A probabilistic attack simulation language for the IT domain *

Sotirios Katsikeas[1 [0000−0001−8287−3160]], Simon Hacks[1 [0000−0003−0478−9347]], Pontus Johnson[1 [0000−0002−3293−1681]], Mathias Ekstedt[1 [0000−0003−3922−9606]], Robert Lagerström[1 [0000−0003−3089−3885]], Joar Jacobsson[2], Max Wällstedt[2], and Per Eliasson[2]

[1] Division of Network and Systems Engineering, KTH Royal Institute of Technology, Stockholm, Sweden
{sotkat|shacks|pontusj|mekstedt|robertl}@kth.se
https://www.kth.se/nse
[2] foreseeti AB, Stockholm, Sweden
{joar.jacobsson|max.wallstedt|per.eliasson}@foreseeti.com
http://www.foreseeti.com

Abstract. Cyber-attacks on IT infrastructures can have disastrous consequences for individuals, regions, as well as whole nations. In order to respond to these threats, the cyber security assessment of IT infrastructures can foster a higher degree of safety and resilience against cyber-attacks. Therefore, the use of attack simulations based on system architecture models is proposed. To reduce the effort of creating new attack graphs for each system under assessment, domain-specific languages (DSLs) can be employed. DSLs codify the common attack logics of the considered domain.

Previously, MAL (the Meta Attack Language) was proposed, which serves as a framework to develop DSLs and generate attack graphs for modeled infrastructures. In this article, we propose coreLang as a MAL-based DSL for modeling IT infrastructures and analyzing weaknesses related to known attacks. To model domain-specific attributes, we studied existing cyber-attacks to develop a comprehensive language, which was iteratively verified through a series of brainstorming sessions with domain modelers. Finally, this first version of the language was validated against known cyber-attack scenarios.

Keywords: Meta Attack Language · Threat Modeling · Attack Simulation · Attack Graphs · Domain Specific Language · IT Infrastructure.

* 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, the Swedish Energy Agency, and the Swedish Governmental Agency for Innovation Systems (Vinnova).

1 Introduction

Today, our society is heavily dependent on IT infrastructures. Another fact is that cyber-attacks on IT infrastructures can have disastrous consequences for individuals, regions, and whole nations. One example are the recent deliberate disruptions of electrical power and energy systems [3, 20], which resulted in real-world catastrophic physical damage, like major power outage or city-wide disruptions of any service that requires electric power. But also, attacks on automated vehicles [21] and internet of things enabled attacks [25, 22] are good examples of IT related cyber-attacks.

It is therefore necessary to keep such critical IT infrastructures secure. In order to respond to these threats, the assessment of IT infrastructure's cyber security can foster a higher degree of safety and resilience against cyber-attacks.

However, such an assessment is difficult. In order to identify vulnerabilities, the security-relevant parts of the system must be understood and all potential attacks have to be identified [17]. There are three challenges related to these needs: First, it is challenging to identify all relevant security properties of a system. Second, it might be difficult to collect this information. Last, the collected information needs to be processed to uncover all weaknesses that can be exploited by an attacker.

Attack graphs have been previously proposed as a method to assess security on larger system architectures. This approach is gaining in popularity, both in academia and in industry the last years.

Hitherto, we have proposed the use of attack graph simulations based on system architecture models (e.g., [4, 9]) to support these challenging tasks. Our approaches facilitate the modeling of systems and simulating cyber-attacks against them, in order to identify the greatest weaknesses. This can be imagined as the execution of a great number of parallel virtual penetration tests. Such an attack simulation tool enables the security assessor to focus on the collection of the information about the system required for the simulations, since the simulation tackles the first and third challenges.

As the previous approaches rely on a static implementation, we propose the use of MAL (the Meta Attack Language) [11]. This framework for domain-specific languages (DSLs) is used to define which information about a system is required and specifies the generic attack logic. Then, MAL automatically generates attack graphs corresponding to security simulations involving the modeled system. Since MAL is a meta language (i.e. the set of rules that should be used to create a new DSL), no particular domain of interest is represented.

Using MAL threat/attack DSLs for many kinds of domains can be defined, as for example industrial control systems, vehicles, IoT, etc. The goal of using DSLs is to make the DSLs as detailed and domain specific as possible in order to be able to get precise results, valid for the specific domains. More specifically, the goal is to capture the specifics of the domains, attack vectors, design strengths and weaknesses. However, these domains also share a lot of common properties and designs. Example of commonalities are that software is executed among all of these domains, execution stacks like virtual machines or operating systems

(OS) are used, software communicates over networks, there are accounts with privileges and many more. Therefore, we can conclude that even if we want to capture specifics, in the end, it would be a redundant waste of effort to capture the same fundamental information more than once for all the different DSLs.

We therefore propose to design a core DSL that will cover the basic and common structure of software systems and IT infrastructure. Our goal with this work is to capture the basic architecture for future DSLs. Therefore, this work aims to create and evaluate a MAL-based DSL, named coreLang, that would have a high level of abstraction and will, therefore, be suitable for modeling generic IT infrastructures. Ideally, the aim for corelang is to cover the basics for all possible domains. Then, due to this higher level of abstraction, coreLang could be extended to create other MAL-based DSLs. This is a large task and our current ambition is to get the fundamentals correct for some basic domains (such as normal enterprise IT and control systems). In this paper, the first release of coreLang will be presented.

