 2018 (SAC 2018),* 9 pages.
https://doi.org/10.1145/3167132.3167336

1 INTRODUCTION

Modern day enterprises are frequently challenged with how to estimate security risks, what to protect and how much to invest in securing their organisation. Furthermore, as highlighted in [15, 27], elicitation of security goals remains cumbersome because of:

- multiple stakeholders with subjective interpretation of the security goals;
- inconsistent terminology, jargon of technical and operational details;
- absence of a security framework that integrates business/organizational goals and security needs.

Recognizing the subjectiveness embedded into the elicitation of security goals, several standards such as Operationally Critical Threat, Asset and Vulnerability Evaluation (OCTAVE) [2] Control Objectives for Information and Related Technologies (COBIT) [14], Security Quality Requirements Engineering (SQUARE) [27] have been developed, that provide generic guidelines in form of dos and donts list, work-flow diagrams and compliance regulations. Though these standards give a telescopic view of security best practices, they do not translate into security goals.

Additionally, a plethora of threat modelling frameworks and analysis methodologies have been developed, such as ADVISE [24], attack trees [18, 19, 30], privilege graphs [10] and attack graphs [13, 16, 31]. These formalisms rely on precise models to quantify the riskful scenarios; however, the security properties over all these aforementioned models are usually expressed ad hoc, i.e., either stated informally or encoded unnaturally in some generic property specification language such as of temporal logics.

To the best of our knowledge, a formal property specification language for security goals is lacking. Such a specification language is pivotal to:

- phrase security requirements, i.e., a set of unambiguous and concise properties that characterize a secure system;
- validate threat models, i.e., to ascertain whether the threat model complies with the stakeholder requirements;
- verify threat models, i.e., to perform a structured qualitative and quantitative risk assessment of the threat models;
- compare threat models, i.e., to reason about different design choices that lead to a secure system.

One of the main challenges here is to embrace all different attributes (cost, time, probability, damage, etc) in framing both qualitative and quantitative goals.

SAC 2018, April 9–13, 2018, Pau, France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167336

Our contributions. We introduce a formal specification language, LOCKS, that allows a security practitioner to directly express security goals. A key design goal of LOCKS is to express security goals over many different security frameworks and threat models. Therefore, we make very few assumptions over the security framework and express LOCKS over a *structural attack model* (SAM, Section 4). A SAM is based on partially ordered sets of attacks, which tells for each successful attack which steps have to be carried out and in which order; for instance, to infect a computer, we must first get a virus file on a system, and then execute the file. To develop LOCKS, we made a literature survey of the popular graphical threat models and filter out the underlying security goal described in these papers. These security goals are then represented formally (Section 3). Based on the recurring constructs occurring in the formalized security goals, we proceed further to develop the LOCKS grammar (in Figure 4). We then describe the static semantics (in Section 5.1) and the denotational semantics (in Section 5.2) to make the terms of LOCKS well-formed and precise. Summarizing, the main contributions of this paper are:

- A domain specific language LOCKS that allows a security practitioner to express as well as compose security goals in a declarative manner. We claim that LOCKS offers the following benefits:
  - It is generic, i.e., not tied to any security architectural framework;
  - It is a formal language with precise syntax and semantics, laying the foundation of a computer language;
  - It encompasses both qualitative and quantitative security goals.
- The structural attack model SAM, encompassing most of the aforementioned graphical threat models and is used as the semantic model for LOCKS.

2 RELATED WORK

There exists a plethora of work on security models, frameworks and policies. For example, generic formal specification languages such as of Z notation, UML+OCL, etc, have been used to model access and authorization policies [8, 25]. Other formal specification languages in security are tailored to specific settings, for example, in authentication [1], information flow [6], distributed systems [32], etc.

A lot of literature is devoted to the graphical threat models, an overview of which (exploiting directed acyclic structures) can be found in [17]. To illustrate their threat modelling framework, authors usually take one or two specific metrics such as probability of a successful attack [12, 18], minimum cost of attack [21], mean effort [10], mean time to security failure [10], attack resistance [34], and trade-offs between the security attributes [4, 20, 21]. In contrast, our paper formalizes all these security metrics over a generic attack model, namely the structural attack model SAM.

Our work distinguishes itself from all aforementioned works in respect that we instead of proposing a new security model, propose a property specification language, that allows to capture many security goals.

Close to our proposed language, there exists a few generic property specification languages such as of the Uppaal query language [7], the PRISM property specification language [23] that exists as part of a particular model checker and is essentially based on propositional temporal logic. Few authors such as in [10, 12, 21, 22, 31], use these generic property specification language to encode the security goals. Attempts to express security properties this way are largely ad-hoc: the encoding is indirect or unnatural. For example, in [21, 22], the authors encode the security goal of obtaining the maximum damage inflicted by an attack within minimum duration as a boolean formula and the optimum value as the bound at which the security property flips its boolean value while constricting the bound. Instead, in our framework the same properties can be framed using straightforward constructs readily available in our proposed language.

In [18], authors classify attributes into attribute domains and choose appropriate semantics (propositional/ multiset/ equational) to perform computations of attributes on attack-defense trees. Their work is limited to computation of only single attribute at a time such as of cost or duration. Our framework encompasses theirs, as we show by taking some of the security goals from the previous paper and formalize them in LOCKS. Furthermore, our proposed language provides the flexibility to frame security goals using user defined attributes.

3 TYPICAL SECURITY GOALS

Below we provide typical security goals taken from the literature. These security goals are stated in natural language and use threat models such as of attack trees [9, 21, 26, 30], attack defense tree [4, 5, 18], attack graphs [28, 33] and attack defense diagrams [12] to model the attacks on the asset(s). Note that we used the aforementioned threat formalisms to illustrate our work, however many other formalisms that exploit the directed acyclic structures, can be translated into SAM (in Section 4), hence, LOCKS can be used over all these formalisms. We take one or more security goals from the aforementioned papers and formally express them in our proposed language LOCKS.

In [30], Schneier et al. in his seminal paper introduces the formalism of attack trees to model the attack scenarios. An attack tree (see Figure 1) starts with a security threat, modelled as the root of the tree, representing the attacker's top level goal. This root is recursively refined into attacker's sub-goals using logical gates, until one arrives at the leaves (when no further refinement is possible or is not required), modelling the attacker atomic actions. Classically, an attack tree uses `AND` and `OR` gates to describe the conjunctive and disjunctive composition of their child nodes. That is, to succeed in an `AND` gate, the attacker has to succeed in all of its child nodes, whereas the `OR` gate requires the attacker to execute at least one child node successfully. Further extensions of attack trees with defense steps have been proposed resulting in attack-defense trees [11, 18] and with complex gates (`SAND`)

in [3] to model the temporal dependency between the child nodes. A model-driven engineering approach to analyse attack trees by using the attack tree and Uppaal meta-models is proposed in [29].

