Computers & Security 130 (2023) 103284

Contents lists available at ScienceDirect

Computers & Security

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

The meta attack language - a formal description

Wojciech Wideł[a], Simon Hacks[b,*], Mathias Ekstedt[a], Pontus Johnson[a], Robert Lagerström[a]

[a] Division of Network and Systems Engineering, KTH Royal Institute of Technology,Stockholm, Sweden
[b] Department of Computer and Systems Sciences, Stockholm University, Stockholm, Sweden

A R T I C L E   I N F O

Article history:
Received 27 January 2023
Revised 24 April 2023
Accepted 28 April 2023
Available online 2 May 2023

Keywords:
Threat modeling
Attack simulation
Attack graphs
Domain specific language

A B S T R A C T

Nowadays, IT infrastructures are involved in making innumerable aspects of our lives convenient, starting with water or energy distribution systems, and ending with e-commerce solutions and online banking services. In the worst case, cyberattacks on such infrastructures can paralyze whole states and lead to losses in terms of both human lives and money.

One of the approaches to increase security of IT infrastructures relies on modeling possible ways of compromising them by potential attackers. To facilitate creation and reusability of such models, domain specific languages (DSLs) can be created. Ideally, a user will employ a DSL for modeling their infrastructure of interest, with the domain-specific threats and attack logic being already encoded in the DSL by the domain experts.

The Meta Attack Language (MAL) has been introduced previously as a meta-DSL for development of security-oriented DSLs. In this work, we define formally the syntax and a semantics of MAL to ease a common understanding of MAL's functionalities and enable reference implementations on different technical platforms. It's applicability for modeling and analysis of security of IT infrastructures is illustrated with an example.

1. Introduction

Assessing the level of security in various forms of computer systems is a challenging activity. In fact, the difficulty starts already at the definition of what security means and how it should be measured. One approach for measuring and analyzing security that lately has gained popularity, both in academia and in practice, is attack graphs. Attack graphs describe how an (imagined) antagonist can move in a system by taking different actions. In its simplest form the actions are combined in a graph by means of OR and AND gates capturing the logical dependencies between the actions. With the attack vectors resulting from this type of modeling, it is possible for security engineers to understand and assess weak links in the overall defense of larger system architectures.

Even though potentially valuable for the security engineers, generally it is quite costly to make use of the attack graphs due to their poor reusability. In previous work (Johnson et al., 2018), we have introduced the Meta Attack Language (MAL) as a meta domain specific language (meta-DSL) to support the development of domain specific languages (DSLs) for attack graph generation and attack simulations. MAL formalizes how modeling languages can be devised so that specific attack graph instances can be automatically generated from system architecture models described according to a specific meta model.

By providing MAL as a meta-DSL, it is easier to create DSLs that can solely focus on representing domain related aspects, while already providing the frame for modelling the security aspects. Even better, by providing an ecosystem of DSLs developed with MAL (Hacks and Katsikeas, 2021), language developers are able to reuse existing concepts and refine them towards the needs of the certain domain. Moreover, an existing tool (Ekstedt et al., 2015) has already been further developed to support MAL-based DSLs to allow efficient computation of attack paths based on the provided instantiation of the MAL DSL.

In this paper, we introduce a formal description of the syntax and semantics of MAL. Thus, we describe the functional aspects of MAL in greater detail and we aim to unify the understanding of MAL among its different stakeholders. Moreover, we want to provide a technology independent description of MAL to enable its implementation in different technologies. Accordingly, we opted for a formalization in set notation and not in technology specific notations as for example xtext.[1]

* Corresponding author.
E-mail addresses: widel@kth.se (W. Wideł), simon.hacks@dsv.su.se (S. Hacks), mekstedt@kth.se (M. Ekstedt), pontusj@kth.se (P. Johnson), robertl@kth.se (R. Lagerström).

[1] https://www.eclipse.org/Xtext/

https://doi.org/10.1016/j.cose.2023.103284

In previous works, syntax and semantics were only briefly sketched (Johnson et al., 2018) to a level that would not allow providing another implementation of MAL. Instead, the idea of MAL was sketched and illustrated by an informal example. Other works did not elaborate on MAL itself but were solely using it to create MAL DSLs (e.g., Hacks et al., 2020; Katsikeas et al., 2019; Katsikeas et al., 2022) or provide methods for the developing of MAL DSLs (e.g., Hacks and Katsikeas, 2021; Hacks et al., 2023; Xiong et al., 2021).

The paper starts off with Section 2 providing an overview of MAL. In Section 3, we report on related work, before we present an example model in Section 4 that is used throughout the main contributions of the paper presented in Section 5 and 6 that cover the syntax and semantics of MAL. In Section 7, we perform some exemplary calculations based on the example model before the paper is concluded.

2. An informal introduction to MAL

MAL could be thought of as a language combining UML class and object diagrams and probabilistic attack-defense graphs. The basic elements of MAL are `Assets`, which are described in a concrete MAL DSL and correspond to classes, that can have `Instances`, which are represented in models adhering to a concrete MAL DSL and correspond to objects. E.g., an `Asset` *Computer* could have an `Instance` *MyMacBook*. `Assets` can have `Associations` to one another so that a *Computer* can *Communicate* over a *Network*. As in class diagrams, the `Associations` feature `Roles` and `Multiplicities`, where the latter govern how many `Instances` can be `Linked` to each other.

In addition to this standard UML structure, MAL also features `Attack steps` that are located on the `Assets`. E.g., a *Computer* could have the `Attack steps` *Connect* and *Compromise*. `Attack steps` are of either AND or OR type and can have parents and children so that they can form attack graphs. Furthermore, MAL includes `Defenses` that can be related to the `Attack steps` when forming the graph. So, our *Computer* might be defended by being *Patched*, and with this defense in place it makes sense to introduce two `Attack steps` on the *Computer: ExploitKnownVulnerability* and *ExploitZeroDayVulnerability*. In order to encode how difficult it is for an attacker to succeed with the different `Attack steps`, for instance given available `Defenses`, MAL also allows the language designer to specify a `Time To Compromise (TTC)` for `Attack steps`. As mentioned earlier, MAL adopts a probabilistic approach, so we may for instance specify that we believe that the time it takes to succeed with *ExploitZeroDayVulnerability* follows a certain probability distribution, perhaps *Exponential(0.01)* days. However, any other probability distribution or combination of probability distributions can be used to describe the likelihood of a successful attack step and the needed effort by the attacker, which need to be defined by the MAL DSL developer following for example (Xiong et al., 2021).

To sum up, one can understand a MAL DSL as a collection of attack trees encoded in assets. These assets can then be combined by the user to create a representation of the organization's reality. In the background, the different attack trees are linked accordingly, enabling the overall security assessment of the model.

Finally, domain specific languages devised in MAL also encode how the attack graphs are generated for specific instance models. For `Attack steps` located on the same `Asset` this is straight forward: *Computer.Connect* can lead to both *Computer.ExploitKnownVulnerability* and *Computer.ExploitZeroDayVulnerability* (depending on *Computer.Patched*), and these two `Attack steps` in turn lead to *Computer.Compromise*. This attack graph fragment is valid for all *Computers*. But, from *Computer.Compromise* we would normally think that the attacker would be able to do the `Attack step` *Transmit* on all the *Networks* that the *Computer* has a *Communication* `Link` to. And once there, the Attacker can do *Computer.Connect* on all (other) *Computers* `Linked` to the *Network*. From such logic a specific attack graph instance can be derived for every instance model and the TTC values in the graph can be aggregated to global TTC values for all the `Attack steps`, given an `Attacker starting point`.

3. Related work

We divide the coverage of related work into security modeling and analysis approaches on the one hand and formalizations of modeling languages on the other.

3.1. Security modeling and analysis

Before discussing existing approaches for security modeling and analysis, we reason about the requirements towards the MAL and compare it to existing solutions:

1. **Causality-based security predictions**. The MAL should support causality-based security predictions, instead of a formal verification of the system's design. The rational behind this requirement is to support the analysis of the quality of the later implementation and configuration vulnerabilities of systems. Moreover, different simulations should be performed considering different defenses effective or not.
2. **Enterprise-wide systems-of-systems analysis**. The MAL should be able to simulate enterprise-wide systems-of-systems to provide an overall security assessment of the entire IT landscape or sub-sets of it.
3. **Individual component analysis**. Additionally to the aforementioned requirement, MAL should also support a more detailed analysis of individual components. This allows the security analyst to inspect also more details of a certain system if necessary.
4. **Enable non-security expert for analysis**. The security analysis of systems is challenging as the analyst needs domain specific knowledge as well as security knowledge. At the same time, the pool of persons having knowledge in both areas is restricted. To address this challenge, MAL should also enable non-security experts to perform the analysis.

The field of model-driven security engineering includes quite a large number of DSLs, such as UMLsec (Jürjens, 2002; 2005), SecureUML (Basin et al., 2011; 2006), SECTET (Alam et al., 2007; Hafner et al., 2006), or STS-ml (Paja et al., 2015). In these languages it is possible to specify a system design in terms of components and interactions between them. Additionally, security properties such as constraints, requirements, or threats can be defined. Security analysis is then conducted using model checking and by searching for constraint violations, while different languages use different formalisms to build up their syntax and semantics.

The approach of these DSLs promotes secure system design or in other words a formal verification of the system. While it is beneficiary to have such a formal verification, these DSLs struggle to easily simulate different configurations of the system, i.e., different vulnerabilities and defenses in place. This can be accomplished by MAL. Moreover, are these solutions not domain-specific in the sense that they are designed for a certain business domain, such as power or health. Instead, they are general purpose languages that require already the respective security knowledge to be used effectively.