Following, we will present related work before a brief introduction to MAL, the framework that we mainly use in this work, this is done in Section 3. Next, we detail the facilitated research method in Section 4. Then, in Section 5 the language that was developed is presented in detail. To give a better understanding of the capabilities of the language, we created an example model of the IT infrastructure part of the Ukrainian cyber-attack scenario, which is presented in Section 6. Finally, the validation and discussion about this work is done in Section 7, which is followed by the conclusion.

2 Related Work

This work is related to three domains of previous work: attack/defense graphs, model-driven security engineering, and information technology (IT) security.

First, as already mentioned, attack/defense graphs are widely applied as a formalism for security analysis. Second, there are DSLs for the security analysis of software and system models defined in the domain of model-driven security engineering. Finally, due to the fact that coreLang is designed to be applied in the domain of IT security, the results of existing IT attack studies are utilized for the evaluation of the language.

The concept of attack trees is based on the works by Bruce Schneier [23, 24]. Attack trees were formalized by Mauw & Oostdijk [16] and extended to include defenses by Kordy et al. [13]. As summarized in [14], there are several approaches to elaborating on attack graphs (e.g., [10, 26]). Based on the theoretical achievements of previously presented papers, various tools using attack graphs have been developed. These tools largely operate by collecting information regarding existing systems or infrastructures and automatically creating attack graphs based on this information. For example, the topological vulnerability analysis (TVA) tool [18] models security conditions in networks and uses a database of exploits as transitions between security conditions.

A sub-domain of attack graph modeling, that is of more interest to us, focuses on probabilistic attack graphs (e.g., facilitating Bayesian networks). In [6], the authors applied the TVA tool to generate attack graphs, transform generated graphs into dynamic Bayesian networks, and enrich the Bayesian networks using probabilities based on common vulnerability scoring system (CVSS) scores. The CVSS was also utilized by [28] to model uncertainties in attack structures, attacker actions, and alert triggering.

Several DSLs have been built in MAL serving as good examples of the capabilities a MAL-based DSL has and how it can be developed. These languages provide the capability to model a system's design based on its components and their interactions. Furthermore, such languages also facilitate the modeling of security properties such as constraints, requirements, or threats. One example of a MAL-based DSL is vehicleLang [12], which is a DSL for modeling cyber-attacks on modern vehicles. Another example is a simplistic core language, that only contains the most common IT entities and attack steps and is included in the presentation of MAL [11]. Finally, another MAL-based DSL that will soon be published but some parts of it were used as inspirational blueprints for developing coreLang, is awsLang, which is a DSL for modeling Amazon Web Services environments [5].

Apart from the languages mentioned before, there exist some security languages which do not support automated analysis purposes [15, 2]. They offer only the capability to model security relevant properties. An analysis needs to be conducted manually without any further support.

Approaches using attack graphs and system modeling have been united in some previous works (e.g., P^2CySeMoL [9], and securiCAD [4]). The core concept of these methods is to generate probabilistic attack graphs automatically from a given system specification. Attack graphs serve as inference engines to produce predictive security analysis results from system models.

MAL, that will be briefly presented in the next section, is a modeling and simulation framework based on graphical models. It combines attack graphs with conceptual graphical software system modeling techniques.

3 MAL

For a detailed overview of the MAL, we refer readers to our original paper, which focuses on core grammar, syntax, formalism, and additional details regarding the MAL [11]. However, for completeness, a short presentation of the MAL is provided below.

First, a DSL created with MAL contains the main elements that are found on the domain under study. Those are called `assets` in MAL. The assets contain `attack steps`, which represent the actual attacks/threats that can happen on them.

An attack step can be connected with one or more following attack steps to create an attack path. Those are used to create attack graphs which are facilitated when the simulation is run. Attack steps can be either of the type OR

or the type AND, indicating that performing any individual parental attack step is required (OR) or performing all parental attack steps is required (AND) for the current step to be performed. Additionally, each attack step can be associated with specific types of risks. The risks can be any combination of confidentiality (C), integrity (I), and availability (A) and are specified in brackets after the attack step name. Furthermore, defenses are entities that do not allow connected attack steps to be performed if they have the value TRUE. Finally, probability distributions can be assigned to the attack steps in order to represent the effort needed to complete the related attack step.

Assets also have `associations` between each other that describe the relations between them. Inheritance between assets is also possible and each child asset inherits all the attack steps of the parent asset. Additionally, the assets can be organized into categories.

Next, a short example of how a MAL-based DSL looks like follows. On this example, four modeled assets can be seen together with the connections of attack steps from one asset to another. For example, if an attacker performs *phish* on the `User`, it is possible then to reach *obtain* on the associated `Password` and as a result finally perform *authenticate* on the associated `Host`. In the last lines of the example the associations between the assets are defined.

```
category System {
  asset Network {
    | access
      -> hosts.connect
  }

  asset Host {
    | connect
      -> access
    | authenticate
      -> access
    | guessPassword
      -> guessedPassword
    | guessedPassword [Exponential(0.02)]
      -> authenticate
    & access {C,I,A}
  }

  asset User {
    | attemptPhishing
      -> phish
    | phish [Exponential(0.1)]
      -> passwords.obtain
  }

  asset Password {
```

```
    | obtain {C}
      -> host.authenticate
  }
}

associations {
  Network [networks]
    * <-- NetworkAccess --> *
    [hosts] Host
  Host [host]
    1 <-- Credentials --> *
    [passwords] Password
  User [user]
    1 <-- Credentials --> *
    [passwords] Password
}
```

4 Method

Design Science Research (DSR) is a widely applied and accepted means of developing artifacts in information systems research. It offers a systematic structure for developing artifacts, such as constructs, models, methods, or instances [8]. The application of DSR is appropriate since research objectives are guiding the development of artifacts. We adopted the approach presented by Peffers et al. [19], which is split into six individual steps and two potential feedback loops, as described below.