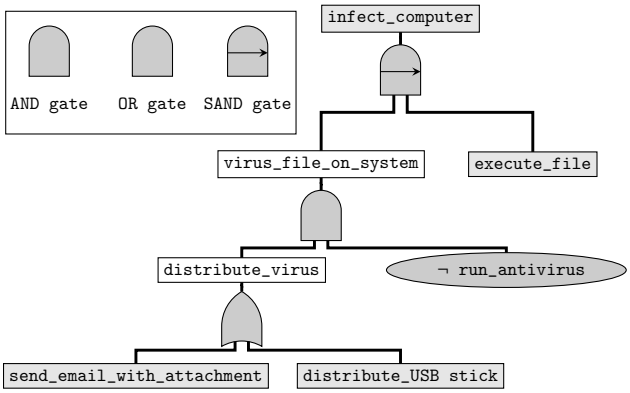

Figure 1: Attack- defense tree modelling the infection of a computer system with a computer virus

In Schneier et al., the leaves of the tree are decorated with single boolean attributes such as of possible/ impossible, easy/ difficult, expensive/ inexpensive, intrusive/ non-intrusive, legal/ illegal, special equipment required/ no special equipment. A typical security goal considered in the above paper is to ask for the attacks satisfying certain predicates, for example, to obtain the attacks which do not require special equipments. We connote this security goal as: the set of attacks, where each attack step in the attack do not require special equipment for its execution (where an attack is basically a sequence of atomic attack steps, some of which can be executed in parallel, and that leads to the top node of the attack tree). This security goal can be written in LOCKS as:

$$\{\mathtt{at} \in \mathtt{A} \mid \forall \mathtt{x} \in \mathtt{at} \setminus \mathtt{DS} : (\neg \mathtt{x.skill\_required})\}$$

Here, `at` is an attack that is contained within the universe of attacks `A` and that satisfy the constraint that each attack step `x` contained in attack `at` do not require special equipments (considering only attack step means neglecting all the defense steps, if there are any, contained in `at` and that belong to the universe of defense steps `DS`). Note that one can easily formulate variants of the above security goal by substituting the label of "requiring special equipments" with other boolean attributes.

In [26], Mauw et al. formalizes attack trees into attack suites and formally defines the criteria over these suites to perform the computation of attributes (cost, probability, damage, etc). In [12, 18], authors enrich the attack modelling framework with defenses, resulting in attack-defense tree/ attack-defense diagrams and use similar example security goals described in previous works. A typical security goal considered in the aforementioned papers is to ask for the feasibility of an attack under the resource constraints (budget, time, etc) of an attacker. Example of such security goal is to obtain attacks that can be executed in less than 1000 US$. This security goal can be written in LOCKS as:

$$\mathtt{Tot\_Cost(at)} = \textstyle\sum\{\mathtt{s.Cost} \mid \mathtt{s} \in \mathtt{at} \setminus \mathtt{DS}\} \ ;$$
$$\exists \mathtt{ak} \in \mathtt{A} : (\mathtt{Tot\_Cost(ak)} < 1000)$$

The security goal is given by first defining a function `Tot_Cost` that maps an attack `at` to a value obtained by summing the cost of each attack step `s` ∈ `at`. This function is then used to check whether an attack `ak` exists that is contained in the universe of all attacks `A` and whose cumulative cost is less than 1000 US$.

In [4, 9, 21], authors propose a multi parameter optimization framework for the annotated attack(-defense) trees. Here, the authors decorate the leaves of the attack(-defense) tree with multiple attributes to answer a wide range of questions such as of the cheapest low-risk attack, most likely nonintrusive attack, etc. A typical security goal here is to obtain the set of attacks that incurs the minimum cost and has the highest probability of success. Note that this security goal is ambiguous, as one can first obtain either the sets of attacks that has the highest probability of success and distill it further to obtain the set of attacks that also incur the minimum cost, or vice versa. This security goal can be written in LOCKS as:

$$\begin{aligned}
\mathtt{Tot_Prob}_{\mathtt{succ}}\mathtt{(at)} = \ & \textstyle\prod\{\mathtt{as.Prob}_{\mathtt{succ}} \mid \mathtt{as} \in \mathtt{at} \cap \mathtt{AS}\} \\
& \times \textstyle\prod\{1 - \mathtt{ds.Prob}_{\mathtt{succ}} \mid \mathtt{ds} \in \mathtt{at} \cap \mathtt{DS}\} \ ; \\
\mathtt{Tot_Cost(at)} = \ & \textstyle\sum\{\mathtt{s.Cost} \mid \mathtt{s} \in \mathtt{at} \setminus \mathtt{DS}\} \ ; \\
\mathtt{ischeapest(ak)} = \ & \forall \mathtt{oat} \in \mathtt{A} : (\mathtt{Tot_Cost(ak)} <= \mathtt{Tot_Cost(oat)} \\
& \wedge \mathtt{Tot_Prob}_{\mathtt{succ}}\mathtt{(ak)} >= \mathtt{Tot_Prob}_{\mathtt{succ}}\mathtt{(oat)}) \ ; \\
\{\mathtt{atk} \in \mathtt{A} \mid & \mathtt{ischeapest(atk)}\}
\end{aligned}$$

The security goal is given by first defining the functions $\mathtt{Tot\_Prob}_{\mathtt{succ}}$ and `Tot_Cost`, that map an attack `at` to a value signifying the probability and the cumulative cost of an attack respectively. Note the probability of attack is given as the product of the probability of success of attack steps, $\mathtt{as.Prob}_{\mathtt{succ}}$, and the inverse probability of success of defense steps, $1 - \mathtt{ds.Prob}_{\mathtt{succ}}$. Subsequently, we define a predicate `ischeapest` to verify whether an attack `ak` is cheapest and has the highest probability of success using the functions `Tot_Cost` and $\mathtt{Tot\_Prob}_{\mathtt{succ}}$ defined earlier. Note that one can ask for similar security goals such as of obtaining the maximum damage with minimum time, obtaining the maximum cost with minimum duration.

In [33], authors use the formalism of attack graphs to model the attacks on a networked system. Here, an attack consists of chaining the initial conditions (preconditions that needs to be satisfied in order to launch an attack) and the exploits (vulnerability existing between two hosts) leading to the node(s) representing the asset(s). A typical security goal considered in the paper is to obtain: Which initial conditions when disabled, ensures a secure system? In our framework, we connote the "initial conditions" as the atomic steps that initiate each attack. This security goal can be written in LOCKS as:

$$\{\mathtt{v} \in \mathtt{at} \mid \mathtt{at} \in \mathtt{A} \wedge \neg \exists \mathtt{w} \in \mathtt{at} : (\mathtt{w} \sqsubset_{\mathtt{at}} \mathtt{v})\}$$

Note to answer the above security goal we need the notion of causality between the steps. Hence, we define a partial ordering relationship, given by $\mathtt{s} \sqsubset_{\mathtt{at}} \mathtt{s'}$. It indicates that step s should come before step s′ in an attack at. If s and s′ are unrelated (i.e., neither $\mathtt{s} \sqsubset_{\mathtt{at}} \mathtt{s'}$ nor $\mathtt{s'} \sqsubset_{\mathtt{at}} \mathtt{s}$), then these steps can be executed in any order, even in parallel. The security goal is then given by collecting all the steps v that initiate each attack at, i.e., there is no step w that precedes the step v in an attack $\mathtt{at} \in \mathtt{A}$ (recall that A is the universe of all attacks).