Moreover, these languages are classically designed to either analyze an organization more holistically or solely provide means to analyze single components such as a single application. Other security languages solely focus on the modeling of security, such as CORAS (Lund et al., 2010), secureTROPOS (Mouratidis et al., 2002), and SecDSVL (Almorsy and Grundy, 2014). These languages capture security properties but do not provide any reasoning logic to analyze models.

Another well-established language for modeling security is attack trees, attributed to Bruce Schneier (Schneier, 1999; 2000), and closely related to threat logic trees (Weiss, 1991). Formal foundations of attack trees have been laid by Mauw and Oostdijk (2005), and the framework was extended to include defenses by Kordy et al. (2010). Since then, many attack-tree-based approaches have been presented (Ingols et al., 2009; Kotenko and Doynikova, 2014; Ou and Singhal, 2011; Williams et al., 2008). Kordy et al. (2014) provide a summary on those approaches, and an overview of the recent advances in attack tree-based modeling of security is given in Widełet al. (2019). Among these advances are (Ivanova et al., 2015; Pinchinat et al., 2014; Vigo et al., 2014), which tackle the issue of semiautomatic generation of attack trees from models of systems of interest.

Attack trees allow the security assessment on a detail level of choice and can be also easily altered to allow the analysis of different system configurations. However, they require a high level of understanding of the concept itself and are no tool that can be provided to non-security experts to analyze their systems.

In each of these works, the burden of describing the attack logic is put on the creator of the model. In the MAL framework, the attack logic is encoded in a DSL created by the domain experts, which we support by a method to design the language itself (Hacks et al., 2022) and methods to determine the vulnerabilities' properties (Rencelj Ling and Ekstedt, 2022; Xiong et al., 2021). Thus, the modeler needs only to specify the objects in the system and the relations between them, while first research has been conducted to support the automated generation of concrete MAL DSL instances (Aldea and Hacks, 2022; Hacks et al., 2021; Rencelj Ling and Ekstedt, 2021). Furthermore, the process of generating attack graphs from models created using MAL is fully automated by the tool support (Ekstedt et al., 2015), which takes the model and the MAL DSL and generates the related attack graph from it.

Apart from research related publications, there exists also a number of attack graph-based tools. These tools require information on existing system or infrastructure and can from this automatically derive attack graphs. The MulVal tool (Homer et al., 2013; Ou et al., 2005) constructs logical attack graphs by associating the vulnerabilities extracted from scans. k-defense Safety (Wang et al., 2014) extends MulVAL for the computation of zero-day attack graphs. NAVIGATOR (Chu et al., 2010) considers identified vulnerabilities as directly exploitable by the attacker, provided that the latter has access to the vulnerable system. The TVA tool (Noel et al., 2009) models networks in terms of security conditions and uses a database of exploits for specifying transitions between these security conditions. Similarly, NetSecuritas (Ghosh et al., 2015) composes scanners output and known exploits to generate attack graphs and corresponding security recommendations.

Probabilistic attack graph modeling is a research subfield of its own. Frigault et al. (2008) translate "raw" attack graphs obtained with the TVA-tool into dynamic Bayesian networks and convert CVSS scores to probabilities. Similarly, Xie et al. (2010) rely on CVSS to model uncertainties in the attack structure, attacker's actions, and alerts triggering. Poolsappasit et al. (2012) use Bayesian attack graphs to estimate the security risk on network systems and produce a security mitigation plan using a genetic algorithm. In the framework developed by Arnold et al. (2014), the basic actions in attack trees are assigned probability distributions of time needed for their execution. While in our setting, values are sampled from such distributions and attack simulations employing these values are performed in order to approximate the overall distribution corresponding to compromise of assets of interest, the goal of Arnold et al. (2014) is to derive the exact distribution for the main goal of the attacker from those distributions. They fulfill the most of the requirements presented before, while still being designed for security-experts and as general purpose tools. Thus, these are also no tools that can be provided to domain experts directly.

Previous work that has led up to the need to develop MAL includes CySeMoL (Sommestad et al., 2013), P^{2}CySeMoL (Holm et al., 2015), and pwnPr3d (Johnson et al., 2016). These are all DSLs that automatically generate probabilistic attack graphs from a given system language specification. However, they do not follow an explicit meta language, making them hard to change and re-use.

3.2. Modeling language formalization

When it comes to the formalization of DSLs much work has been conducted on defining semantics of UML diagrams, especially class diagrams. Szlenk (2006) recognized that the semantics of models written in UML is not precisely defined, which might lead to incorrect interpretation of models and negatively influence the verification of a model and its transformation. He formally defined the syntax of UML class diagrams and its semantics. Our formalization of MAL is strongly based on his approach. Costal et al. (2011) took a closer look at the semantics of sub setting, specialization, and redefinition relations in UML diagrams. Feinerer and Salzer (2014) defined semantics for multiplicities of associations. Abdulganiyyi and Ibrahim (2014) developed a semantic abstraction by a logical approach to trace back former relations in evolved diagrams. Another stream of syntax and semantics in UML diagrams focuses on the definition of algebra for class diagrams. For instance, Enjo et al. (2009) present a syntax of a class diagram describing a data model for syntactical foundation for class diagram algebra and the related operations.

The endeavors of the scientific community are not restricted to the formalization of UML diagrams. There is also a plenty of research related to the formalization of DSLs. Jackson and Sztipanovits (2009) develop formal foundations for the structural semantics DSLs. A higher level of abstraction is presented by Jiang and Wang (2012) focusing on formalizing meta DSLs. They propose a formal representation of domain specific meta modeling languages. They use first-order logic to verify the consistency of their language and meta-models built based on the language.

Similarly to our work, Stappers et al. (2012) formalize an already existing DSL. However, their language supports the definition of predictive and reactive rules to optimally allocate manufacturing activities to mechatronic subsystems over time while subjected to dynamic constraints. They follow a two steps approach. First, they project the existing concrete syntax onto a formal abstract syntax that defines the language operators and process terms. Second, they define the dynamic operational semantics using structural operational semantics.

4. A MAL example

In this section we provide an example to illustrate the practical use of MAL. Due to space restrictions the example is small and does not cover the full semantic scope of MAL. Instead, the purpose of this section is to illustrate that MAL can be employed practically for threat modeling using available tools.[2]

[2] All tools used here are available on https://mal-lang.org

4.1. An example language

The domain for our example language is basic computer networks and it is developed around the defense mechanisms of credential encryption, software patching, and antivirus software. We provide two alternative specifications of the language. The first one is based on the human-readable syntax of MAL, as used by MAL practitioners and as accepted by existing MAL compilers. The second one employs the syntax defined in Section 5.

```
abstract asset Machine {
| connect
-\gt  authCompromise
| authenticate
-\gt  authCompromise
& authCompromise
-\gt  compromise
| compromise
-\gt  executees.connect,
storedCreds.access,
networks.communicate
}
asset Hardware extends Machine {
}
asset Software extends Machine {
| compromise
+\gt  executor.connect
}
asset OperatingSystem extends Software {
| connect
+\gt  attemptZeroDayExploit,
attemptExploitKnownVulnerability

& attemptZeroDayExploit
[Exponential(0.005)]
-\gt  executeZeroDayExploit,
bypassAVwithZeroDayExploit
& attemptExploitKnownVulnerability
[Exponential(0.1)]
-\gt  executeKnownExploit,
bypassAVwithKnownVulnExploit
& bypassAVwithZeroDayExploit
[Bernoulli(0.9)]
-\gt  enableExploitZeroDayVulnerability
& bypassAVwithKnownVulnExploit
[Bernoulli(0.5)]
-\gt  enableExploitKnownVulnerability
| enableExploitZeroDayVulnerability
-\gt  executeZeroDayExploit
| enableExploitKnownVulnerability
-\gt  executeKnownExploit
& executeZeroDayExploit
-\gt  compromise
& executeKnownExploit
-\gt  compromise
# patched
-\gt  attemptExploitKnownVulnerability
E antiVirusProtected
\lt - executees[AntiVirus]
-\gt bypassAVwithKnownVulnExploit,
bypassAVwithZeroDayExploit
!E notAntiVirusProtected
\lt - executees[AntiVirus]
-\gt enableExploitKnownVulnerability,
enableExploitZeroDayVulnerability
}
asset AntiVirus extends Software {
}
asset Network {
| communicate
-\gt  parties.connect
}
asset Credentials {
| access
-\gt  compromiseUnencrypted,
crack
& compromiseUnencrypted
-\gt  compromise
& crack [Gamma(25, 1)]
-\gt  compromise
| compromise
-\gt  authenticates.authenticate
# encrypted
-\gt  compromiseUnencrypted
}
associations {
Machine [executor]
1 \lt  -- Execution -- \gt *
[executees] Software
Machine [parties]
* \lt  -- Communication -- \gt *
[networks] Network
Machine [stores]
* \lt  -- Storage -- \gt *
[storedCreds] Credentials
Machine [authenticates]
* \lt  -- Access -- \gt *
[authCreds] Credentials
}
```

As a starting point the language defines `Machine`, which is an abstract representation of items that can execute code. `Machine` is extended, in several layers, into a number of concrete assets; `Hardware`, `Software`, `OperatingSystem`, and `AntiVirus`. The specification also includes computer `Network` and `Credentials`. In terms of associations, `Machine` can execute `Software` and all `Software` requires a machine to execute on. Moreover, `Machines` can communicate over `Networks` and they can store `Credentials`, and `Credentials` can allow access to `Machines`.