1 **Identify Problem & Motivate:** In the introduction section of this paper the importance of cyber security assessment of IT infrastructure's was highlighted. Additionally, MAL is already proposed as a tool to provide an environment for security assessors including already known attacks on assets. Therefore, the problem that we try to solve is to devise a tool, in our case a DSL, that will be able to model the common elements found in almost all IT infrastructures using MAL.
2 **Define Objectives:** To identify vulnerabilities, the security-related elements of an IT infrastructure must be understood and all potential attack types must be identified. There are three main challenges related to these goals. First, it is difficult to collect specific information regarding all potential attacks on each security-related element. Second, collected information must be processed and modeled in a language in a manner that accurately models potential attacks and their outcomes. Finally, a key objective of this work, and a challenge to overcome, is to find the appropriate level of modeling abstraction so that the developed language could be reused as a foundation for other future MAL-based DSLs.
3 **Design & Development:** To design artifacts, we based our attack language on MAL [11]. To model domain specific properties in our language,

we relied on brainstorming sessions with people from foreseeti AB which can be considered as domain experts, since they are in close contact with IT architects and security officers of many different industries and they provided us with lessons learned from over five years of development and usage of the securiCAD tool which contains an attack graph generating DSL. During those sessions, we presented our perception of the corresponding topic under study and they provided comments back to us. The comments we got were used as an early indication if we are moving towards the right direction. These workshops were conducted for two hours on a weekly basis for five months.

4&5 **Demonstration & Evaluation:** To demonstrate our approach, we created a set of use case models. First, we model typical IT attacks that are well known and relevant to the part of the language that was under active development at that time. Thus, for every new part that was developed in the language, demonstrating use case models were created. Second, we created a model of the IT infrastructure part of the Ukrainian cyber-attack scenario [3].

The evaluation of the use case models created was done through evaluation sessions with the same domain experts. The results of the simulations on the created models were discussed with domain experts and relevant feedback was given back to us. If the feedback suggested that improvements or changes are needed in the language to better reflect the reality, an iteration of the step 3 was done.

Finally, we also see the presented, on this paper, version of the coreLang as the result of a the first development iteration.

6 **Communication:** This paper is the first official communication of coreLang. Our research is communicated by the publication of the paper itself and presentation at the conference.

coreLang is an open-source project, and the code behind the language is publicly available on our GitHub repository[3].

5 CoreLang

Six different main asset categories have been included in coreLang (see Figure 1): system, vulnerability, user, identity and access management (IAM), data resources, and networking. In this section, those categories and the related design decisions will be explained in detail.

5.1 System

The first category *System* is the collection of assets that usually represent the computing instances in an environment, and thus are the natural attack surface.

First, an asset called `Object` was created (inspired by the Java object) that provides common functionality to all inheriting assets. Basically, an `Object` is

[3] https://github.com/mal-lang/coreLang/tree/stable

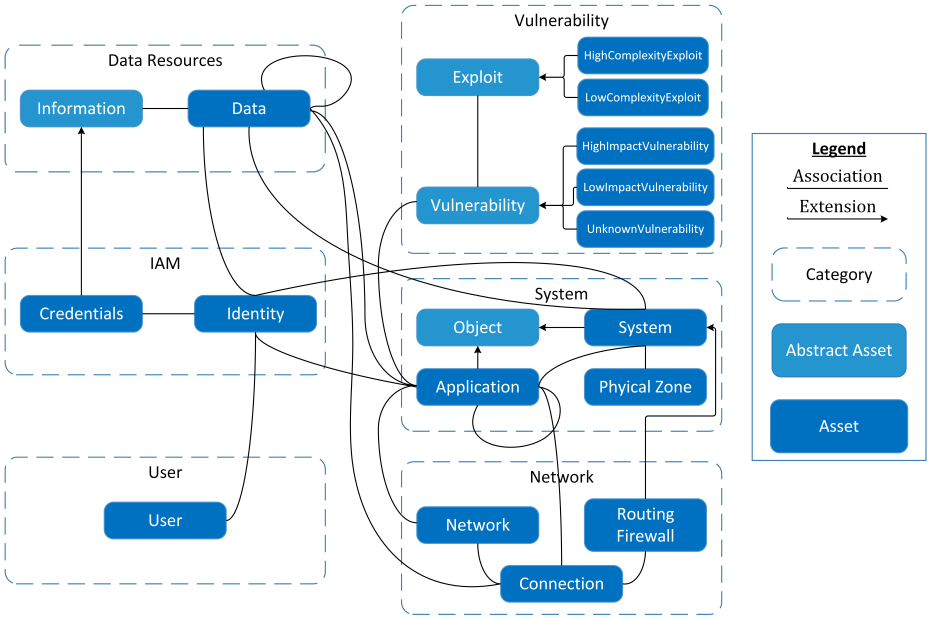

Fig. 1. Overview of assets and their associations in coreLang

the simplest form of an asset that can have a `Vulnerability`. Then, `Object` is specialized into two child assets, `System` and `Application`.