In [19], authors propose that a well formed security question, stated in natural language, can be split into following parts: modality, notion, owner and execution style. A typical question asked in this paper is to obtain the minimal duration of the attack, when all the actions are executed one after the another. In this question, the *modality*, i.e., the characteristic of the attack/ defense value, is minimizing the attack value min, the *notion* is the attribute name duration, the *owner* is the player for whom the security goal is formulated, in this case the attacker and the execution style is sequential. This security goal can be written in LOCKS as:

$$\mathtt{Tot\_Duration(at)} = \textstyle\sum\{\mathtt{s.Duration} \mid \mathtt{s} \in \mathtt{at} \setminus \mathtt{DS}\} \ ;$$
$$\mathtt{Min}\{\mathtt{Tot\_Duration(ak)} \mid \mathtt{ak} \in \mathtt{A}\}$$

The security goal is given by first defining a function Tot_Duration (similar to Tot_Cost seen above), that sums the duration of the atomic attack steps in an attack at, which is then used to obtain the minimum duration among all successful attacks. Similarly, one can ask for other security goals such as of most cheapest/ most damaging/ most likely attacks.

Another variant of this security goal considering a different execution style (parallel execution of attack steps) is to ask: What is the minimum duration of the attack, when all the actions are executed simultaneously? This security goal can be written in LOCKS as:

$$\mathtt{Tot\_Duration(sas)} = \textstyle\sum\{\mathtt{s.Duration} \mid \mathtt{s} \in \mathtt{sas} \setminus \mathtt{DS}\} \ ;$$
$$\mathtt{Max}\{\mathtt{Tot\_Duration(sas)} \mid \mathtt{sas} \subseteq \mathtt{at} \wedge$$
$$\forall \mathtt{a}, \mathtt{b} \in \mathtt{sas} : (\mathtt{a} \sqsubset_{\mathtt{at}} \mathtt{b} \vee \mathtt{b} \sqsubset_{\mathtt{at}} \mathtt{a}) \wedge \mathtt{at} \in \mathtt{A}\}$$

This first defines a function Tot_Duration (defined previously) and then obtaining the maximum value among all the subsets of attack steps that are considered to be total ordered.

In [28], authors use attack graphs to model attacks on a networked system. A typical security goal here is to check whether there exists a minimal set of initial conditions that ensures network safety. We interpret the initial conditions as the defense steps in our framework. This security goal can be written in LOCKS as:

$$\mathtt{makes\_safe(sas)} = \neg\exists \mathtt{at} \in \mathtt{A} : (\mathtt{sas} \subseteq \mathtt{at} \wedge \mathtt{sas} \subseteq \mathtt{DS}) \ ;$$
$$\exists \mathtt{sds} \in \mathtt{A} : (\mathtt{makes\_safe(sds)} \wedge \neg\exists \mathtt{ss} \subset \mathtt{sds} : \mathtt{makes\_safe(ss)})$$

This first defines a function make_safe that checks whether the set of defense steps sas ensures network safety, i.e., there is no attack at, where the set of defense steps $\mathtt{sas} \subseteq \mathtt{at}$. Subsequently, make_safe is used as a predicate to verify whether the set of defense steps sds is minimal, i.e., there is no proper subset $\mathtt{ss} \subset \mathtt{sds}$ that also ensures network safety.

Another typical security goal asked over attack graphs is to count the total number of vulnerabilities in the shortest attack path [13]. In our framework, we connote vulnerability as an atomic step. This security goal can be written in LOCKS as:

$$\mathtt{Min}\,\{\mathtt{card}\{\mathtt{at}\} \mid \mathtt{at} \in \mathtt{A}\}$$

This first counts the number of vulnerabilities (steps) in an attack at and subsequently finds the smallest value among the set of values. Note that similar to the above security goal, one can also formulate security goal that ask for the total number of attack scenarios.

From the discussions above, we conclude that:

- Most threat models as described above, base itself on the representation of attack scenarios using few common concepts, such as of directed acyclic graphs, cause-consequence relationship, disabling the defense steps or the safeguards, chaining of attacker actions/ vulnerabilities leading to one or more asset(s). Typically these formalism takes the input of one or more attributes. Hence, in Section 4, we propose a generic attack model, the structural attack model SAM, that encompasses all these formalisms and represent the attack scenarios with partially ordered sets of successful attacks.
- Most security goals as described above, base itself on few common constructs such as of obtaining the numeric value(s) or the boolean value(s) by performing different arithmetic and logical operations on the attributes (depending on type of attributes and modality of interest like counting the number of occurrence/ maximum attribute value/ cumulative attribute value/ expected attribute value/ multi-objective optimization etc). Furthermore, we observe that there are security goals that ask for the set of attacks/ set of attack steps/ set of defense steps fulfilling multiple constraints such as of budget, time, minimality, certain prerequisites, etc. Yet, another security goals are concerned with the logical ordering of steps in an attack (like precedence, successive, etc), for which one requires the notion of the partial ordering of steps to be defined. Furthermore, we see there exists a multitude of attributes (and possibly more, when provided the flexibility to define the user-defined attributes), opening up a wide range of possibilities to define, combine and perform operations on the attributes. Considering and accommodating all the above factors, we present the LOCKS grammar (in Section 5).

4 STRUCTURAL ATTACK MODELS (SAM)

As explained in the Section 1, this paper introduces a specification language to formally state the security goals like the ones surveyed in Section 3, which we call LOCKS. We evaluate LOCKS terms over a generic attack model namely the structural attack model (SAM). As decribed previously, SAM

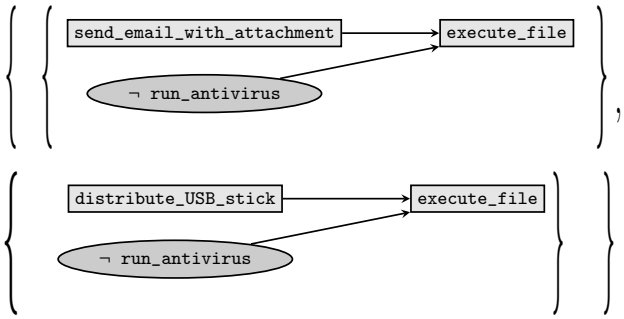

Figure 2: Set of successful attacks A for the attack-defense tree shown in Figure 1.

is based on partially ordered sets of successful attacks that dictates which steps needs to be carried out and in which order. Furthermore, each atomic step in SAM is decorated with a number of attribute values such as cost of execution, difficulty of attack, etc. To show how a specific graphical threat formalism can be translated into the generic structure of SAM, we take an example of an attack-defense tree shown in the Figure 1.