On the `Machine` asset, we first find the attack step `connect`, representing the attacker gaining logical access to the `Machine` and thus being allowed to send basic requests to it, and the attack step `compromise` indicating that the attacker has full control over the `Machine`. Furthermore, we find attack logic for taking over the `Machine` by means of using credentials. The `authCompromise` attack step is of type AND and requires that the attacker both have done `connect` and `authenticate`. In order to know how the attacker is able to `authenticate` we in turn need to look to the `Credentials` asset to see that this is achieved by succeeding by a `compromise` there, since this attack steps leads to `authenticate` on all the `Machines` that this `Credentials` instances provides access to, specified by the role `authenticates`. To `compromise` `Credentials` in turn is achieved by one out of two attack vectors depending on if the defense mechanism `encrypted` is true or false. When `encrypted`, the attack logic forces the attacker to do the attack step `crack`, where we also have encoded TTC distribution for this operation. Logically, the `encrypted` defense is thus disabling the `compromiseUnencrypted` attack step since this is an ANDstep requiring all parents to be true. Analogously to `Machine`,

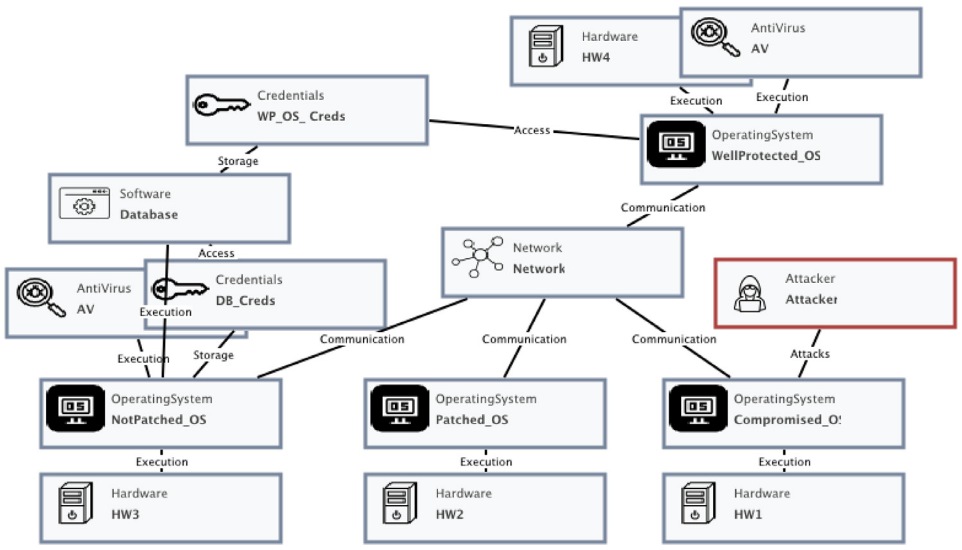

Fig. 1. The example model.

the attack vectors on the Credentials cannot be initiated unless the attacker managed to access it. And if we continue tracing the logic backwards this step is achieved by compromising any Machine that has a storage association to the Credentials.

The other main chunk of the attack logic is found on the OperatingSystem. Here, two main attack vectors are encoded, one is the exploitation of a known vulnerability and the other is the exploitation of a zero-day vulnerability. Analogously to the logic found on the credentials, the patched defense disables the attemptExploitKnownVulnerability step and, thus, the on-following steps in that vector. The final defense mechanism considered in the example is antivirus. This is encoded using the *exist* and *does not exist* steps. Intuitively, if OperatingSystem is executing Software of type AntiVirus, then the attacker can try to bypass the antivirus. Technically, this means that in the attack graph corresponding to the model the node antiVirusProtected will be compromised, and so the nodes representing bypassAVWithKnownVulnExploit and bypassAVWithZeroDayExploit steps will be reachable by the attacker. If there is no AntiVirus being executed by the OperatingSystem, then this node will not be compromised, and so the two aforementioned nodes will not be reachable by the attacker.[3]

Finally, the compromise of a Machine allows that attacker to communicate on the Networks it is connected to, compromised Software can connect the Machine it is executed on, in addition to the attack steps enabled as specified in Machine.

4.2. An example model

In Fig. 1, a small example model created in the above-described language is shown. It contains a single network with four computers on it. In our scenario we consider the computer running the OperatingSystem *WellProtected_OS* being the valuable target we want to protect, and thus the focus of our analysis. The model also contains a computer assumed to have already been compromised by the attacker. Moreover, there are two additional computers, one being patched and the other not being patched, but instead protected by antivirus. The latter computer contains a database in which credentials to the target computer are stored. The target operating system is both patched and guarded by antivirus.

In the following sections, we describe the overall syntax and semantics of MAL. To ease the understanding of MAL, we refer to this example and use it to illustrate the concrete instantiation.

5. The syntax of MAL

Next, we will next present our formalization. Therefore, we will state first the needed preliminaries, followed by the abstract syntax of MAL illustrated by a concrete example based on the previous section.

5.1. Preliminaries

The set of natural numbers is denoted by $\mathbb{N}$. For a set X, the powerset of X is denoted by 2^X, and the set of all functions from X to a set Y, by Y^X. The notation $X \subset Y$ means that X is a subset of Y and $X \neq Y$. If f is a function from $X' \subseteq X$ to Y, then it is a *partial function* from X to Y, denoted $f : X \rightharpoonup Y$; it does not necessarily map every element of X on an element of Y. The domain of f is denoted by $\textbf{\textit{dom}}(f)$. For a tuple or a list l, the function $\pi_i(l)$ returns the ith element of l.

For $p \in [0, 1]$, we use Bernoulli(p) for the probability distribution of a random variable that takes value 1 with probability p, and value 0 with probability $1 - p$. Finally, if ϕ is a probability distribution such as exponential or gamma, then by $x \leftarrow \phi$ we mean that the value of x is obtained by sampling from the distribution ϕ.

5.2. Abstract syntax

To define the abstract syntax of MAL, let

- Classifiers be the set of all *classifiers* (asset names),
- Steps denote the set of all the *steps* that might appear in languages,
- Roles denote the set of all the *roles* that might appear in languages,
- Types denote the set $\{\&, |, \#, E, !E\}$ of *types of steps*,
- $\mathcal{F}$ denote the set of all probability distributions.

The types & and | are used for annotating *AND attack steps* and *OR attack steps*, respectively. Intuitively, a step of type & can be executed only if all of its parent steps have been executed. To be able

[3] The reason why we use both *exist* and *does not exist* steps here is to both enable and disable attack vectors. In the case of an unpatched machine, it is still possible to deploy a zero-day vulnerability exploit, whereas bypassing an antivirus system cannot be done if it is not present.

to execute an attack step of type |, the attacker needs to execute at least one of its parent steps. Steps of type # are *defense steps*. These represent security measures that can be implemented in the system to counter particular attack steps. The last two types symbols, E and $!E$, stand for *exists* and *does not exist* quantifiers. The steps of these types are used for modeling dependencies between availability of some attack steps to the attacker and the existence of some assets or lack thereof.

Relations between assets and steps within the assets are described with terms generated by the grammar

$$\begin{aligned} term_r ::= \;& role \,|\, (term_r \wedge term_r) \,|\, (term_r \vee term_r) \\ & |\, (term_r \setminus term_r) \,|\, (term_r.term_r) \\ & |\, term_r[class], \\ terms_s ::= \;& step \,|\, term_r.step, \end{aligned} \tag{1}$$

with $role \in \texttt{Roles}, class \in \texttt{Classifiers}$ and $step \in \texttt{Steps}$. The set of all terms of the form $term_r$ and the set of all terms of the form $term_s$ are denoted by $Term_r$ and $Term_s$, respectively. The parentship relation between attack steps and the relation of defense steps countering attack steps are specified using the terms in $Term_s$. The terms in $Term_r$ are used for expressing the preconditions for the steps of types E and $!E$.

A language expressed with MAL is a tuple

$$L = (\text{classifiers}_L, \text{assetSteps}_L, \text{stepSpec}_L, \\ \text{ends}_L, \text{mults}_L, \text{roles}_L, \text{extends}_L),$$

where

1. classifiers_L is a set of classifiers of the language. It can be partitioned into sets of *assets* and *associations*, denoted by assets_L and associations_L, respectively. Among the assets, there are *abstract assets*, the set of which we denote by abstrAssets_L. That is,

$$\text{classifiers}_L \subseteq \texttt{Classifiers},$$
$$\text{classifiers}_L = \text{assets}_L \cup \text{associations}_L,$$
$$\text{assets}_L \cap \text{associations}_L = \emptyset,$$
$$\text{abstrAssets}_L \subseteq \text{assets}_L.$$

2. assetSteps_L is a function assigning to each of the assets in L a set of steps related to this asset. That is,

$$\text{assetSteps}_L : \text{assets}_L \to 2^{\texttt{Steps}}.$$

3. stepSpec_L is a function specifying properties of steps in the context of particular assets: their types, their prerequisites and consequences, and the probability distributions of their TTCs. That is,

$$\text{stepSpec}_L : \text{assets}_L \times \texttt{Steps} \rightharpoonup \\ \texttt{Types} \times 2^{Term_r} \times 2^{Term_s} \times \mathcal{F},$$

and stepSpec_L is defined for every pair $(asset, step)$ such that $asset \in \text{assets}_L$ and $step \in \text{assetSteps}_L(asset)$.
For an asset $asset$ in assets_L and a step $step \in \text{assetSteps}_L(asset)$, we let

$$\text{type}_\text{L}(asset, step) := \\ \pi_1(\text{stepSpec}_L(asset, step))$$

to denote the type of step $step$ in the context of the asset $asset$. The following is required of stepSpec_L: for every asset $asset \in \text{assets}_L$ and every step $step \in \text{assetSteps}_L(asset)$ satisfying $\text{stepSpec}_L(asset, step) = (t, R, S, f)$, it must hold that
 - if $step' \in S$, then

$$step' \in \text{assetSteps}_L(asset), \text{ and} \\ \text{type}_\text{L}(asset, step') \in \{\&, |\},$$

 - if $t \in \{\&, |, \#\}$, then $R = \emptyset$,
 - if $t \in \{E, !E\}$, then $R \neq \emptyset$.