The `System` asset specifies the hardware on which `Applications` can run. After achieving *physical access*, the attacker can try to *authenticate* on it and/or perform a *denial of sevice* attack on all the `Applications` that are executed on it. Except *physical access*, two more levels of access are modeled on a `System`. The first one is the *specific access*, which models the ability to locally connect to the hosted applications after authenticating. Then, there is also the *full access*, which is gained after a "high-privilege" `Identity` authenticates itself or is compromised.

On the other hand, the `Application` asset specifies everything that is executed or can execute other applications. For that reason, the `Application` asset is more complex and includes a wider range of attack steps. With the same way that is modeled for the `System` asset, in order to get *access* on an `Application` two previous attack steps need to be compromised, the first one is some kind of *connect* and second a successful *authentication*. There are three possible ways of "connecting" to an `Application`: i) either via *local connect*, which occurs because any identity with "low-privilege" access on the executing instance is assumed to be able to locally (i.e., on the same host application, using loopback) interact with the executed applications, ii) via *network connect*, which can happen when an application is exposed on a network, or iii) via *identity local interaction* which happens when the associated "low-privilege" `Identity` is compromised or authenticates itself. More details about the `Identities` will

be provided in the corresponding subsection 5.4. It is worth noting that some attack steps (e.g. *codeExecution*) were adopted from awsLang [5].

To clarify the definition of the `Application` asset, the relevant MAL code snippet is presented below. Additionally, how the attack steps of the `Application` asset are connected with the attack steps of other assets, with which `Application` has associations with, is represented on the MAL generated graph in Figure 2.

```
asset Application extends Object
{
  | localConnect
    ->  localAccess,
        connectLocalInteraction,
        attemptUseVulnerability

  & localInteraction
    ->  appExecutedApps.localConnect,
        attemptUseVulnerability

  | attemptUseVulnerability
    ->  vulnerabilities.attemptAbuse

  | networkConnect
    ->  networkAccess,
        connectLocalInteraction,
        attemptUseVulnerability

  | accessNetworkAndConnections
   -> networks.access,
      appConnections.applications.networkConnect,
      appConnections.transmit,
      appConnections.transmitResponse

  | authenticate
    ->  localAccess,
        networkAccess

  | access {C,I,A}
    ->  read,
        modify,
        deny,
        appExecutedApps.access,
        containedData.attemptAccess,
        accessNetworkAndConnections,
        hostApp.localConnect

  | codeExecution
```

```
    ->  access,
        executionPrivIds.assume,
        modify

  | read {C}
    ->  containedData.attemptRead

  | modify {I}
    ->  containedData.attemptAccess

  | deny {A}
    ->  containedData.attemptDelete
...
}
```

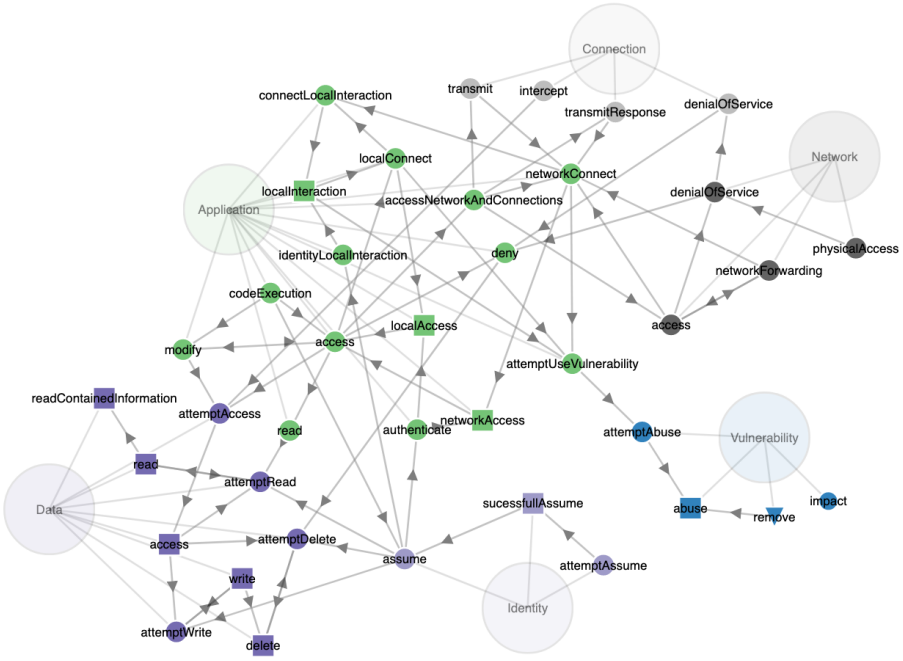

Fig. 2. Graph representing the attack steps, and their connections, of the Application asset in coreLang

Lastly, this category contains `PhysicalZone`, which is the location where `Systems` are physically deployed. If *physical access* is performed on a `PhysicalZone`, then the attacker is able to *connect* and get *physical access* on the `Systems` that are part of the `PhysicalZone`.

5.2 Vulnerability

The basic idea of creating a MAL-based language is to provide a set of already known attack steps to the modeler. However, this incorporates two types of shortcomings. First, we concentrate on known attack steps. But, there are also attack steps that are not known yet. Second, the level of abstraction selected for coreLang is another shortcoming. Because of that, we cannot provide all possible attack steps upfront, as the attack steps are very diverse for different assets.