Example 4.1. Refer to the attack-defense tree (ADTree) shown in Figure 1, adapted from [5]. It represents how an attacker can infect a computer with a virus, as modelled in the root. The atomic steps represent the attacker/ defender actions, modelled as the leaves of the tree. The attack steps are shown as rectangles and the defense steps as ellipses. The logical gates (`AND`, `OR`, `SAND`) show how atomic steps are combined to launch a multi-stage attack to reach the root. An `OR` gate is compromised if at least one of its children is compromised; an `AND` gate is compromised if all of its children are compromised; a `SAND` gate is compromised if all of its children are compromised, where the $i+1^{\text{th}}$ child is only executed after the i^{th} child was compromised. In order to launch an attack, an adversary must perform the attack steps and overcome the defense steps, i.e., to reach a sub-goal of `virus_file_on_system`, an attacker must successfully execute `distribute_virus` and overcome the defense step `run_antivirus`.

In order to obtain the SAM model for the ADTree described in Figure 1, one need to traverse the tree from bottom-up, starting from any leaf to the top node and respecting the constraints imposed by the logical gates to obtain an attack. Thus, one can start with either `send_email_with_attachment` or `distribute_USB_stick` (dictated by the `OR` gate in the ADTree), and must overcome `run_antivirus` (dictated by the `AND` gate), followed by `execute_file` (dictated by the `SAND` gate) leading to the top goal of `infect_computer`, thus resulting in a set `A` consisting of two attacks, shown in Figure 2.

Consider one of the set of steps, shown in Figure 3, whose successful execution in an order, leads to the top goal. Note that here the execution of steps: `distribute_USB_stick` and `run_antivirus`, precede the execution of the step `execute_file`. However, there are no constraints on the execution of steps : `distribute_USB_stick` and ¬ `run_antivirus`,

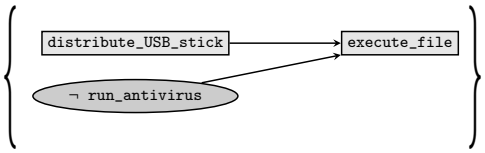

Figure 3: Set of steps in the attack-defense tree shown in Figure 1, whose execution in an order, leads to the top goal.

hence the execution of these steps can be carried out in any order, even in parallel.

Furthermore, we can decorate each step with attributes, for example a boolean attribute `skills_required` that indicates whether special skills are required to execute the step, for example:

```
send_email_with_attachment.skills_required = True
      distribute_USB_stick.skills_required = False
             run_antivirus.skills_required = False
              execute_file.skills_required = False
```

Other graphical threat formalisms such as attack graphs can be translated similarly into SAM.

We now provide the notations and formal definitions of the concepts introduced above. These are used throughout the paper to define the static and the denotational semantics of LOCKS (see Sections 5.1 and 5.2).

Notation. We use $2^{\mathtt{G}}$ to indicate the power set of a set `G` and `PO(G)` for the set of all partial ordered sets over `G`. The disjoint union of two sets `B` and `B'` is denoted by $\mathtt{B} \uplus \mathtt{B'}$. A partial function is denoted as $\mathtt{f}\colon \mathtt{B} \nrightarrow \mathtt{B'}$, where `B` and `B'` are the domain and the codomain. Finally, $\mathtt{f}[\mathtt{a} \mapsto \mathtt{b}]$ denotes the function `f` updated such that it maps `a` to `b`.

Attack and defense steps. A *step* `s` denotes an atomic action available to an attacker or a defender. We consider a global universe of attack and defense steps $\mathtt{S} = \mathtt{AS} \uplus \mathtt{DS}$, where `AS` is the universe of all attack steps and `DS` is the universe of all defense steps. Defense steps are conceptually the same as attack steps, except that they have to be overcome in order to be part of a successful attack.

Attack. A successful *attack* $(\mathtt{W}, \sqsubset) \in \mathtt{PO(S)}$ is a partially ordered set of atomic steps leading to the successful compromise of one or more assets. Here, $\mathtt{W} \subseteq \mathtt{S}$ is a set of atomic steps that have to be carried out, while $\sqsubset \subseteq \mathtt{W} \times \mathtt{W}$ constrains the order in which those steps should be executed.

Attribute names and values. We assume a global universe of *attribute names* given by `Name`. It includes common attributes like `cost`, `damage`, `time`, and `equipment_required`.

We define the type of attributes using the function $\mathtt{Type\_Attr} : \mathtt{Name} \rightarrow \mathit{AtomType}$ where $\mathit{AtomType} = \{\mathit{Bool}, \mathit{Real}, \mathit{Prob}, \mathit{Step}\}$. Here, *Bool*, *Real*, *Prob*, *Step* are the available types that respectively refer to the set of boolean values, real numbers, probability values and elementary attack/ defense steps. *AtomType* is later used in Section 5.1 to specify the static semantics of LOCKS.

Attributes are evaluated in the universe $\mathtt{Val} = \mathtt{PrimVal} \uplus \mathtt{SetVal}$ of all attribute values. This is partitioned into `PrimVal` =

```
Goal    ::= Func(Var) = Expr ; Goal
          | Expr
Expr    ::= Unop Expr
          | Expr Binop Expr
          | Var
          | Literal
          | Var.Label
          | Func (Var)
          | {Expr | Expr}
          | A
          | AS
          | DS
          | Quant Var ∈ Expr : (Expr)
          | Quant Var, Var ∈ Expr : (Expr)
          | Quant Var ⊆ Expr : (Expr)
          | Expr ⊏_Expr Expr
Var     ::= ID
Func    ::= ID
Label   ::= ID
Literal ::= BOOL
          | REAL
          | PROB
```

```
Unop    ::= Max
          | Min
          | ∑
          | ∏
          | Card
          | ¬
Binop   ::= ∩
          | ∪
          | \
          | ∈
          | ⊆
          | ⊂
          | ≤
          | ≥
          | >
          | <
          | =
          | +
          | −
          | ×
          | ∧
          | ∨
Quant   ::= ∀
          | ∃
```

Figure 4: LOCKS Grammar

$\texttt{Bool} \uplus \texttt{Real} \uplus \texttt{Prob} \uplus \texttt{S}$, where $\texttt{Bool} \in \{\texttt{True}, \texttt{False}\}$ is the set of boolean values, Real the set of real numbers, $\texttt{Prob} = [0, 1]$ the set of probability values and S the universe of attack and defense steps; and $\texttt{SetVal} = \texttt{PO}(\texttt{PrimVal})$, the set of *sets* of primitive values.

Definition 4.2 (Structural Attack Model). A structural attack model (SAM) is a tuple $\langle \texttt{A}, \texttt{Attr} \rangle$, where $\texttt{A} \subseteq \texttt{PO}(\texttt{S})$ is a set of all successful attacks and $\texttt{Attr} : \texttt{Name} \rightarrow (\bigcup \texttt{A} \nrightarrow \texttt{PrimVal})$ is a function that returns the attribute value for a given step and attribute name, if the attribute is defined for the step. Recall that AS and DS are the global universes of attack and defense steps and $\texttt{S} = \texttt{AS} \uplus \texttt{DS}$.