Intuitively, the first of the above requirements states that if there is a relation between two steps (i.e., $step$ is a parent attack step of an attack step $step'$, or $step$ is a defense step countering an attack step $step'$) that does not exploit any associations between assets, then both these steps are related to the same asset. The remaining two provide a distinction between the *exists* and *does not exist* steps and the other steps. Theses two steps specify conditions under which some other steps are available for the attacker to execute, the conditions being the presence or lack of some instances of assets. Such instances are specified using terms in $Term_r$, stored in the set R. Thus, the set R is required to be non-empty for these two steps, and to be empty for the other steps.

4. ends_L is a function specifying for a given association the assets related to each other through the association. That is,

$$\text{ends}_L : \text{associations}_L \to \text{assets}_L \times \text{assets}_L.$$

5. mults_L is a function specifying multiplicities of association ends, i.e., the possible numbers of assets instances in association instances. It does so by assigning to each of the associations a pair of non-empty sets of natural numbers. That is,

$$\text{mults}_L : \text{associations}_L \to \\ (2^{\mathbb{N}\setminus\{0\}} \setminus \{\emptyset\}) \times (2^{\mathbb{N}\setminus\{0\}} \setminus \{\emptyset\}).$$

6. roles_L is a labeling function specifying the roles of the assets in associations:

$$\text{roles}_L : \text{associations}_L \to \texttt{Roles} \times \texttt{Roles}.$$

7. extends_L describes the *extension relation*: if an asset is an extension of another asset, then it is mapped by extends_L to the latter. If an asset extends another asset, then it inherits all steps of the latter, together with their types. The remaining properties of the steps might be different. That is,

$$L.\text{extends}_L : \text{assets}_L \rightharpoonup \text{assets}_L,$$

for every $asset \in \textbf{\textit{dom}}(\text{extends}_L)$ it holds that

$$\text{assetSteps}_L(asset) \supseteq \\ \text{assetSteps}_L(\text{extends}_L(asset)),$$

and for every $step \in \text{assetSteps}_L(\text{extends}_L(asset))$ it holds that

$$\text{type}_\text{L}(asset, step) = \\ \text{type}_\text{L}(\text{extends}_L(asset), step).$$

Furthermore, intuitively speaking, the extending asset participates in all associations in which the extended asset participates, i.e., for every $asset \in \textbf{\textit{dom}}(\text{extends}_L)$ and every $assoc \in \text{associations}_L$ satisfying

$$\text{extends}_L(asset) = \pi_i(\text{ends}_L(assoc))$$

for some $i \in \{1, 2\}$, there is an association $assoc' \in \text{associations}_L$ such that

$$\begin{aligned} \text{mults}_L(assoc') &= \text{mults}_L(assoc), \\ \text{roles}_L(assoc') &= \text{roles}_L(assoc), \\ \pi_i(\text{ends}_L(assoc')) &= asset, \\ \pi_j(\text{ends}_L(assoc')) &= \pi_j(\text{ends}_L(assoc)), \end{aligned}$$

with j satisfying $\{i, j\} = \{1, 2\}$.

Finally, none of the assets can be an extension of itself (neither direct nor indirect). Formally, for every *asset* $\in$ assets$_L$ and every $n \in \mathbb{N}$, $n \neq 0$, it must hold that

$$\text{extends}_L^n(asset) \neq asset.$$

Example language

For a description employing our formal syntax, let

$$L = (\text{classifiers}_L, \text{assetSteps}_L, \text{stepSpec}_L, \text{ends}_L, \text{mults}_L, \text{roles}_L, \text{extends}_L)$$

be the language presented above. The classifiers of the language L are

$$\text{classifiers}_L = \text{assets}_L \cup \text{associations}_L,$$

with

$$\text{assets}_L = \{\texttt{Machine}, \texttt{Hardware}, \texttt{Software}, \texttt{OperatingSystem}, \texttt{AntiVirus}, \texttt{Network}, \texttt{Credentials}\},$$
$$\text{abstrAssets}_L = \{\texttt{Machine}\}, \text{ and}$$
$$\text{associations}_L = \{\texttt{Execution}, \texttt{Communication}, \texttt{Storage}, \texttt{Access}\}.$$

The extends$_L$ function is defined as

$$\text{extends}_L(\texttt{Hardware}) = \texttt{Machine},$$
$$\text{extends}_L(\texttt{Software}) = \texttt{Machine},$$
$$\text{extends}_L(\texttt{OperatingSystem}) = \texttt{Software},$$
$$\text{extends}_L(\texttt{AntiVirus}) = \texttt{Software}.$$

We will not give the full description of the remaining components of L, but instead provide examples illustrating how this description can be derived from the human-readable one.

Generally, steps related to a particular asset are listed within the curly brackets following the asset's name, in the lines starting with one of the type symbols $\&, |, \#, E, !E$. For instance,

$$\text{assetSteps}_L(\texttt{Machine}) = \{\texttt{connect}, \texttt{authenticate}, \texttt{authCompromise}, \texttt{compromise}\}.$$

In the case of assets extending other assets, the steps having exactly the same properties as in the context of the extended asset are not listed. For example, there are no steps present in the description of the Hardware asset. Since Hardware extends Machine, this fact should be interpreted as

$$\text{assetSteps}_L(\texttt{Hardware}) = \text{assetSteps}_L(\texttt{Machine}).$$

The lines starting with arrows contain the prerequisites (<-; only for the *exists* and *does not exist* steps) and the consequences (->) of steps. The special arrow +> is used for listing additional consequences that a step has in the context of the asset, but not in the context of the parent asset. The probability distributions of steps in the context of particular assets are specified within square brackets; if a distribution is not stated explicitly, then it is assumed that the execution of the step is instantaneous, and so its TTC distribution is Bernoulli(0). For instance,

$$\text{stepSpec}_L(\texttt{Software}, \texttt{compromise}) = (|, \emptyset, \{\texttt{executees.connect}, \texttt{storedCreds.access}, \texttt{networks.communicate}, \texttt{executor.connect}\}, \text{Bernoulli}(0)),$$
$$\text{stepSpec}_L(\texttt{OperatingSystem}, \texttt{antiVirusProtected}) = (E, \{\texttt{executees[AntiVirus]}\}, \{\texttt{bypassAVwithKnownVulnerability}, \texttt{bypassAVwithZeroDayExploit}\}, \text{Bernoulli}(0)).$$

The properties of associations are listed in the block at the end. The asterisk means that any non-zero number of assets of particular type can participate in the association. In other words, the symbol * represents the set $\mathbb{N} \setminus \{0\}$. Hence, the formal description of the Execution association is

$$\text{ends}_L(\texttt{Execution}) = (\texttt{Machine}, \texttt{Software}),$$
$$\text{mults}_L(\texttt{Execution}) = (\{1\}, \mathbb{N} \setminus \{0\}),$$
$$\text{roles}_L(\texttt{Execution}) = (\texttt{executor}, \texttt{executees}),$$

and the formal specification of the remaining associations can be derived analogously.

6. The semantics of MAL

The syntax given in the previous section describes the structure of DSLs created using MAL. In this section, we focus on models that can be expressed using such DSLs, and we define a procedure for automatic transformation of these models into attack graphs.

6.1. Model

The existing instances of a classifier are called its *extent*. If we let Instances denote the set of possible instances of classifiers, then a *model* $\mathcal{M}$ can be seen as a tuple

$$\mathcal{M} = (\text{instances}_\mathcal{M}, \text{ends}_\mathcal{M}, \text{entryPoints}_\mathcal{M}, \text{defenses}_\mathcal{M}),$$

where

1. instances$_\mathcal{M}$ is a partial function that maps classifiers to their extents (intuitively, it specifies which of the instances present in the model are instantiations of which classifiers). That is,
$$\text{instances}_\mathcal{M} : \texttt{Classifiers} \rightharpoonup 2^{\texttt{Instances}}.$$
2. ends$_\mathcal{M}$ is a function that maps each of the instances of associations (links) to a pair of sets of instances of assets (objects) connected by the link. That is,
$$\text{ends}_\mathcal{M} : \texttt{Instances} \rightharpoonup 2^{\texttt{Instances}} \times 2^{\texttt{Instances}}.$$
3. entryPoints$_\mathcal{M}$ is a set of pairs $(instance, step) \in \text{instances}_\mathcal{M} \times$ Steps representing attack steps known to have been already executed by the attacker.
4. defenses$_\mathcal{M}$ is a function
$$\text{defenses}_\mathcal{M} : \text{instances}_\mathcal{M} \times \texttt{Steps} \rightharpoonup \mathcal{F}$$
assigning to each of the defense steps present in the model a Bernoulli probability distribution describing the likelihood of the defense being implemented.