To overcome these issues, we provide a set of `Vulnerabilities` and `Exploits`. On the one hand, these assets can be used as a foundation for other language developers. On the other hand, we provide a standard and abstract set of `Vulnerability` and `Exploit` that represent three discrete levels of importance. These can be used by the end-user to model attack steps that are not known at the time of creating the language. Basically, any `Object` can have a `Vulnerability` that leads to different levels of *impact* to the vulnerable `Object`. This `Vulnerability` can then be facilitated by an `Exploit` that can have different levels of *complexity*, for example a `Low Complexity Exploit` can be exploited in order to abuse a `High Impact Vulnerability`.

5.3 User

This category contains the representation of a `User`. The `User` serves as attack surface for *social engineering attacks*. The most apparent attack that is modeled in this asset is the *phishing* attack of the `User`, which can lead to either *credential theft* or *takeover* of the user's computer. The latter one allows a malicious backdoor connection to be opened to the user's computer, which the attacker can then use to further compromise the same machine or perform lateral movement.

5.4 IAM

Identity and access management (IAM) is an accepted concept to manage different identities representing users and their access to certain applications [27]. Therefore, the IAM category in coreLang is comprised of the `Identity` asset that represents a user group, and the `Credentials` asset that can be associated with one or more `Identities`. After legitimate authentication or an illegitimate compromise of an `Identity`, the attacker *assumes* its privileges. Thus, both legitimate and illegitimate access is represented. As already mentioned, access to an `Identity` is usually secured by means of `Credentials`. Those `Credentials` can be stolen/guessed by the attacker directly (e.g., due to brute-force) or the `User` can be convinced to enter them by themselves (e.g., due to social engineering, like phishing, as mentioned previously).

`Identities` are, however, not only associated with `Credentials` but also with `Users` and `Objects`, like `Systems` and `Applications`, as seen in Figure 1. An `Identity` associated with a `User` models the usage of that `Identity` by a `User` or by an `Application` running under the identity's privileges. Additionally,

an `Identity` that is associated with a `System` or an `Application` represents the privileges that the `Identity` has over it.

When it comes to IAM on a `System`, two different levels of Identity-System associations are modeled. First there is the *Low Privilege Access* which provides individual level access on a `System` from an `Identity`. Second, there is the *High Privilege Access*, which is equal to gaining access on the `System` as every possible associated `Identity`. The reason for these two levels of privileges is caused by the common separation between simple and admin users was done.

On the `Application` side, there are three different levels of Identity-Application associations modeled. First, there is the *Low Privilege Application Access*, which only provides local interaction with the `Application`. But, this simple interaction is the only prerequisite for many `Vulnerabilities` and, therefore, can result in severe compromise of the whole infrastructure [29]. Second, there is the *Execution Privilege Access*, which represents the fact that every `Application` is executed with the privileges of an `Identity`. In this case, if the `Application` is compromised, then the privileges of the associated `Identity` should also be compromised. Finally, there is the *High Privilege Application Access*, which models the higher level of privileges over an `Application` and if such privileges get compromised, all the child/executed applications should also be compromised.

5.5 Data Resources

This category groups the assets that are usually communicated. First, the `Information` asset is defined as a conceptually abstract concept that is then incorporated in the `Data` asset. The `Data` asset represents any form of data that can be stored or transmitted. This asset was heavily based on the homonymous asset found on awsLang [5]. An attacker can perform the classical actions of *read*, *write*, and *delete*, which all are modeled as attack steps. Those attack steps can be reached either by compromising the `Identity` that is associated with the `Data` or by compromising the asset that contains those `Data`, as for example the `Connection` or the `Application` asset.

5.6 Networking

The last category is concerned with networking related assets. First, the `Network` asset is defined. An attacker that has *physical access* to a `Network` can perform a *denial of service* attack by physically destroying the network medium. But if the attacker has network *access*, it is able to *network connect* and perform *denial of service* to all the network exposed `Applications` as well as attempt *network forwarding* to other neighbouring `Networks`.

The border of every `Network` is defined by a `RoutingFirewall`, which specifies a border router with firewall capabilities that can interconnect many networks. The `RoutingFirewall` is modeled as a `System` and is therefore subject to possible `Vulnerabilities`. A `Vulnerability` can lead to *full access* which results in bypassing of all the network rules defined by the firewall.

Lastly, there is the `Connection` asset, which specifies the existence of a connection between `Applications` or `Networks` and could consequently be used for lateral movement by an attacker. Each `Connection` that is associated with the `RoutingFirewall` represents a *connection rule* meaning that the firewall allows the forwarding of the associated traffic. If a `Connection` is not associated with the `RoutingFirewall`, then the corresponding traffic is prohibited.

While `Applications` can be associated with connections in a single way, `Networks` have three different types of associations with a `Connection` in regard to the three possible rules that can be found on a firewall. Those are, first the simple *Network Connection*, which models a bidirectional connection rule, second the *Out Network Connection*, which models a uni-directional connection rule that solely allows outgoing traffic of this `Network`, and third the *In Network Connection* which models the uni-directional connection rule that allows only incoming traffic into that `Network`.

6 Example Model

As already mentioned, coreLang aims to provide a high level of abstraction in the models created. For that reason, it is suitable for modeling a wide variety of IT infrastructures with a high level of abstraction. To demonstrate the application of coreLang, we use it to model a simplified version of the Ukrainian cyber-attack scenario and the way we interpreted it from an analysis that was published on it [3], that described how the attackers got a foothold on the internal networks.

The model was created in securiCAD [4], which is a software tool developed by foreseeti AB for performing virtual attack simulations on models of IT architectures, and which also allows to create MAL-based models. In Figure 3, the created model is presented.