Example 4.3. Consider our running example 4.1. Here, the SAM is a tuple $\langle \texttt{A}, \texttt{Attr} \rangle$ where A is the set of successful attacks in Figure 2 and Attr is a function mapping each atomic step in $\bigcup$ A to a boolean attribute of `skill_required`. Querying the security goal of obtaining a set of attack steps that do not require special skills over A, we obtain $\{\boxed{\texttt{distribute\_USB\_stick}}, \boxed{\texttt{execute\_file}}\}$.

5 SPECIFICATION LANGUAGE FOR SECURITY GOALS (LOCKS)

We distill the constructs from both the example security goals (Section 3) and the structural attack model SAM (Section 4) to construct the grammar of LOCKS, shown in Figure 4. The

```
Type     ::= AtomType
           | SetType
           | FuncType
AtomType ::= Bool
           | Real
           | Prob
           | Step
SetType  ::= Set(AtomType)
           | Set(SetType)
FuncType ::= Set(Step) →AtomType
```

Figure 5: LOCKS type grammar

top level non-terminal is Goal, which represents a security goal. It is one of the following:

- Func(Var) = Expr; Goal, which involves first the compilation of Func(Var) = Expr that defines a user-defined function Func, which is subsequently used in the following Goal. We limit ourselves to functions that take sets of steps as input parameters.
- An expression Expr that is defined recursively. We define each term of the expression formally and with examples when discussing the static semantics (in Section 5.1) and denotational semantics (in Section 5.2) of LOCKS.

Other terms in our grammar in Figure 4 consists of terminal symbols that are denoted by the capital letters and include:

- Identifiers ID, used as names of functions, labels and variables.
- BOOL, REAL, STRING, PROB, which are the classes of constants representing data values.
- A, which stands for the set of successful attacks contained in the given instance of the SAM.
- AS and DS, which stand for the fixed universes of attack and defense steps.

The unary and binary operators are standard, except for expressions $\texttt{Expr}_1 \sqsubset_{\texttt{Expr}} \texttt{Expr}_2$. This assumes that both $\texttt{Expr}_1$ and $\texttt{Expr}_2$ evaluate to atomic steps between which a partial ordering relationship exists in the attack to which Expr evaluates. Whether this relationship holds is looked up in the given SAM model.

5.1 Static semantics of LOCKS

Our language is strongly typed. This ensures that the terms in LOCKS are well-formed. The set of all semantic types, denoted *Type*, is defined in Figure 5.

The available types are:

- *AtomType*: *Bool*, *Real*, *Prob*, *Step*, which respectively refer to the boolean values, real numbers, probability values and elementary attack/ defense steps in the structured attack model;
- *SetType*, denoting a set whose elements are *AtomType* or *SetType*;
- *FuncType*, denoting a function that returns a value of *AtomType* when applied to an argument of *SetType*.

We use a contextual typing function $\mathcal{T}$ that partially assigns types to identifiers: i.e, $\mathcal{T} : \mathit{ID} \nrightarrow \mathit{Type}$. We use the

notation $[\![\mathsf{Term}]\!]_{\mathcal{T}}$ to surround the syntactic Term, to be read as: the type of Term under the typing context $\mathcal{T}$.

Type rules for Identifiers. The type rule for identifiers is $[\![\mathsf{ID}]\!]_{\mathcal{T}} = \mathcal{T}(\mathsf{ID})$ iff $\mathcal{T}$ is defined on ID. The literals generated by BOOL, REAL and PROB have their usual types.

Type rules for Goal. The type of Goal terms of the form Expr is defined recursively. The type of Goal terms of the form $\mathsf{Func(Var)} = \mathsf{Expr}_1 \ ; \ \mathsf{Expr}_2$ requires first inferring the type of $\mathsf{Func(Var)} = \mathsf{Expr}_1$ in typing environment $\mathcal{T}$ and using it subsequently to infer the type of Expr_2 in the updated typing environment:

$$[\![\mathsf{Func(Var)} = \mathsf{Expr}_1 \ ; \ \mathsf{Expr}_2]\!]_{\mathcal{T}} = [\![\mathsf{Expr}_2]\!]_{\mathcal{T}[\mathsf{Func} \leftarrow T]} \quad \text{where } T = \mathit{Set}(\mathit{Step}) \rightarrow [\![\mathsf{Expr}_1]\!]_{\mathcal{T}[\mathsf{Var} \leftarrow \mathit{Set}(\mathit{Step})]} \tag{1}$$

Example 5.1. Consider our running example 4.1 and 4.3. Suppose we want to obtain the attack that inflicts the maximum damage in the given structural attack model $\langle \mathtt{A}, \mathtt{Attr} \rangle$, where A is the set of successful attacks given in figure 2 and Attr is a function that annotates each step s with an attribute damage whose value is a real number signifying the monetary damage inflicted by execution of the step. The security goal is then given by equation 2.

$$\mathsf{Tot\_Damage(at)} = \textstyle\sum\{\mathsf{s.damage} \mid \mathsf{s} \in \mathsf{at} \setminus \mathsf{DS}\} \ ; \ \mathsf{Max}\{\mathsf{Tot\_Damage(ak)} \mid \mathsf{ak} \in \mathsf{A}\} \tag{2}$$

This goal is of the form $\mathsf{Func(Var)} = \mathsf{Expr}_1; \mathsf{Expr}_2$, where Func = Tot_Damage is a function that maps a successful attack at whose type is *Set*(*Step*) to a value of type *Real* obtained by evaluating Expr_1, i.e. $\sum\{\mathsf{s.damage} \mid \mathsf{s} \in \mathsf{at} \setminus \mathsf{DS}\}$. It is followed by another expression $\mathsf{Expr}_2 = \mathsf{Max}\{\mathsf{Tot\_Damage(ak)} \mid \mathsf{ak} \in \mathsf{A}\}$ that obtains the maximum value among all successful attacks by using the function Tot_Damage defined earlier.

Type rules for expressions. The type of the expressions of the form Max Expr is *Real* if the type of Expr is *Set*(*Real*), refer to equation 3. The type rules for expressions with other unary operators Min, $\sum$ and $\prod$ can be defined similarly.

$$[\![\mathsf{Max\ Expr}]\!]_{\mathcal{T}} = \mathit{Real} \ \text{ if } \ [\![\mathsf{Expr}]\!]_{\mathcal{T}} = \mathit{Set}(\mathit{Real}) \tag{3}$$

The type of expressions of the form Card Expr is *Real*, provided the type of the Expr is *Set*(*S*) where *S* is *Set*(*Step*) or *Step*. The type of expressions ¬Expr is *Bool* provided the type of Expr is *Bool*. The type of the expression of the form Var is *Step* or *Set*(*Step*), provided by the typing context $\mathcal{T}$.

The type of the expressions of the form Expr_1 Binop Expr_2, with binary operators is given in table 1. Each cell in the table shows the type of the evaluated expression under the given binary operator Binop. For example, the type of the expressions with binary relational operators ($\leq$, $\geq$, $>$, $<$, $=$) is *Bool*, provided the type of both its operands is same and is either *Real* or *Prob*.