6.2. Model respecting a DSL

If we let Languages and Models denote the set of all languages as defined in Section 5 and the set of all models as defined in Section 6.1, respectively, then for a given language $L \in$ Languages and a model $\mathcal{M} \in$ Models we say that $\mathcal{M}$ *respects* L, denoted

$$Resp(L, \mathcal{M}),$$

if and only if:

1. $\mathcal{M}$ contains instantiations of classifiers of L, and no others. That is,

   $$\boldsymbol{dom}(\text{instances}_{\mathcal{M}}) \subseteq \text{classifiers}_L.$$

2. An instance of a given association respects the specification of this association, i.e., it connects the correct numbers of instances of assets appearing in this association. That is, for every association $assoc \in \text{associations}_L$, every link $link \in \text{instances}_{\mathcal{M}}(assoc)$ and for $i \in \{1,2\}$ it holds that

   $$|\pi_i(\text{ends}_{\mathcal{M}}(link))| \in \pi_i(\text{mults}_L(assoc))$$

   and

   $$\pi_i(\text{ends}_{\mathcal{M}}(link)) \subseteq \text{instances}_{\mathcal{M}}(\pi_i(\text{ends}_L(assoc))).$$

3. The extent of an association contains at most one link connecting a given pair of sets of assets instances. That is, for every association $assoc \in \text{associations}_L$ and every two links $link_1, link_2 \in \text{instances}_{\mathcal{M}}(assoc)$ satisfying $link_1 \neq link_2$ it holds that

   $$\pi_1(\text{ends}_{\mathcal{M}}(link_1)) \neq \pi_1(\text{ends}_{\mathcal{M}}(link_2))$$

   or

   $$\pi_2(\text{ends}_{\mathcal{M}}(link_1)) \neq \pi_2(\text{ends}_{\mathcal{M}}(link_2)).$$

4. An instance of an asset that extends another asset is also an instance of the latter. That is, if $asset' = \text{extends}_L(asset)$, with $asset, asset' \in \text{assets}_L$, it holds that $\text{instances}_{\mathcal{M}}(asset) \subseteq \text{instances}_{\mathcal{M}}(asset')$.
5. Every instance of an abstract asset in the model $\mathcal{M}$ is also an instance of a non-abstract asset extending the former. That is, for every abstract asset $abstrAsset \in \text{abstrAssets}_L$ and every instance $instance \in \text{instances}_{\mathcal{M}}(abstrAsset)$ there is an asset $asset \in \text{assets}_L$ and a number $n \in \mathbb{N}$, $n > 0$, such that

   $$\text{extends}_L^n(asset) = abstrAsset,$$
   $$instance \in \text{instances}_{\mathcal{M}}(asset).$$

6. Every entry point describes an attack step that can be performed in the modeled infrastructure. That is, if $(instance, step) \in \text{entryPoints}_{\mathcal{M}}$, then there is an asset $asset \in \text{assets}_L$ such that $instance \in \text{instances}_{\mathcal{M}}(asset)$, $step \in \text{assetSteps}_L(asset)$ and $\text{type}(asset, step) \in \{\&, |\}$.
7. The $\text{defenses}_{\mathcal{M}}$ function assigns a Bernoulli probability distribution to each of the defense steps related to assets instantiated in the model, i.e.,

   $$\begin{aligned}&\boldsymbol{dom}(\text{defenses}_{\mathcal{M}}) = \\ &\{(instance, step) : \text{ there is } asset \in \text{assets}_L \\ &\text{and } step \in \text{assetSteps}_L(asset) \text{ such that} \\ &instance \in \text{instances}_{\mathcal{M}}(asset) \text{ and} \\ &\text{type}(asset, step) = \#\},\end{aligned}$$

   and for every $(instance, step) \in \boldsymbol{dom}(\text{defenses}_{\mathcal{M}})$ it holds that

   $$\text{defenses}_{\mathcal{M}}((instance, step)) \in \{\text{Bernoulli}(p) : p \in [0,1]\}.$$

Example model The specification of the model employing the notation previously introduced can be easily derived from the above description and from Fig. 1. Denoting the model by $\mathcal{M}$, we have for example

$$\text{instances}_{\mathcal{M}}(\texttt{Hardware}) = \{\text{HW1, HW2, HW3, HW4}\}$$

and

$$\text{entryPoints}_{\mathcal{M}} = \{(\text{Compromised\_OS}, \texttt{compromise})\}.$$

The fact that some defenses are functional and others are not, is modeled by assigning to the defenses appropriate Bernoulli probability distributions. For instance,

$$\text{defenses}_{\mathcal{M}}(\text{Patched\_OS}, \texttt{patched}) = \text{Bernoulli}(1),$$
$$\text{defenses}_{\mathcal{M}}(\text{NotPatched\_OS}, \texttt{patched}) = \text{Bernoulli}(0).$$

6.3. Evaluation of terms for models

Terms generated using the grammar (1) that appear in a DSL provide a general description of relations between assets and steps related to assets. In the context of a specific model, this description will be translated into direct relations between particular instances of assets and particular steps. The functions that will be used for performing such translation rely on the intuitively obvious notion of *an instance playing a role in a link.*

Definition 1. Let $L \in \texttt{Languages}$ be a language, let $\mathcal{M}$ be a model respecting L and let $role \in \texttt{Roles}$. For an instance $instance \in \text{instances}_{\mathcal{M}}(asset)$ of an asset $asset \in \text{assets}_L$ and an instance $link \in \text{instances}_{\mathcal{M}}(assoc)$ of an association $assoc \in \text{associations}_L$, we say that *instance* plays role *role* in *link* if there is $i \in \{1,2\}$ such that $\pi_i(\text{roles}_L(assoc)) = role$ and $instance \in \pi_i(\text{ends}_{\mathcal{M}}(link))$.

For an instance of an asset and a term $term_r \in Term_r$, the objects reachable from the instance in a particular model via links specified by $term_r$ are determined using the function

$$\begin{aligned}eval_R :& \texttt{Languages} \times \texttt{Models} \times \texttt{Instances} \times Term_r \\ &\to \texttt{Instances}\end{aligned}$$

defined recursively in Table 1. For determining steps within such reachable objects that a step belonging to the instance leads to, the function

$$\begin{aligned}eval_S :& \texttt{Languages} \times \texttt{Models} \times \texttt{Instances} \\ &\times \texttt{Steps} \times Term_s \\ &\to \texttt{Instances},\end{aligned}$$

defined in Table 2 is used.

6.4. Semantics

The semantics defined in this section assigns a function describing transformation of models specified with a MAL-created DSL into attack graphs. By an attack graph, we mean a tuple $(V, A, \text{type}, \text{ttc})$, where V is a set of *nodes*, $A \subseteq V \times V$ is a set of *arcs*, and $\text{type} : V \to \{|, \&, \#\}$ and $\text{ttc} : V \to \mathcal{F}$ are functions associating nodes with their types and probability distributions, respectively. Let $\mathcal{AG}$ denote the set of all attack graphs.

For a language $L \in \texttt{Languages}$, let f_L be the function mapping models respecting L on attack graphs, i.e.,

$$f_L : \{\mathcal{M} \in \texttt{Models} : Resp(L, \mathcal{M})\} \to \mathcal{AG},$$

defined as follows. For a model $\mathcal{M}$ respecting L, the image $f_L(\mathcal{M})$ of $\mathcal{M}$ by f_L is the attack graph $(V_1 \cup V_2, A, \text{type}, \text{ttc})$ defined by

$$\begin{aligned}V_1 =& \{(instance, step) : \text{ there is } asset \in \text{assets}_L \text{ s.t.} \\ &instance \in \text{instances}_{\mathcal{M}}(asset), \\ &step \in steps_L(asset) \\ &\text{and type}_L(asset, step) \in \{|, \&, \#\}\}, \\ V_2 =& \{(instance, step) : \text{ there is } asset \in \text{assets}_L \text{ st} \\ &instance \in \text{instances}_{\mathcal{M}}(asset), \\ &step \in steps_L(asset),\end{aligned}$$

Table 1
Evaluation rules for terms belonging to $Term_r$.

$eval_R(L, \mathcal{M}, instance, role) =$	$\{instance'$: there is $assoc \in$ associations$_L$ and $link \in$ instances$_{\mathcal{M}}(assoc)$ such that $instance'$ plays role $role$ in $link$ and $instance$ plays the other role in $link\}$
$eval_R(L, \mathcal{M}, instance, (term_1 \wedge term_2)) =$	$eval_R(L, \mathcal{M}, instance, term_1) \cap eval_R(L, \mathcal{M}, instance, term_2)$
$eval_R(L, \mathcal{M}, instance, (term_1 \vee term_2)) =$	$eval_R(L, \mathcal{M}, instance, term_1) \cup eval_R(L, \mathcal{M}, instance, term_2)$
$eval_R(L, \mathcal{M}, instance, (term_1 \setminus term_2)) =$	$eval_R(L, \mathcal{M}, instance, term_1) \setminus eval_R(L, \mathcal{M}, instance, term_2)$
$eval_R(L, \mathcal{M}, instance, (term_1.term_2)) =$	$\bigcup_{instance' \in eval_R(L, \mathcal{M}, instance, term_1)} eval_R(L, \mathcal{M}, instance', term_2)$
$eval_R(L, \mathcal{M}, instance, term_r[class]) =$	$eval_R(L, \mathcal{M}, instance, term_r) \cap$ instances$_{\mathcal{M}}(class)$

Table 2
Evaluation rules for terms belonging to $Term_s$.