The attack description below is based on the simulation results which are in accordance to what the analysis reports from that specific cyber-attack state [3].

The simulated attack scenario, which is presented on the generated attack graph in Figure 4, is the following. First, the attacker performs a *social engineering attack* towards the `User` by sending a malicious payload file attached on a Microsoft Office Word document sent via email. Then, the `User` opens the document and executes the payload. Due to a `Vulnerability` on the Office Word `Application`, the malicious payload is executed and the vulnerability is *exploited* allowing the attacker to successfully *execute code* and take control of the user's Office Word application. The attacker has *assumed* the privileges of the `Identity` associated with the Office Word application and, therefore, has *access* on the `Application`. That, in turn, allows a *local connect* to the Windows operating system, which is an `Application`. Unluckily enough, the operating system is also vulnerable and the attacker can attempt exploit this also to gain *access* on it. Next, due to poor security policy enforcement, the `Credentials` for another workstation located in the same office `Network` is stored in the operating system in a text file and is accessible by the attacker. The attacker can also *access* the office `Network` and *network connect* to the second workstation. By having

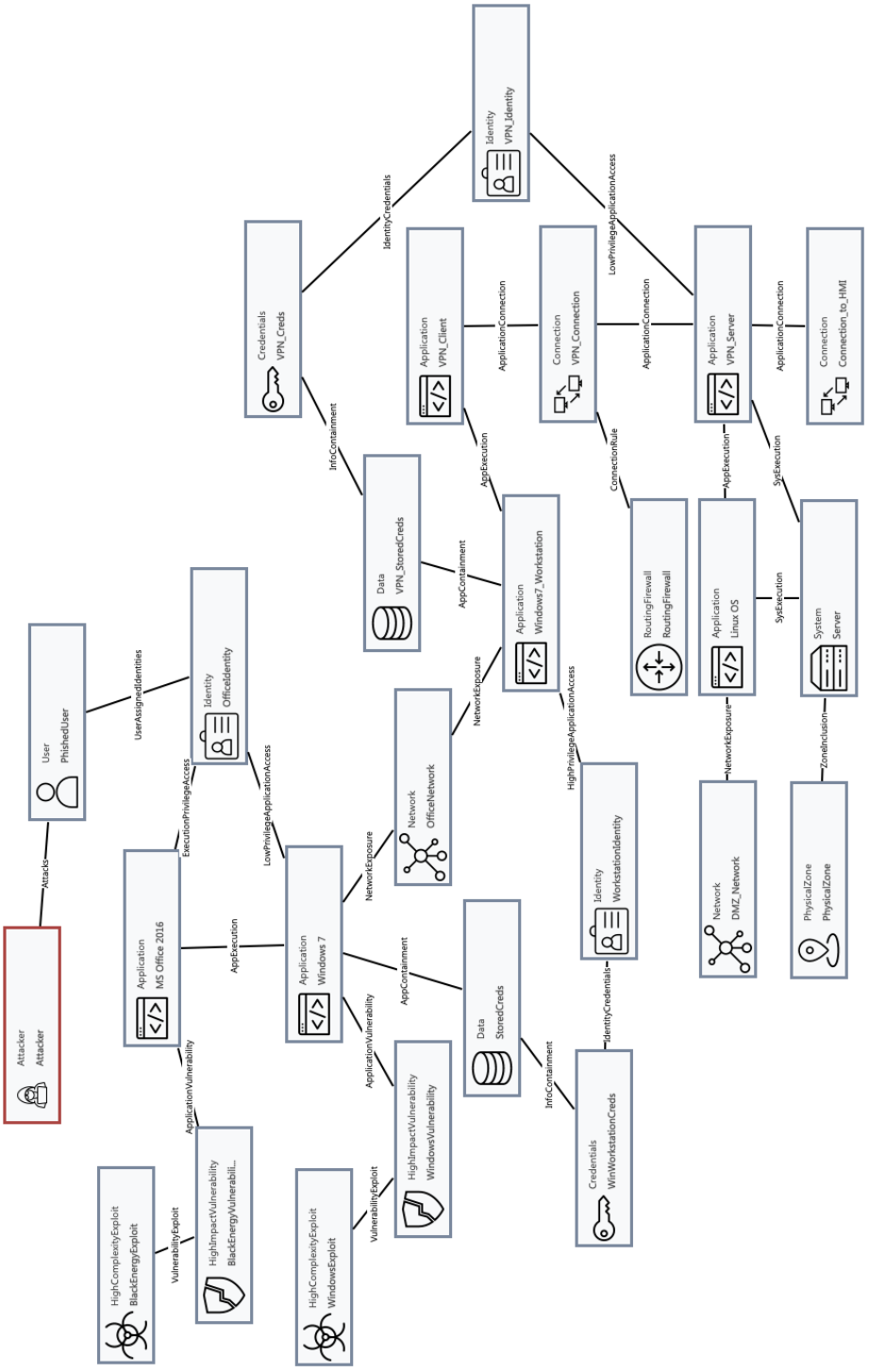

Fig. 3. coreLang model in securiCAD of the IT infrastructure of the Ukrainian cyber-attack example model