The type of an expression Var.Label is Type_Attr(Label), where Type_Attr is the attribute type function introduced in Section 4, and Var is of type *Step*:

$$[\![\mathsf{Var.Label}]\!]_{\mathcal{T}} = \mathsf{Type\_Attr(Name)} \ \text{ if } \ \mathcal{T}(\mathsf{Var}) = \mathit{Step} \tag{4}$$

T_1 \ T_2	*Prob/Real*	*Bool*	*Step*	*Set*(*T*)
Prob/ Real	($\leq$, $\geq$, $>$, $<$, $=$) $\rightarrow$ *Bool* if $T_1 = T_2$ ($+$,$\times$) $\rightarrow$ *T* where $T = T_1 = T_2$	-	-	$\in \rightarrow$ *Bool* where $T = T_1$
Bool	-	($\wedge$, $\vee$) $\rightarrow$ *Bool*	-	$\in \rightarrow$ *Bool* where $T =$ *Bool*
Step	-	-	-	$\in \rightarrow$*Bool* where $T =$ *Step*
Set(*T*)	-	-	-	($\subseteq$, $\subset$) $\rightarrow$ *Bool* ($\cup$, $\cap$, $\setminus$) $\rightarrow$ *Set*(*T*) where $T_1 = T_2$

Table 1: Type matrix for expressions with binary operators

The type of an expression of the form Func(Var) is *T*, provided Func is a function mapping a set of steps to type *T* and the type of Var is *Set*(*Step*):

$$[\![\mathsf{Func(Var)}]\!]_{\mathcal{T}} = T \ \text{ if } \ \mathcal{T}(\mathsf{Var}) = \mathit{Set}(\mathit{Step}) \ \text{ and } \ \mathcal{T}(\mathsf{Func}) = \mathit{Set}(\mathit{Step}) \rightarrow T \tag{5}$$

The type of expressions of the form $\{\mathsf{Expr}_1 \mid \mathsf{Expr}_2\}$ is *Set*(*S*), where the type *S* is inferred under a typing context $\mathcal{U} \supseteq \mathcal{T}$ in which the type of Expr_2 equals *Bool*:

$$[\![\{\mathsf{Expr}_1 \mid \mathsf{Expr}_2\}]\!]_{\mathcal{T}} = \mathit{Set}([\![\mathsf{Expr}_1]\!]_{\mathcal{U}}) \ \text{ with } \ \mathcal{U} \supseteq \mathcal{T} \ \text{ and } \ [\![\mathsf{Expr}_2]\!]_{\mathcal{U}} = \mathit{Bool} \tag{6}$$

Note that the type of the expression Expr_1 under the typing environment $\mathcal{U}$ is not always uniquely defined and may require additional contextual information. For example, consider an expression {x | Card(x)=1}. It is of the form $\{\mathsf{Expr}_1 \mid \mathsf{Expr}_2\}$, but, the type of Expr_1 is not uniquely defined and for example can be a set of steps or a set of real numbers. However, such contrived cases do not occur in our example security goals.

Example 5.2. Consider the expression s.damage in Example 5.1. It is of the form Var.Label and returns a value of type *Real* signifying the damage inflicted by the execution of atomic step s, provided damage is an attribute name whose type is *Real* and *s* is an atomic step of type *Step*. The security goal also contains an expression Tot_Damage(at) of the form Func(Var) that returns a value of type *Real* signifying the cumulative damage of an attack at.

In the same example, consider the expression {s.damage | $\mathsf{s} \in \mathsf{at} \setminus \mathsf{DS}$}. It is of the form $\{\mathsf{Expr}_1 \mid \mathsf{Expr}_2\}$. It evaluates to a set of values where each value is of type *Real* obtained by evaluating the expression s.damage, provided that the atomic step s satisfies the expression Expr_2, i.e., $\mathsf{s} \in \mathsf{at} \setminus \mathsf{DS}$.

The type of expression of the form A is *Set*(*Set*(*Step*)). The type of expressions of the form AS and DS is *Set*(*Step*). The type of expression of the form $\mathsf{Quant\ Var} \in \mathsf{Expr}_1 : (\mathsf{Expr}_2)$ is *Bool*, provided the type of Expr_1 is *Set*(*S*) and the type of Expr_2 is *Bool* inferred on first substituting Var with *S*:

$$[\![\mathsf{Quant\ Var} \in \mathsf{Expr}_1 : (\mathsf{Expr}_2)]\!]_{\mathcal{T}} = \mathit{Bool} \ \text{ if } \ [\![\mathsf{Expr}_1]\!]_{\mathcal{T}} = \mathit{Set}(S) \ \text{ and } \ [\![\mathsf{Expr}_2]\!]_{\mathcal{T}\mathsf{Var} \leftarrow S)} = \mathit{Bool} \tag{7}$$

The type of expression of the form $\mathsf{Quant\ Var}_1, \mathsf{Var}_2 \in \mathsf{Expr}_1 : (\mathsf{Expr}_2)$ is *Bool*, provided the type of Expr_1 is *Set*(*S*) and the type of Expr_2 is *Bool* inferred on first substituting Var_1 and

Var_2 with S. The type of the expression Quant Var $\subseteq$ Expr_1 : (Expr_2) is defined similarly.

Example 5.3. Consider our running example 4.3 where we want to obtain the set of attacks in the structured attack model $\mathcal{M}$ that do not require special skills. The security goal is given by:

$$\{\mathsf{at} \in \mathsf{A} \mid \forall \mathsf{s} \in \mathsf{at} \setminus \mathsf{DS} : \neg \mathsf{s.skill\_required}\} \tag{8}$$

Here, the expression $\forall \mathsf{s} \in \mathsf{at} \setminus \mathsf{DS} : \neg \mathsf{s.skill\_required}$ is of the form Quant Var $\in$ Expr_3 : Expr_4 and returns true provided all atomic attack steps forming the successful attack `at` satisfy Expr_4, i.e., do not require special skills.

The type of the expression of the form $\mathsf{Expr}_1 \sqsubset_{\mathsf{Expr}} \mathsf{Expr}_2$ is *Bool*, provided the type of both Expr_1 and Expr_2 is *Step* and the type of Expr is *Set*(*Step*).

5.2 Denotational Semantics of LOCKS

The meaning of the terms in LOCKS depends on the following entities:

- A Value domain `Val` is the semantic domain of LOCKS (see Section 4).
- A Declaration function $\mathcal{D}$ with a signature $\mathcal{D}$: $ID \nrightarrow$ (`Val` $\rightarrow$ `Val`), partially maps an identifier to a function where both the domain and the co-domain are in the value domain.
- An assignment function $\mathcal{A}$ with signature $\mathcal{A}$: $ID \nrightarrow$ `Val` that partially maps identifier that stands for a variable to its value in `Val`.