$eval_S(L, \mathcal{M}, instance, step_1, step_2) =$	$\{(instance, step_2)\}$
$eval_S(L, \mathcal{M}, instance, step_1, term_r.step_2) =$	$\{(instance', step_2)$: $instance' \in eval_R(L, \mathcal{M}, instance, term_r)$ and $step_2 \in steps_L(asset)$ for every $asset$ such that $instance' \in$ instances$_{\mathcal{M}}(asset)\}$

$$\begin{aligned}
&\text{type}_L(asset, step) = E, \text{ and there is}\\
&term_r \in \pi_2(\text{stepSpec}_L(asset, step)) \text{ such that}\\
&eval_R(L, \mathcal{M}, instance, term_r) \neq \emptyset\}\\
&\cup\\
&\{(instance, step) : \text{ there is } asset \in \text{assets}_L \text{ st}\\
&instance \in \text{instances}_{\mathcal{M}}(asset),\\
&step \in steps_L(asset),\\
&\text{type}_L(asset, step) = !E, \text{ and there is}\\
&term_r \in \pi_2(\text{stepSpec}_L(asset, step)) \text{ such that}\\
&eval_R(L, \mathcal{M}, instance, term_r) = \emptyset\},
\end{aligned}$$

$$\begin{aligned}
A = \{\big((instance, step), (instance', step')\big) \in V \times V :\\
&\text{there is } asset \in \text{assets}_L \text{ such that}\\
&instance \in \text{instances}_{\mathcal{M}}(asset) \text{ and}\\
&step \in \text{steps}_L(asset),\\
&\text{and there is } term_s \in \pi_3(\text{stepSpec}_L(asset, step))\\
&\text{such that}\\
&(instance', step') \in\\
&eval_S(L, \mathcal{M}, instance, step, term_s)\},
\end{aligned}$$

$$\text{type}((instance, step)) = \begin{cases} \text{type}_L(asset, step), & \text{if } instance \in \text{instances}_{\mathcal{M}}(asset)\\ & \text{and } (instance, step) \in V_1,\\ \&, & \text{otherwise,} \end{cases}$$

and

$$\text{ttc}((instance, step)) = \pi_4(\text{stepSpec}_L(asset, step)),$$

where $asset$ in the last formula is the unique asset such that $instance \in \text{instances}_{\mathcal{M}}(asset)$ and $instance \notin \text{instances}_{\mathcal{M}}(asset')$ for every $asset' \in \text{assets}_L$ satisfying $\text{extends}_L^n(asset') = asset$ for some $n \in \mathbb{N}$.

The semantics of a language $L \in \texttt{Languages}$ is defined as

$$[\![L]\!] := f_L.$$

7. An application of the MAL semantics for security analysis

The MAL semantics defined in the previous section transforms a MAL model into an attack graph describing the possible behavior of the attacker in the modeled infrastructure. For the purpose of security analysis various methods can be applied to the attack graphs, representing different types of attacker behavior. We could for instance imagine a rational attacker choosing the overall shortest path, or one that chooses the cheapest next step, or just attacking at random. Although, the goal of this work is to provide formal foundations for generating attack graphs from MAL-based DSLs rather than the attack graphs per se are used for quantitative security analysis, we briefly sketch one such method here. This section describes an implementation found in the MAL tooling[4] as of the date of writing, but the objective of this section is merely to hint towards the possibilities of MAL-based attack graphs rather than to develop a full-fledged security analysis framework.

The attack simulation method considered here consists of four stages. In the first of them, it is established which of the defense measures are in place. This is done by sampling from the defense steps' Bernoulli distributions. In the second stage, this knowledge is combined with the information on the attack steps already performed by the attacker (with some of them specified explicitly in the model as the entry points, and others derived from the preconditions of E and $!E$ steps), and used for determining attack steps reachable by the attacker. For such steps, TTC values are sampled from their corresponding probability distributions. Finally, these values are propagated throughout the attack graph, yielding the overall time needed for compromising particular assets.

Formally, this method can be described as follows: Let $L \in \texttt{Languages}$ be a language, let $\mathcal{M}$ be a model respecting L and let$(V_1 \cup V_2, A, \text{type}, \text{ttc}) = [\![L]\!](\mathcal{M})$ be the attack graph representing security scenario modeled with $\mathcal{M}$. To reason conveniently about the possible defenses against an attack step, we set

$$\text{Defs}(v) = \{v' \in V_1 : (v', v) \in A, \text{type}(v') = \#\},$$

for $v \in V_1$ satisfying $\text{type}(v) \in \{|, \&\}$.

For describing which of the defenses are implemented, define the function

$$imp : \{v \in V_1 : \text{type}(v) = \#\} \to \{0, 1\}$$

by setting $imp(v) \leftarrow \text{defenses}_{\mathcal{M}}(v)$.

Finally, we say that nodes in $\text{entryPoints}_{\mathcal{M}} \cup V_2$ are the *compromised nodes*.

A node $v \in V_1$ satisfying $\text{type}(v) \in \{|, \&\}$ is *reachable by the attacker* if it satisfies all of the following conditions.

1. There is a path from at least one of the compromised nodes to v.

[4] Avialable at https://mal-lang.org.

2. None of the defenses attached to v is implemented, i.e., $imp(v') = 0$ for every $v' \in \text{Defs}(v)$.
3. If $\text{type}(v) = \&$, then every parent of v is reachable by the attacker.
4. If $\text{type}(v) = |$, then at least one of the parents of v is reachable by the attacker.

For a node v reachable by the attacker, we use $\text{T}_{\text{loc}}(v)$ and $\text{T}_{\text{glob}}(v)$ to denote the local and the global time to compromise, respectively. Furthermore, the parents of v that are reachable by the attacker are denoted by $\text{Parents}(v)$. The values of the local time to compromise are obtained by setting

$$\text{T}_{\text{loc}}(v) = \begin{cases} 0, & \text{if } v \in \text{entryPoints}_{\mathcal{M}} \cup V_2, \\ \leftarrow \text{ttc}(v) & \text{otherwise.} \end{cases}$$

The global time to compromise, that is, an estimate of a time needed by the attacker for execution of a particular step and all of its prerequisites, is computed as

$$\text{T}_{\text{glob}}(v) = \text{T}_{\text{loc}}(v) + \min_{v' \in \text{Parents}(v)} \text{T}_{\text{glob}}(v'),$$

if $\text{type}(v) = |$, and as

$$\text{T}_{\text{glob}}(v) = \text{T}_{\text{loc}}(v) + \max_{v' \in \text{Parents}(v)} \text{T}_{\text{glob}}(v'),$$

if $\text{type}(v) = \&$.

The motivation behind the choice of operations for global time to compromise computations is as follows. Execution of an attack step takes time equal to the step's local time to compromise. In the case of an OR attack step, its execution requires also executing at least one of its parents, and in the case of an ANDstep – all of the parents. Thus, the minimal of the global times to compromise of the parents is added in computations for an OR node, and the maximal – in computations for an ANDnode.[5]

There are many different ways to calculate the TTC based on preferences for a certain way to for example calculate the shortest path between the attack surface and the target for which the TTC should be calculated. All these different calculations adhere still to the here described semantics of MAL. Accordingly, we do not want to restrain concrete implementations of MAL to a certain implementation how to calculate or use the generated attack graph.

Example for time to compromise Using existing tools, which transform MAL models into attack graphs and use the algorithm sketched in Section 7 for computing global TTC values, we obtained the optimal attack vector depicted in Fig. 2. In the figure, we see that the attacker from the *Network* can *Connect* to the *WellProtected_OS* and the *NotPatched_OS*. On the latter they are exploiting a known vulnerability, as the OS is not patched. From there, the *Database* is compromised since its credentials is directly accessible on the OS. In the database, the encrypted credentials to the *WellProtected_OS* are found, and then *Crack*ed. With both a *Connection* and the *Compromise*d credentials, the attacker is finally able to *Compromise* the *WellProtected_OS* by means of authentication. In Fig. 3, the empirical cumulative distribution function of the global TTC for the target is plotted. It has been derived from results of a hundred runs of the algorithm presented in Section 7, employing the local TTC distributions specified in the language.

8. Conclusions

In this article, we have presented a formal syntax and semantics of the Meta Attack Language (MAL). As with other domain specific languages, MAL enables security engineers and researchers to develop languages that realize many advantages as compared to other more open and general-purpose approaches. With a tailored semantic space, it is easier for language developers to understand, develop, validate, and maintain their work. The same also applies to the security engineers using the languages. Of course, just like most software development this requires good coding discipline from the developers and users with documentation of e.g., design decisions and overall working of the encoded logic.

Within the security domain, MAL also enables language developers to both take on a top-down approach with coarsely defined assets and attack steps that (likely) includes plenty of probabilistic time to compromise distributions with high uncertainties (as the example in this paper), as well as a bottom-up approach with a narrower domain but more determinism stemming from logical necessities in the system configurations.