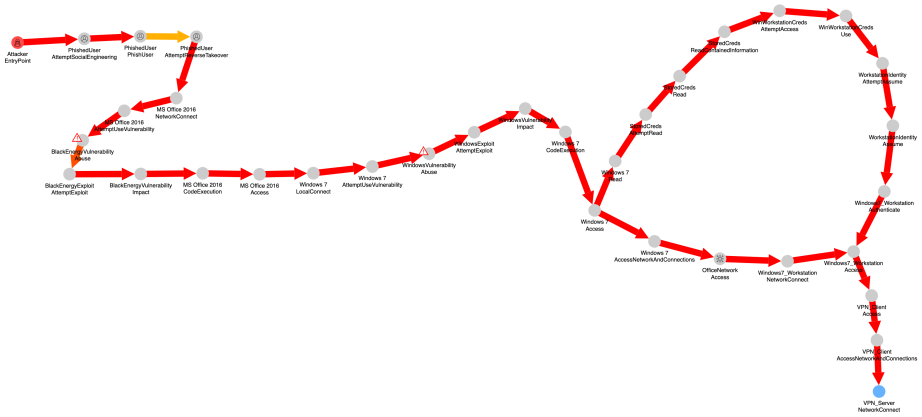

Fig. 4. coreLang generated attack graph for the Ukrainian cyber-attack example model

the `Credentials` that attacker is also able to *authenticate* and gain *access* on this workstation. Gaining *access* on this workstation is proven to be resourceful since a VPN client `Application` is executed on that operating system. Again, the attacker was lucky, since the VPN `Credentials` are stored as file on that workstation. By accessing them, is then able to *network connect* on the VPN server on the DMZ `Network` and then use the `Connection` to human-machine interface (HMI) in order to send control commands on the HMI controlling the power grid.

7 Validation & Discussion

According to Hevner et al. [8], five methods can be used to evaluate the output of DSR: observations, analysis, experiments, tests, and descriptions. Because developing coreLang was similar to developing source code, tests were selected as an evaluation method. This decision was made based on the fact that testing is widely used in application development and commonly accepted as a means for ensuring that an application behaves as intended.

In our work, the tests were implemented as use case tests [7]. More specifically, as the development of the language proceeded through different asset categories, some real-world use cases that describe a variety of common IT attacks, that should be supported by the language, were provided to us by our collaborators and domain experts from foreseeti AB. Then, models for those test use cases were created and simulations were ran. The results of those simulations were then discussed with the same domain experts in evaluation sessions and feedback was provided back to us. If the feedback suggested that improvements or changes are needed in the language to better reflect the reality, an iteration of the development phase for this asset category and evaluation session was done.

By using use cases for validation, we ensure that the generated language fulfills the requirements of having a high level of abstraction while it retains its correctness. Additionally, the language still covers real-world scenarios that are typically requested from the IT infrastructure modelers that will eventually be the users of this language.

Through the evaluation sessions we had, the goal was to use the experiences of our collaborators in order to improve coreLang. Those experiences were related to parts of the models that were previously cumbersome to model using the existing tools and MAL languages, and also experiences about cases that were not at all modelable previously. Some examples of such incomplete modeling cases are IAM and networking.

One problem that occurred during development was caused by the higher level of abstraction that we wanted to retain in the language. Some common IT elements, interactions or relations could not be explicitly modeled. Our solution to this problem was to make assumptions in the design of the language that allow the more specific cases to be modeled with a higher level of abstraction. These assumptions are documented in the language itself. One characteristic example is that on coreLang, there is no asset specifying an operating system nor a guest operating system, both cases can be modeled by having two application assets associated with each other in a hierarchical manner where one is the executor and the other is the executee. This type of recursive design approach can be considered a strength of the language since it allows the modeling of different nested execution cases (e.g. the case of a guest OS running on a VM under a host OS and the case of an OS running an software application) using a single solution.

Another example of such an assumption that was made is related to the three different application privilege levels that are available in the language. More specifically, the use of the two discrete access levels, namely low and high privilege application access, was inspired by the fact that typically an application is either being executed under a simple user's (low) or root/admin's (high) privileges and it always is associated with one type of them, in our case called execution privilege.

Another problem that we had to solve, again related to the level of abstraction, was that it would not be clear to users of the language to understand how exactly each of the included assets should be used in a model. To solve this problem, first, a proper name for each one of the assets was selected, then, second, a short documentation text about each asset was included in the language.

8 Conclusion & Future Work

Assessing the cyber security of IT infrastructures is becoming increasingly important as the number of IT security issues and cyber-attacks increases. This article presented coreLang, which is a MAL-based domain specific language for the abstract IT domain.

coreLang supports a high level of modeling abstraction and is therefore suitable for modeling generic IT infrastructures. This higher level of abstraction makes the developed language easier to expand since it is easier to use it as a foundation for many different MAL-based DSLs. Finally, coreLang is an open-source project to which anyone can contribute[4].

There are several potential directions for future work and future work is something expected since coreLang is still a work in progress and this is the first release.

First, coreLang could be used as a foundation for the creation of extensions that will allow the language to become more specific when needed. Since coreLang captures the basic IT architecture and has a high level of abstractions it could be used as a foundation for future MAL-based DSLs. For example, some new assets could be added in an extension file that will enhance the language with capabilities for better specific operating system and software modeling.

Second, given the fact that software vulnerabilities are covered in a comprehensive way an extension could add an on par with the common vulnerability scoring system (CVSS) [1] representation of software vulnerabilities. Then, another future addition on the language would be to add defenses that are able to either completely stop or make the attacks, that are already modeled, harder to perform. This would allow more flexible models to be simulated without having to change the assets that are placed in the model.