Semantics of LOCKS terms. We use the notation of double brackets $[\![\_]\!]$ to enclose the syntactic term and use the contextual environments $\mathcal{A}, \mathcal{D}$ to assign the appropriate denotations to the variables and the function identifiers. Thus, the meaning of expression `E` is denoted as $[\![\mathtt{E}]\!]_{\mathcal{A},\mathcal{D}}$. The binary operators ($\cup$, $\cap$, $\setminus$, $\in$, $\subseteq$, $\subsetneq$, $\sqsubseteq$, $\geq$, $\leq$, $>$, $<$, $=$, $+$, $-$, $\times$, $\wedge$, $\vee$), unary operators (Max, Min, $\sum$, $\prod$, Card, $\neg$) and the quantifiers ($\forall$, $\exists$) have their usual meanings. The meaning of the identifiers is a value given by its assignment, sometimes $\mathcal{D}$ (when an identifier stands for a function) and sometimes $\mathcal{A}$ (when an identifier stands for a variable).

Semantics of Goals. The semantics of the Goal terms of the form Expr is defined recursively. The semantics of the Goal terms of the form Func(Var) = Expr_1 ; Expr_2 involves first the compilation of Func(Var)=Expr_1 used subsequently to evaluate the following expression term Expr_2. Here, Func(Var)=Expr_1 defines the function Func, mapping the set of steps Var to a value obtained by evaluating the expression Expr_2 when all free occurrences of Var in Expr_1 are substituted by the value of Var. It is subsequently used to evaluate the expression term Expr_2, refer to equation 9. An illustrative example of this form was provided in example 5.1.

$$[\![\mathsf{Func(Var)} = \mathsf{Expr}_1 \ ; \ \mathsf{Expr}_2]\!]_{\mathcal{A},\mathcal{D}} = [\![\mathsf{Expr}_2]\!]_{\mathcal{A},\mathcal{D}[\mathsf{Func} \leftarrow F]} \tag{9}$$
$$\text{where } F : S \mapsto [\![\mathsf{Expr}_1]\!]_{\mathcal{A}[Var \leftarrow S],\mathcal{D}}$$

Semantics of Expressions. The semantics of expressions of the form Unop Expr and Expr Binop Expr is obtained by evaluating the expressions with unary/ binary operators under the type constraints, defined in Section 5.1. The meaning of an expression of the form Var is given by its assignment. The meaning of an expression of the form Literal is its usual interpretation. The meaning of an expression of the form Var.Label is a value returned by the `Attr` function (recall `Attr` is a function that maps defined in definition 4.2), provided Var is an atomic step given by its assignment and `Label` is an attribute name, both of which can be looked up in the instance of the structural attack model $\mathcal{M}$, refer to equation 10. An illustrative example of this form is provided in example 5.1. The meaning of an expression of the form Func(Var) is provided by the declaration of Func and the assignment of Var, as shown in equation 11.

$$[\![\mathsf{Var.Label}]\!]_{\mathcal{A},\mathcal{D}} = \mathtt{Attr}(\mathcal{A}(\mathsf{Var}))(\mathsf{Label}) \tag{10}$$
$$[\![\mathsf{Func\ (Var)}]\!]_{\mathcal{A},\mathcal{D}} = \mathcal{D}(\mathsf{Func})(\mathcal{A}(\mathsf{Var})) \tag{11}$$

The meaning of expressions of the form AS, DS is the fixed universe of attack and defense steps. The meaning of the expression A is the set of successful attacks provided by the user as $\mathcal{M}$, where $\mathcal{M}$ is the instance of SAM, defined in Section 4. The meaning of the expression of the form $\{\mathsf{Expr}_1 \mid \mathsf{Expr}_2\}$ is a set of values, each obtained by evaluating an expression Expr_1 under a contextual environment $\mathcal{P} \supseteq \mathcal{A}$ and $\mathcal{Q} \supseteq \mathcal{D}$, that simultaneously satisfy the expression Expr_2 under the same contextual environment, given in equation 12. An illustrative example of this form is provided in example 5.2.

$$[\![\{\mathsf{Expr}_1 \mid \mathsf{Expr}_2\}]\!]_{\mathcal{A},\mathcal{D}} = \{[\![\mathsf{Expr}_1]\!]_{\mathcal{P},\mathcal{Q}} \mid \mathcal{P} \supseteq \mathcal{A}, \mathcal{Q} \supseteq \mathcal{D}, \ [\![\mathsf{Expr}_2]\!]_{\mathcal{P},\mathcal{Q}} = \mathtt{true}\} \tag{12}$$

The meaning of an expression of the form Quant Var $\in$ Expr_1 : (Expr_2) with `Quant` instantiated by the universal quantifier is true if and only if for all objects $x \in \mathsf{Expr}_1$, the evaluation of $\mathsf{Expr}_2(x)$ holds true when all free variables Var in Expr_2 is mapped to x, given in equation 13. Similarly, Quant Var_1, Var_2 $\in$ Expr_1 : (Expr_2) with `Quant` instantiated by the universal quantifier is true if and only if for any pair of objects $x, y \in \mathsf{Expr}_1$, the evaluation of $\mathsf{Expr}_2(x, y)$ holds true when variables Var_1 in Expr_2 is mapped to x and Var_2 in Expr_2 is mapped to y. The semantics of the expressions of the form Quant Var $\in$ Expr_1 : Expr_2 with existential and negated quantifiers can be defined similarly.

$$[\![\forall \ \mathsf{Var} \in \mathsf{Expr}_1 : (\mathsf{Expr}_2)]\!]_{\mathcal{A},\mathcal{D}} = \forall x \in [\![\mathsf{Expr}_1]\!]_{\mathcal{A},\mathcal{D}} : [\![\mathsf{Expr}_2]\!]_{\mathcal{A}(\mathsf{Var} \mapsto \mathsf{x}),\mathcal{D}} \tag{13}$$

The meaning of an expression of the form $\mathsf{Expr}_1 \sqsubset_{\mathtt{at}} \mathsf{Expr}_2$ with binary operator $\sqsubset_{\mathtt{at}}$ is true, provided Expr_1 is a atomic step that comes before another atomic step Expr_2 in an attack `at` in $\mathcal{M}$, where $\mathcal{M}$ is the instance of the SAM.

6 CONCLUSION

In this paper, we addressed a lack of formalized security goals in the literature. To fill this gap, we proposed a formal specification language LOCKS, that allow security practitioners

to formulate both qualitative and quantitative security goals embracing wide number of attributes such as cost, damage, probability, etc. The security goals in LOCKS are expressed as queries over an attack model, namely the structural attack model SAM. As most prominent threat models such as attack trees and attack graphs can be translated to generic structures of SAMs, our proposed language can express security goals over all these frameworks. We show the practical value of our work by taking several informally stated security goals from the literature and formulating them in our proposed language.

Laying the formal foundations to define the security goals, we see several directions extending our work. One is to make our language tool supported. Another direction is to enrich the structural attack model with an attacker profile and attributes which are functionally dependent on each other like probabilities over time. This will enable us to answer questions such as: Will Ethan and Ervin, two adversaries collaborate to launch an attack? or What is the probability of success to launch a multi-stage attack in 1 year? In parallel, we plan to endow LOCKS with a more user friendly syntax with typical domain-specific constructs. Furthermore, we plan to validate our framework with the security practitioners.