This article provides no evaluation of the proposed formalisms. However, MAL has proven itself as useful in several occasions as manifested in MAL DSLs for different domains such as general

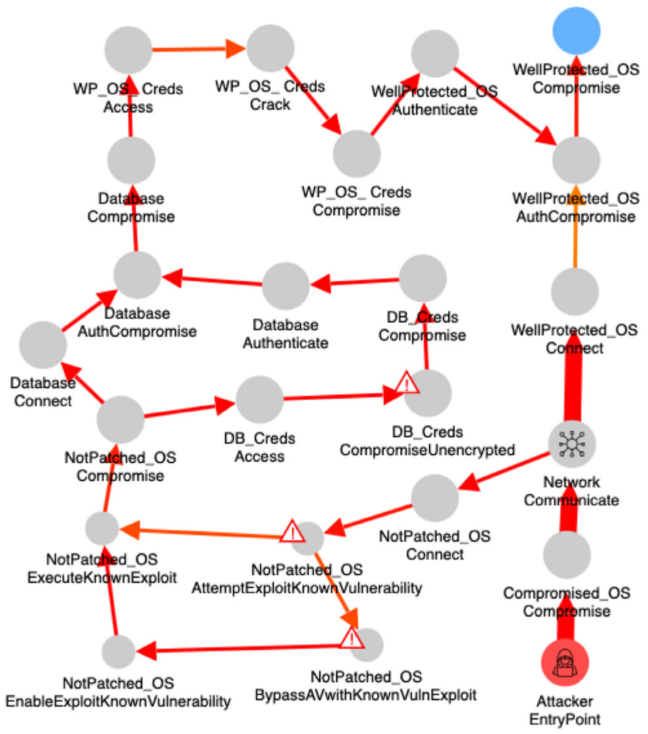
WP_OS_Creds Access
WP_OS_Creds Crack
WellProtected_OS Authenticate
WellProtected_OS Compromise
Database Compromise
WP_OS_Creds Compromise
WellProtected_OS AuthCompromise
Database AuthCompromise
Database Authenticate
DB_Creds Compromise
WellProtected_OS Connect
Database Connect
NotPatched_OS Compromise
DB_Creds Access
DB_Creds CompromiseUnencrypted
Network Communicate
NotPatched_OS Connect
NotPatched_OS ExecuteKnownExploit
NotPatched_OS AttemptExploitKnownVulnerability
Compromised_OS Compromise
NotPatched_OS EnableExploitKnownVulnerability
NotPatched_OS BypassAVwithKnownVulnExploit
Attacker EntryPoint

Fig. 2. The shortest attack vector to the target operating system.

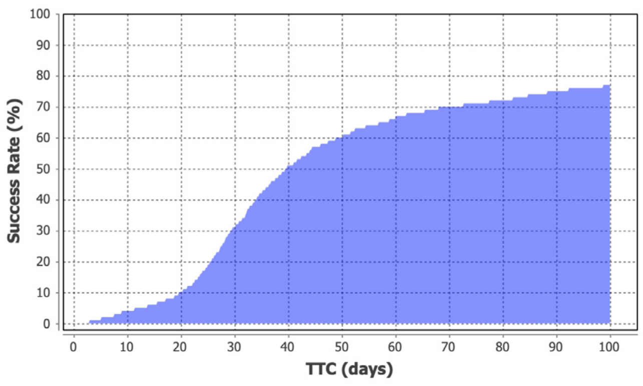

Fig. 3. The cumulative distribution function of the global TTC for the target attack step *Compromise* on the asset *WellProtected_OS*.

[5] Should all the parents of each of the ANDattack steps be independent, one could consider taking the sum of their times to compromise instead of the maximal value. In practice, however, such independence cannot be guaranteed, and the obtained result would be a potentially hazardous overestimate of the actual time to compromise.

IT (Katsikeas et al., 2020), power (Hacks et al., 2020; Rencelj Ling and Ekstedt, 2021), or vehicles (Katsikeas et al., 2022). Additionally, MAL was successfully facilitated in different projects like EnergyShield[6] or SOCCRATES[7].

Declaration of Competing Interest

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

CRediT authorship contribution statement

Wojciech Wideł: Conceptualization, Methodology, Writing – original draft. **Simon Hacks:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Mathias Ekstedt:** Conceptualization, Methodology, Supervision, Funding acquisition, Writing – original draft. **Pontus Johnson:** Conceptualization, Funding acquisition. **Robert Lagerström:** Conceptualization, Funding acquisition.

Data availability

Data will be made available on request.

Acknowledgments

This work has received funding from the Swedish Civil Contingencies Agency through the research centre Resilient Information and Control Systems (RICS), European Union's H2020 research and innovation programme under the Grant Agreements no. 833481 and no. 832907, and the Swedish Governmental Agency for Innovation Systems (Vinnova).

References

Abdulganiyyi, N., Ibrahim, N., 2014. Semantic abstraction of class diagram using logical approach. In: 2014 4th World Congress on Information and Communication Technologies (WICT 2014), pp. 251–256. doi:10.1109/WICT.2014.7077274.

Alam, M., Breu, R., Hafner, M., 2007. Model-driven security engineering for trust management in SECTET. JSW 2 (1), 47–59.

Aldea, A., Hacks, S., 2022. Analyzing enterprise architecture models by means of the meta attack language. In: Advanced Information Systems Engineering: 34th International Conference, CAiSE 2022, Proceedings. Springer, pp. 423–439. Leuven, Belgium, June 6–10

Almorsy, M., Grundy, J., 2014. SecDSVL: a domain-specific visual language to support enterprise security modelling. In: Software Engineering Conference (ASWEC). IEEE, pp. 152–161. 2014 23rd Australian

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

Basin, D., Clavel, M., Egea, M., 2011. A decade of model-driven security. In: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies. ACM, pp. 1–10.

Basin, D., Doser, J., Lodderstedt, T., 2006. Model driven security: from UML models to access control infrastructures. ACM Trans. Softw. Eng. Methodol. (TOSEM) 15 (1), 39–91.

Chu, M., Ingols, K., Lippmann, R., Webster, S., Boyer, S., 2010. Visualizing attack graphs, reachability, and trust relationships with NAVIGATOR. In: Proc. of the 7th Int. Symp. on Visualization for Cyber Security. ACM, pp. 22–33.

Costal, D., Gómez, C., Guizzardi, G., 2011. Formal semantics and ontological analysis for understanding subsetting, specialization and redefinition of associations in UML. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (Eds.), Conceptual Modeling – ER 2011. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 189–203.

Ekstedt, M., Johnson, P., Lagerström, R., Gorton, D., Nydrén, J., Shahzad, K., 2015. securiCAD by foreseeti: a CAD tool for enterprise cyber security management. In: Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International. IEEE, pp. 152–155.

Enjo, H., Tanabu, M., Iijima, J., 2009. A step toward foundation of class diagram algebra for enterprise service systems. In: 2009 6th International Conference on Service Systems and Service Management, pp. 456–460. doi:10.1109/ICSSSM.2009.5174926.

Feinerer, I., Salzer, G., 2014. Numeric semantics of class diagrams with multiplicity and uniqueness constraints. Softw. Syst. Model. 13 (3), 1167–1187. doi:10.1007/s10270-012-0294-4.

Frigault, M., Wang, L., Singhal, A., Jajodia, S., 2008. Measuring network security using dynamic Bayesian network. In: Proc. of the 4th ACM Workshop on Quality of Protection. ACM, pp. 23–30.

Ghosh, N., Chokshi, I., Sarkar, M., Ghosh, S.K., Kaushik, A.K., Das, S.K., 2015. NetSecuritas: an integrated attack graph-based security assessment tool for enterprise networks. In: Proc. of the 2015 Int. Conf. on Distributed Computing and Networking. ACM, p. 30.

Hacks, S., Katsikeas, S., 2021. Towards an ecosystem of domain specific languages for threat modeling. In: International Conference on Advanced Information Systems Engineering. Springer, pp. 3–18.

Hacks, S., Katsikeas, S., Ling, E., Lagerström, R., Ekstedt, M., 2020. PowerLang: a probabilistic attack simulation language for the power domain. Energy Inf. 3 (1), 1–17.

Hacks, S., Katsikeas, S., Rencelj Ling, E., Xiong, W., Pfeiffer, J., Wortmann, A., 2022. Towards a systematic method for developing meta attack language instances. In: Enterprise, Business-Process and Information Systems Modeling: 23rd International Conference, BPMDS 2022 and 27th International Conference, EMMSAD 2022, Held at CAiSE 2022, Proceedings. Springer, pp. 139–154. Leuven, Belgium, June 6–7, 2022

Hacks, S., Lagerström, R., Ritter, D., 2021. Towards automated attack simulations of BPMN-based processes. In: 2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC). IEEE, pp. 182–191.

Hacks, S., Persson, L., Hersén, N., 2023. Measuring and achieving test coverage of attack simulations extended version. Softw. Syst. Model. 22 (1), 31–46.

Hafner, M., Breu, R., Agreiter, B., Nowak, A., 2006. SECTET: An extensible framework for the realization of secure inter-organizational workflows. Internet Res. 16 (5), 491–506.

Holm, H., Shahzad, K., Buschle, M., Ekstedt, M., 2015. P^2CySeMoL: predictive, probabilistic cyber security modeling language. IEEE Trans. Dependable Secure. Comput. 12 (6), 626–639. doi:10.1109/TDSC.2014.2382574.

Homer, J., Zhang, S., Ou, X., Schmidt, D., Du, Y., Rajagopalan, S.R., Singhal, A., 2013. Aggregating vulnerability metrics in enterprise networks using attack graphs. J. Comput. Secur. 21 (4), 561–597.