Finally, it must be noted that the goal of coreLang is to be able to model the common IT infrastructures but not to provide the correct probabilities and probability values for all the included attack steps. This should be considered as more work that needs to be done separately or be part of the DSL that will use coreLang as their foundation.

References

1. CVSS v3.1 Specification Document, https://www.first.org/cvss/v3.1/specification-document
2. Almorsy, M., Grundy, J.: Secdsvl: A domain-specific visual language to support enterprise security modelling. In: Software Engineering Conference (ASWEC), 2014 23rd Australian. pp. 152–161. IEEE (2014)
3. Defense Use Case: Analysis of the cyber attack on the Ukrainian power grid. Electricity Information Sharing and Analysis Center (E-ISAC) (2016)
4. Ekstedt, M., Johnson, P., Lagerström, R., Gorton, D., Nydrén, J., Shahzad, K.: securiCAD by foreseeti: A CAD tool for enterprise cyber security management. In: Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International. pp. 152–155. IEEE (2015)
5. Engström, V., Johnson, P., Lagerström, R.: Automating Cyber Attack Simulations Against Amazon Web Services Environments (To be published) (2020)
6. Frigault, M., Wang, L., Singhal, A., Jajodia, S.: Measuring network security using dynamic bayesian network. In: Proc. of the 4th ACM workshop on Quality of protection. pp. 23–30. ACM (2008)

[4] https://mal-lang.org/coreLang/

7. Hasling, B., Goetz, H., Beetz, K.: Model based testing of system requirements using uml use case models. In: 2008 1st International Conference on Software Testing, Verification, and Validation. pp. 367–376. IEEE (2008)
8. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS quarterly **28**(1), 75–105 (2004)
9. Holm, H., Shahzad, K., Buschle, M., Ekstedt, M.: P^2CySeMoL: Predictive, probabilistic cyber security modeling language. IEEE Transactions on Dependable and Secure Computing **12**(6), 626–639 (2015). https://doi.org/10.1109/TDSC.2014.2382574
10. Ingols, K., Chu, M., Lippmann, R., Webster, S., Boyer, S.: Modeling modern network attacks and countermeasures using attack graphs. In: Computer Security Applications Conference, 2009. ACSAC'09. Annual. pp. 117–126. IEEE (2009)
11. Johnson, P., Lagerström, R., Ekstedt, M.: A meta language for threat modeling and attack simulations. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. p. 38. ACM (2018)
12. Katsikeas, S., Johnson, P., Hacks, S., Lagerström, R.: 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 (2019)
13. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of attack–defense trees. In: International Workshop on Formal Aspects in Security and Trust. pp. 80–95. Springer (2010)
14. Kordy, B., Pièvre-Cambacédès, L., Schweitzer, P.: Dag-based attack and defense modeling: Don't miss the forest for the attack trees. Computer science review **13**, 1–38 (2014)
15. Lund, M.S., Solhaug, B., Stølen, K.: Model-driven risk analysis: the CORAS approach. Springer Science & Business Media (2010)
16. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: International Conference on Information Security and Cryptology. pp. 186–198. Springer (2005)
17. Morikawa, I., Yamaoka, Y.: Threat tree templates to ease difficulties in threat modeling. In: 2011 14th International Conference on Network-Based Information Systems. pp. 673–678 (Sep 2011). https://doi.org/10.1109/NBiS.2011.113
18. Noel, S., Elder, M., Jajodia, S., Kalapa, P., O'Hare, S., Prole, K.: Advances in topological vulnerability analysis. In: Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology. pp. 124–129 (Mar 2009). https://doi.org/10.1109/CATCH.2009.19
19. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems **24**(3), 45–77 (2007)
20. Petermann, T., Bradke, H., Lüllmann, A., Poetzsch, M., Riehm, U.: Was bei einem Blackout geschieht: Folgen eines langandauernden und großflächigen Stromausfalls, vol. 662. Büro für Technikfolgen-Abschätzung (2011)
21. Petit, J., Shladover, S.E.: Potential cyberattacks on automated vehicles. IEEE Transactions on Intelligent Transportation Systems **16**(2), 546–556 (2015)
22. Prokofiev, A.O., Smirnova, Y.S., Silnov, D.S.: The internet of things cybersecurity examination. In: 2017 Siberian Symposium on Data Science and Engineering (SSDSE). pp. 44–48 (2017)
23. Schneier, B.: Attack trees. Dr. Dobb's journal **24**(12), 21–29 (1999)
24. Schneier, S.: Lies: digital security in a networked world. New York, John Wiley & Sons **21**, 318–333 (2000)

25. Stellios, I., Kotzanikolaou, P., Psarakis, M., Alcaraz, C., Lopez, J.: A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services. IEEE Communications Surveys Tutorials **20**(4), 3453–3495 (2018)
26. Williams, L., Lippmann, R., Ingols, K.: GARNET: A graphical attack graph and reachability network evaluation tool. Springer (2008)
27. Witty, R.J., Allan, A., Enck, J., Wagner, R.: Identity and access management defined. Research Study SPA-21-3430, Gartner (2003)
28. Xie, P., Li, J.H., Ou, X., Liu, P., Levy, R.: Using Bayesian networks for cyber security analysis. In: Dependable Systems and Networks (DSN), 2010 IEEE/IFIP Int. Conf. on. pp. 211–220. IEEE (2010)
29. Yan, D., Liu, F., Jia, K.: Modeling an information-based advanced persistent threat attack on the internal network. In: ICC 2019 - 2019 IEEE International Conference on Communications (ICC). pp. 1–7 (2019)