ACKNOWLEDGMENTS

This research has been partially funded by STW and ProRail under the project ArRangeer (grant 12238), STW under the project SEQUOIA (grant 15474), NWO under the projects BEAT (grant 612001303) and SamSam (grant 50918239), and the EU under the projects SUCCESS and TREsPASS (318003).

REFERENCES

[1] M. Abadi and A. D. Gordon. 1999. A Calculus for Cryptographic Protocols: The Spi Calculus. *Inf. Comput.* 148, 1 (1999), 1 – 70.
[2] C. Alberts, A. Dorofee, J. Stevens, C. Woody, A. Dorofee, and C. Scondras. 2003. *Introduction to the OCTAVE Approach.* Technical Report. SEI, CMU.
[3] F. Arnold, D. Guck, R. Kumar, and M. Stoelinga. 2015. *Sequential and Parallel Attack Tree Modelling.* Springer, 291–299.
[4] Z. Aslanyan and F. Nielson. 2015. Pareto Efficient Solutions of Attack-Defence Trees. In *Principles of Security and Trust (POST) 4th Int. Conf.* Springer, 95–114.
[5] Z. Aslanyan, F. Nielson, and D. Parker. 2016. Quantitative Verification and Synthesis of Attack-Defence Scenarios. In *IEEE 29th Computer Security Foundations Symposium, CSF.* IEEE, 105–119.
[6] M. Y. Becker, C. Fournet, and A. D. Gordon. 2010. SecPAL: Design and Semantics of a Decentralized Authorization Language. *J. Comp. Security* 18, 4 (2010), 619–665.
[7] G. Behrmann, A. David, and K. G. Larsen. 2004. A Tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems, Int. School on Formal Methods for the Design of Computer, Comm. and Software Systems, SFM-RT.* Springer, 200–236.
[8] A. Boswell. 1995. Specification and Validation of a Security Policy Model. *IEEE Tran. Softw. Engg* 21, 2 (1995), 63–68.
[9] A. Buldas, P. Laud, J. Priisalu, M. Saarepera, and J. Willemson. 2006. Rational Choice of Security Measures via Multi-parameter Attack Trees. In *Critical Information Infrastructures Security (CRITIS).* Springer, 235–248.
[10] M. Dacier and Y. Deswarte. 1994. Privilege Graph: An Extension to the Typed Access Matrix Model. In *European Symp. on Research in Computer Security (ESORICS).* Springer, 319–334.
[11] M. Fraile, M. Ford, O. Gadyatskaya, R. Kumar, M. Stoelinga, and R. Trujillo-Rasua. 2016. Using Attack-Defense Trees to Analyze Threats and Countermeasures in an ATM: A Case Study. In *The Practice of Enterprise Modeling - PoEM Proc.* Springer, 326–334.
[12] H. Hermanns, J. Krämer, J. Krcál, and M. Stoelinga. 2016. The Value of Attack-Defence Diagrams. In *Principles of Security and Trust (POST) 5th Int. Conf., Eindhoven, The Netherlands.* Springer, 163–185.
[13] N. Idika and B. Bhargava. 2010. Extending Attack Graph-Based Security Metrics and Aggregating Their Application. *IEEE Trans. on Dependable and Secure Computing* 9, 1 (2010), 75–85.
[14] Isaca. 2012. *Cobit 5.* ISA.
[15] W. Jansen. 2010. Directions in security metrics research. (2010).
[16] S. Jha, O. Sheyner, and J. Wing. 2002. Two formal analyses of attack graphs. In *15th IEEE Comp. Security Foundations (CSF).* IEEE, 49–63.
[17] B. Kordy, L. P. Cambacédès, and P. Schweitzer. 2014. DAG-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review* 13-14 (2014), 1–38.
[18] B. Kordy, S. Mauw, Sasa Radomirovic, and P. Schweitzer. 2010. Foundations of Attack-Defense Trees. In *Formal Aspects of Security and Trust (FAST).* Springer, 80–95.
[19] B. Kordy, S. Mauw, and P. Schweitzer. 2012. Quantitative Questions on Attack-Defense Trees. In *Info. Security and Cryptology (ICISC).* Springer, 49–64.
[20] R. Kumar, D. Guck, and M. Stoelinga. 2015. Time Dependent Analysis with Dynamic Counter Measure Trees. *arXiv preprint arXiv:1510.00050* (2015).
[21] R. Kumar, E. Ruijters, and M. Stoelinga. 2015. Quantitative Attack Tree Analysis via Priced Timed Automata. In *Formal Modeling and Analysis of Timed Systems (FORMATS).* Springer, 156–171.
[22] R. Kumar and M. Stoelinga. 2017. Quantitative Security and Safety Analysis with Attack-Fault Trees. In *Int. Symp. on High Assurance System Engg. (HASE).* IEEE, 25–32.
[23] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proc. 23rd Int. Conf. on Computer Aided Verification (CAV'11).* Springer, 585–591.
[24] E. LeMay, M. D. Ford, K. Keefe, W. H. Sanders, and C. Muehrcke. 2011. Model-based Security Metrics Using ADversary VIew Security Evaluation (ADVISE). In *Quantitative Evaluation of Systems (QEST).* IEEE, 191–200.
[25] T. Lodderstedt, D. A. Basin, and J. Doser. 2002. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proc.of the 5th Int. Conf. on The Unified Modeling Language.* Springer, 426–441.
[26] S. Mauw and M. Oostdijk. 2006. Foundations of Attack Trees. In *Info. Security and Cryptology (ICISC).* Springer, 186–198.
[27] N. Mead. 2013. SQUARE Process https://buildsecurityin.us-cert.gov/articles/best-practices/requirements-engineering/square-process. (2013).
[28] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. 2003. Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs. In *Annual Comp. Security App. Conference (ACSAC).* IEEE, 86–95.
[29] S. Schivo, B. M. Yildiz, E. Ruijters, C. Gerking, R. Kumar, S. Dziwok, A. Rensink, and M. Stoelinga. 2017. How to Efficiently Build a Front-End Tool for UPPAAL: A Model-Driven Approach. In *Dependable Software Engineering. Theories, Tools, and App. - Third Int. Symp., SETTA, Proc.* Springer, 319–336.
[30] B. Schneier. 1999. Attack Trees. *Dr. Dobb's Journal* (1999).
[31] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. 2002. Automated Generation and Analysis of Attack Graphs. In *Symp. on Security and Privacy.* IEEE, 273–284.
[32] D. von Oheimb and S. Mödersheim. 2012. ASLan++ — A Formal Security Specification Language for Distributed Systems. In *Formal Methods for Components and Objects (FMCO).* Springer, 1–22.
[33] L. Wang, S. Noel, and S. Jajodia. 2006. Minimum-cost network hardening using attack graphs. *Computer Comm.* (2006), 3812–3824.
[34] L. Wang, A. Singhal, and S. Jajodia. 2007. Measuring the Overall Security of Network Configurations Using Attack Graphs. In *Data and Applications Security (DBSec).* Springer, 98–112.