Ingols, K., Chu, M., Lippmann, R., Webster, S., Boyer, S., 2009. Modeling modern network attacks and countermeasures using attack graphs. In: Computer Security Applications Conference, 2009. ACSAC'09. Annual. IEEE, pp. 117–126.

Ivanova, M.G., Probst, C.W., Hansen, R.R., Kammüller, F., 2015. Transforming graphical system models to graphical attack models. In: Graphical Models for Security - Second International Workshop, GraMSec 2015, pp. 82–96. Verona, Italy, July 13, 2015, Revised Selected Papers

Jackson, E., Sztipanovits, J., 2009. Formalizing the structural semantics of domain-specific modeling languages. Softw. Syst. Model. 8 (4), 451–478. doi:10.1007/s10270-008-0105-0.

Jiang, T., Wang, X., 2012. Formalizing domain-specific metamodeling language XMML based on first-order logic. J. Softw. 7 (6). doi:10.4304/jsw.7.6.1321-1328.

Johnson, P., Lagerström, R., Ekstedt, M., 2018. A meta language for threat modeling and attack simulations. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. ACM, New York, NY, USA doi:10.1145/3230833.3232799.

Johnson, P., Vernotte, A., Ekstedt, M., Lagerström, R., 2016. pwnPr3d: an attack-graph-driven probabilistic threat-modeling approach. In: Availability, Reliability and Security (ARES), 2016 11th International Conference on. IEEE, pp. 278–283.

Jürjens, J., 2002. UMLsec: extending UML for secure systems development. In: International Conference on The Unified Modeling Language. Springer, pp. 412–425.

Jürjens, J., 2005. Secure Systems Development with UML. Springer Science & Business Media.

Katsikeas, S., Hacks, S., Johnson, P., Ekstedt, M., Lagerström, R., Jacobsson, J., Wällstedt, M., Eliasson, P., 2020. An attack simulation language for the it domain. In: International Workshop on Graphical Models for Security. Springer, pp. 67–86.

Katsikeas, S., Johnson, P., Hacks, S., Lagerström, R., 2019. Probabilistic modeling and simulation of vehicular cyber attacks: an application of the meta attack language. In: Proceedings of the 5th International Conference on Information Systems Security and Privacy.

Katsikeas, S., Johnsson, P., Hacks, S., Lagerström, R., 2022. VehicleLang: a probabilistic modeling and simulation language for modern vehicle IT infrastructures. Comput. Secur. 117, 102705.

Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P., 2010. Foundations of attack–defense trees. In: International Workshop on Formal Aspects in Security and Trust. Springer, pp. 80–95.

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

Kotenko, I., Doynikova, E., 2014. Evaluation of computer network security based on attack graphs and security event processing. J. Wirel. Mob. Netw. Ubiquitous Comput. Dependable Appl. (JoWUA) 5 (3), 14–29.

Lund, M.S., Solhaug, B., Stølen, K., 2010. Model-Driven Risk Analysis: The CORAS Approach. Springer Science & Business Media.

Mauw, S., Oostdijk, M., 2005. Foundations of attack trees. In: International Conference on Information Security and Cryptology. Springer, pp. 186–198.

Mouratidis, H., Giorgini, P., Manson, G., Philp, I., et al., 2002. A natural extension of tropos methodology for modelling security. In: Proceedings Agent Oriented Methodologies Workshop, Annual ACM Conference on Object Oriented Programming, Systems, Languages (OOPSLA). Citeseer. Seattle-USA

[6] https://energy-shield.eu/

[7] https://www.soccrates.eu/

Noel, S., Elder, M., Jajodia, S., Kalapa, P., O'Hare, S., Prole, K., 2009. Advances in topological vulnerability analysis. In: Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology, pp. 124–129. doi:10.1109/CATCH.2009.19.

Ou, X., Govindavajhala, S., Appel, A.W., 2005. MulVAL: a logic-based network security analyzer. USENIX Security.

Ou, X., Singhal, A., 2011. Attack graph techniques. Quant. Secur. Risk Assess. Enterprise Netw. doi:10.1007/978-1-4614-1860-3_2.

Paja, E., Dalpiaz, F., Giorgini, P., 2015. Modelling and reasoning about security requirements in socio-technical systems. Data Knowl. Eng. 98, 123–143.

Pinchinat, S., Acher, M., Vojtisek, D., 2014. Towards synthesis of attack trees for supporting computer-aided risk analysis. In: Software Engineering and Formal Methods - SEFM 2014 Collocated Workshops: HOFM, SAFOME, OpenCert, MoKMaSD, WS-FMDS, pp. 363–375. Grenoble, France, September 1–2, 2014, Revised Selected Papers

Poolsappasit, N., Dewri, R., Ray, I., 2012. Dynamic security risk management using Bayesian attack graphs. IEEE Trans. Dependable Secure Comput. 9 (1), 61–74. doi:10.1109/TDSC.2011.34.

Rencelj Ling, E., Ekstedt, M., 2021. Generating threat models and attack graphs based on the IEC 61850 system configuration description language. In: Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems, pp. 98–103.

Rencelj Ling, E., Ekstedt, M., 2022. Estimating the time-to-compromise of exploiting industrial control system vulnerabilities. In: 8th International Conference on Information Systems Security and Privacy-ICISSP, Vol. 1, pp. 96–107.

Schneier, B., 1999. Attack trees. Dr. Dobb's J. 24 (12), 21–29.

Schneier, S., 2000. Lies: Digital Security in a Networked World Vol. 21, 318–333.

Sommestad, T., Ekstedt, M., Holm, H., 2013. The cyber security modeling language: a tool for assessing the vulnerability of enterprise system architectures. IEEE Syst. J. 7 (3), 363–373.

Stappers, F.P.M., Weber, S., Reniers, M.A., Andova, S., Nagy, I., 2012. Formalizing a domain specific language using SOS: an industrial case study. In: Sloane, A., Aßmann, U. (Eds.), Software Language Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 223–242.

Szlenk, M., 2006. Formal semantics and reasoning about UML class diagram. In: Proceedings of the International Conference on Dependability of Computer Systems, pp. 51–59. doi:10.1109/DEPCOS-RELCOMEX.2006.27.

Vigo, R., Nielson, F., Nielson, H.R., 2014. Automated generation of attack trees. In: IEEE 27th Computer Security Foundations Symposium, CSF 2014, pp. 337–350. Vienna, Austria, 19–22 July, 2014

Wang, L., Jajodia, S., Singhal, A., Cheng, P., Noel, S., 2014. k-Zero day safety: a network security metric for measuring the risk of unknown vulnerabilities. IEEE Trans Dependable Secure Comput 11 (1), 30–44. doi:10.1109/TDSC.2013.24.

Weiss, J.D., 1991. A system security engineering process. In: 14th Annual NCSC/NIST National Computer Security Conference, pp. 572–581.

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

Williams, L., Lippmann, R., Ingols, K., 2008. GARNET: A Graphical Attack Graph and Reachability Network Evaluation Tool. Springer.

Xie, P., Li, J.H., Ou, X., Liu, P., Levy, R., 2010. Using Bayesian networks for cyber security analysis. In: 2010 IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN). IEEE, pp. 211–220.

Xiong, W., Hacks, S., Lagerström, R., 2021. A method for assigning probability distributions in attack simulation languages. Complex Syst. Inf. Model. Q. (26) 55–77.

Wojciech Widel received the PhD degree in mathematics from the AGH University of Science and Technology, Kraków, Poland, in 2017, and the second PhD degree in computer science from INSA Rennes, France, in 2019. He is working as a Statistician at TIER Mobility, Poland. Previously, he worked as a Postdoctoral Researcher with the Division of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden. His research was focused on analysis methods for graphical security models and structural graph theory.

Simon Hacks is senior lecturer for information systems at Department of Computer and Systems Sciences, Stockholm University, Sweden. His research interests lie in the quality of Enterprise Architecture (EA) and its models. Therefore, he has coined the term of EA Debt as an extension of Technical Debt, which provides a holistic view on organizations beyond technical aspects. Moreover, he performs research on the reuse of different models for attack simulations. In line, he facilitates the Meta Attack Language (MAL) that allows to provide domain specific languages (DSL) tailored to the needs of different stakeholders. Finally, he aims to provide tooling around MAL to ensure the quality of the created MAL DSLs. He received his PhD from RWTH Aachen University, Germany in the field of EA modeling.

Pontus Johnson is a professor at the Royal Institute of Technology (KTH) in Stockholm, Sweden. His research interests mainly lie in the area of cyber security and the analysis of architectural models of computer networks - in particular simulating cyber attacks on such networks. Pontus supervises a number of PhD students and holds courses on Ethical Hacking. He is, since 2020, the director of the Center for Cyber Defense and Information Security at KTH. He received his MSc from the Lund Institute of Technology in 1997 and his PhD and Docent titles from the Royal Institute of Technology in 2002 and 2007. He was appointed professor in 2009.

Mathias Ekstedt received the MSc, PhD, and Docent degrees from KTH Royal Institute of Technology, in 1999, 2004, and 2010, respectively. He has been a Professor in industrial information and control systems with the KTH Royal Institute of Technology since 2015. Much of his research interest includes developing formalisms for analyzing security vulnerabilities of system architectures through the means of probabilistic attack/defense graphs. He is the founder and director of KTH's Master program in cybersecurity.

Robert Lagerström is a professor at the School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Sweden. His research is focused on enterprise security, threat modeling, attack simulations, vehicle security, and viable cities. Robert is a member of the Young Academy of Sweden and one of the founders of foreseeti.